Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Learning Segmentation from Multiple Datasets with Different Label sets

Diploma Thesis

Elnaz Babayeva

Supervisor: Ing. Milan Šulc

Master program: Open Informatics
Specialization: Artificial Intelligence

Prague, May 24, 2019

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Babayeva** Jméno: **Elnaz** Osobní číslo: **399335**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Učení segmentace z několika datasetů s odlišnými množinami labelů

Název diplomové práce anglicky:

Learning segmentation from multiple datasets with different label sets

Pokyny pro vypracování:

1. Review the state-of-the-art in deep learning methods for image segmentation.
2. Review existing datasets for image segmentation.
3. Review learning methods (for diverse tasks) for training on multiple datasets with different label sets.
4. Based on 1.-3., focus on either semantic or instance segmentation.
5. Propose a procedure that improves performance if additional datasets with different label sets are available. Optionally, considered datasets with incomplete annotation (e.g. bounding boxes instead of segmentation mask). Towards this end, analyse relations between label sets of the selected datasets.
6. Implement the proposed method, evaluate it on publicly available data, and compare the results with the state-of-the-art.

Seznam doporučené literatury:

[1] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN." International Conference on Computer Vision 2017.
[2] Ronghang Hu, Piotr Dollár, Kaiming He, Trevor Darrell, and Ross Girshick. "Learning to Segment Every Thing." arXiv preprint arXiv:1711.10370 (2017).
[3] Ian Goodfellow, Yoshua Bengio, Aaron Courville. "Deep Learning." MIT Press, 2016.
[4] Damien Fourure, Rémi Emonet, Elisa Fromont, Damien Muselet, Alain Trémeau, and Christian Wolf. "Semantic segmentation via multi-task, multi-domain learning." In Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), pp. 333-343. Springer International Publishing, 2016.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Milan Šulc, skupina vizuálního rozpoznávání FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **09.02.2018** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2019**

Ing. Milan Šulc podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomantka bere na vědomí, že je povinna vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

07.05.2019

Datum převzetí zadání

Podpis studentky

Declaration

I hereby declare I have written this diploma thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

In Prague, May 24, 2019

.....
Elnaz Babayeva

Abstract

The thesis deals with the task of object instance segmentation, which aims to learn a per-pixel mask of every object instance of defined classes within an image. State-of-the-art convolutional neural networks require large quantities of annotated training data. Segmentation masks are costly to annotate.The work focuses on training instance segmentation from multiple datasets with different label sets, and addresses the problem of missing annotations: instances of a class, which are labeled only in some datasets, may be considered the background in a dataset where the class is not labeled.

As the main contribution of the thesis, two semi-supervised methods to distinguish between unlabeled instances and background are proposed. One is based on feature similarity of object proposals, second trains a separate classification head for each dataset and uses all heads to discover unlabeled instances. Both methods were implemented as an extension of Mask R-CNN and tested on two splits of the MS COCO dataset: the first divides images into three datasets with overlapping label sets, the second divides images into three datasets with disjoint label sets. On the first split with overlapping labels, the baseline method performed 5% worse in terms of mAP compared to a fully supervised setting. The proposed methods KNN and ϵ-MHC improved mAP in the semi-supervised scenario by 1.75% and 0.72% of mAP respectively, which means 35% and 14% increase towards the bound given by full annotation. Interestingly, on the second set with disjoint labels, the proposed methods performed the same as baseline. Analysis of individual errors cases did not show noticeable changes in categorical errors or in the type of errors.

Keywords: Computer vision, segmentation, detection, missing annotations, neural network, deep learning.

Abstrakt

Diplomová práce se zabývá úlohou segmentace instancí objektů, jejíž cílem je naučit se v obrázku označit pixely oblasti každé instance objektu z definovaných tříd. Moderní konvoluční neuronové sítě vyžadují velké množství anotovaných trénovacích dat. Anotace segmentačních masek je nákladná. Práce se zaměřuje na trénování segmentačních sítí z několika datových sad s různými množinami označených tříd, a věnuje se problému chybějících anotací: instance z třídy anotované pouze v některé datové sadě, může být v jiné dataové sadě, kde anotována není, považována za pozadí.

Hlavním přínosem diplomové práce je návrh dvou metod pro rozeznávání neoznačených instancí objektů od pozadí. První metoda je založena na podobnosti příznaků detekovaných oblastí zájmu. Druhá metoda vytvoří pro každou množinu označených tříd vlastní klasifikační hlavu, a neoznačené instance vyhledává pomocí všech hlav. Obě metody byly implementovány jako rozšíření Mask R-CNN a testovány na datové sadě MS COCO rozdělené dvěma způsoby: v prvním případě byly obrázky rozděleny do tří datových sad s částečným překryvem anotovaných tříd, v druhém případě do tří datových sad s disjunktními množinami anotovaných tříd. V prvním případě s překryvem tříd by naivní řešení znamenalo zhoršení o 5% mAP oproti plné anotaci. Navržené metody KNN a ϵ-MHC zlepšují učení s chybějícími anotacemi o 1.75% mAP a 0.72% mAP, což posouvá výsledek o 35% a 14% směrem k výsledkům s plnou anotací. V druhém případě s disjunktními množinami anotovaných tříd dosahovaly navrhované metody stejných výsledků jako naivní řešení. Analýza jednotlivých chyb neodhalila významné rozdíly v jejich druhu či ve výskytu chyb v jednotlivých třídách.

Klíčová slova: Počítačové vidění, segmentace, detekce, částečná anotace, neuronové sítě, hluboké učení.

Acknowledgements

Foremost, I want to express my deep gratitude to my supervisor Milan Šulc. He guided and educated me in the fields of object detection and computer vision. His support and interest in my work helped me to pursue it further and push the boundaries. I appreciate his time, which he spent consulting my work and his valuable advice as in academia as in day-to-day life.

I would like to thank professor Matas for giving me the opportunity of joining CMP and giving me a second chance when I had hard times. I am very grateful to members of the CMP group, especially to Ahmet, Ondra, Filip, Tomáš, Klára, Matěj for creating a great, fun, and friendly environment for learning and working.

Moreover, I want to thank Stratosphere Lab, especially to Sebas, Mari, Vero, and Jan, for moral support and encouragement which I have received from them during the whole studies. Special thanks go to wITches for allowing me to become an educator and to encourage me making the world a little better. I appreciate all the moral support from my friends: Nastya L, Nastya A, Kamila K, Jakub, Kuanysh, Sasha, Dasha, Polina.

Last but not least, I am thankful to my mom and dad for their love and support and to my sisters Kamila, Leila, and Laura, who are always there for me. Most importantly, I would like to thank Yura for his love and encouragement and not letting me give up.

Contents

Chapter 1

Introduction

Visual recognition has been rapidly improving with deep learning systems using Convolution Neural Networks(CNN) [1]. We are interested in the task of object instance segmentation, which aims to learn a per-pixel mask of every object instance of defined classes within an image. State-of-the-art CNN's [2]–[7] have millions of parameters, and large quantities of labeled training data are required to learn the model. Labeling a large number of images with segmentation masks is very costly and time-consuming: a trained annotator spent 90 minutes per image with full pixel-wise semantic segmentation of one image in the Cityscapes dataset [8].

There are existing large-scale datasets such as Pascal VOC [9] and Microsoft COCO [10]. The Microsoft COCO dataset is one of the largest datasets for object instance detection and segmentation. It contains 100, 000 images with 81 predefined classes. Increasing the number of the predefined classes improves the ability to understand the whole scene captured in the image.

The objective of this thesis is to improve object instance segmentation by utilizing existing datasets with different sets of labeled classes (label sets). The resulting model should benefit from a larger number of training samples and recognize classes from all training datasets. Using multiple dataset during training is challenging: instances of a class, which is labeled only in some datasets, may be considered the background in a dataset where the class is not labeled. We denote this problem as *missing annotations*. During training of object detection network, detection of each unlabeled instance is penalized as false positive.

As the main contribution of the thesis, two semi-supervised methods to distinguish between unlabeled instances and background are proposed for instance segmentation based on object detection.

The first method is based on feature similarity of object proposals. In each epoch, a database of all annotated object instances is created. If the network predicted an instance

of an un-annotated class c, similarity to other instances of class c in the database is used to distinguish between un-annotated instances and background.

The second proposed semi-supervised method extends detection based methods by adding a separate head with classification and regression layer for each label set. The loss for proposals, which are classified as background by the head corresponding to the image label set, is dropped if the proposals have high classification score in any other head.

Both methods were implemented into the Mask R-CNN framework. Experiments are done on different splits of the MS COCO dataset: one split has datasets with distinct label sets, another has datasets with overlapping label sets. On the first set with overlapping labels, the baseline method performed 5% worse in terms of mean average precision (mAP) compared to a fully supervised setting. The proposed methods KNN and ϵ-MHC, improved mAP in the semi-supervised scenario by 1.75% and 0.72% of mAP respectively, which means 35% and 14% increase towards the bound given by full annotation.

The thesis is structured as follows: Chapter 2 provides an overview of the large-scale datasets, the state-of-the-art-approaches to instance segmentation, and the semi-supervised approaches to training from several datasets. The problem statement is described in detail in Chapter 3. The proposed methods are presented in Chapter 4, and experiments are shown in Chapter 5. The final conclusion are drawn in Chapter 6.

Chapter 2

Related Work

Deep convolutional neural networks brought significant improvements to the field of computer vision, especially in the areas of object classification, object detection, semantic segmentation, and instance segmentation tasks [4], [5], [7].

This chapter begins with the definition of the computer vision tasks that are relevant to this work (Section 2.1). The existing real-scene datasets with semantic levels of annotations are presented in Section 2.2. The state-of-the-art methods in instance segmentation are reviewed in Section 2.3, along with with the detailed explanation of Mask R-CNN [5] in Section 2.5. Section 2.6 discusses the semi-supervised approaches in joining datasets.

2.1 Relevant Tasks in Computer Vision

The task of **object detection** consists in localizing and classifying instances of every object of a predefined class within an image. Typically, the output of an object detection algorithm are the bounding boxes around the objects, and classes assigned to it.

The task of **semantic segmentation** consists in labeling each pixel of an image with a corresponding semantic label. Semantic segmentation does not perform object detection with a bounding box. The output of the semantic segmentation task is a mask, with the same size as the input image, where each pixel is classified to one of the predefined semantic labels.

The task of **instance segmentation** can be interpreted as a combination of object detection and semantic segmentation. It consists in labeling each pixel of an image with its object instance, i.e. not only finding the category it belongs to but also differentiating objects of the same category.

Figure 2.1 shows examples of different levels of annotations for object detection, semantic segmentation, and instance segmentation.

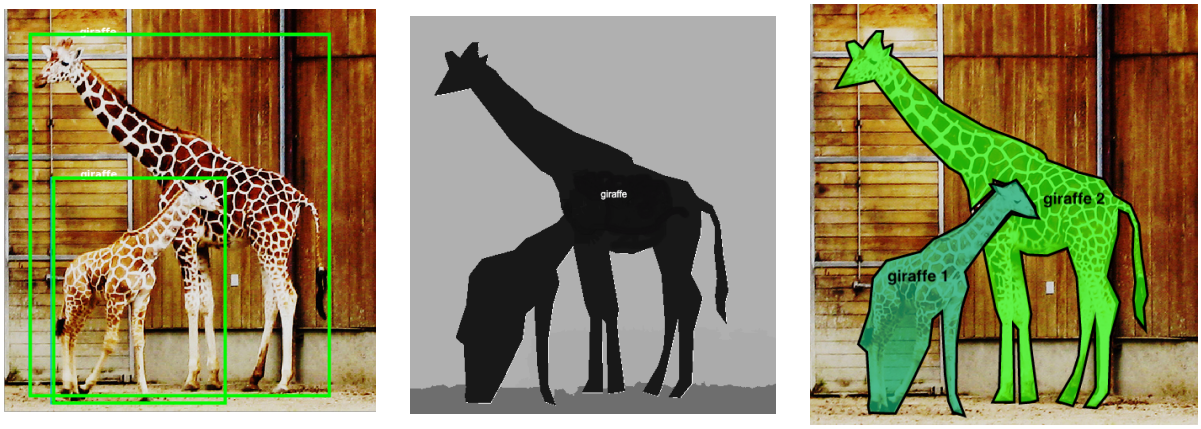


Figure 2.1: Example of different levels of annotations in the Microsoft COCO dataset. The left image shows bounding box annotations for the object detection task. The center image has per-pixel annotations for the semantic segmentation: black pixels are labeled as giraffe, and gray pixels are labeled as background. The right image shows a segmentation instance mask. Instance segmentation differentiates pixels of two giraffes as two different objects, while semantic segmentation takes both giraffes as one blob.

2.2 Segmentation Datasets Overview

This section describes the following generic large-scale segmentation datasets:

- Microsoft Common Object in Context (MS COCO) [10]
- PASCAL Visual Object Classes (PASCAL VOC) [9]
- PASCAL Context [11]
- COCO Stuff [12]
- CityScape [8]
- ADE20K [13]

All these datasets have semantic annotations. The MS COCO and Pascal VOC contain instance segmentation annotations (e.g., objects vs background), while others have per-pixel whole scene annotations without instances. All datasets will be described in detail in the following sybsections.

2.2.1 PASCAL Visual Object Classification

PASCAL Visual Object Classification (PASCAL VOC) [9] is a well-known baseline dataset for object classification, object detection, instance segmentation, person layout, and action classification. Eight annual challenges were held from 2005 to 2012 focusing on

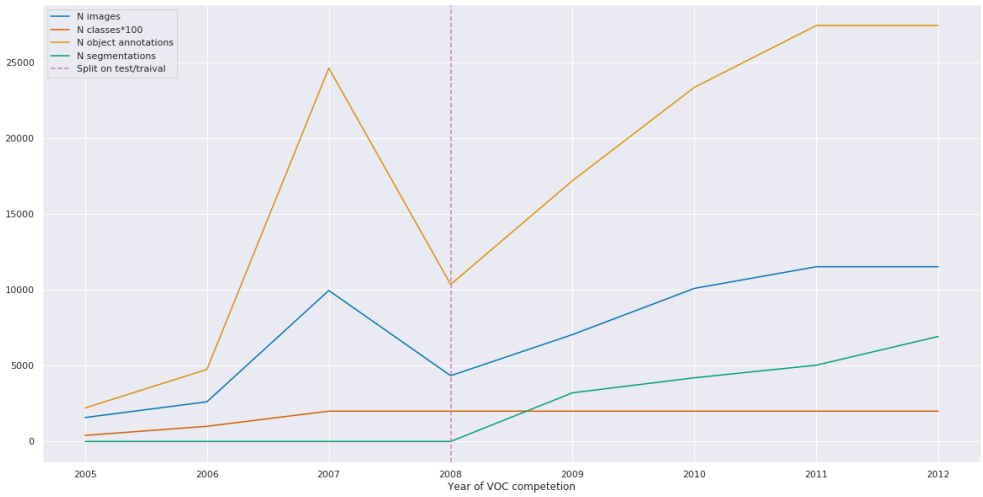


Figure 2.2: Expansion of the PASCAL VOC dataset from 2005 to 2012. Before 2008 the test dataset was publicly available, then it was split 50% as test and /50% training-validation.

improving these datasets. Each year the dataset was updated with more images, classes, and annotations. The expansion of the dataset through the years is shown in Figure 2.2.

The final version of PASCAL VOC was made in 2012. It consists of 11, 530 images and 27, 000 bounding boxes in total. It is equally split into validation and training sets. For the segmentation problem, there are 6, 900 labeled images. The quality of the segmented objects is precise. In order to have decisive segmentation annotations, the border regions of the objects are marked with the "void" label indicating that they could objects or background.

Although the PASCAL VOC dataset contains only 20 categories (plus the background category), it is still used as a reference dataset in the object detection and segmentation problems, partially because the images have a wide range of viewing conditions such as angle, and lighting. The annotations on the dataset only cover 29.3% of the pixels on each images in order to create realistic scenes, as opposed to solely putting an object in the center of the image. Images were collected from the Flickr image database, and annotated with the following 20 classes: Aeroplane, Bicycle, Bird, Boat, Bottle, Bus, Car, Cat, Chair, Cow, Dining table, Dog, Horse, Motorbike, Person, Potted plant, Sheep, Sofa, Train, TV/monitor.

2.2.2 Microsoft Common Objects in Context

Microsoft COCO, also known as Microsoft Common Objects in Context or MS COCO [10] is a large-scale dataset, focused on research problems in scene understanding such as

detecting objects which are not in canonical view, understanding the relationship between objects and precise localization of objects. The dataset was created in 2014 for detection and segmentation challenges and consists of 164, 000 images with more than a million labeled instances. There are 80 object classes divided into 11 hierarchical parent classes. The main idea of the dataset was to collect common objects of everyday life scenes. To define what is a *common* object, MS COCO authors asked children aged 8-11 to name objects in outdoors and indoors environments, then the authors chose 272 final classes, ranked them manually and kept those classes with more than 5 000 instances. The final classes and their number of instances are displayed in Figure 2.3. The classes are organized into a hierarchy, e.g., superclass *vehicle* includes classes *boat, track, car, and bus*. All images were collected using Google search engine, Bing search engine, and Flickr. Annotations were done using Amazon Mechanical Turk with strict annotation rules. The semantic annotations on this dataset have a single label per pixel while bounding boxes are overlapping.

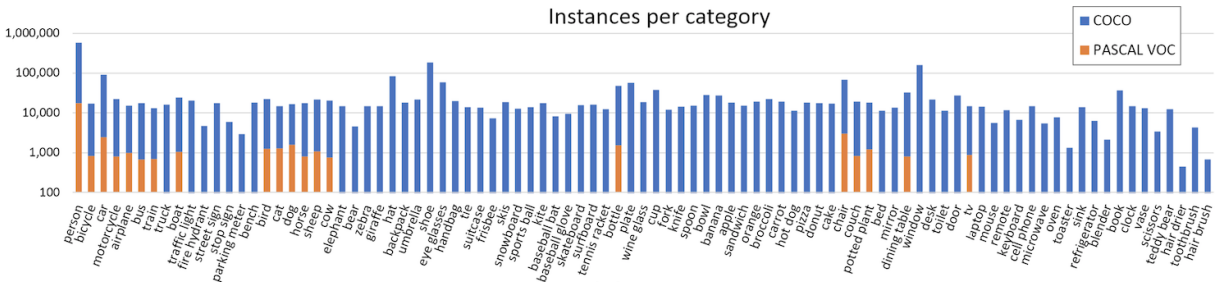


Figure 2.3: Classes and their number of instances in the COCO dataset. The number of instances per class is imbalanced. Class *person* has almost one million instances, while classes *toaster and hair drier* have less than 10 000.

2.2.3 PASCAL VOC Context

The PASCAL VOC Context [11] dataset, also known as VOC Context, is an extension of the PASCAL VOC dataset, which goes beyond the original instance segmentation task by providing annotations for the whole scene. The number of classes is increased from 20 to 520. The classes are divided into three types: objects, stuff, and hybrids. Objects are classes that are defined by shape. VOC Context includes the original 20 PASCAL categories and it enlarges them with rare classes such as *accordion, wood, and fork*. Stuff classes are amorphous background regions such as *sky, and water*. Hybrid classes are classes which shape is so variable that they cannot be easily modeled, such as *road*.

The classes in this dataset are not distributed uniformly. Taking into account 59 of the most frequent classes and assigning the rest to the background label, 87.2% of the pixels are labeled as foreground. The disadvantage of PASCAL VOC Context is the use

of free-form labels for annotation rules. As a result, classes such as *bridge* and *footbridge* are introduced in VOC Context as a parent-child relationship. This leads to inconvenient merging of labels: the annotators named the same thing using different labels. The main distinction of VOC Context from the previous datasets is that it allows multi-class labels; e.g. the pixels of a tree seen through a window are labeled both tree and window. Figure 2.4 shows images of the VOC Context and their ground truth annotations. The images are diverse, and the whole scene is annotated, which leads to coarse annotations.

Figure 2.4: Example of the whole scene annotations from VOC Context dataset[11].

2.2.4 Microsoft COCO Stuff

The Microsoft COCO Stuff [12] dataset, also known as COCO Stuff, extends the MS COCO dataset with a *stuff* category for labeling amorphous background regions, such as grass and sky, and therefore providing better scene understanding. COCO Stuff uses the same images as MS COCO but it extends the amount of classes to 172: 80 classes are the same as the original MS COCO dataset, 91 classes belong to the new *stuff* category, and 1 new class *unlabeled*. The unlabeled class is used when the pixel does not belong to any predefined class or the annotator was not able to infer the class of a pixel. Authors manually selected mutually exclusive 90 *stuff* classes, which cover the majority of pixels, only 6% of pixels are unlabeled. As in COCO, *stuff* classes are also organized into label hierarchy: parent class *textile* has *cloth* and *curtain* as children. The *stuff* part of the dataset is labeled by a semi-supervised superpixel method with the final call from a human annotator. In Figure 2.5 MS COCO images with their corresponding mask are shown. Object annotations of original MS COCO correspond to annotations from the MS COCO dataset.

Figure 2.5: Example of whole scene segmentation on COCO Stuff dataset.

2.2.5 ADE20K

The ADE20K dataset has densely annotated images where every pixel has a semantic label using a large and an unrestricted open vocabulary. The dataset consists of 25, 000 images with approximately 434, 000 labeled instances and 2, 693 classes. ADE20K supports multi-class labels by labeling not only the object itself but also its parts. For example the class *door* contains the class *knob* so the pixels of the *doorknob* are labeled both as *door* and *knob*. The total part hierarchy reaches three levels of deepness. All images in ADE20K contain at least five objects. The maximum number of object instances in one image is 419 for objects with parts, and 273 for objects without parts. This shows that the ADE20K dataset has highly complex annotations, but at the same time it is coarse and inconsistent because of the semantics of the labels and the dense annotations.

2.2.6 CityScapes

The Cityscapes dataset is a large-scale dataset with pixel-level and instance-level semantic annotations of a diverse set of stereo video sequences recorded in streets from 50 different cities [8]. The images were recorded in similar weather condition in order to be consistent.

It has 20, 000 coarsely-annotated images and 5, 000 precise annotations with 60, 000 instances. Around 97% of all labeled pixels in the coarse annotations were assigned the same class as in the fine annotations, showing the high accuracy of labeling.

There are 30 predefined classes for annotations, which are grouped into eight categories: *flat, construction, nature, vehicle, sky, object, human,* and *void*. The classes were selected based on their frequency, relevance, and compatibility with existing datasets.

2.3 State-of-the-art Instance Segmentation Methods

Instance segmentation is the combination of object detection and semantic segmentation tasks. There are usually two common approaches for instance segmentation: the first is based on object detection methods, where at the beginning the proposals of objects are generated, and then they are segmented and classified; the second method starts from per-pixel semantic classification and attempts to divide the pixels of the same category into different instances.

For object detection there is a family of R-CNN methods such as R-CNN [14], Fast R-CNN [15] and the state-of-the-art Faster R-CNN [4], which are based on proposal generation and their classification. In these methods, regions of interest are used to localize the object within the image. You Only Look Once (YOLO) [16] is another state-of-the-art object detection single convolutional network, which predicts the bounding boxes and their probabilities without region proposals, but as one regression model.

The state of the art semantic segmentation algorithm U-Net [17] is based on the Fully Convolutional Network (FCN) [6], which is one of the first neural networks trained end-to-end for semantic segmentation tasks. The U-Net algorithm consists of two parts: the downsampling path which learns semantic information and the upsampling one that restores spatial information. The output of the network is the map with the class prediction for each pixel in the input image.

Driven by the effectiveness of object detection R-CNN methods, many methods for instance segmentation problem are based on segmenting the proposals. In *Instance-aware Semantic Segmentation via Multi-task Network Cascades* [18], a complex network with multiple stages was introduced. The network generates bounding box proposals before segmentation followed by classification. DeepMask [19] learns to propose segmentation masks, which are then classified by Fast R-CNN.

Another family of solutions to the instance segmentation problem is driven by the success of semantic segmentation. For example, in InstanceCut [20] two approaches are combined: semantic segmentation and edge detection. The semantic segmentation map is cut into instance segmentation masks using the edge detection network.

Mask R-CNN [5] is a conceptually simple and flexible framework which is based on Faster R-CNN object detection method. Mask R-CNN adds a branch for mask prediction in parallel with a prediction of object localization and object classifications. Mask R-CNN surpassed all the approaches in instance segmentation MS COCO challenge in 2017.

Considering the semi-supervised approach in instance segmentation, Mask^X R-CNN is a training paradigm with a weight transfer function, that enables training instance segmentation models on a large set of categories all of which have box annotations, but only a small fraction of which have a mask annotations [21]. Mask^X R-CNN transfers parameters of the bounding box prediction to the parameters of mask prediction, allowing to train network with a combination of strongly supervised samples (e.g., mask annotations) and weakly supervised annotations (e.g., bounding boxes).

In this work, we focus on the Mask R-CNN approach because it is a flexible framework for the instance segmentation challenge. Since Faster R-CNN is the primary architecture for Mask R-CNN, in Section 2.4 we discuss the Faster R-CNN in more detail, and in Section 2.5 we explain the difference between Mask R-CNN and Faster R-CNN. The semi-supervised approach Mask^X R-CNN is discussed in Section 2.8.

2.4 Faster R-CNN

Faster R-CNN is one of the state-of-the-art networks in object detection. Faster R-CNN consists of two modules: Region Proposal Network(RPN) for generating proposals (e.g., region of interest) and Faster R-CNN detection network that uses these proposals to detect and classify objects. Faster R-CNN is the continuation of R-CNN and Fast R-CNN [14], [15] detection networks.

The most significant contribution of Faster R-CNN is the Region Proposal Network for proposal generation, which is faster than Selective Search [22] used previously. RPN is represented as a fully convolutional network, which allows training Faster R-CNN end-to-end.

The whole pipeline of Faster R-CNN is shown in Figure 2.6. The backbone network of Faster R-CNN is shared as for the RPN so for the detection network. The backbone network is based on a common CNN architecture such as VGG or ResNet (usually pre-trained on ImageNet) [2], [3]. The output of the backbone is a convolutional feature map. RPN simultaneously regresses regions of different scales and aspect ratios, and its objectness score at each location on a regular grid of feature map.

Faster R-CNN optimization function consists of two losses: RPN loss and R-CNN loss, which will be discussed further subsections.

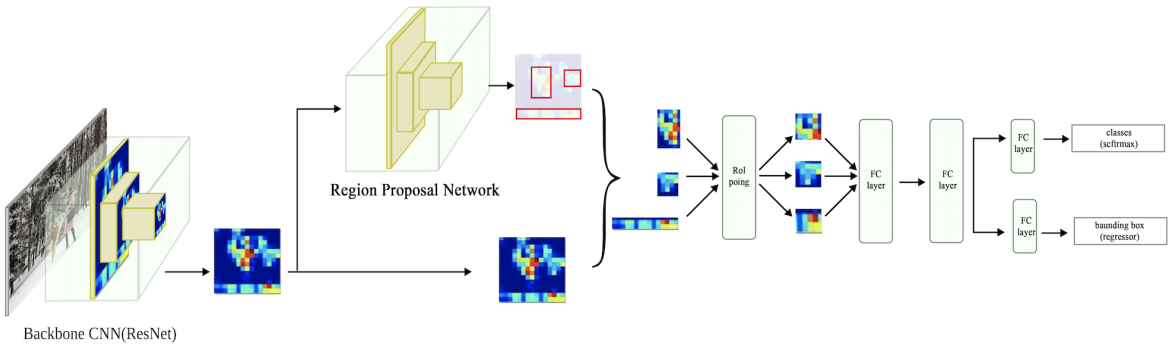


Figure 2.6: Faster R-CNN pipeline as a single network for object detection. In our settings, the backbone architecture is ResNet. The output of the backbone is the feature map. Region Proposal Network in applied on the feature map and outputs the rectangular object proposals with an objectness score. Region of Interest (ROI) pooling converts arbitrary size proposals to the fixed size vector, which is then applied to regression and classifications layer of the object detection network. The output of the pipeline is the bounding box parameters for the objects and its corresponding classification score.

2.4.1 Region Proposal Network

Region Proposal Network takes an image of any size as input and outputs a set of the rectangular proposals and their corresponding objectness score - a measure of belonging to the object class vs. background.

To generate region proposals, RPN slides a fully convolutional network of size $n \times n$, which is applied on the last shared convolutional map of the backbone architecture. Each sliding window is mapped to the low dimensional features, which are sent to a regression layer reg_{rpn}, predicting the bounding box for the proposal, and to a classification layer cls_{rpn} predicting if the region is an object. RPN pipeline is shown on Figure 2.7. Note that RPN operates as a sliding window, the fully-connected layers are shared across all spatial locations. The $n \times n$ convolutional layer is followed by two sibling 1×1 convolutional layers reg_{rpn} and cls_{rpn}.

Anchors

At each sliding window location, RPN predicts k proposals of different aspect ratios and scales, which are parameterized relatively to its k reference boxes, called anchors.

These k anchors are centered with the sliding window and associated with a scale and an aspect ratio. In Faster R-CNN there is a setting with 3 scales and 3 aspect ratios, creating in total 9 anchors at each sliding window position. For a feature map with size $H \times W$ there are $H \times W \times k$ anchors in total.

The output of the regression layer reg_{rpn} is a vector with $4k$ neurons, which encodes of the coordinates of k boxes, the class output layer cls_{rpn} is a $2k$ vector the object-

ness/background score for each of k proposals.

Anchors and their proposals are translation invariant. If the object translates in the image, the proposal should also be translated, and the same function should be able to predict the proposal in either location.

Another advantage of the RPN is a pyramid of anchors, which regresses bounding boxes with respect to different size/scale anchors, using the same convolutional feature map of the single scale, which is more computationally efficient.

Some RPN proposals are overlapping. To avoid a similar proposal, non-maximum suppression (NMS) method is applied. NMS thresholds the proposal with low objectness score and removes proposals which have high Intersection over Union (IoU) with each other. In general, the NMS method keeps 2000 proposals per image.

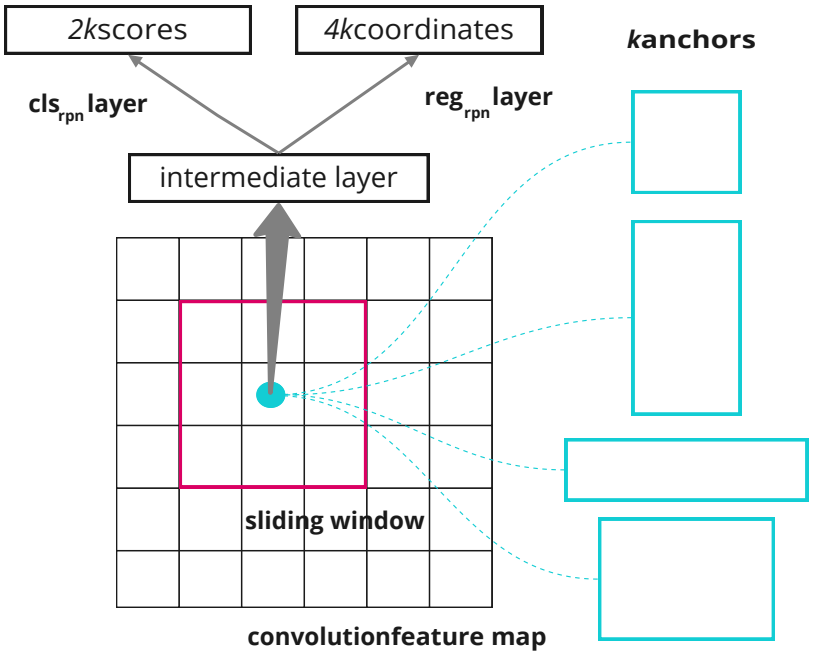


Figure 2.7: Region Proposal Network. Each sliding window is mapped to a lower-dimensional feature map, which is sent to the classification and regression layer. At each point of the sliding window, k proposals are predicted, each of them assigned to one of the anchors. Anchors have different scale and aspect ratio.

RPN Loss Function

Each anchor used in training is assigned a binary label (i.e., object or not object) with the following procedure:

- The **positive label** is assigned to the anchor with the highest IoU with the ground truth, and to all the anchors with IoU overlap higher than 0.7 with any ground truth. The positive label of one ground-truth box may be assigned to multiple anchors.

- **Negative label** is assigned to non-positive anchors if its IoU is lower than 0.3 for all ground truth boxes.

- Anchors which are not positive and not negative do not participate in training.

The objective function for RPN consists of two parts: regression loss and classification loss for each proposal. The classification loss is logarithmic loss over two classes: object or not object. 'The classification loss for a proposal and its corresponding anchor is defined in Equation 2.1, where :

- i is the index of an anchor in a mini-batch

- p_i is the objectness score of an anchor (object vs. background)

- $p_i^* = 1/0$ the ground truth label whether the anchor is positive/negative

$$L_{cls^{rpn}}(p_i, p_i^*) = -(p_i^* \cdot log(p_i) + (1 - p_i^*) \cdot log(1 - p_i)) \tag{2.1}$$

For bounding box regression, the network learns the offsets of predicted bounding box proposals with respect to the ground truth and anchors bounding boxes. For the network, it is simpler to learn offsets from anchors then predict parameters of proposals as center coordinates, width, and height. The parametrization is shown in Equation 2.2.

$$\begin{gathered} \phi((x^*, y^*, w^*, h^*), (x, y, w, h)) = (t^x, t^y, t^w, t^h) \\ t^x = \frac{x^* - x}{w} \\ t^y = \frac{y^* - y}{h} \\ t^w = log(\frac{w^*}{w}) \\ t^h = log(\frac{h^*}{h}) \end{gathered} \tag{2.2}$$

The regression loss for a proposal i is defined in Equation 2.3. For accounting different sizes and aspects ratios, k bounding box regressors are learned. Each regressor is responsible for one scale and aspect ratio, so k regressors do not share any weights.

$$L_{reg}(t_i, t_i^*) = \sum_{j \in x,y,w,h} L^1_{smooth}(t_i^j - t_i^j *) \tag{2.3}$$

where

- t_i is a parameterized offset vector of the predicted bounding box with respect to its anchor. $(t^x, t^y, t^w, t^h) = \phi((x, y, w, h), (x_a, y_a, w_a, h_a))$: where x, y is the center of predicted proposal and w, h the width and height of the proposal. (x_a, y_a) are coordinates of the anchor's center, and (w_a, h_a) is width and height of the anchor.

- t_i^* is a parameterized offset vector of an anchor with respect to it's assigned ground truth bounding box. $(t^{x^*}, t^{y^*}, t^{w^*}, t^{h^*}) = \phi((x^*, y^*, w^*, h^*), (x_a, y_a, w_a, h_a))$, where (x^*, y^*, w^*, h^*) are the parameters of the ground truth bounding box, and (x_a, y_a, w_a, h_a) are parameters for an anchor.

- L^1_{smooth} is a combination of L_1 and L_2 losses. It is more robust to outliers.

$$L^1_{smooth}(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases} \tag{2.4}$$

The loss function for a proposal is defined as:

$$L_{rpn}(p_i, t_i, t_i^*, p_i^*) = \frac{1}{N_{batch}} \sum_i L_{cls}(p_i, p_i^*) + \lambda * \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{2.5}$$

Note, that L_{reg} contributes to L_{rpn} only for positive anchors, so they are objects and have ground truth bounding boxes. λ is a balancing term, a trade-off between classification and regularization loss. N_{batch} is mini-batch size, and N_{reg} is the number of anchor locations($W \times H$).

2.4.2 Region of Interest Pooling

The output of RPN are the object proposals with no class assigned to them. Faster R-CNN reuses the existing convolutional feature map and RPN proposals by extracting fixed-sized feature maps for each proposal using the region of interest pooling(RoIPool). Fixed size feature maps are needed to classify the proposals into a fixed number of classes. RoI max(average) pooling divides $h \times w$ RoI window into a $H \times W$ grid of sub-windows of approximate size $\frac{h}{H}$x$\frac{w}{W}$ and then computes maximum (average) values from each sub-window, creating a fixed-size feature map. The disadvantage is that RoIPool quantizes a floating-number RoI to the discrete values of the feature map. The process of RoIMaxPool is shown on Figure 2.8

2.4.3 Region-based Convolution Neural Network

Region-based convolutional neural network (R-CNN) is the last step in the pipeline of Faster R-CNN. The R-CNN follows two objectives: to classify proposals into object classes (+ background) and to adjust the proposals coordinates. There are two separate fully connected layers: one for classification with $N_C + 1$(number of classes+background) neurons and another for a regression branch with $4 * N_C$ neurons(4 bounding box parameters

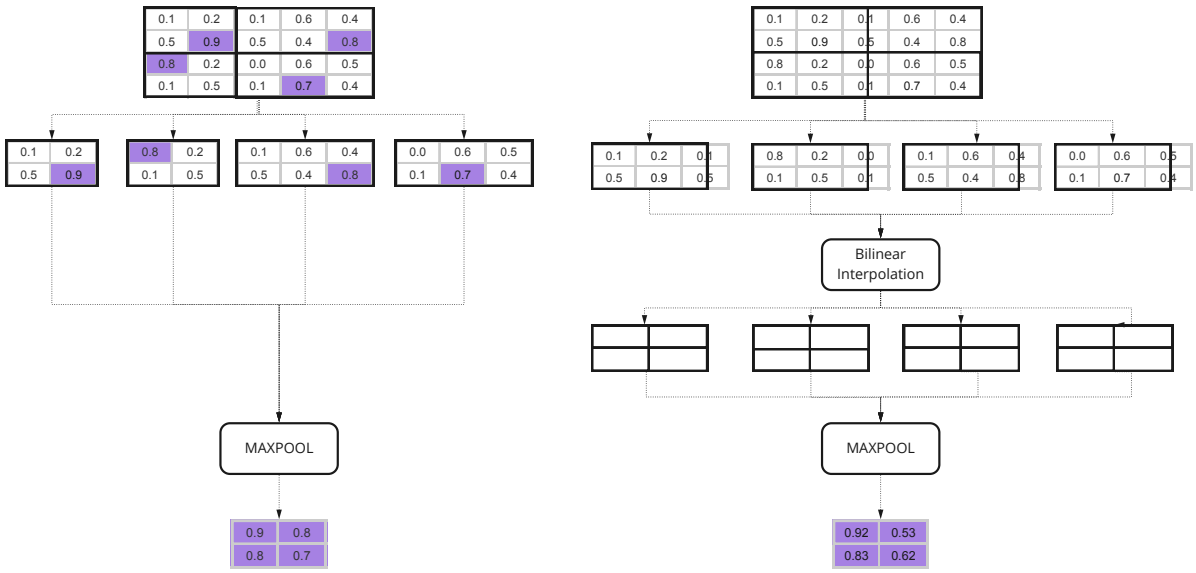


Figure 2.8: Example of RoIPool and RoIAlign. The matrix represents a proposal; the solid lines are an ROI with 2×2 bins. **Left**. RoI max pooling is applied to the proposal. RoI Pool divides feature map into 4 bins and computes the maximum value of each of them. There is a misalignment of segments due to quantization. **Right**. RoI Align divides feature map into 4 equal bins. In each bin 2 sampling points are calculated by bilinear interpolation from the nearby grid points on the feature map. The final fixed size map is created by taking the maximum value of each bin. No quantization is performed on any coordinates involved in the RoI, its bins, or the sampling points.

for each class), trained with the multi-task loss which is similar to the RPN loss. The regression branch outputs bounding-box regression offsets for each class.

R-CNN Loss Function

Each training RoI is labeled with a ground-truth class u and a ground-truth bounding-box regression target v. A **positive label** is assigned to the bounding box, which has at least 0.5 IoU overlap with ground truth box. **Negative** bounding boxes are sampled from the remaining RoIs which have maximum IoU with any ground truth bounding box in the interval (0.1, 0.5). (Note: in RPN the negative were RoI with less than 0.3 IoU with ground truth).

The final network's detection loss consists of classification loss L_{cls}^{r-cnn} and a regression loss L_{reg}. $L_{cls}^{r-cnn}(p,u) = -logp_u$ is the log loss for the ground truth class u, where p is prediction vector for $N+1$ classes $p = (p_0, p_1, p_2, ..., p_N)$, where 0 is an index of the background.

$$L_{R-CNN}(p,u,t^u,t*) = L_{cls}^{r-cnn}(p,u) + \lambda \cdot [u \geq 1] \cdot L_{reg}(t^u,t*) \tag{2.6}$$

Note, that the bounding-box regressor predicts parameters for each class, t_u is the

predicted bounding box offsets for ground truth class u. The regression loss is the same as for RPN $L_{reg}(t^u, t*)$, where t^u is predicted tuple for ground truth, and t^* is a tuple of ground truth bounding-box regression target. The Iverson bracket indicator function $[u \geq 1]$ is present because the regression loss is defined only for the object classes. λ is a normalization parameter which controls the trade-off between classification and regression loss.

The final Faster R-CNN loss is a sum of L_{R-CNN} and L_{RPN}. The pseudocode for the Faster R-CNN algorithm is shown on Algorithm 1.

Algorithm 1: Faster-RCNN algorithm.

```
train_faster_rcnn(model,train_data, total_epochs) model.create
while current_epoch ≤ total_epochs do
    for batch in train_data do
        feature_map = model.resnet(batch)
        proposals, objectness_scores = model.rpn(feature_map)
        fixed_size_proposals = model.roi_align(feature_map, predicted_proposals)
        bboxes, cls_scores = model.faster_rcnn(fixed_size_predicted_proposals)
        loss = compute_loss(batch, bboxes, cls_scores, proposals, objectness_scores)
        model.update_loss(loss)
    end
end
```

2.4.4 Feature Pyramid Networks

Even with RPN characteristics, detecting objects in different scales is a challenging task, especially for small objects. In order to improve the detection on different scales, feature pyramid network is intruded into the detection network.

Feature Pyramid Network (FPN) [23] is a feature extractor designed for pyramid concept. It replaces the feature extractor of detectors like Faster R-CNN and generates multiple feature maps on different scales with better quality information. The motivation is to combine low-resolution semantically strong features with high-resolution semantically weak features with the help of lateral connections.

FPN pipeline consists of two parts: bottom-up and top-down pathway. The bottom-up pathway is a fully convolutional network for feature extraction. With the downsampling approach the spatial resolution of a feature map decreases but the semantic value of each level increases. The top-down pathway reconstructs higher resolution layers from semantically richer but lower resolution layers with an upsampling method.

The reconstructed layers are semantically strong, but the locations of objects are not precise after all the downsampling and upsampling. The lateral connections between

reconstructed layers and the corresponding feature maps are added. This helps the predictor to improve detection of the spatial locations and also simplifies training by acting like skip connections (similar to ResNet).

In Faster R-CNN, FPN is used as feature detector. In FPN, a pyramid of multi-scale feature maps is generated from different levels of backbone architecture such as ResNet. The feature map layer is selected in the most proper scale to extract the feature region based on the width w and height h of the RoI, produced by RPN. The pipeline of the process is shown on Figure 2.9. FPN improves Faster R-CNN accuracy by 8% on the large object detection and by 12.5% on small object detection[23].

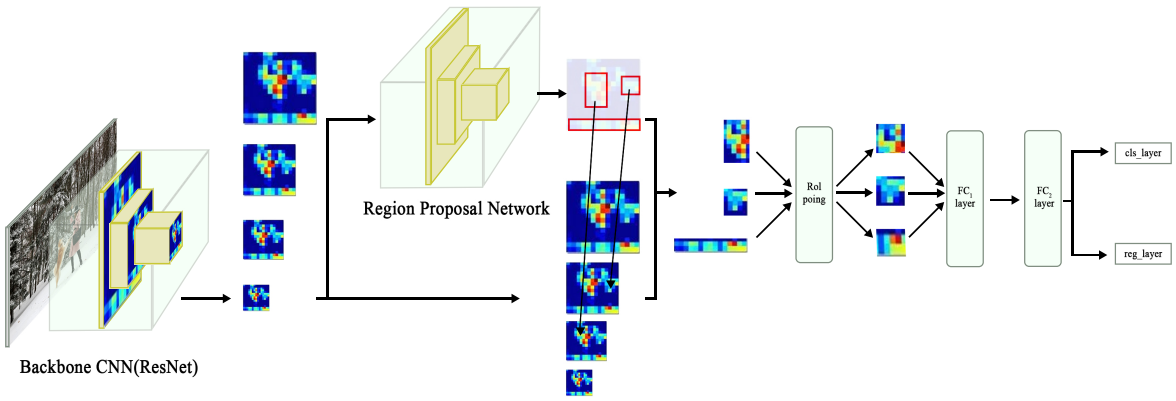


Figure 2.9: Feature Pyramid Network with Faster R-CNN pipeline. FPN generates N feature maps. Proposals generated by RPN extracted from their assigned feature map: small object extract features from the smaller maps.

2.5 Mask R-CNN

Mask R-CNN is a simple, flexible system which surpasses state-of-the-art methods in instance segmentation task. Mask R-CNN[15] expands Faster R-CNN by adding a branch that predicts a segmentation mask for each Region of Interest(RoI), shown in Figure 2.10. The Mask R-CNN predicts the masks in parallel with bounding box regression and classification. The mask branch is a small Fully Convolutional Network(FCN)[6] applied to each RoI. In the previous Section 2.4, we discussed Faster R-CNN in depth because Mask R-CNN is based on it.

Formally, Mask R-CNN consists of Faster R-CNN loss and newly added mask loss L_{mask}. The mask branch has Cm^2 dimensional output for each RoI, which encodes C binary masks of size $m \times m$, one for each class of C classes. L_{mask} is activated only for ground-truth class k associated with the RoI; others mask do not contribute to the loss (similar as for regression layer), so there is no competition among classes for mask prediction. The class-specific mask can be changed to class-agnostic mask, (i.e., predicting

a single $m \times m$ output regardless of class). Class-agnostic map setting is only 0.6% AP (average precision) less accurate than class-specific.

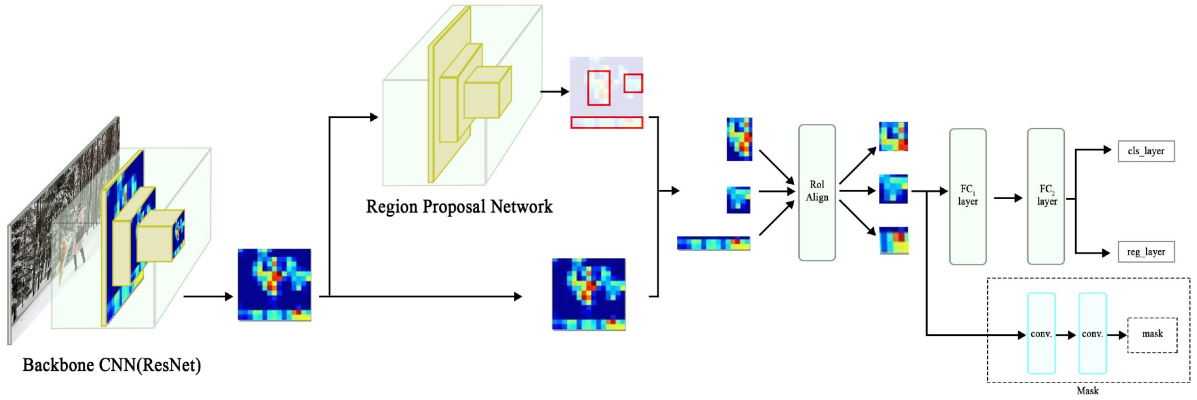


Figure 2.10: Mask R-CNN pipeline. Mask branch is added after RoIAlign. Mask consists of small Fully Convolutional Network and outputs a binary mask for each RoI.

A mask preserves the spatial layout of the input. For each RoI, a mask is predicted by FCN. Because FCN preserves spatial information, each RoI should precisely encode information from the feature map. To extract a feature map without any quantization, the authors introduce RoIAlign instead of RoIPool.

2.5.1 RoIAlign

RoIPool is a standard operation for extracting feature maps from each RoI. Quantization creates a misalignment between RoI and the extracted features. Instead of rounding the real-valued boundaries of the RoI to the nearest location on a coarse grid, as it was done in RoIPool, RoIAlign method is introduced. RoIAlign uses bilinear interpolation to compute the values of the input features at four regularly sampled locations in each RoI bin and aggregates the result (using max or average). RoIAlign increases AP from 3% to 10% percent in different settings. A detailed example of RoIAlign is shown in Figure 2.8.

2.6 Semi-supervised Learning for Dataset Combination

Most of the deep learning approaches require a large number of training datasets. Combination of datasets utilizes as much diverse training data as possible and increases the number of recognizable classes.

Multiple heterogeneous datasets were combined for training a CNN with hierarchical classifier for the semantic segmentation task [24]. The classes of the datasets are semantically connected, and for hierarchical classification 108 labels were manually separated

in three levels of hierarchy, which is not always possible due to a large number of classes in other scenarios.

A selective loss function, integrated into CNN to use multiple training datasets with possibly different label sets, is introduced in *Semantic Segmentation via Multi-task, Multi-domain Learning* [25]. The datasets used in the experiments contain the images from the same set of images; only annotations were done for different tasks, so the label sets are different.

Semi-supervised and weakly-supervised learning for semantic segmentation task is discussed in [26], [27]. In these semi-supervised approaches, datasets with different levels of supervision are combined, for example, a dataset with image-level information and a dataset with high-quality semantic segmentation label. In our task, we focus on a combination of datasets with different label sets, but with the same level of annotation's supervision.

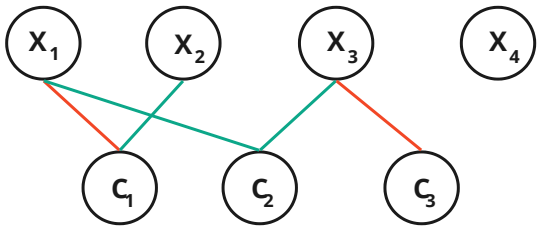
The problem of partially labeled datasets is discussed in *Multi-label learning with Missing Labels* [28]. Based on the sample-level/class-level similarity, the full label assignment for each partially labeled sample is recovered. The core idea is based on feature similarity: if two feature vectors are similar to each other, then their labels would also be similar. In Section 2.7 there are more details about this approach.

2.7 Multi-Label Learning with Missing Labels

For the semi-supervised approach, we focus on the problem of missing annotations, where instead of assuming a complete set of label assignments, an only partial set of labels is available, while others are missing or not provided. A similar problem is discussed in Multi-Label Learning with Missing Labels(MLML) [28].

In MLML each sample can be assigned to multiple classes,(e.g., multi-label learning) and be partially labeled. Labels are separated into three groups: positive label - class is present for the sample, negative label - class is absent for the sample, and missing class - there is no information about whether the class is present or not. More formally, if x_i is labeled $(c_1, \neg c_2, ?c_3)$, it means that x_i is assigned to c_1 and not to c_2, but there is no information about c_3, so c_3 is a *missing label* and x_i is called partially labeled sample. The goal is to predict the complete label assignments of all partially labeled and unlabeled samples. The task is based on two assumptions: label consistency and label smoothness. Label consistency is responsible for predicted label assignments to be consistent with initial positive and negative labels. Label smoothness consists of two parts: sample-level smoothness, where two samples with similar features have similar labels, and class-level smoothness, where two semantically similar classes have similar instances. Based on

these assumptions, the problem of multi-label with missing labels is converted to a linear program problem, which is possible to solve with Sylvester equation [29].



	x1	x2	x3	x4
c1	-1	1	0	0
c2	1	1	0	0
c3	0	0	-1	0

Figure 2.11: The graph on the left describes the label assignments of classes c_1, c_2, c_3 to the samples x_1, x_2, x_3, x_4. The red link denotes negative label where x_i is not labeled with class c_i. The green link denotes positive label where x_i is labeled with class c_i. No link means a missing label. The initial label matrix is presented on the right, where one column vector corresponds to one sample node, and one-row vector corresponds to one class node.

Task Definition

Dataset $X = (x_1, x_2...x_n)$, $x_i \in R^{d\times 1}$, where each sample can be assigned to m classes $C = \{c_1, c_2...c_m\}$ simultaneously, so the labels for x_i is represented as a vector $y_i = (y_{i,1}, y_{i,2}, .., y_{i,m})$, where $y_{i,j} \in \{-1, 0, 1\}$. Positive label $y_{i,j} = 1$ means that sample x_i is labeled by as c_j; negative label $y_{i,j} = -1$ means that sample x_i is not labeled by as c_j; missing label $y_{i,j} = 0$ means that there is no information about whether or not x_j is labeled as c_j. The sample x_i is unlabeled when y_i is zero vector; x_i is fully labeled, when y_i is consisted of non-zero entries; x_i is is partially labeled, when y_i consists from zero and non-zero entries. Initial label matrix $Y \in \{1, 0, -1\}^{m\times n}$ represents a column vector of label assignments y_j for vector x_i. The goal is to predict fully labeled matrix $Z \in \{-1, 1\}^{n\times m}$ with the following two assumptions:

- **label consistency**. The predicted label matrix Z should be consistent with the initial label matrix Y.
- **label smoothness**. The smoothness assumption consists of the sample-level and class-level smoothness:
  - **sample-level smoothness**. If two samples $x_i \sim x_j$, where $\sim$ denotes similarity, then corresponding **columns** $z_i \sim z_j$.
  - **class-level smoothness**. If two classes $c_i \sim c_j$, then corresponding **rows** $z_i \sim z_j$.

Based on above assumption the MLML problem defined as follows:

$$Z^* = argmin_Z ||Z - Y||_F^2 + \frac{\lambda_X}{2} tr(Z L_X Z^T) + \frac{\lambda_C}{2} tr(Z^T L_C Z) \tag{2.7}$$

The first term $||Z-Y||_F^2$ indicates the label consistency, where $||*||_F$ is the Frobenius norm. $Z \in \{-1, 1\}$ and $Y \in \{-1, 0, 1\}$, so the minimum of the $||Z - Y||_F^2$ is when for $\forall y \in \{-1, 1\}, z_{i,j} = y_{i,j}$.

$tr(ZL_XZ^T)$ represents sample-level smoothness and $tr(Z^TL_CZ)$ is a class-level smoothness, λ_X and λ_C are hyperparameters for the trade-off between label and class level smoothness.

L_X is dependent on sample similarity matrix $V_X^{n \times n}$ - a pairwise correlation of X computed based on k-nn graph. V_X is shown in Equation 2.8. $d_{i,j}$ is the Euclidean distance, between x_i and x_j. Note, that x_j is not within k nearest neighbour of x_i then $d_{i,j} = 0$. $\sigma_i = d(x_i, x_h)$, where x_h is the h-th nearest neighbour of x_i.

$$V_X = \exp\left(-d^2\left(x_i, x_j\right) / \sigma_i \sigma_j\right) \tag{2.8}$$

L_C is dependent on class similarity matrix V_C, which embeds the semantic correlation among the classes C described in Formula 2.9. $\overline{Y}_{\cdot i} = (Y_{1i}, Y_{2i}, \ldots, Y_{li})$ is a sub vector of $Y_{\cdot i}$ and l is a number of partially labeled samples. Since the unlabeled sample consists only from zero entries, it can not provide useful information for the semantic correlations. So, the entries corresponding to the completely unlabeled samples are ignored in Equation 2.9.

$$V_C(i, j) = \exp\left(-\eta\left[1 - \frac{\left\langle \overline{Y}_{\cdot i}, \overline{Y}_{\cdot j} \right\rangle}{\left\| \overline{Y}_{\cdot i} \right\| \left\| \overline{Y}_{\cdot j} \right\|}\right]\right) \tag{2.9}$$

The more detailed explanation of sample-level and class-level smoothness is described in [29].

Based on these class-level and sample-level assumptions, partially labeled samples are being fully classified. The idea of sample-level smoothness is applied in our semi-supervised methods.

2.8 Learning to Segment Every Thing

For instance segmentation, most of the methods require all samples from the dataset to be labeled with segmentation mask. In *Learning to Segment Every Thing* [21], the authors propose a partially supervised training method, that enables to train instance segmentation models on a large set of categories all of which have bounding box annotation, but

only a small set of samples have mask annotation. It is done with the help of a novel weight transfer function, which allows predicting category mask based on its bounding box detection parameters.

The partially supervised instance segmentation problem task is formulated as follows:

- All the training samples have bounding box annotations
- A small subset of categories have the mask annotations
- The instance segmentation algorithm should utilize bounding box annotations and available mask annotations to fit a model that can segment instances of **all object categories**.

For the partial instance segmentation method a transfer function is built on Mask R-CNN [5]. As discussed in Section 2.5, Mask R-CNN decomposes the task of the mask prediction and class/bounding box prediction. Because all three branches are trained together, the parameters of the bounding box learn the visual embeddings of the class objects. This enables to transfer this information to the mask branch. The weight transfer function is trained to predict the category mask based on its bounding box detection parameters. It can be trained end-to-end in Mask R-CNN, taking samples with masks annotations as supervisions. At testing time, the weight transfer function can predict segmentation mask for every category, including those which did not have mask annotations during the training phase.

Task Definition

Let C be the set of object classes. $C = A \cup B$, where examples from A has mask annotations and examples from B has **only** bounding box annotations. Note, the mask annotations can be easily converted to bounding box annotations, so there is also available bounding boxes for dataset A. The task becomes partially supervised because categories from B are weakly supervised (e.g., no mask annotations) available. Mask^X R-CNN [5] transfers category-specific information from the model's bounding box detectors to its instance mask predictors.

In Mask R-CNN, the class-specific bounding box parameters and mask parameters are learned independently, more in Section 2.5. Instead, Mask^X R-CNN predicts category-specific mask parameters from its bounding box parameters using generic class-agnostic weight transfer function, which can be trained simultaneously with Mask R-CNN. Specifically, for a given class $c \in C$, w^c_{det} is a class-specific object detection parameters of the last layer bounding box branch. w^c_{seg} is a class-specific weight of mask branch. In Mask R-CNN w^c_{det} and w^c_{seg} are independent, while in Mask^X R-CNN $w^c_{seg} = T(w^c_{det}; \theta)$,

where θ are parameters of class-agnostic mask. The transfer function T is applied to any category $c \in C$ and should be generalized to classes whose masks the network has not seen during the training. For this, class-agnostic parameters θ are responsible for the generalization, while w_{det}^{c} are seen as an appearance-based visual embedding of the class. T is implemented as a small, fully connected network. The whole pipeline is shown in Figure 2.12.

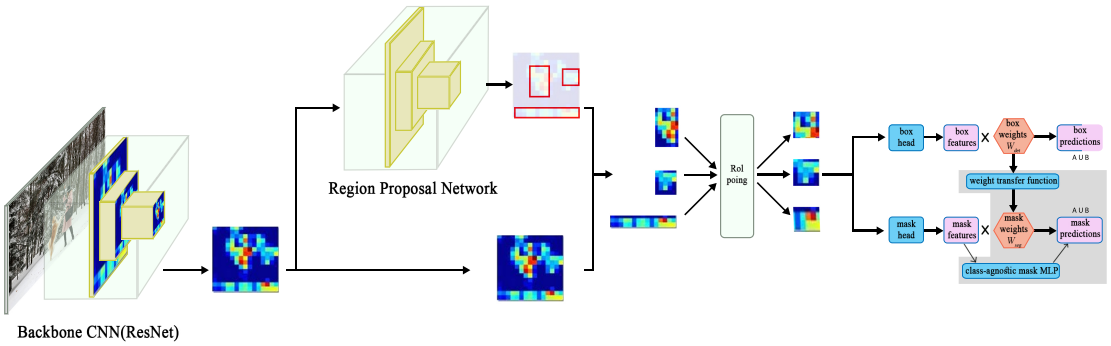


Figure 2.12: **Mask^X R-CNN**.Mask^X R-CNN is an extension of Mask R-CNN network, the extension is shown in grey. Mask^X R-CNN predicts a category's segmentation parameters w_{seg}^{c} from it box detection parameters w_{det}^{c}, using weight transfer function. During training, the weight transfer function needs as mask as bounding box annotation.

Chapter 3

Problem Statement

3.1 Notation

- C - the set of all labeled classes.
- I - the set of all images.
- D - the set of datasets. Dataset $D_d \in D$ is defined by the tuple (I_d, C_d^+), $I_d \in I$, $C_d^+ \in C$.

  C_d^+ - *label set*, set of classes which are annotated in the images I_d.

  $C_d^- = C \setminus C_d^+$ - the set of *missing classes* which may be present in images I_d, but not annotated.

  If $C_d^- = \emptyset$ then the dataset D_d is *fully labeled*, if $C_d^- \neq \emptyset$ then the dataset is *partially labeled*.
- N_C, N_I, N_D - total number of classes, images and datasets.
- d - an index of the dataset. $d \in \{1, .., N_D\}$
- A - the set of the annotated instances. $a^c \in A$ an instance of class $c \in C$.
- P - the set of all generated proposals by object detection network
- X - the set of the feature vectors. Each proposal p_i has its corresponding feature vector x_i.

3.2 Problem Definition

There are several instance segmentation datasets $D_1, D_2, ...D_{N_D}$ with similar visual content and different label sets. To utilize as much training data as possible and to in-

crease the number of recognized labels $C = \bigcup C_d^+,\ d \in \{1, .., N_D\}$, all datasets are used for training. The **goal** is to segment masks of objects in $c \in C$ using all image sets $I_d,\ d \in \{1, .., N_D\}$ for training. Combining datasets with different label sets for detection/segmentation task is a semi-supervised scenario because instances of classes $c \in C_d^-$ are not annotated in dataset D_d. Object instance segmentation methods are often based on object detection. Each region predicted by the object detection method is considered correct or incorrect prediction based on its IoU with annotated instances.

In a supervised object detection, the following cases happen:

- *TP* - true positive: region of class c is annotated and predicted
- *FP* - false positive: region of class c is not annotated but predicted
- *TN* - true negative: region of class c is neither annotated nor predicted
- *FN* - false negative: region of class c is annotated but not predicted

In our semi-supervised scenario, two more cases appear because of dataset combination:

- T_uP - true un-annotated positive: region of class $c \in C_d^-$ is not annotated, although it is present in the image, and it is predicted
- F_uN - false un-annotated negative: region of class $c \in C_d^-$ is not annotated, although it is present in the image, and it is not predicted

Without missing annotations, T_uP is evaluated to be *FP* and F_uN is considered to be *TN*.

We focus on problem of *missing annotations*, described in Section 3.3. In addition, we discuss other problems in dataset combination in Section 3.4.

3.3 Missing Annotations

For simplicity, consider two datasets D_1, D_2 with two corresponding sets of images I_1, I_2 and two corresponding sets of labeled classes C_1^+, C_2^+. The **goal** is to learn to segment masks of objects in $C = C_1 \cup C_2$.

One possible naive approach is to apply supervised learning on each dataset separately: train network M_1 on dataset D_1 and train M_2 on dataset D_2. M_1 and M_2 learned to predict objects from classes C_1, C_2 respectively, so the overall ensemble of the models is able to predict classes C. During testing time, both models M_1 and M_2 predict their corresponding set of classes C_1 and C_2, and the final output is the union of the predictions.

The disadvantage of this approach is the need to evaluate several models, which is time-consuming and expensive in GPU-hours. Moreover this approach is inefficient, because similar features are learned separately in each network.

Another naive approach is to concatenate datasets into one and train one system for all D datasets. As it was mentioned in Section 2.5, methods such as Mask R-CNN first generate proposals and then classify them into C classes and background. Because of missing annotations across the datasets, objects of the same class c are penalized differently. If there is an object of class $c \in C_d^-$ in an image in I_d, it is considered as background, while if it belongs to an image in $I_{\bar{d}}$, it is considered as positive proposal of class c.

To distinguish between background and objects of missing annotations, we propose two semi-supervised methods to instance segmentation. The proposed methods reduce the number of samples considered as false positive, by discovering true unlabeled positives T_uP.

3.4 Label Inconsistency

Most of deep learning approaches in classification tasks contain an output layer with N neurons, which aim to classify an input vector to N classes [5], [15]. Usually, classification is performed with *softmax* classifier which minimizes the cross-entropy between the estimated class probabilities and the *true* distribution, which in this interpretation is the distribution where all the probability mass is on the correct class [30]. In that case, classes compete with each other.

A naive approach to train classification systems with multiple training datasets is to change the classification layer to size N_C, where $C = \bigcup_{d=1}^{N_D} C^+$ and each neuron of the classification layers corresponds to one of the classes in C. The problems with this approach is that classes across the datasets can be semantically similar or be in a hierarchical relationship. This causes label inconsistencies when different labels correspond to one object in different datasets. Due to *softmax* classifier, inconsistent labels compete with each other, and it causes slow and not accurate training.

We distinguish three problems in label inconsistency:

- **Semantic similarity**. In different datasets, the classes, which are semantically similar, can be called different names. For example, the Cityscapes dataset distinguishes between classes *caravan, truck*, where in the COCO dataset they are merged into the class *truck*. A similar situation arises with the class *person*, which in the autonomous driving car dataset is called *pedestrian* or *rider*.

- **Synonyms**. Equivalent objects are called by synonyms. The PASCAL VOC datasets include classes such as *airplane, motorbike*, but in the COCO dataset there is *airplane* and *motorcycle*.

- **Hierarchical relationship**. Some labels can be parts of other labels across datasets. In ADE20K the class *doorknob* is presented. Class *doorknob* would be a part of the class *door* in the PASCAL VOC Context dataset. In this case, labels are not semantically similar to each other, but one is part of each other.

When dealing with a small set of labels, such as 80 classes in the COCO dataset and 20 classes in the PASCAL VOC dataset, it is possible to eliminate these problems by manually changing their labels to use a single set of new labels defined [21], [26]. With an increasing number of labels, setting them manually is not an option.

Chapter 4

Method

As a baseline model, we have chosen Mask R-CNN, a state-of-the-art system for the instance segmentation task. Mask R-CNN is an extension of Faster R-CNN with a segmentation mask as described in Section 2.5.

By combining datasets with different label sets the network receives negative reward for T_uP.

During Mask R-CNN training, P proposals are generated. If a proposal p_d is classified as class $c \in C_d^-$ and its annotation is *background* then there are two cases which can occur:

- It is an instance of the missing class for this dataset, i.e, T_uP prediction.
- The proposal p is a background, i.e, FP prediction.

In classical supervised learning for object detection problem, these two cases are identical: Mask R-CNN generates a loss for misclassification. In our semi-supervised scenario, the first case is the consequence of joining datasets with different label sets. Mask R-CNN produces a loss for T_uP predictions, which forces the network to "unlearn" objects of missing labels. F_uN predictions do not contribute to loss so unlabeled objects which were predicted as background do not influence the training process. The following two semi-supervised methods, described in Section 4.1 and in Section 4.2, focus on discovering T_uP predictions, which would otherwise be considered FP and would penalize the network for actually correct predictions.

The mask loss L_{mask} in Mask R-CNN is not penalized for false positive cases and therefore the semi-supervised training can be described only in terms of Faster R-CNN.

4.1 Mask R-CNN with KNN Search

The first method was inspired by Multi-Label Learning with Missing Labels (MLML) [28], discussed in Section 2.7.

To distinguish between false positives (background proposals) and T_uP, we apply the MLML *sample-level assumption*, if instances are similar they belong to the same class, i.e. if $x_1 \sim x_2$, then their corresponding classes are the same: $c_{x_1} = c_{x_2}$, where $c \in C$.

Faster R-CNN+KNN method is based on feature similarity of object proposals. In each epoch, a database of all annotated objects is created. If a proposal p is classified as class $c \in C_d^-$ and its annotation is background, the minimum distance of the corresponding feature vector x and the feature vector of a predefined database is computed. If the minimum distance surpasses the threshold, then the loss for this proposal is set to zero.

Generated proposals are represented in the form of bounding boxes. In Faster R-CNN each proposal p is transformed into the feature vector x_p by the fully-connected layer fc_2 in Figure 2.6. This feature vector is then sent to the following classification and regression layers where the network predicts the class and the bounding box offsets. It has been shown that the penultimate fully-connected layers provide reasonable feature representation [31]. Because of the sample-level assumption, features can be fine-tuned with metric learning, which will pull features of the same class closer to each other. For simplicity, in this work we use "off-the-shelf" features.

The database of feature vectors of all the ground truth instances is created offline after each epoch. The ground truth bounding boxes are sent through the same pipeline as proposals generated by RPN, so the output for each ground truth instances a_k is a feature vector x_k. The database building is shown in Algorithm 2. Data about each training instance are inserted into the database containing information about the corresponding image, class, dataset, bounding box, and feature vector.

Algorithm 2: Database of ground truth instances.

```
1 Function create_feature_db(train_instances, model):
2     for image in train_instances do
3         gt_box, gt_class, gt_dataset, gt_image = get_ground_truth_info (image)
4         backbone_feature_map = model.backbone(image)
5         fixed_size_boxes = model.roi_align(feature_map, gt_boxes)
6         _, _ , features = model.r-cnn(fixed_size_boxes)
7         feature_db[idx] = [gt_image, gt_dataset, gt_class, gt_box, feature]
8     end
9     return feature_db
```

The threshold which is applied to detect the similarity between the predicted proposals and ground truth objects is computed for each class $c \in C$. Computing the threshold

is shown in Algorithm 3: for each sample in the database, k nearest neighbors search is applied. We find the minimum distance to a neighbor of the same class as the query sample, as shown in Lines [5-7] in Algorithm 3. If there is no neighbor of the same class as a query sample, which means that the query is similar more to the features of a different class, then such queries do not contribute to the threshold calculation, for a threshold to be robust. The threshold is set as a median of the minimum distances for class.

Algorithm 3: Calculating distance threshold for each class.

```
1  Function get_threshold_per_class(feature_db):
2      classes = set(feature_db.classes)
3      for sample in feature_db do
4          k_idx, k_distances = knn(sample.feature, feature_db.features, 5)
5          for idx in k_idx do
6              k_class = feature_db[idx].class
7              if k_class == sample.class then
8                  dists_per_class[sample.class].add(k_distances[idx])
9                  break
10             end
11         end
12     end
13     for class in dist_per_class do
14         median_dist_per_class = median(dist_per_class[class])
15     end
16     return median_dist_per_class
```

The whole pipeline of Mask-RCNN+KNN is shown in Algorithm 4. At the beginning of each epoch, the database of annotated instances and thresholds is updated. Lines [8-11] correspond to the Faster R-CNN, where in addition to predicted scores and proposals, the region object detection network returns feature vectors x for each proposal p. Moreover, the loss function, described in Algorithm 5, multiplies loss of instances by a binary vector of weights, since the loss of ambiguous samples is dropped. RPN loss and Faster R-CNN are implemented as it was described in Equation 2.5 and Equation 2.6.

The semi-supervised part of the algorithm is shown in Lines [15-26]. If a proposal is classified into $c \in C^{-}_{dataset\_id}$ and its annotation is background, then the nearest neighbor search is applied. If the distance from the nearest neighbor to the proposal is less than the precomputed threshold for the class c, then the R-CNN and RPN loss for the proposal

is dropped.

Algorithm 4: K-NN applied with Faster R-CNN

```
Function train_faster_rcnn(model,train_data, total_epochs):
    for current_epoch in total_epochs do
        features_db = create_feature_db(train_data.all_instances, model)
        distance_threshold_per_class = get_threshold_per_class(feature_db)
        for image in train_data do
            dataset_id = image.dataset_id
            C+_dataset_id, C-_dataset_id = get_label_sets(sample, dataset_id)
            backbone_feature_map = model.backbone(image)
            proposals, objectness_scores = model.rpn(backbone_feature_map)
            fixed_size_proposals = model.roi_align(backbone_feature_map,
              predicted_proposals)
            bboxes, cls_scores, features = model.r-cnn(fixed_size_proposals)
            classes = argmax(cls_scores)
            N = len(classes)
            weights_for_loss = ones(N)
            for idx in N do
                class = classes[idx]
                gt_class = get_ground_truth_class(sample, bboxes[idx]) // ground
                   truth class of predicted bbox, 0 - background
                if class ≠ 0 and gt_class=0 and class ∈ C-_dataset_id then
                    class_features = get_class_feature_from_db (features_db, class)
                    nn_idx, nn_distance = knn (features[idx], class_features, 1)
                    threshold = distance_threshold_per_class[class]
                    if nn_distance ≤ threshold then
                        weights_for_loss[idx] = 0
                    end
                end
            end
            loss = compute_loss(image, bboxes, cls_scores, proposals,
              objectness_scores, weights_for_loss)
            model.update(loss)
        end
    end
```

Algorithm 5: Loss function.

```
Function compute_loss(image, bboxes, cls_scores, proposals, objectness_scores,
  weights = None):
    N = len(cls_scores)
    if weights==None then
        weights = ones(N)
    end
    gt_proposals, gt_objecetness_score, gt_cls_scores, gt_bboxes =
     get_ground_truth_info(image, bboxes, cls_scores, proposals,
     objectness_scores)
    rpn_loss, r_rcnn = 0,0
    for id in N do
        rpn_loss += weights[id](rpn_cls_loss(objectness_scores[id],
         gt_objectness_scores[id]) + reg_loss(proposals[id], gt_proposals[id]))
        faster_rcnn += weights[id](faster_cls_loss(cls_scores[id],
         gt_cls_scores[id]) + reg_loss(bboxes[id], gt_bboxes[id]))
    end
    total_loss = rpn_loss+ faster_rcnn
    return total_loss;
```

4.2 Faster R-CNN with Multiple Classification Heads

We extended Faster R-CNN by adding a classification and regression layer for each dataset separately. The original classification and regression layers were removed, and $2N_D$ layers were added: the classification and regression layers for each dataset $d \in D$. The global feature extractor is shared for all datasets, but the classification for each dataset is performed on the corresponding head during training. The setting with multiple classification heads is denoted as Faster R-CNN with MCH - multi-head classification.

We proposed a semi-supervised approach Faster R-CNN with ϵ-MCH, where we drop the loss for proposals which are classified as background in their corresponding head but have high classification score in one of the other branches. If the object does not have a predicted class c in the label set, e.g., $c \in C_d^-$, it can be classified by a classifier trained on different label sets.

For Faster R-CNN with MCH, the change in architecture of R-CNN is changed. Changes are shown in Figure 4.1 and in Algorithm 6. For each dataset $d \in D$ a branch with two layers is created: cls_layer_d and reg_layer_d with corresponding size of $N_{C^{+1}}$ neurons and $4N_C$ neurons respectively and the background is included in each classification

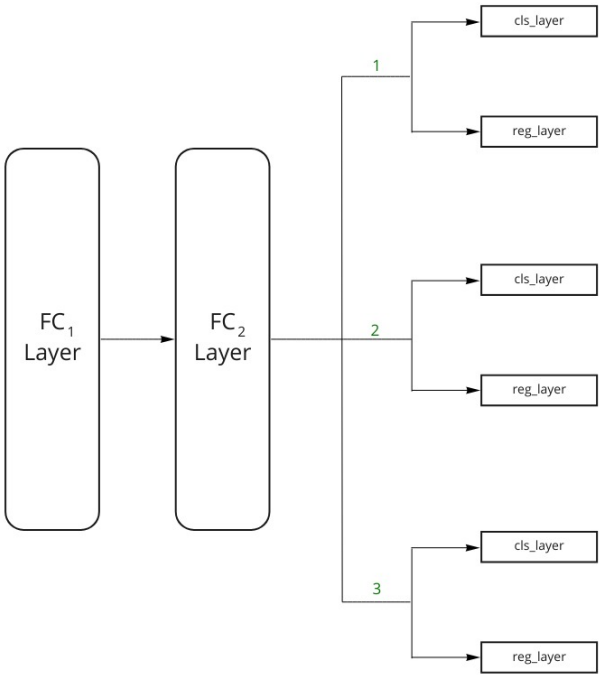


Figure 4.1: Multi-headed classification for each dataset. Proposals from datasets 1,2,3 update only its branch during training. First two layers of R-CNN is shared for all datasets.

branch.

Algorithm 6: R-CNN with multiple heads.

1 **Function** `model.multihead_r-cnn.create(`*datasets*`):`
2 fc_1 = fc_layer()
3 fc_2 = fc_layer()
4 faster_rcnn.multiheads = {}
5 **for** *dataset_id* **in** *datasets* **do**
6 C^+ = dataset.C^+
7 N_{C^+} = len(C^+)
8 cls_layer_d= fc_layer(N_{C^+} + 1)
9 reg_layer_d= fc_layer(4 N_{C^+})
10 `model.r-cnn_heads` [dataset.id] = [fc_1, fc_2, cls_layer_d, reg_layer_d]
11 **end**

Faster R-CNN with $\epsilon-$ MCH is described in Algorithm 8 and has the similar pipeline as the original Faster R-CNN.

Function `model.rcnn` in Algorithm 1 is changed to `model.multihead_r-cnn`, which contains a head for each dataset and outputs the binary weights for the proposals, which are then sent to the loss function in Algorithm 5.

The main changes are done in `model.multihead_r-cnn` in Algorithm 7. Each *image* has the information about the dataset it belongs to. In Line 5 the proposals are sent

to the classification and regression layers of the corresponding branch to the dataset. If one of the proposals is predicted as a background but has higher classification score than $\epsilon_{objectness}$ the proposal is sent to the branches corresponding to the other datasets. Each of the head predicts $class_{head}$ and cls_score_{head} for this proposal. If $class_{head}$ is not background and cls_score_{head} is higher than the predefined threshold for this class then the weights of the loss for this proposal is set to zero.

Algorithm 7: Faster R-CNN with multiple heads.

```
1  Function model.multihead_r-cnn(proposals, image, objectness_scores,
     thresholds_db):
2      dataset_id = image.dataset_id
3      gt_classes = get_ground_truth_class (image, proposals)
4      N = len(proposals)
5      bboxes, cls_scores = model.r-cnn_heads(dataset_id)(proposals)
6      classes = argmax(cls_scores)
7      thresholds_db = update_threshold_db(classes, cls_score, gt_classes)
8      classes_threshold = get_threshold (thresholds_db)
9      weights_for_proposals= ones(N)
10     for idx in N_proposals do
11         class = classes[idx]
12         proposal = proposals[idx]
13         if class == 0  and objectness_scores ≥ ε_objectness  and class ∈ C_dataset_id
            then
14             for head_id in datasets do
15                 if head_id ≠ dataset_id then
16                     bbox_head, cls_score_head = model.r-cnn_heads(head_id)(proposal)
17                     cls_head = argmax(cls_score_head) cls_thresh =
                        classes_threshold[cls_head]
18                     if cls_head ≠ 0  and cls_head ≥ cls_thresh_head then
19                         weights_for_proposals[idx] = 0
20                     end
21                 end
22             end
23         end
24     end
25     return bboxes, classes, weights_for_proposals, thresholds_db
```

Algorithm 8: Faster R-CNN algorithm with multi-head classification.

```
1  Function train_faster_rcnn(model, train_data, total_epochs):
2      model.remove_r-cnn
3      model.multihead_r-cnn.create (train_data.datasets)
4      for current_epoch in total_epochs do
5          for image in train_data do
6              feature_map = model.backbone(batch)
7              proposals, objectness_scores = model.rpn(feature_map)
8              fixed_size_proposals = model.roi_align(feature_map,
                predicted_proposals)
9              bboxes, classes, weights_for_proposals, thresholds_db =
                model.multihead_r-cnn(fixed_size_predicted_proposals,image,
                objectness_scores, thresholds_db)
10             loss = compute_loss(image, bboxes, cls_scores, proposals,
                objectness_scores, weights_for_proposals)
11             model.update_loss(loss)
12         end
13     end
```

The $thresholds\_db \in (0,1)^{C,50}$ is a matrix, which saves the 50 most recent classification scores for each correctly classified class. The threshold for each class is set to the mean of its saved scores. In the beginning the threshold for each class is set to infinity to limit the number of false positives. During the training, classification scores are getting higher, so the thresholds should also increase. Function `get_threshold` computes a median of *thresholds_db* for each class.

Chapter 5

Experiments

Our proposed methods were tested on Faster R-CNN where class-agnostic masks were learned separately on top of the pretrained Faster R-CNN detectors.

5.1 Datasets

5.1.1 Existing Large Scale Datasets

In order to evaluate the proposed methods, several datasets are required with different label sets. Section 2.1 presented 6 different datasets with semantic and instance segmentation annotations. The number of images for each dataset is shown in Figure 5.2. The number of classes and number of instances for each dataset is shown in Figure 5.1. The MS COCO and MS COCO Stuff datasets have the largest set of images. The ADE20K dataset has more than 3, 000 classes with 600, 000 annotated instances. The MS COCO Stuff dataset consists of object annotations coming from the MS COCO dataset and about 300, 000 annotations of *stuff* classes.

Datasets	VOC Context	VOC	COCO	ADE20K	COCO Stuff	CityScapes
VOC Context	460	20	41	325	74	15
VOC	20	20	20	20	20	5
COCO	41	20	80	59	80	8
ADE20K	325	20	59	3039	111	23
COCO Stuff	74	20	80	111	172	11
CItyScapes	15	5	8	23	11	40

Table 5.1: Intersection of label sets of the different datasets.

Table 5.1 shows the intersection between label sets of the different datasets. PASCAL

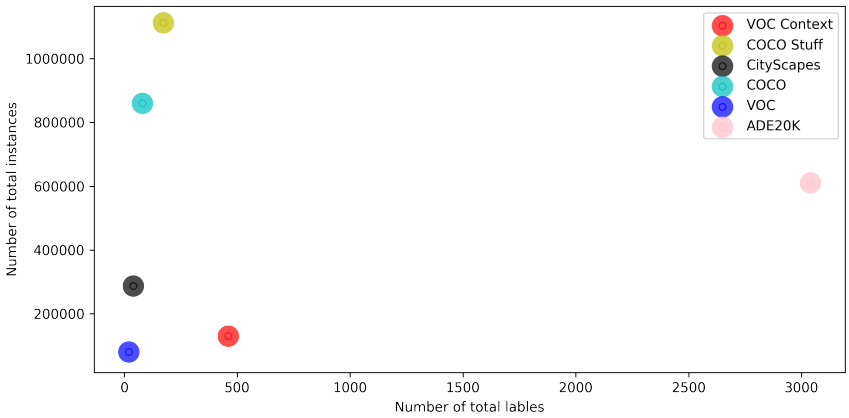


Figure 5.1: Number of annotation with the corresponding number of labels for each dataset. Because the COCO Stuff dataset and the PASCAL VOC dataset have per pixel annotations, we took a region of one class as one annotation.

VOC dataset is a subset of the PASCAL VOC Context, and the MS COCO dataset is a subset of MS COCO Stuff. PASCAL VOC is also a subset of all datasets except Cityscapes. There are only six labels which are present in all datasets: person, car, train, bus, bicycle, and motorbike.

The MS COCO, ADE20K, and PASCAL VOC datasets have instance segmentation annotations. Cityscapes has only instance annotations for some classes, and PASCAL VOC Context and MS COCO Stuff are augmented datasets with stuff classes of the original PASCAL VOC and MS COCO datasets. Examples of the *dogs* annotations in MS COCO, PASCAL VOC, and ADE20K are shown in Figure 5.3.The MS COCO and PASCAL VOC datasets contain precise instance segmentation annotations, while the ADE20K dataset has coarse annotations.

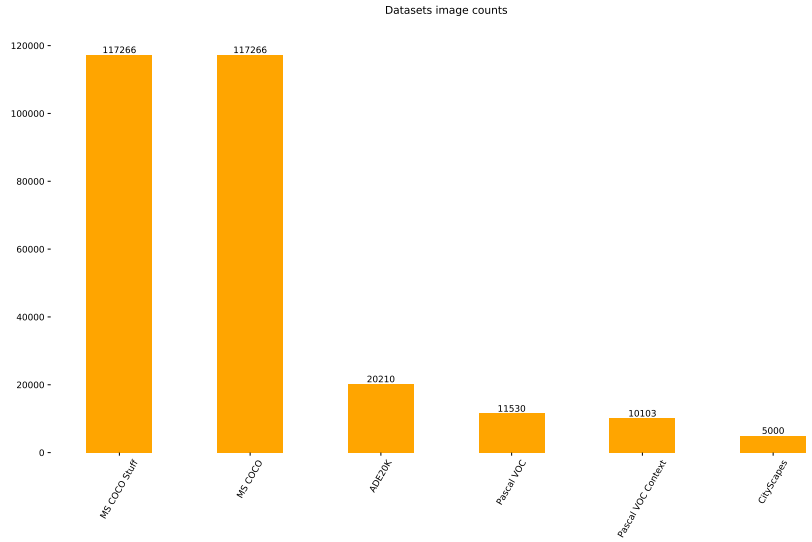


Figure 5.2: Number of images in the different datasets.

Figure 5.3: Examples of *dogs* annotations in different datasets. First, second an third row corresponds to annotations from PASCAL VOC, MS COCO, ADE20K respectively.

5.1.2 Training Datasets

Our problem demands several datasets with similar visual content and different labels sets. ADE20K has coarse annotations and not well defined classes, Pascal VOC label set is subset of the MS COCO label set. In order to have the same quality of annotations and different label sets, we decide to split the MS COCO dataset into 3 datasets with different label sets. To avoid inconsistency of the datasets and to test the ideas described in our methods, two splits of MS COCO training dataset are created: *coco_disjoint* and *coco_overlap*. They contain all the images and classes from the original MS COCO dataset, and corresponding annotations of the split are a subset of annotations in the MS COCO dataset.

coco_disjoint and *coco_overlap* are divided into three subsets each to create a multi-dataset setting. For simplicity datasets corresponding to *coco_disjoint* split are denoted as $coco\_disjoint_0$, $coco\_disjoint_1$, $coco\_disjoint_2$. The same indexation is applied for *coco_overlap*. In *coco_disjoint* split the classes across the datasets are disjoint, while in *coco_overlap* label sets across datasets overlapping. Separation of the classes into different datasets for each split is shown in Figure 5.4. Both cases simulate the real world datasets:*coco_disjoint* datasets have nothing in common so that they could come from different tasks, while *coco_overlap* datasets have shared classes, so their combination should raise the variety of shared classes samples and bring new information from disjoint classes.

Images across the datasets do not overlap. So each image from the MS COCO dataset

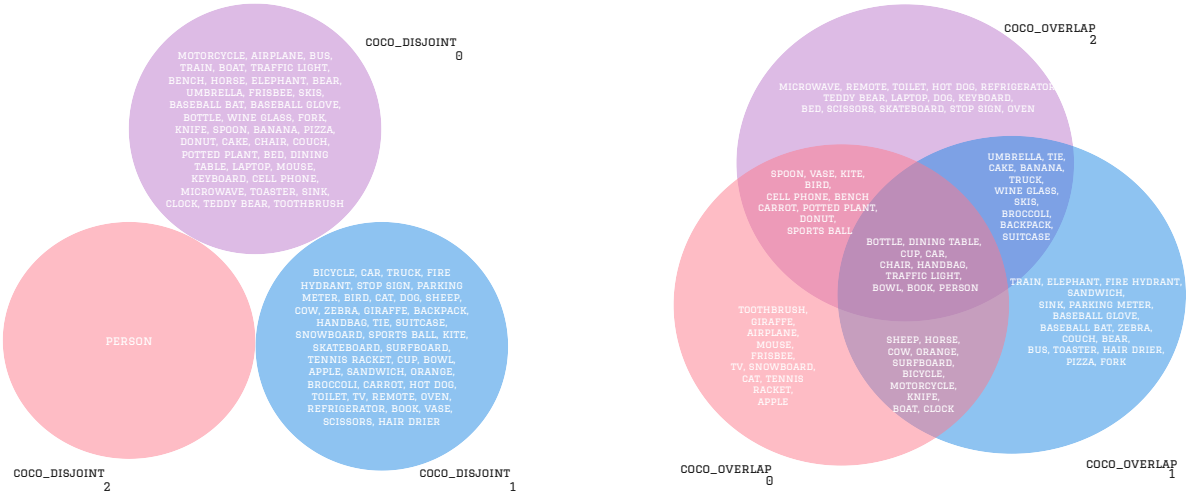


Figure 5.4: Separations of the original COCO classes in three datasets for each split. **Right**. *coco_disjoint* has a dataset for *person* class, due to large number of instances in the original MS COCO dataset. **Left**. *coco_overlap* classes are distributed among three datasets, 10 most frequent classes are represented for all datasets.

belongs only to one dataset. Images are divided in the following procedure: if object of class $c \in C_d^+$ and object of class $c \int C_{\bar{d}}^-$ are annotated in image $i \in I_{COCO}$, i is randomly assign to datasets D_d or $D_{\bar{d}}$, and annotations of classes not belonging to the chosen dataset are removed from the image. The example of removing annotations is shown in Figure 5.5.

Figure 5.5: The difference in annotations of COCO dataset in the left image and it's annotations subset $coco\_disjoint_0$ in the right image. The classes as *person*, *tv*, *book*, *refrigerator*, *vase* are not labeled in $coco\_disjoint_0$ dataset, so their annotations are removed.

A number of images, annotations, classes for each dataset of the splits and the MS COCO dataset are shown in Table 5.2. All images and categories from COCO dataset are used in the datasets, while 51.5% in *coco_disjoint* and 22.9% *coco_overlap* annotations from original COCO datasets and are lost due to not overlapping image approach.

The histograms of most frequent class annotations for *coco_overlap* is shown in Figure 5.6. *coco_overlap* covers all the annotation of most frequent classes. The ratio of instances

Datasets	N images	N categories	N instances
coco_disjoint$_0$	43558	39	144135
coco_disjoint$_1$	46315	40	164325
coco_disjoint$_2$	27393	1	108806
coco_overlap$_0$	39012	40	230678
coco_overlap$_1$	40408	46	221426
coco_overlap$_2$	37846	44	210132
coco_disjoint	117266	80	417266
coco_overlap	117266	80	662236
COCO	117266	80	860001

Table 5.2: Number of images, number of categories and instances in training split for *coco_disjoint* and *coco_overlap* split. *coco_disjoint* and *coco_overlap* cover all the images and categories from COCO datasets, but *coco_disjoint* and *coco_overlap* covers 51.5% and 77% of total number of COCO annotations. *coco_overlap* datasets has more uniformly distributed number of images than *coco_disjoint*.

for each class between the datasets corresponds to the ratio of the total number of instances between the dataset.

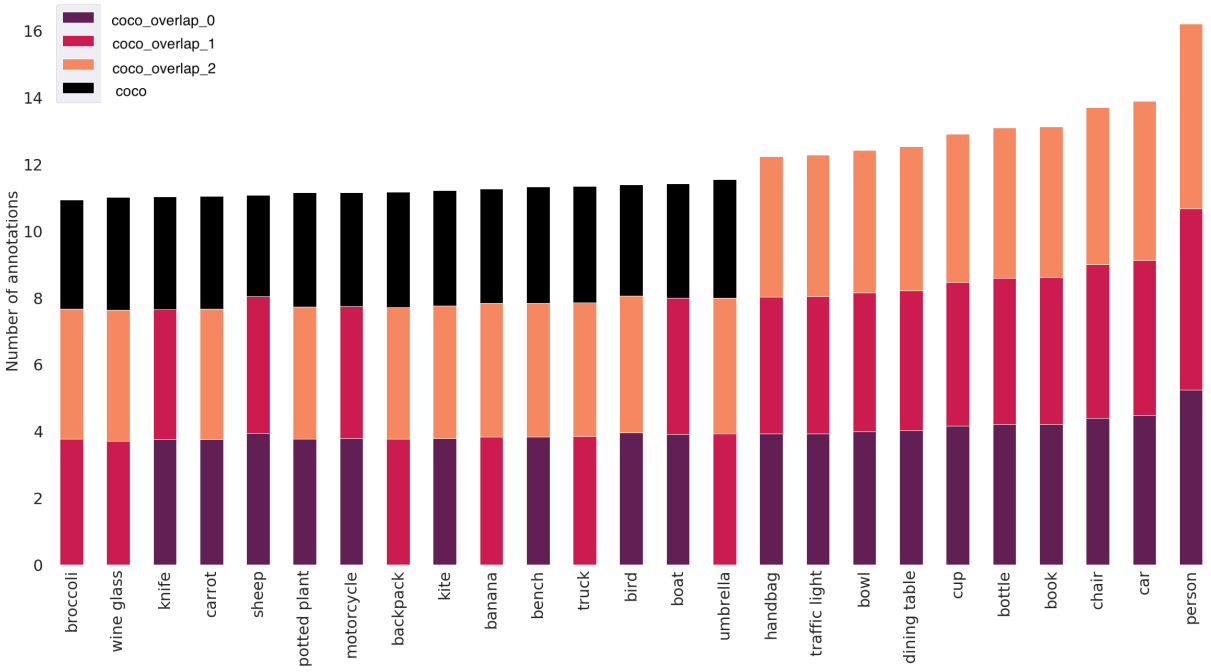


Figure 5.6: Histogram represent s number of annotations per class for datasets in *coco_overlap*. Three datasets are shown in different colors. Number of annotations which were dropped from COCO dataset for each class are shown in black.

5.1.3 Validation Datasets

Validation set correspond to the MS COCO validation split.

In addition, to compute the accuracy of the individual datasets, *coco_val* was split in *coco_disjoint_val*$_i$ where annotations of classes which are not in *coco_disjoint_val*$_i$ label set were dropped. The same settings were done for *coco_overlap_val*$_i$.

Datasets	N images	N categories	N instances
coco_disjoint_val$_0$	5000	39	12422
coco_disjoint_val$_1$	5000	40	13355
coco_disjoint_val$_2$	5000	1	11004
coco_overlap_val$_0$	5000	40	28564
coco_overlap_val$_1$	5000	46	29650
coco_overlap_val$_2$	5000	44	29118
coco_val	5000	80	36781

Table 5.3: Number of annotations, images and classes in each validation dataset split. All datasets are subsets of *coco_val* dataset. All images participate in validation but each dataset contains annotations of object of its corresponding label sets.

5.2 Implementation Details

For our task, we used existing Mask R-CNN baseline, implemented in pytorch. There is an existing official version of Mask R-CNN network, which part of detection system **Detectron**, published by Facebook AI Research center [32]. Detectron is built in Caffe2 framework, which uses static computational graph created in the beginning and which is not possible to change during runtime. For more efficient computation with dynamic computational graph, which allows making changes in the network during running, we have used pytorch Mask R-CNN implementation [33]. `https://github.com/roytseng-tw/Detectron.pytorch`.

For Faster R-CNN+KNN, the database of 860, 000 ground truth instances was created. To compute KNN search of query vector of size 1024 in such a large database is time-consuming. In order to increase computational power, we have used **faiss** library [34] for efficient similarity search. **faiss** supports GPU computation, which enables to do exact nearest neighbor search efficiently. If the number of annotated instances increases, it is possible to use approximate search based on the partition-based method.

All the experiments as for training as for validation were computed with the NVIDIA 1080Ti GPU support. Mask R-CNN implementation supports multi-GPU training. For Faster R-CNN+KNN a distributed GPU training was applied: one GPU supports *faiss* library for KNN search and the second one can be applied for Faster R-CNN. This method does not support multi-GPU training.

5.2.1 Training

For all the experiments, we have used Faster R-CNN with FPN, where the backbone is ResNet50. ResNet50 is pretrained on ImageNet dataset. Input images are rescaled as their shorten edge is 800 pixels. There is one image per batch with 512 sampled RoI, and the ratio of positive and negative RoIs is 1:3. For anchors RPN uses 4 scales and 3 aspect ratios with 2000 proposals kept after applying NMS. Anchors of a single scale assigned to a corresponding features map created by FPN. Then RoIAlign extracts a features map of size 7×7 for each RoI.

We train the network on one GPU for 6 epochs, with a learning rate of 0.02 decreasing it by 10 at 4th and 5th epochs.

At testing time, the number of proposals before NMS and post NMS is set to 1000.

5.2.2 Approximate Joint Training of Faster R-CNN

Most of Faster R-CNN implementations, like an official one in Detectron, do not use end-to-end training, because the training time is not efficient. Instead, approximate joint training is introduced. In each SGD iteration, RPN generates the proposals which are treated as fixed in Fast R-CNN detector. During backward path shared layers are updated from both RPN loss and Fast R-CNN loss. Derivatives of predicted bounding boxes with respect to the proposals are ignored.

In our proposed methods, we set RPN and Faster R-CNN loss to zero for some proposals. Because the training is not end-to-end we tracked proposals' anchors and corresponding feature maps, produced by FPN, to set drop the proposal's loss for RPN.

5.2.3 Hyper-parameters

In Faster R-CNN with ϵ-MCH we introduce the objectness score threshold $\epsilon_{threshold} = 0.4$. The threshold is not changed during the training because, in the next step, the following sample is thresholded by another dynamic threshold.

In Faster R-CNN+KNN, while calculating the threshold, we use KNN search with five nearest neighbors. Because we search for the closest sample of the same class, five neighbors are sufficient. Metric for similarity is L_2.

5.2.4 Mean Average Precision

In the MS COCO challenge, the main measure in object detection task is mean average precision(mAP), a 101-point interpolated AP with average over multiple IoU and categories.

The general definition for the Average Precision (AP) is the area under the precision-recall curve:

$$\text{AP} = \int_0^1 p(r)dr \tag{5.1}$$

For COCO, AP is the average over 10 IoU from 0.5 to 0.95 with a step size of 0.05 on 80 categories. We use the mAP as our primary metric.

5.3 Results

For the evaluation we have trained 5 systems and evaluated them for each split *coco_overlap* and *coco_disjoint*:

- FS Baseline is Faster R-CNN,trained on fully annotated COCO dataset.
- Baseline 1 is Faster R-CNN trained on a partially annotated split *coco_overlap* or *coco_disjoint*.
- Baseline 2 is Faster R-CNN trained on each dataset of the partially annotated split separately.
- Baseline 3 is Faster R-CNN + MHC (multi-headed classifier). An extended version of Faster R-CNN with a corresponding classifier to each dataset, trained on partially annotated split.
- Faster R-CNN + KNN. Our proposed semi-supervised method, based on feature similarity, described in Section 4.1 trained on partially annotated split.
- Faster R-CNN + ϵ-MHC. Our proposed semi-supervised method, based on classification through different heads, described in Section 4.2, trained on partially annotated split.

We compare Faster R-CNN results for *coco_overlap* in Table 5.4 and for *coco_disjoint* in Table 5.5.

Faster R-CNN trained on fully annotated datasets is 5% and 4% better than Faster R-CNN trained on *coco_overlap* and *coco_disjoint* respectively. Interestingly, result of Baseline 1 on *coco_disjoint* is better than *coco_overlap*, even though in *coco_disjoint* there are 33% less annotations present. The expectation was that *coco_disjoint* will learn better the most frequent class *person* which is the only object class of $coco\_disjoint_2$. The mAP of *person* in *coco_disjoint* for Baseline 1 is 44%, while mAP for class *person* in *coco_overlap* is close to 50%, as shown Figure 5.7.

For *coco_overlap* split Faster R-CNN + KNN surpasses the baseline with joint training. Mean average precision for each class for baseline and Faster R-CNN + KNN is shown in Figure 5.7. There is not noticeable correlation between the frequency of the labels and it's prediction. Our proposed method was able to predict classes such *hair drier*, while baseline did not discover it. Examples of the proposals generated by baseline Faster R-CNN and Faster R-CNN+ KNN are shown in Figure 5.8.

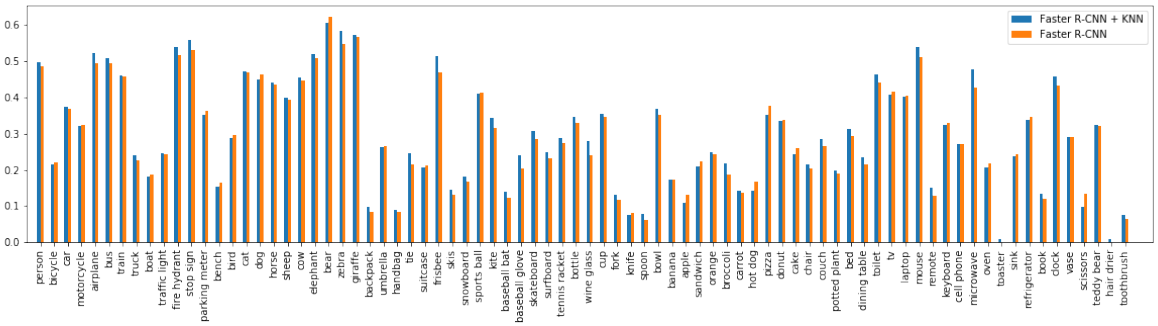


Figure 5.7: mAP for each class predicted by Baseline Faster R-CNN and out proposed method Faster R-CNN+KNN. Both methods were trained on *coco_overlap*. Labels are sorted by the frequency in ground truth.

Our methods did not improve training Faster R-CNN with *coco_disjoint* dataset. Due to a small number of instances KNN search was not precise in the feature similarity. Only 20% of the ground truth instances in *coco_disjoint* have the nearest neighbor the instance of the same class. Error analysis on Faster R-CNN + KNN for *coco_disjoint* is described in Section 5.3.2.

Dataset	Faster R-CNN					
	FS Baseline	Baseline 1	+KNN	Baseline 2	+ ϵ-MHC	Baseline 3
coco_val	0.32	0.27	**0.29**	0.27	0.28	
coco_overlap_val$_0$	0.31	0.28	0.30	0.28	0.29	0.27
coco_overlap_val$_1$	0.31	0.27	0.28	0.27	0.29	0.26
coco_overlap_val$_2$	0.32	0.26	0.27	0.28	0.26	0.24

Table 5.4: All the methods, except FS Baseline, were trained on *coco_overlap* split and evaluated on all *coco_val* and validation datasets for each label set. FS Baseline was trained on fully annotated MS COCO train dataset. Baseline 1 is joint training of Faster R-CNN on *coco_disjoint* datasets. Baseline 2 is Faster R-CNN trained on each dataset of split separately. Baseline 3 is Faster R-CNN with multiple classification heads.

Faster R-CNN + MHC and Faster R-CNN + ϵ-MHC showed only local improvements on both splits. During last epoch on Faster R-CNN + ϵ-MHC on *coco_overlap* only 12% of unlabeled instances were sent to the classification layer of the different branches, and only 3% passed the threshold and were classified correct.

Figure 5.8: Bounding box predictions are shown in green rectangles, their classes and classification scores are written in white above the bounding box. **Right** column has predictions from naive Faster R-CNN trained on *coco_overlap*. **Left** column corresponds to the predictions from Faster R-CNN + KNN trained on *coco_overlap*.

Dataset	Faster R-CNN					
	FS Baseline	Baseline 1	+KNN	Baseline 2	+ ϵ-MHC	Baseline 3
coco_val	0.32	**0.28**	0.28	0.28	0.28	
coco_disjoint_val$_0$	0.32	0.28	0.28	0.28	0.28	0.22
coco_disjoint_val$_1$	0.30	0.28	0.28	0.28	0.28	0.32
coco_disjoint_val$_2$	0.48	0.44	0.43	0.43	0.43	0.49

Table 5.5: Experiments completed on *coco_disjoint*. All the methods were trained on split *coco_disjoint* and evaluated on all *coco_val* and validation datasets for each label set. FS Baseline was trained on fully annotated MS COCO train dataset. Baseline 1 is joint training of Faster R-CNN on *coco_disjoint* datasets. Baseline 2 is Faster R-CNN trained on each dataset of split separately. Baseline 3 is Faster R-CNN with multiple classification heads.

5.3.1 Mask R-CNN

For Faster R-CNN and Faster R-CNN + KNN, the mask branch was added and fine tuned on *coco_overlap* dataset. Faster R-CNN weights were frozen, and only class-agnostic mask branch was trained. Both network were trained on 6 epochs. The results are shown in Table 5.6. The detection mAP has not been change, since their weight were not updated. Mask's mAP for out method is 1% higher than for baseline solution, 27% vs 26%. In Figure 5.9 are examples of segmented mask.

	Mask R-CNN fine tuned on	
	Faster R-CNN	Faster R-CNN + KNN
Mask AP	0.264	0.270
Detection AP	0.27	0.29

Table 5.6: Mask and Detection mAP for Mask R-CNN fined t

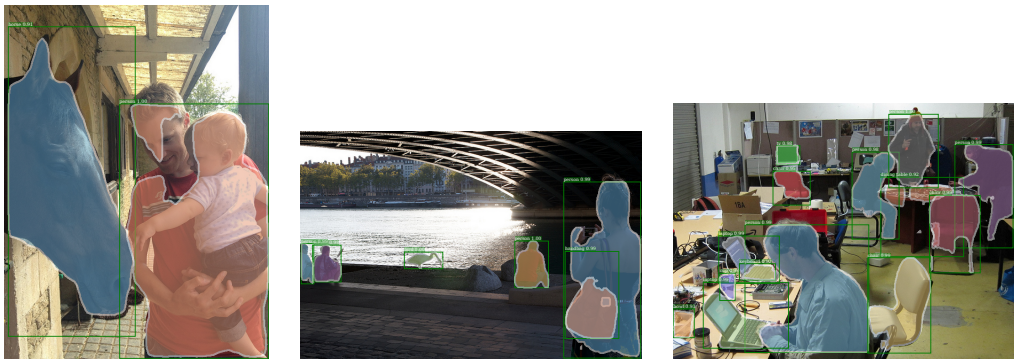


Figure 5.9: Masks prediction produced by Mask R-CNN fine tuned on Faster R-CNN + KNN on *coco_overlap* dataset.

5.3.2 Error Analysis of Faster R-CNN + KNN

We apply KNN search on features of the ground truth instances extracted from pretrained Faster R-CNN on *coco_disjoint*. KNN search is used with L_2 norm and $k = 4$ neighbors. The results are shown in Figure 5.11. In the 2nd row, KNN not only detected the neighbors of the same class as search query *person* but also saved the context of the scene. The last neighbor in this search has ground truth class as *chair*. It is caused by the high IoU of the bounding box of the chair and person, sitting on it. In the 3rd row, the second nearest neighbor to an instance *knife* is a *person*, which is visually similar to *knife*: the body is shaped like a blade, and the leg looks like a handle. In the 4th row, neighbors of *horse* instance were all misclassified. But, the neighbours of search query *horse* have similar features: *chair* has the same colors as query, *zebra* has similar shape. In the 5th row, the query object *truck* is barely recognizable by the human eye, but the KNN search still found appropriate two objects of the same class.

We have calculated frequency of neighbour k being the same class as a search query. At least 22% of samples do not have a sample of the same class among 4 nearest neighbours and 20% of samples have the nearest neighbour the sample of the same class. It means, that the features of the same class are not close to each other in L_2 distance. To prevent this, it is possible to apply metric learning during Faster R-CNN training, and to pull the features of the same class together. To visualize the similarity between samples, we apply

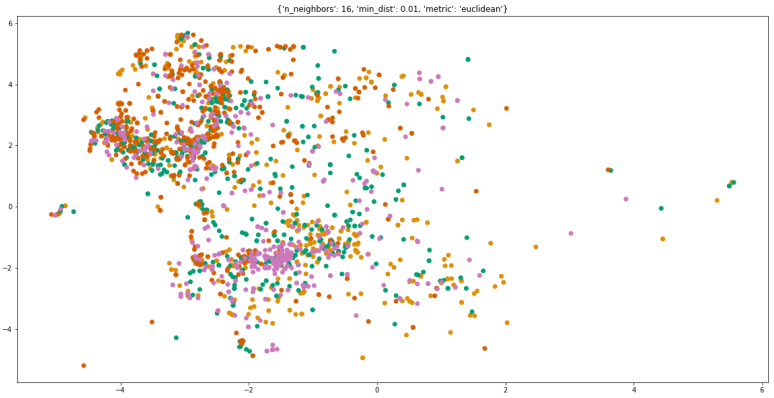


Figure 5.10: UMAP visualization of ground truth instances extracted from pretrained Faster R-CNN.

Uniform Manifold Approximation and Projection (UMAP) [35], a dimension reduction technique. UMAP was trained on 417, 266 instances with standard parameters. The visualization for the most frequent classes is shown in Figure 5.10. The samples do not create clusters, but some classes tend to be closer to each other such as class *person*. Class *person* in 69% of cases have the nearest neighbour as sample of the same class. As

a feature work, it is possible to apply metric learning during Faster R-CNN training, and to push the features of the same class together.

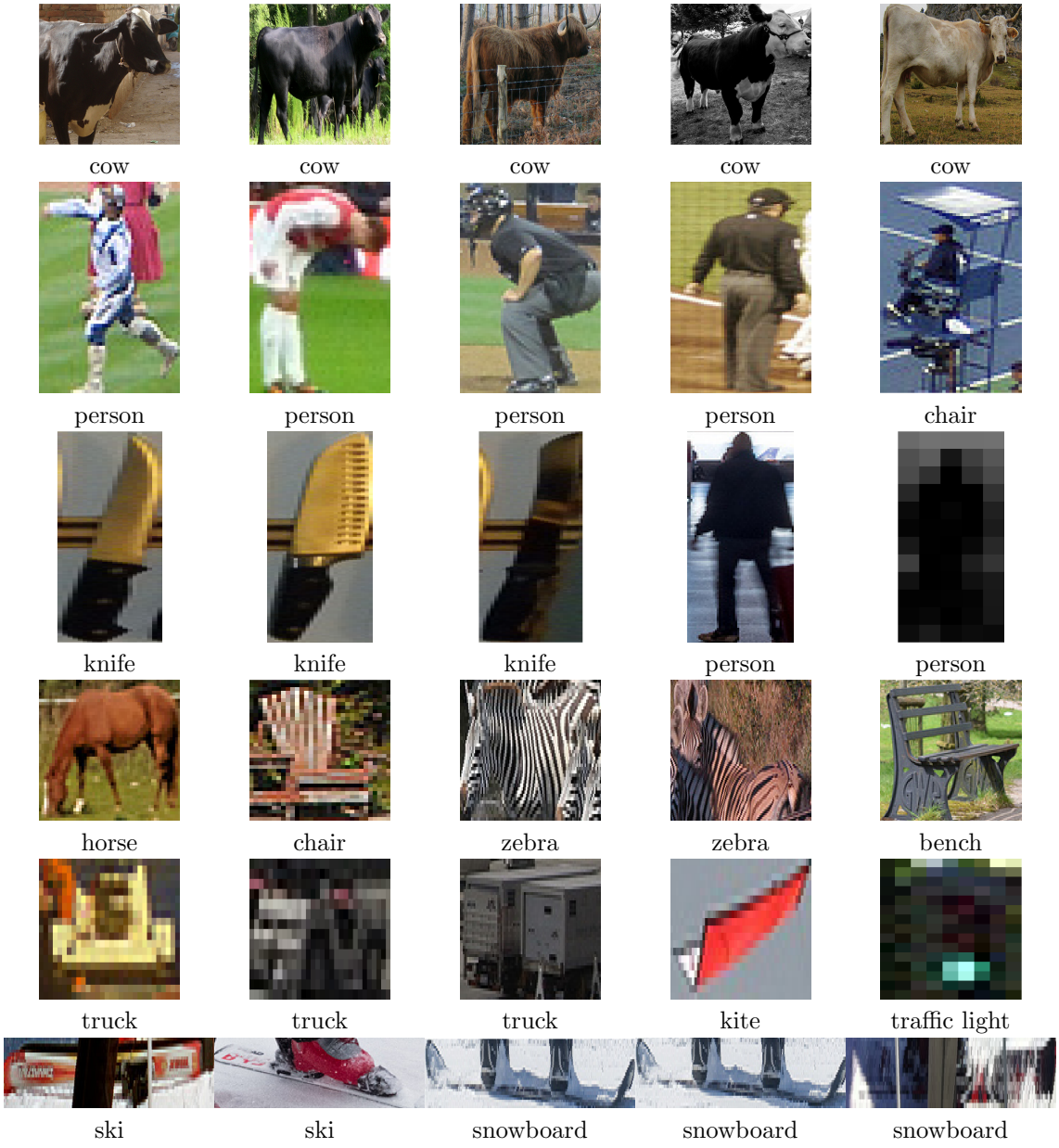


Figure 5.11: Nearest neighbour search on the ground truth objects extracted from pre-trained Faster R-CNN. The first image is a search query, 4 images next to it is the nearest neighbours computed by KNN with $L2$ metric. Sign below the pictures is the ground truth class of the proposal.

Chapter 6

Conclusion

The work focuses on training instance segmentation from multiple datasets with different label sets, and addresses the problem of missing annotations.

Two novel semi-supervised methods were proposed for object detection based networks. These methods can use both images with instance segmentation annotations and images with bounding box annotations only.

The first method is based on feature similarity of object proposals. If the network predicted an instance as class $c \in C^-$ (not belonging to the image label set), similarity to other annotated instances of class c is used to distinguish between unlabeled instances and background.

The second proposed semi-supervised method extends the object detection method by adding a separate classification head for each label set. The loss for proposals, which are classified as background by the head corresponding to the image label set, is dropped if the proposals have a high classification score in any other head.

Both methods were implemented as an extension of Mask R-CNN. Experiments are done on two splits of the MS COCO dataset: the first divides images into three datasets with overlapping label sets, the second divides images into three datasets with distinct label sets. In the first split, only 24% of annotations were lost, while in the second split, almost half of the annotations were lost. All images and classes kept as in original COCO.

On the first set with overlapping labels, the baseline method performed 5% worse in terms of mAP compared to a fully supervised setting. The proposed methods KNN and ϵ-MHC improved mAP in the semi-supervised scenario by 1.75% and 0.72% of mAP respectively, which means 35% and 14% increase towards the bound given by full annotation. Interestingly, on the second set with distinct labels, the proposed methods performed the same as baseline. Given the significantly lower number of object instances (417266 vs. 662236), the thresholds in both methods may benefit from a different setting. Analysis of individual error cases did not show noticeable changes in categorical errors or

in the type of errors.

6.1 Future Work

In Experiments in Chapter 5, first an object detection network was trained with the proposed methods, and an instance class-agnostic segmentation mask branch was added and fine-tuned afterwards. Comparison against class-specific segmentation and against end-to-end training may bring further insights.

The method based on similarity search, Faster R-CNN + KNN, heavily relies on the distances between feature embeddings. The quality of this metric may be improved by adding a metric learning objective to Faster R-CNN loss.

For Faster R-CNN + $\epsilon-$ MHC, correct proposals of an annotated class c can be used as a background example for branches which do not contain c in their label set. The thresholding of Faster R-CNN + $\epsilon-$ MHC should be revised: only 12% of unlabeled annotations passed the thresholds during training.

Bibliography

[1] J. Schmidhuber, "Deep learning in neural networks: An overview", *Neural networks*, vol. 61, pp. 85–117, 2015.

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

[3] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks", in *Advances in neural information processing systems*, 2015, pp. 91–99.

[5] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN", in *Computer Vision (ICCV), 2017 IEEE International Conference on*, IEEE, 2017, pp. 2980–2988.

[6] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[8] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[9] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge", *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[10] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context", in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[11] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille, "The role of context for object detection and semantic segmentation in the wild", in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[12] H. Caesar, J. Uijlings, and V. Ferrari, "COCO-Stuff: Thing and stuff classes in context", in *Computer vision and pattern recognition (CVPR), 2018 IEEE conference on*, IEEE, 2018.

[13] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 633–641.

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

[15] R. Girshick, "Fast R-CNN", in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[17] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", in *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, pp. 234–241.

[18] J. Dai, K. He, and J. Sun, "Instance-aware semantic segmentation via multi-task network cascades", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3150–3158.

[19] P. O. Pinheiro, R. Collobert, and P. Dollár, "Learning to segment object candidates", in *Advances in Neural Information Processing Systems*, 2015, pp. 1990–1998.

[20] A. Kirillov, E. Levinkov, B. Andres, B. Savchynskyy, and C. Rother, "Instancecut: From edges to instances with multicut", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5008–5017.

[21] R. Hu, P. Dollár, K. He, T. Darrell, and R. Girshick, "Learning to segment every thing", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4233–4241.

[22] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders, "Selective search for object recognition", *International Journal of Computer Vision*, 2013. DOI: 10.1007/s11263-013-0620-5. [Online]. Available: http://www.huppelen.nl/publications/selectiveSearchDraft.pdf.

[23] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2117–2125.

[24] P. Meletis and G. Dubbelman, "Training of convolutional networks on multiple heterogeneous datasets for street scene semantic segmentation", in *2018 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2018, pp. 1045–1050.

[25] D. Fourure, R. Emonet, E. Fromont, D. Muselet, A. Trémeau, and C. Wolf, "Semantic segmentation via multi-task, multi-domain learning", in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, Springer, 2016, pp. 333–343.

[26] G. Papandreou, L.-C. Chen, K. P. Murphy, and A. L. Yuille, "Weakly-and semi-supervised learning of a deep convolutional network for semantic image segmentation", in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1742–1750.

[27] Q. Tian and B. Li, "Simultaneous semantic segmentation of a set of partially labeled images", in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, 2016, pp. 1–9.

[28] B. Wu, Z. Liu, S. Wang, B.-G. Hu, and Q. Ji, "Multi-label learning with missing labels", in *2014 22nd International Conference on Pattern Recognition*, IEEE, 2014, pp. 1964–1968.

[29] G. Chen, Y. Song, F. Wang, and C. Zhang, "Semi-supervised multi-label learning by solving a sylvester equation", in *Proceedings of the 2008 SIAM International Conference on Data Mining*, SIAM, 2008, pp. 410–419.

[30] [Online]. Available: http://cs231n.github.io/linear-classify/.

[31] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: An astounding baseline for recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014, pp. 806–813.

[32] Facebookresearch, *Facebookresearch/detectron*, 2019. [Online]. Available: `https://github.com/facebookresearch/Detectron`.

[33] Roytseng-Tw, *Roytseng-tw/detectron.pytorch*, 2019. [Online]. Available: `https://github.com/roytseng-tw/Detectron.pytorch`.

[34] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs", *arXiv preprint arXiv:1702.08734*, 2017.

[35] L. McInnes, J. Healy, N. Saul, and L. Grossberger, "Umap: Uniform manifold approximation and projection", *The Journal of Open Source Software*, vol. 3, no. 29, p. 861, 2018.

Appendix A

Attachments

A.1 Table of CD Contents

An attached CD contains git repository of the extended Mask R-CNN pipeline with the following five branches:

- **master** original Mask R-CNN method
- **faster_rcnn_knn** implemented semi-supervised Faster R-CNN + KNN approach
- **faster_rcnn_mhc** implemented Faster R-CNN with multi-classification heads
- **faster_rcnn_mhc_eps** implemented ϵ - Faster R-CNN with multi-classification heads
- **faster_rcnn_mhc_test** testing pipeline for Faster R-CNN with multi-classification heads methods

The repository is also available at `https://github.com/elnazavr/Detectron.pytorch.git`.