An investigation of regression as an avenue to find precision-runtime trade-off for object segmentation.

Arun Rajendra Prabhu

Publisher: Dean Prof. Dr. Wolfgang Heiden

Hochschule Bonn-Rhein-Sieg – University of Applied Sciences, Department of Computer Science

Sankt Augustin, Germany

November 2020

Technical Report 05-2020
This work was supervised by

Paul G. Plöger
André Hinkenjann
Stefan Eickeler
This report has been derived from the work done as a part of the master’s thesis with the same title. The blue colored texts in this report (except the chapter headings) represent the changes made after the submission of the master’s thesis report.
Abstract

The ability to finely segment different instances of various objects in an environment forms a critical tool in the perception tool-box of any autonomous agent. Traditionally instance segmentation is treated as a multi-label pixel-wise classification problem. This formulation has resulted in networks that are capable of producing high-quality instance masks but are extremely slow for real-world usage, especially on platforms with limited computational capabilities. This thesis investigates an alternate regression-based formulation of instance segmentation to achieve a good trade-off between mask precision and run-time. Particularly the instance masks are parameterized and a CNN is trained to regress to these parameters, analogous to bounding box regression performed by an object detection network.

In this investigation, the instance segmentation masks in the Cityscapes dataset are approximated using irregular octagons and an existing object detector network (i.e., SqueezeDet) is modified to regresses to the parameters of these octagonal approximations. The resulting network is referred to as SqueezeDetOcta. At the image boundaries, object instances are only partially visible. Due to the convolutional nature of most object detection networks, special handling of the boundary adhering object instances is warranted. However, the current object detection techniques seem to be unaffected by this and handle all the object instances alike. To this end, this work proposes selectively learning only partial, untainted parameters of the bounding box approximation of the boundary adhering object instances. Anchor-based object detection networks like SqueezeDet and YOLOv2 have a discrepancy between the ground-truth encoding/decoding scheme and the coordinate space used for clustering, to generate the prior anchor shapes. To resolve this disagreement, this work proposes clustering in a space defined by two coordinate axes representing the natural log transformations of the width and height of the ground-truth bounding boxes.

When both SqueezeDet and SqueezeDetOcta were trained from scratch, SqueezeDetOcta lagged behind the SqueezeDet network by a massive $\approx 6.19$ mAP. Further analysis revealed that the sparsity of the annotated data was the reason for this lackluster performance of the SqueezeDetOcta network. To mitigate this issue transfer-learning was used to fine-tune the SqueezeDetOcta network starting from the trained weights of the SqueezeDet network. When all the layers of the SqueezeDetOcta were fine-tuned, it outperformed the SqueezeDet network paired with logarithmically extracted anchors by $\approx 0.77$ mAP. In addition to this, the forward pass latencies of both SqueezeDet and SqueezeDetOcta are close to $\approx 19$ ms. Boundary adhesion considerations, during training, resulted in an improvement of $\approx 2.62$ mAP of the baseline SqueezeDet network. A SqueezeDet network paired with logarithmically extracted anchors improved the performance of the baseline SqueezeDet network by $\approx 1.85$ mAP.

In summary, this work demonstrates that if given sufficient fine instance annotated data, an existing object detection network can be modified to predict much finer approximations (i.e., irregular octagons) of the instance annotations, whilst having the same forward pass latency as that of the bounding box predicting network. The results justify the merits of logarithmically extracted anchors to boost the performance of any anchor-based object detection network. The results also showed that the special handling of image boundary adhering object instances produces more performant object detectors.
To begin with, I would like to extend my sincere gratitude to my advisor, Prof. Dr. Paul G. Plöger, for his constant encouragement and motivation during this thesis as well as my overall studies at the Hochschule Bonn-Rhein-Sieg. Many thanks to Prof. Dr. André Hinkelmann for agreeing to co-advice me. I would like to thank Dr. Stefan Eickeler for being ever-present throughout this project. I also thank him for helping me stay calm during the countless moments of panic experienced during this project.

I would also like to collectively thank Prof. Dr. Paul G. Plöger and Prof. Dr. Rudolf Berrendorf and the Platform for Scientific Computing at Hochschule Bonn-Rhein-Sieg, for granting me access to use the cluster for this project.

I would also like to extend my deep and earnest gratitude to Dr. Theresa Bick and Sushant Chavan whose valuable insights helped me chip off the rough edges of this report. I would also like to thank M.Sc. Alex Mitrevski, M.Sc. Santosh Thodula, Dharmin Bakaraniya, and Pranav Megarajan for all their help and valuable feedback which helped to improve this project. In addition to this, I would also like to thank all the MAS faculty at the Hochschule Bonn-Rhein-Sieg for the countless things that I have learned during my time here.

I would now like to thank Dr. Sheetal. Pattanshetti and Dr. Vishwanath. Pattanshetti for providing me the confidence to pursue masters abroad. I would like to express my gratitude to my parents. Their support and blessings have always enabled me to perform to the best of my abilities. I would also like to thank my friends and colleagues. Without their support, none of this would have been possible.

Finally, I would like to thank the Almighty for providing me with everything I needed to get myself through this endeavor.
Contents

1 Introduction
   1.1 Motivation ............................................. 2
   1.2 Problem Statement ..................................... 5

2 State of the Art ........................................... 7
   2.1 Semantic Segmentation ................................. 7
   2.2 Object Detection ............................ 12
   2.3 Instance Segmentation .............................. 24
   2.4 Related Work ........................................... 29
   2.5 Summary ................................................ 32

3 Methodology ............................................ 33
   3.1 Shortlisting of the dataset .......................... 34
   3.2 Shortlisting of the object detection network ........ 36
   3.3 Evaluation metrics ..................................... 38
      3.3.1 Mean Average Precision (mAP) .................. 38
      3.3.2 mAP calculation scenarios ....................... 43
      3.3.3 Run-time ............................................ 45
   3.4 Data pre-processing .................................... 46
   3.5 Experimental Design ................................... 48
      3.5.1 Experimental placeholders ....................... 48
      3.5.2 Experiment design process ....................... 49
      3.5.3 Training and evaluation strategies ............. 53

4 Solution ................................................ 57
   4.1 The quest for the finest anchors ..................... 57
      4.1.1 Anchor design as a clustering problem ........ 57
      4.1.2 Non-linearly extracted anchors .................. 59
   4.2 Boundary adhesion considerations ..................... 61
      4.2.1 Detecting boundary adhering object instances ... 61
      4.2.2 Untainted information extraction from boundary adhering object annotations .... 63
   4.3 Ground-truth transformation design .................... 64
      4.3.1 Center-offset encoding/decoding scheme .......... 64
      4.3.2 Anchor-offset encoding/decoding schemes ........ 66
   4.4 Architectural modifications ............................ 70
   4.5 Loss modifications ..................................... 74
5 Results and Evaluation

5.1 Experiment 1 ......................................................... 79
  5.1.1 mAP Analysis ................................................. 80

5.2 Experiment 2 ......................................................... 81
  5.2.1 mAP Analysis ................................................. 81
  5.2.2 Class-wise AP analysis ................................. 82
  5.2.3 Run-time Analysis ....................................... 84

5.3 Experiment 3 ......................................................... 85
  5.3.1 mAP Analysis ................................................. 85
  5.3.2 Run-time Analysis ....................................... 88

5.4 Experiment 4 ......................................................... 88
  5.4.1 mAP Analysis ................................................. 88
  5.4.2 Run-time Analysis ....................................... 95

5.5 Experiment 5 ......................................................... 97
  5.5.1 mAP Analysis ................................................. 98
  5.5.2 Run-time Analysis ....................................... 101

6 Conclusions ......................................................... 103
  6.1 Contributions ...................................................... 106
  6.2 Lessons learned .................................................. 106
  6.3 Limitation ......................................................... 107
  6.4 Future work ....................................................... 108

Appendix A Boundary analysis ................................................. 111

Appendix B Interpreting the plots .............................................. 113
  B.1 Parallel categorical plots ..................................... 113
  B.2 Sun-burst plots .................................................. 115

Appendix C Qualitative results ................................................. 117

References ................................................................. 119
## List of Figures

1.1 Illustrations of the different image segmentation sub-cATEGORIES adopted from Kirillov et al. [22]. **Top left:** object detection, **top right:** semantic segmentation, **bottom left:** instance segmentation, **bottom right:** panoptic segmentation. ........................................... 2
1.2 Expected inference speed vs mask quality plot .................................................. 3
1.3 **Top:** Cityscape dataset (Cordts et al. [7]) **bottom:** sample from the Mapillary Vistas Dataset (Neuhold et al. [39]) dataset .......................................................... 4
1.4 Instance segmentation as fine-grained object detection: (a) The ground truth mask to be learned. This can also be considered as the mask which an instance segmentation approach would generate. (b) Bounding box as a coarse mask generated by an object detection network. (c) Bounding octagonal mask which is finer than the coarse mask generated by the object detection. ........... 6
2.1 Feature map fusing scheme employed by Long et al. [35] ............................................ 7
2.2 ParseNet contexture module introduced by Long et al. [35] ........................................ 8
2.3 Chen et al. [5] DeepLab network overview. ............................................................. 8
2.4 Deconvolution Network for Semantic Segmentation introduced by Noh et al. [41]. ............. 10
2.5 SegNet introduced by Badrinarayanan et al. [1]. .................................................... 10
2.6 A simple example of a high-resolution network introduced by Sun et al. [57]. ...................... 10
2.7 Overview of the CFNet proposed by Zhang et al. [66]. ............................................. 11
2.8 U-net architecture proposed by Ronneberger et al. [49]. ......................................... 12
2.9 Overview of PSPNet proposed by Zhao et al. [68]. .................................................. 12
2.10 Overview of R-CNN proposed by Girshick et al. [14]. .......................................... 13
2.11 SPP layer introduced by He et al. [16]. ................................................................. 14
2.12 Overview of R-FCN proposed by Dai et al. [8]. ..................................................... 15
2.13 Illustration of position-sensitive pooling operation proposed by Dai et al. [8].  
   The ROI is divided into 9 equal sub-regions, corresponding to each sub-region, there are 9 position sensitive score maps. 
   Position sensitive pooling selectively gathers information from these maps (marked by continuous yellow boxes) and merges them to get a single pooled map. ........................................... 15
2.14 **Top:** FPN (Pinheiro et al. [43]) where predictions are made only using finest level features. 
   **Bottom:** FPN (Lin et al. [30]) where the features at all the levels are used for making predictions. 
   The rectangular box illustrates the building block of this FPN. .................................... 16
2.15 Basic idea behind YOLO architecture introduced by Redmon et al. [47]. ..................... 17
2.16 Comparison between SSD architecture by Liu et al. [33] and YOLO architecture by Redmon et al. [47]. ............................................................... 17
2.17 RetinaNet architecture proposed by Lin et al. [31] ................................................ 18
2.18 Deconvolutional module proposed by Fu et al. [11]. .............................................. 20
2.19 Comparison between original SSD architecture (top) by Liu et al. [33] and DSSD architecture (bottom) by Fu et al. [11].

2.20 Prediction modules (a) is the one used by Liu et al. [33], modules (b) to (d) are proposed by Fu et al. [11].

2.21 Comparison of different FPN networks. (d) is the MLFPN introduced by Zhao et al. [69].

2.22 RefineDet architecture proposed by Zhang et al. [67].

2.23 CornerNet architecture proposed by Law and Deng [26].

2.24 ExtremeNet architecture proposed by Zhou et al. [71].

2.25 Top: DeepMask architecture proposed by Pinheiro et al. [42]. Bottom: Positive and Negative image patches and their associated masks.

2.26 (a) DeepMask (Pinheiro et al. [42]) style network. (b) FCN (Long et al. [35]) style network. (c) and (d) SharkMask network and the refinement module proposed by Pinheiro et al. [43].

2.27 Prediction head architectures. (a) The original DeepMask (Pinheiro et al. [42]) head. (b-d) Different prediction head variations with increasing simplicity and speed.

2.28 Mask R-CNN architecture proposed by He et al. [18].

2.29 PANet architecture introduced by Liu et al. [32]. (a) Typical FPN backbone, (b) Bottom-up Path Aggregation, (c) Adaptive feature pooling, (d) Box branch, (e) Fully-connected fusion.

2.30 YOLACT architecture introduced by Bolya et al. [2].

2.31 ESE-Seg (Xu et al. [64]) mask contour predictions.

2.32 ESE-Seg network architecture proposed by Xu et al. [64].

2.33 Network architecture proposed by Jetley et al. [21].

2.34 Left: Octagonal masks constructed by ExtremeNet (Xu et al. [64]) from the extreme point predictions. Right: Actual instance masks.

3.1 Ground-truth extraction examples. Left: Cityscape (Cordts et al. [7]) fine pixel-wise annotation, Right top row: Bounding box extraction for “CAR”, Right middle row: Octagonal mask extraction for “CAR”, Right bottom row left: Bounding box extraction for “PERSON” and Right bottom row right: Octagonal mask extraction for “PERSON”. Significant amount of false positive pixels are avoided by using octagon as compared to bounding box.

3.2 Precision and Recall illustration by Walber [61].

3.3 IoU illustration inspired by Rosebrock [50]. The figure shows how IoU can be calculated when the ground truth type varies between {bounding box, instance mask} and detection type varies between {bounding box, octagonal mask}.

3.4 Illustrations by Rafael Padilla and da Silva [44].

3.5 mAP-(Box) x (Box) calculation scenario.

3.6 mAP-(Mask) x (Box or Octagon) calculation scenario.

3.7 Inference pipeline of a typical anchor based object detection network.

3.8 Class instance distribution in the Cityscape dataset.

3.9 Grouped instance distribution as a percentage of the total sample size.
4.1 Linearly extracted anchors .................................................. 58
4.2 Logarithmically extracted anchors ........................................... 58
4.3 Bounding box parameterization comparison ............................. 63
4.4 Parameterizations used by center-offset encoding/decoding scheme. The green bounding-box/polygon represents the ground-truth and the red bounding-box represents the anchor box. .............. 65
4.5 Parameterizations used by anchor-offset linear encoding/decoding scheme. The green bounding-box/polygon represents the ground-truth and the red bounding-box represents the anchor box. . . 67
4.6 Parameterizations used by anchor-offset non-linear encoding/decoding scheme. The green bounding-box/polygon represents the ground-truth and the red bounding-box represents the anchor box. . . 68
4.7 Network modifications needed for octagonal mask prediction. . . . 71
4.8 Fire module introduced by Iandola et al. [19] ............................ 72

5.1 Flow-diagram illustrating the dependency of mAP on the candidate network placeholder values for experiment 1. Only placeholders with unique values are indicated in the figure to reduce clutter. 80
5.2 Flow-diagram illustrating the dependency of mAP on the candidate network placeholder values for experiment 2. Only placeholders with unique values are indicated in the figure to reduce clutter. 82
5.3 Plot of difference in the average precision values vs the sample size for the two mAP calculation scenarios (specified in the individual titles). The size of the markers is proportional to the sample size. ................................................................. 83
5.4 Sun-burst plots representing the run-time comparison of the different candidate networks. . . . 84
5.5 Flow-diagram illustrating the dependency of mAP on the candidate network placeholder values for experiment 3. Only placeholders with unique values are indicated in the figure to reduce clutter. 86
5.6 Flow-diagram illustrating the dependency of mAP on the candidate network (bounding box predicting only) placeholder values for experiment 4. Only placeholders with unique values are indicated in the figure to reduce clutter. ........................................ 89
5.7 Flow-diagram illustrating the dependency of mAP on the candidate network (octagonal mask predicting only) placeholder values for experiment 4. Only placeholders with unique values are indicated in the figure to reduce clutter. ........................................ 91
5.8 Flow-diagram illustrating the dependency of mAP on the candidate network placeholder values for experiment 4. Only placeholders with unique values are indicated in the figure to reduce clutter. 93
5.9 Plot of difference in the average precision values vs the sample size for different mAP calculation scenarios. Top row: Plots for linear candidate pair. Bottom row: Plots for non-linear candidate pair. The size of the markers is proportional to the sample size. ................................................................. 94
5.10 Sun-burst plots representing the run-time comparison of the different candidate networks. . . . 96
5.11 Flow-diagram illustrating the dependency of mAP on the linear candidate network placeholder values for experiment 5. Only placeholders with unique values are indicated in the figure to reduce clutter. ........................................ 98
5.12 Flow-diagram illustrating the dependency of mAP on the non-linear candidate network placeholder values for experiment 5. Only placeholders with unique values are indicated in the figure to reduce clutter. ........................................ 99
5.13 Sun-burst plots representing the run-time comparison of the different candidate networks. . . . . 102

6.1 Summary of all experiment conducted as a part of this work. . . . . . . . . . . . . . . . . . . . . . . . 105
6.2 Octagonal mask predictions overlapping each other in a cluttered environment. . . . . . . . . . 107
6.3 Left: Distorted mask predictions. Right: Post-process correction. Notice the knot and the
associated correction indicated by ellipse . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 108

A.1 Margin vs image sample number in Cityscape dataset. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 111
A.2 Illustrations for the bottom margin values. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112

B.1 Parallel categorical plot explained. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 113
B.2 Sun-burst plot explained. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 115


<table>
<thead>
<tr>
<th>Table Number</th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Comparison of prominent autonomous driving datasets.</td>
<td>35</td>
</tr>
<tr>
<td>3.2</td>
<td>Candidate networks which will be used for the experiment 1.</td>
<td>50</td>
</tr>
<tr>
<td>3.3</td>
<td>Candidate networks which will be used for the experiment 2.</td>
<td>51</td>
</tr>
<tr>
<td>3.4</td>
<td>Candidate networks which will be used for the experiment 3.</td>
<td>51</td>
</tr>
<tr>
<td>3.5</td>
<td>Candidate networks which will be used for the experiment 4.</td>
<td>52</td>
</tr>
<tr>
<td>3.6</td>
<td>Candidate networks which will be used for the experiment 5.</td>
<td>53</td>
</tr>
<tr>
<td>4.1</td>
<td>Linear anchors.</td>
<td>59</td>
</tr>
<tr>
<td>4.2</td>
<td>Log anchors.</td>
<td>59</td>
</tr>
<tr>
<td>4.3</td>
<td>Cityscape dataset (Cordts et al. [7]) image margins.</td>
<td>61</td>
</tr>
<tr>
<td>4.4</td>
<td>Encoding and decoding transformations employed by center-offset encoding/decoding scheme.  ( \epsilon = 10^{-8} ) is a small value added to the offsets to prevent invalid values inside logarithm.</td>
<td>66</td>
</tr>
<tr>
<td>4.5</td>
<td>Encoding and decoding transformations employed by anchor offset linear encoding/decoding scheme</td>
<td>68</td>
</tr>
<tr>
<td>4.6</td>
<td>Encoding and decoding transformations employed by anchor offset non-linear encoding/decoding scheme. Based on experiments ( \rho = 0.5 ) produced stable training and hence is used for all the experiments.</td>
<td>69</td>
</tr>
<tr>
<td>5.1</td>
<td>Candidate networks that were used for experiment 1.</td>
<td>80</td>
</tr>
<tr>
<td>5.2</td>
<td>Candidate networks that were used for experiment 2.</td>
<td>81</td>
</tr>
<tr>
<td>5.3</td>
<td>Candidate networks that were used for experiment 3.</td>
<td>85</td>
</tr>
<tr>
<td>5.4</td>
<td>Candidate networks that were used for experiment 4.</td>
<td>88</td>
</tr>
<tr>
<td>5.5</td>
<td>Candidate networks that were used for experiment 5.</td>
<td>97</td>
</tr>
</tbody>
</table>
1

Introduction

Image segmentation can be defined as the task of dividing an image/video frame into distinct segments by grouping pixels representing semantically similar entities. It is an extremely broad category of problems in computer vision and has been divided into several fine-grained sub-categories in recent years. Predominantly image segmentation comes in the following flavors,

1. **Semantic Segmentation**: This is the simplest sub-category of image segmentation. It deals with partitioning an image/video frame into groups of semantically similar entities without any regard to the individual instances of those entities. In other words, semantic segmentation does not delineate between different instances of entities occurring in images/video frames. This is illustrated by the top right sub-figure in Figure 1.1.

2. **Instance Segmentation**: Instance segmentation deals with the task of partitioning an image/video frame into groups of individual instances of semantically similar entities. This is illustrated by the bottom left sub-figure in Figure 1.1.

3. **Object Detection**: Object detection is defined as the task of detecting and localizing instances of objects in an image/video frame typically using bounding boxes. In other words, object detection can be considered as a coarser variant of instance segmentation where the segmentation masks are in the form of rectangular bounding boxes. Hence it can be considered as another sub-category of image segmentation. This is illustrated by the top left sub-figure in Figure 1.1.

4. **Panoptic Segmentation**: Panoptic segmentation is the most challenging of all the image segmentation subcategories. It is a new area of image segmentation research. It is a combination of semantic segmentation and instance segmentation. Typically, entities having large areas are segmented using semantic segmentation and individual instances of semantically similar entities with smaller areas are segmented using instance segmentation. This is illustrated by the bottom right sub-figure in Figure 1.1. Panoptic segmentation is essential for complete scene understanding, however, using it in the perception pipeline increases the latency of the pipeline. A lot of the current research in panoptic segmentation is targeted towards addressing this high latency issue. For many practical applications instance/semantic segmentation is most often sufficient.
1.1. Motivation

Before 2012 (the year of AlexNet by Krizhevsky et al. [25]), most of the image segmentation algorithms were built on top of application-centric handcrafted features. Post-2012, deep learning based techniques showed significant prowess, achieving state-of-the-art performances on challenging image segmentation datasets, e.g., PASCAL VOC (Everingham et al. [10]), MS COCO (Lin et al. [29]), etc.

The real world is dynamic and fast response (close to real-time\textsuperscript{1}) is crucial for autonomous systems operating in it. The perception module is just one of the many sub-systems in the entire autonomous system. The ability to finely segment different instances of different objects in a scene swiftly is essential if the hard real-time deadline needs to be met.

1.1 Motivation

Instance segmentation task can be formulated in two different ways,

- Generalized semantic segmentation, which takes into consideration the different instances of the objects in the scene.

- Fine-grained object detection, where the mask generated is finer than the bounding box.

\textsuperscript{1}Real-time here means 30 frames per second
Chapter 1. Introduction

Currently, instance segmentation is treated as a binary multi-label pixel-wise classification task. Approaches like Mask R-CNN by He et al. [18] rely on RPN\(^2\) to generate ROIs\(^3\) which are then pixel-wise classified by another sub-network making it extremely slow for systems with limited onboard computing capabilities. YOLACT by Bolya et al. [2] achieves an impressive 33.5fps on MS COCO (Lin et al. [29]) dataset with an image resolution of 550 × 550. However, many real-world datasets like Cityscape (Cordts et al. [7]) have image dimensions of 2048 × 1024 and object instances of different scales. Rescaling the images to 550 × 550 squashes the object belonging to smaller scales. Increasing the input dimensions of the YOLACT decreases the frame-rate of the network. On the contrary, low latency object detectors operating on large input image resolutions are widely available [Wu et al. [62], Lin et al. [31], Zhou et al. [70]]. Hence most autonomous systems operating in real-world scenarios rely on object detection as a means to segment the different objects in the environment. However, the segmentation achieved by rectangular axis-aligned bounding boxes is extremely coarse, especially in cluttered environments.

In this work, the primary objective is to try to achieve the right trade-off between the quality of the segmentation and the run-time. The instance segmentation task is envisioned as fine-grained object detection where the mask generated is finer than the bounding box (but is still coarser than the pixel-wise instance masks). For this proof-of-concept, an irregular octagon was considered as an ideal shape which could be used to approximate an instance mask. Formally speaking, instance segmentation is considered as a regression task as opposed to a multi-label pixel-wise classification task. This formulation facilitates by-passing the use of an additional mask calculating sub-network. In other words, it is expected that this formulation will enable the generation of instance masks finer than bounding boxes using already available fast object detection networks with a nominal increase in run-time. Figure 1.2 illustrates this expectation.

![Figure 1.2: Expected inference Speed vs mask quality plot](image)

\(^2\)Region Proposal Network
\(^3\)Region Of Interest
While attempting to achieve the above mentioned primary objective, two limitations of the current regression-based object detectors have come to light.

**Field of view restrictions and associated annotation noise:**

Most of the state-of-the-art object detection or segmentation (instance/semantic) techniques are supervised. These techniques learn the correspondence between the image-annotation pairs in the dataset. Hence the performances of these networks are limited by the quality of annotations. A great deal of time and human effort has been spent in formulating the standard image segmentation datasets like MS COCO (Lin et al. [29]), Cityscape (Cordts et al. [7]), KITTI (Geiger et al. [12]), etc.

Despite the best efforts of humans, there is one limitation in the inherent annotation process which can limit the learning capabilities of the network. It is a fairly obvious fact that a human annotator can label only the object portions he/she can perceive in the image. In other words, some objects might only be partially visible in the current scene (since the camera has a finite field of view), and this results in the human annotating the object only partially. Such examples are widely seen in almost all the standard image segmentation datasets. A few examples from some of the popular autonomous driving segmentation datasets are provided in Figure 1.3. This observation is also true for object detection datasets like KITTI (Geiger et al. [12]).

These potential problematic object instances are seen at the borders of the image frames. In both, the images in Figure 1.3, the cars on the right are the problematic object instances. One might argue that occluded images also qualify as partial object instances and might also be problematic for the network to learn from. However, occluded object instances though partial are not problematic. The reason for this distinction is that most of the state-of-the-art approaches for image segmentation are based on Convolutional Neural Network (CNN). These networks have a fixed limited receptive field using which they can scan the input image much like a sliding window. The network is capable of learning to segment the partially occluded object instances by scanning the surroundings of the occluded object. At the image boundaries, however, there is nothing for the network to scan to make sense of the abrupt partial object instance. This work hypothesizes that if these partial object instances at the image boundaries are included during training, they can limit the segmentation performance of the network. To the best of our
knowledge, this has gone unnoticed by the research community. 

Addressing this limitation should improve the performance of any regression-based image segmentation model.

Inconsistency between anchor generation and ground-truth encoding:

Deep-learning based object detection techniques can be broadly classified into two categories,

1. Anchor based object detectors: In these networks, the input image is discretized by a grid. Each grid center is associate with a fixed number of rectangular boxes of varying aspect ratios. These boxes are called anchor boxes or just anchors. One drawback of using anchors is that they introduce additional hyper-parameters (like the anchor’s widths and heights) which need to be tuned for the task at hand. Faster R-CNN (Ren et al. [48]), SSD (Liu et al. [33]) and YOLO (Redmon et al. [47]) are a few anchor-based object detectors.

2. Anchorless object detectors: In these networks, the need for prior anchor boxes and the hyper-parameters associated with them are eliminated. The networks instead use a relatively large resolution heat-map to localize the different instances of the objects in the image. CornerNet (Law and Deng [26]), ExtremeNet (Zhou et al. [71]) and CenterNet (Zhou et al. [70]) are a few anchor-less object detectors.

The aspect ratios of the anchors were hand-picked by Ren et al. [48] in Faster R-CNN. Wu et al. [62] and Redmon and Farhadi [45] proposed to set the aspect ratios of these anchor boxes by using the distribution of the bounding boxes in the dataset. This enabled a way to incorporate prior knowledge about the data distribution into the network design. This facilitated faster convergence and better performing object detectors. The clustering proposed by Wu et al. [62] and Redmon and Farhadi [45] was in the cartesian space defined by the width and the height of the bounding boxes, as the two co-ordinate axes. However, the object detector network predicts a non-linearly transformed (logarithmic) version of the ground-truth bounding box width and height. This results in a discrepancy between the ground-truth encoding transformation performed by the object detector network and the coordinate space used for clustering.

Addressing this limitation should improve the performance of any anchor-based image segmentation model.

1.2 Problem Statement

The primary goal of this work is to quantitatively prove that a good trade-off between precision and run-time for object instance segmentation can be achieved by formulating the instance segmentation as a regression problem as opposed to a multi-label pixel-wise classification problem. This can be achieved by modifying an existing object detector network to predict the parameters of an octagon instead of a bounding box. In Figure 1.4, if the region inside the bounding box and bounding octagon are considered to represent the instance mask, the red pixels represent the False Positive pixels. By using a bounding octagon, the number of False Positive pixels is reduced and the generated mask more closely resembles the actual instance mask.
To address the issue of the problematic object instances at the borders of the image, elaborated in section 1.1 there are two possible courses of action,

1. Completely ignore the problematic object instances.
2. Selectively learn only partial parameters of the object instance.

Since annotated data is extremely valuable, the first course of action is not suitable especially for sparse datasets. Hence the second course of action is chosen to address the issue of problematic object instances at the image borders.

To address the issue of the inconsistency between anchor generation and ground-truth encoding mentioned in section 1.1, this work proposes clustering in a co-ordinate system obtained by performing non-linear logarithmic transformation of the width and the height of the ground-truth bounding boxes.
Over the years the image segmentation sub-categories described in chapter 1 have seen simultaneous parallel development. This work lies in the intersection of object detection and instance segmentation. However, many of the design principles incorporated by the research conducted in these two sub-categories have been influenced by the research in semantic segmentation. Hence in this chapter, the most influential research in semantic segmentation, object detection, and instance segmentation will be reviewed.

2.1 Semantic Segmentation

Deep-learning based models, typically formulate the semantic segmentation as a multi-label pixel-wise binary classification problem. Fully Convolutional Networks (FCN) introduced by Long et al. [35] was one of the first effective deep-learning based techniques for semantic segmentation. The authors modified back-bone feature extractors like VGG-16 (Simonyan and Zisserman [54]) and GoogLeNet (Szegedy et al. [59]) by replacing all the fully-connected layers by convolutional layers similar to the Overfeat object detection network by Sermanet et al. [53]. This enables the network to produce a spatial output. The network uses skip connections and element-wise sum to fuse the appearance information from the shallower layers with the semantic information from the deeper layers. The low-resolution coarser feature maps from deeper layers are up-sampled using learnable deconvolutional layers while merging with high-resolution feature maps from lower layers. The information fusing scheme used by Long et al. [35] is as illustrated in Figure 2.1.

![Figure 2.1: Feature map fusing scheme employed by Long et al. [35]](image)

Out of all the outputs in Figure 2.1, FCN-8s produces the most accurate results however, it is slower.
2.1. Semantic Segmentation

because of the added operations. Long et al. [35] through this work demonstrated that deep convolutional neural networks can be used in an end-to-end trainable manner for semantic segmentation. Lack of global context while making predictions and slow inference speed (not real-time) are the main drawbacks of this technique.

To address the issue of lacking global context in the FCN (Long et al. [35]), Liu et al. [34] introduced a new variant of FCN called as ParseNet. They introduced a module to incorporate the global context called the ParseNet contexture module which is as shown in Figure 2.2.

![Figure 2.2: ParseNet contexture module introduced by Long et al. [35]](image)

The feature maps from a particular layer are globally pooled to get a context vector. The context vector is L2 Normalized, Unpooled, and then concatenated with the L2 Normalized version of the standard feature map. Liu et al. [34] observed that the scales of the features from different layers of the network are different and this introduces issues while combining the feature maps. To overcome this, they proposed to scale each of the feature maps using scaling factors which are learned during backpropagation. The ParseNet improves on the accuracy of the FCN networks whilst adding only a small computational overhead.

Chen et al. [5] observed that the response from the deeper layer of the CNN have poor localization capabilities and hence are not suitable for dense pixel-wise segmentation.

![Figure 2.3: Chen et al. [5] DeepLab network overview.](image)

To counter this, the authors proposed combining CNNs with fully connected Conditional Random Fields (CRF) with gaussian edge potentials (Krähenbühl and Koltun [23]). The network overview is as
shown in Figure 2.3. A fully convolutional variant of a deep CNN like VGG-16 (Simonyan and Zisserman [54]) or ResNet-101 (He et al. [17]) is used as a feature extractor. To reduce the degree of down-sampling performed by the feature extractor, Chen et al. [5] proposed to replace standard convolutions with atrous convolution. The output feature maps from the CNN are up-sampled to the input image resolution using bilinear interpolation and fed to a Fully connected CRF. CRF is a class of discriminative probabilistic undirectional graphical models which are generally used in applications where the context information is essential to make accurate predictions. For the task of semantic segmentation, the semantic label at one pixel location is highly dependent on the semantic label of the pixels in its neighborhood. Each pixel in the upsampled image is considered as a vertex in a fully connected graph. Corresponding to each vertex, a semantic label is to be predicted. Each edge or a pairwise connection between vertices in this graph is associated with a potential. The pairwise potentials are defined as a linear combination of Gaussian kernels. These kernels can have arbitrary shape and can also be formulated on an arbitrary feature space. The inference algorithm for the fully connected CRFs is based on mean field approximation. Instead of predicting the exact probability distribution \( P(X) \) over all the label assignments, the mean field approximation algorithm computes another simpler distribution \( Q(X) = \prod_i Q(x_i) \) that minimizes the KL divergence between the exact distribution \( P(X) \) and the approximate distribution \( Q(X) \). The labels \( \hat{x}_i \) can then be approximated as the maximum of the independent marginals i.e. \( Q(x_i) \). The mean field approximation is an iterative optimization process. Empirically it was found that the KL divergence converges after around 10 iterations.

With this setup, the model was able to accurately delineate the boundaries between different segments. A similar idea of augmenting CNNs with fully connected CRFs has also been investigated by Lin et al. [28].

Schwing and Urtasun [52] claimed that the semantic segmentation techniques that use a combination of CNNs for feature extraction and a probabilistic graphical model for refinement of the segmentation masks, train the models in a piece-wise fashion. To address this issue, they introduced a technique in which the parameters of the VGG-16 (Simonyan and Zisserman [54]) and the fully connected CRF could be jointly trained.

Noh et al. [41] proposed a semantic segmentation algorithm based on deconvolution operation. The network is partitioned into an encoder and a decoder sub-networks. The encoder is a CNN based on VGG-16 (Simonyan and Zisserman [54]) architecture. The decoder sub-network reverses the processing done by the encoder sub-network. Since the encoder mainly consists of convolutional and pooling layers, the decoder contains deconvolutional and unpooling layers. Figure 2.4 represents an overview of the combined network.

Badrinarayanan et al. [1] proposed optimizations to the architecture proposed by Noh et al. [41] and the resulting network was called SegNet. It is as shown in Figure 2.5. The network architecture of SegNet is almost similar to the one proposed by Noh et al. [41], except for the mechanism used by SegNet to up-samples the feature maps in the decoder sub-network. SegNet keeps track of the pooling indices for every pooling layer in the encoder sub-network and then it uses these pooling indices to perform a form of non-linear up-sampling in the decoder sub-network. Hence the need to learn the up-sampling using deconvolutional layers is avoided.
2.1. Semantic Segmentation

An alternate approach to obtain a high-resolution representation was employed by Sun et al. [57]. Instead of trying to recover high-resolution representation from low and medium-resolution feature maps like SegNet (Badrinarayanan et al. [1]) and Deconvolution Network for Semantic Segmentation (Noh et al. [41]), Sun et al. [57] first employ high-resolution convolutions to maintain a high-resolution representation and then fortifies this representation using lower resolution parallel streams. They adopt the HRNet framework, (originally proposed for human pose estimation by Sun et al. [56]), as the base and modify it to use all the parallel convolutional stream outputs for making predictions. They call this modified network as HRNetv2. The overview of the high-resolution network is as shown in Figure 2.6.

HRNetv2 achieves state-of-the-art results on PASCAL Context (Mottaghi et al. [38]), Cityscape (Cordts et al. [7]) and LIP (Gong et al. [15]) datasets. The authors also use this high resolution features as input to the Faster R-CNN (Ren et al. [48]) object detector network. It shows considerable improvement, especially in detecting small objects.
Another line of research looks into the probable advantages of not just incorporating global context but also including relational context between different entities in the scene for pixel-wise labeling tasks. Zhang et al. [66] try to model the relational context between different entities by analyzing the co-occurrence between different features. They claim that the spatial invariance property of CNNs makes them incapable of taking into account co-occurring feature information. To counter this, the authors probabilistically model the feature co-occurrence. In other words, the feature co-occurrence is modeled as a probability distribution over the feature space, conditioned on a specific target feature. This model is called Co-occurrence Feature Model (CFM). This model uses the similarity between the target feature and the other features to determine how likely the target feature will co-exist with the other features in a given scene. Since this relationship is highly dependent on the scene being processed, Zhang et al. [66] model the scene context as contextual priors defined as a Mixture of Softmaxes. The combined model is termed as Co-occurrence Feature Network (CFNet) and is as shown in Figure 2.7.

In the field of medical image segmentation, encoder-decoder based architectures have attracted a lot of interest. One of the most influential deep-learning based network for biomedical image segmentation was proposed by Ronneberger et al. [49] and is called U-Net. The architecture is as shown in Figure 2.8.

It consists of encoder sub-network which extracts semantic information at the expense of localization accuracy. The decoder sub-network restores the localization information using the skip connections from the encoder. 3×3 convolutional operator is used for the encoder. The up-sampling in the decoder is performed with the help of the deconvolutional operator. Finally a 1×1, convolution is used to generate the segmentation map. Variations of U-Net have been employed in diverse domains ranging from road segmentation in the autonomous driving domain to deep-fake creation and detection.

Motivated towards effective incorporation of global context into the learning process, Zhao et al. [68] introduced a new pooling module called Pyramid Pooling Module. Using this module at its core, they introduced a segmentation network called as PSPNet. Figure 2.9 illustrates the architecture of PSPNet.

The PSPNet contains a ResNet (He et al. [17]) with the dilation strategy as the backbone feature extractor. The extracted features are passed through the Pyramid Pooling Module, which gathers
2.2 Object Detection

State-of-the-art deep-learning based models formulate object detection as a combination of classification and bounding box regression problem. Object detection landscape has evolved considerably over the last decade. In the year 2013, Sermanet et al. [53] provided a unified framework for using a single CNN network for classification, localization, and detection called Overfeat. It can be considered as the first single-stage

```latex
\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{unet.png}
\caption{U-net architecture proposed by Ronneberger et al. [49].}
\end{figure}
```

```latex
\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{pspnet.png}
\caption{Overview of PSPNet proposed by Zhao et al. [68].}
\end{figure}
```

representations at different contexts. These are passed through a $1 \times 1$ convolution to reduce the dimension along the channel. The feature maps are then up-sampled using bilinear interpolation. The up-sampled feature maps are then concatenated with the feature map outputs from the backbone network and then passed through a convolution layer to get the final output segmentation maps.

2.2 Object Detection

State-of-the-art deep-learning based models formulate object detection as a combination of classification and bounding box regression problem. Object detection landscape has evolved considerably over the last decade. In the year 2013, Sermanet et al. [53] provided a unified framework for using a single CNN network for classification, localization, and detection called Overfeat. It can be considered as the first single-stage
convolutional object detection network. Most of the object detection techniques before it used different sliding window crops of the input image and then passed these crops through a CNN to determine if the crop contained a particular class or not. This was done because the early CNN classification networks like AlexNet (Krizhevsky et al. [25]) needed a fixed size input. This restriction was imposed by the fully connected layers used in these early CNN architectures. Sermanet et al. [53] found that by replacing the fully connected layers using $1 \times 1$ convolutions, the restrictions on fixed input size can be eased and the entire image can be passed through the CNN irrespective of the image resolution. A CNN inherently works by scanning the image analogous to a sliding window and it produces a spatial output if the input image size is larger than the receptive field of the network. In Overfeat, a 6 scale image pyramid is constructed to process the images at multiple scales. All the scales of the image pyramid are passed through the CNN feature extractor. The output features are then passed into two branches, one for classification and another for direct bounding box regression. The output volumes of each of these branches depend on the scale of the image in the image pyramid.

R-CNN introduced by Girshick et al. [14] formulated the object detection as a two-stage coarse-to-fine refinement problem. It uses selective search (Uijlings et al. [60]) to generate around $2k$ object proposals from the input image. Object proposals are class-agnostic candidate boxes which have a large probability of containing objects. Each of these proposals is first resized and then passed through an AlexNet (Krizhevsky et al. [25]) model, pre-trained on ILSVRC2012 (Deng et al. [9]) classification dataset. One linear SVM per class is applied to the features extracted by CNN to recognize the object categories present in the object proposals. Bounding box refinement of the object proposals is performed using a linear regression model. The overview of the network is as shown in Figure 2.10. Some of the drawbacks of R-CNN are as follows,

- It uses a multi-stage training schedule which is extremely slow.
- Inference is extremely slow.
- Selective search is not trainable, hence bad candidates can increase latency.

SPP-net was introduced by He et al. [16] to address the large latency issue of the R-CNN (Girshick et al. [14]). The AlexNet (Krizhevsky et al. [25]) feature extractor used by R-CNN required the images to have a fixed input size (e.g., $224 \times 224$). This restriction was imposed by the use of fully connected layers in the original AlexNet (Krizhevsky et al. [25]) implementation.
To counter this issue, He et al. [16] introduced a new pooling layer termed as Spatial Pyramid Pooling (SPP). This pooling layer used pooling bins of sizes proportional to input feature map sizes. This enabled the layer to produce fixed-length output features irrespective of the size of the input features. This in-turn facilitated the computing of the features for the entire image only once. The fixed-length features for any subset/crop of the input image could then be quickly extracted using Spatial Pyramid Pooling which is as shown in Figure 2.11.

Girshick [13] noticed the following open-issues with the SPP-net architecture (He et al. [16]),

- He et al. [16] still employed slow multi-stage training.
- The network parameters below the spatial pyramid pooling layer are fixed and are not tuned during training.

Girshick [13] observed that the spatial pyramid pooling layer introduced by He et al. [16] was not differentiable hence it could not be trained. In Fast R-CNN, Girshick [13] countered this problem by employing a differentiable pooling layer called ROI pooling. Girshick [13] claims that the ROI pooling layer was a special-case of the SPP layer, which contained only one pyramid level. In other words, the input image is first passed through a convolutional feature extractor. The object proposals are projected on to the extracted feature maps. The projected object proposals are then passed through the ROI pooling layer to get fixed-length features. These features corresponding to each object proposal are then passed through a series of fully connected layers to get two outputs.

1. The first output is a discrete probability distribution over \( K + 1 \) output classes (\( K \) classes and an additional class for background).

2. The second output is bounding box regression offsets.

Girshick [13] employed a single multi-task loss, which is a combination of log loss for classification and smooth L1 loss for bounding box regression. This enabled joint training of the classifier and the regressor.

In Fast R-CNN (Girshick [13]) the speed bottleneck was due to region proposal network. Ren et al. [48] introduced Faster R-CNN to counter this bottleneck by replacing the selective search based region proposal with a trainable CNN network called Region Proposal Network (RPN). It is considered as the first end-to-end learnable deep-learning based two-stage object detection network. Because of the additional network, the training process now involves 4 losses i.e., RPN classification loss, RPN regression loss, Final classification loss, Final bounding box regression offset loss.
CNN architectures developed after VGG (Simonyan and Zisserman [54]) and AlexNet (Krizhevsky et al. [25]) are fully convolutional. Dai et al. [8] tried to incorporate these newer architectures into the Faster R-CNN setup. It was found that the same architectures which achieved state-of-the-art performance on the classification task struggled to achieve comparable performance on object detection. Dai et al. [8] argued that this was due to the contradicting objective dilemma between increasing translational invariance needed for image classification and respecting translational variance for object detection. To overcome this dilemma they proposed using position-sensitive score maps for each ROI. The resulting network is termed as Region-based Fully Convolutional Network (R-FCN) and is as shown in Figure 2.12.

Each ROI proposal predicted by RPN is divided into 9 sub-regions i.e., top-left, top-center, top-right, center-left, center-center, center-right, bottom-left, bottom-center and bottom-right. Corresponding to these 9 regions, there are 9 position-sensitive score maps for each of the $C + 1$ classes ($C$ actual classes and 1 additional background class). These score maps are pooled using a position-sensitive ROI pool operation. This pooling operates on each class separately, giving a combined output volume of $3 \times 3 \times (C + 1)$. Figure 2.13 illustrates the position-sensitive pooling operation. The global average pool of these feature maps is calculated and this represents the $C + 1$ class scores. A softmax is applied to the class score values and the class with the maximum probability value is the chosen class. For bounding box regression a sibling branch is used which produces an output volume if $3 \times 3 \times 4$ (4 because, 4 parameters are used to describe one bounding box) each ROI proposal. This is then converted to a 4-dimensional vector by averaging.

The two-stage object detection networks though accurate, were not able to match the inference speed of the single-stage counterparts. Li et al. [27] observed that in-spite of using shallower backbone networks for the RPN, there was no significant gains in inference speed in the Faster R-CNN (Ren et al. [48]) and R-FCN (Dai et al. [8]) frameworks. They state that this observation was due to the use of heavy
2.2. Object Detection

detection heads in these frameworks. Faster R-CNN (Ren et al. [48]) uses two fully connected layers for ROI recognition and R-FCN (Dai et al. [8]) uses large resolution position-sensitive score maps. To counter this problem, Li et al. [27] propose a new two-stage network that redesigns the detection heads to be light. The resulting network is rightly termed as Light-Head R-CNN. It uses thin feature maps obtained by applying large separable convolutions on the output feature maps from the backbone network. To improve the speed further, it incorporates a shallow computationally cheap variant of R-CNN detection head which contains ROI pooling and a single fully connected layer.

In the year 2017, Lin et al. [30] proposed using Feature Pyramid Networks (FPN) in the Faster R-CNN (Ren et al. [48]) setup. It enables processing objects at different scales effectively. Most of the object detectors, performed detection using the feature maps from the deeper ends of the network. These feature maps were ideally suited for classification tasks as they have high location invariance and contain high semantic information. Feature maps from shallower layers of the network have high location information but less semantic meaning. Through FPN, Lin et al. [30] enabled a way to merge the location information and semantic information at different scales. FPN variant of a backbone network, with a very small run-time overhead as compared to the normal backbone network, is able to generate information-rich features. When these features were used in the Faster R-CNN (Ren et al. [48]) setup, the resulting object detector was able to achieve state-of-the-art results for a single model on MS COCO (Lin et al. [29]) dataset. The development of FPNs has been a critical milestone in object detection research.

Motivated towards generating an real-time object detection network, Redmon et al. [47] introduced YOLO to the world in 2016. It is considered as the first deep-learning based end-to-end learnable single-stage object detection system (Minaee et al. [37]). YOLO demonstrated that object detection can be done without the need for region proposals. Figure 2.15 represents the overview of the YOLO architecture. Redmon et al. [47] discretizes the input image into a grid of size $S \times S$. If the center of an object falls within a particular grid cell, then that cell is responsible to detect that particular object. The network predicts $B$ bounding boxes for each grid cell. The network in a single pass produces an output volume of size $S \times S \times (B \times (4 + 1) + C)$ i.e., 4 bounding box parameters for B bounding boxes, 1 confidence score for each of the B bounding box and C dimensional class probability distribution conditioned over the fact that the grid cell contains an object. YOLO achieved an mAP of 63.4% on PASCAL VOC (Everingham et al. [10]) dataset whilst running at an impressive 45 fps. The main drawback of YOLO architecture is poor localization.
Chapter 2. State of the Art

Single Shot Detector (SSD) by Liu et al. [33] is another important single-stage anchor-based object detection network. The main contribution of SSD is the use of prior default anchor boxes not just across different aspect ratios but also across different scales. Liu et al. [33] associates feature maps outputs of different layers of the network with prior anchor boxes of different scales. Hence the specific feature maps respond better to specific scales of the objects. The network in a single pass predicts class probability scores and bounding box offsets relative to the default anchor boxes. The authors noticed that during training, a large number of anchor bounding boxes do not contain any object. This might overwhelm the network. Hence they adopt hard negative mining. In hard negative mining, the default boxes having high confidence loss are selected as candidates for negative examples. The ratio between the negative to positive examples is set to 3:1. Liu et al. [33] also introduced a data-augmentation scheme which led to considerable gains in the performance. SSD with input size of 300 × 300 achieves an mAP of 74.3% on PASCAL VOC (Everingham et al. [10]) 2007 test dataset at an impressive 59 fps on Nvidia Titan X GPU. SSD with input size of 512 × 512 outperforms Faster R-CNN (Ren et al. [48]) by achieving an mAP of 76.9% on PASCAL VOC (Everingham et al. [10]) 2007 test dataset.

Figure 2.16: Comparison between SSD architecture by Liu et al. [33] and YOLO architecture by Redmon et al. [47].
2.2. Object Detection

SqueezeDet is a small but powerful fully convolutional anchor-based single-stage object detection network introduced by Wu et al. [62] particularly for the autonomous driving domain. The standout feature of this network was that it achieves an impressive 57.2 fps with an input image resolution of 1242×375. Wu et al. [62] reduce the model size by employing the tiny SqueezeNet architecture introduced by Iandola et al. [19] as the backbone feature extractor and a small fully convolutional detection head. The network is validated using the KITTI object detection dataset (Geiger et al. [12]).

Single-stage object detectors though fast, consistently lagged behind the two-stage object detectors in terms of accuracy. Lin et al. [31] discovered that the sub-par accuracy of single-stage detectors can be attributed to the class imbalance between the foreground and background during training. To overcome this class imbalance problem, Lin et al. [31] introduced a modification of the standard cross-entropy loss called as Focal loss. The Focal loss focuses on training the network using a sparse set of hard ill-classified examples. Lin et al. [31] implements focal loss by down-weighting the loss assigned to the well-classified examples. The resulting network is called as RetinaNet and its architectural overview is as shown in Figure 2.17.

![RetinaNet architecture proposed by Lin et al. [31]](image)

The different variations of RetinaNet either use a ResNet-101 (He et al. [17]) with FPN or ResNeXt-101 (Xie et al. [63]) with FPN as the backbone feature extractors. The different levels of features extracted by the FPN backbone network are passed through two sub-networks, a Class sub-network, which predicts the K class scores for each prior anchor box and a Box sub-network, which predicts 4 bounding-box offsets for each prior anchor box. RetinaNet bridges the accuracy gap between two-stage and single-stage networks whilst maintaining low latency characteristics of single-stage networks.

Redmon and Farhadi [45] provided an incremental update to the original YOLO architecture and the resulting network is called YOLOv2. The network at input resolution of 544×544 achieves 78.6 mAP on PASCAL VOC 2007 (Everingham et al. [10]) dataset at 40 fps. It incorporates the following design decisions,

1. YOLOv2 adds batch normalization on all the convolutional layers which results in a 2% increase in mAP. Since batch normalization also regularizes the model, the dropout layers are removed from the model.

2. The original YOLO (Redmon et al. [47]) implementation first trains a classifier network using a
Chapter 2. State of the Art

resolution of $224 \times 224$ and then increases the resolution to $448 \times 448$ for object detection. Redmon and Farhadi [45] realized that during training the network has to adjust to the increased resolution and simultaneously learn to detect objects, which can be tricky. Hence they first fine-tune the classifier with input size of $448 \times 448$ for 10 epochs on ImageNet (Deng et al. [9]) dataset. The resulting network is then used for object detection training. This results in a 4% mAP increase.

3. YOLOv2 removed the fully connected layers used in the original YOLO (Redmon et al. [47]) for making predictions.

4. YOLOv2 incorporates prior anchor boxes into the framework. The authors propose to perform clustering on the training dataset using k-means to determine the prior anchor box dimensions. The distance metric used is given by the following formula,

$$d(box, centroid) = 1 - IOU(box, centroid)$$

5. The bounding boxes are encoded with respect to the grid centers.

6. Skip connections are used to pass through fine-grained features from earlier layers for improving small object detection.

7. YOLOv2 is completely convolutional and hence can operate on images of any input size. Every 10 batches, the network randomly chooses an input dimension from the range of 320 to 608 in increments of 32, to make it more robust to input size variations.

8. The authors also introduces a new classification backbone network called as Darknet-19 which achieves good accuracy while requiring less number of operations.

YOLOv3 is an incremental update to YOLOv2 proposed by Redmon and Farhadi [46]. The main updates are as follows,

1. To adapt to more complex datasets that contain overlapping labels, like Open Images Dataset (Krasin et al. [24]), Redmon and Farhadi [46] used independent logistic classifiers instead of softmax classifiers in the YOLOv3 network.

2. It introduces a new backbone feature extractor called Darknet-53 which draws inspiration from ResNet (He et al. [17]).

3. In YOLOv3, multi-scale features are used to predict bounding boxes at 3 different scales.

From the accuracy point of view, YOLOv3 lags behind RetinaNet (Lin et al. [31]) by a considerable margin on the MS COCO (Lin et al. [29]) dataset (new mAP metric). However, it is much faster than RetinaNet (Lin et al. [31]).
2.2. Object Detection

Through the Deconvolutional Single Shot Detector (DSSD) network, Fu et al. [11] try to address the prime drawback of SSD (Liu et al. [33]) i.e., poor performance on small scale objects. The architectural comparison between original SSD and DSSD is as shown in Figure 2.19. They first replace the VGG (Simonyan and Zisserman [54]) feature extractor used by original SSD with ResNet-101 (He et al. [17]). To introduce additional context information, Fu et al. [11] augment the SSD with deconvolutional layers, an idea borrowed from the hourglass encoder-decoder style architectures used in semantic segmentation (Noh et al. [41]) and human pose estimation (Newell et al. [40]). The deconvolutional module is as shown in Figure 2.18. It up-samples the information from deeper layers and merges the information from lower layers using element wise product. Fu et al. [11] found that replacing the backbone network, actually degrades the performance on PASCAL VOC (Everingham et al. [10]) dataset however, the incorporation of a new prediction module provides significant improvements. The Figure 2.20 illustrates the different prediction modules proposed by DSSD. DSSD adds a residual block to each prediction layer as shown in Figure 2.20. It outperformed the then
Chapter 2. State of the Art

state-of-the-art R-FCN (Dai2016) on PASCAL VOC 2007/2012 (Everingham et al. [10]) test sets and MS COCO (Lin et al. [29]) datasets.

Figure 2.20: Prediction modules (a) is the one used by Liu et al. [33], modules (b) to (d) are proposed by Fu et al. [11].

Zhao et al. [69] claims that the FPN backbone networks used by the modern-day object detectors were designed for the task of image classification and hence limit the accuracy of the object detectors. Hence they propose an alternate way to construct Feature Pyramid Network for the task of multi-scale object detection called Multi-Level Feature Pyramid Network (MLFPN). The architectural comparison of the different FPN networks is as shown in Figure 2.21. Incorporating the newly introduced MLFPN network in the SSD framework, Zhao et al. [69] introduced a single-stage detector called M2Det.

RefineDet was also introduced with a view of reducing the accuracy gap between the two-stage and single-stage object detectors by Zhang et al. [67]. Similar to Lin et al. [31], Zhang et al. [67] are also of the opinion that the class imbalance is the reason for the poor performance of the single-stage object detectors. They try to combine the advantages of the two-stage detectors and single-stage detectors into a single network. The architecture is as shown in Figure 2.22.

It mainly consists of two interconnected modules,

1. The Anchor Refinement Module (ARM) which intends to address the class imbalance problem by removing negative anchors. In addition to this, it refines the positive anchors to provide a good starting point for further refinement.

Figure 2.21: Comparison of different FPN networks. (d) is the MLFPN introduced by Zhao et al. [69].
2. The Object Detection Module (ODM) takes the refined anchors from the ARM and produces the final bounding box regression offsets and multi-class classification.

The transfer of features from the ARM to the ODM module is controlled by Transfer Control Block (TCB). TCBs are connected to only those layers in ARM which are associated with anchors. It also combines the high-level context information and the low-level semantic information from the different layers in the ARM module so that the ODM operates on high-quality feature maps to make the final bounding box refinement and classification. It uses VGG-16 (Simonyan and Zisserman [54]) and ResNet-101 (He et al. [17]) pre-trained on ILSVRC (Deng et al. [9]) Classification-Localization dataset for its experiments.

Single-stage object detectors have seen a paradigm shift in recent years. The object detection problem is reformulated as a key-point estimation problem (widely used in human pose estimation), to eliminate the need for prior anchor boxes. Such techniques have shown impressive performance on the challenging COCO datasets (Lin et al. [29]). By eliminating the need for anchor boxes, a significant number of hyper-parameters like anchor aspect ratios, the number of anchor boxes, and anchor scales are avoided. Key-point estimation using CNNs usually involves a fully convolutional encoder-decoder network which predicts a multi-channel heat map (e.g., for human pose estimation one channel of the heat-map might be for the head, another channel can be for the left shoulder and so on.).

CornerNet is a single-stage anchorless object detection technique introduced by Law and Deng [26]. Each object bounding box is encoded as a pair of key-points representing the top-left corner and the bottom-right corner. Law and Deng [26] use a single CNN to predict two heat-maps, one for the top-left corner key-points and another for the bottom-right corner key-points for all the objects present in the image. In addition to the heat-maps, the network also predicts an embedding vector for each corner. This embedding vector is used to find which top-left corner key-point and which bottom-right corner key-point belong to the same object.
The network is trained so that it predicts similar embeddings for the top-left and bottom-right key-points belonging to the same object. The network also predicts offset values to refine the bounding boxes extracted from the heat-maps. The network contains two stacked hourglass (Newell et al. [40]) modules as a backbone feature extractor. This is followed by two prediction modules, i.e., one for the top-left corner and another for the bottom-right corner. The overview of the CornerNet architecture is as shown in Figure 2.23.

Zhou et al. [71] follow the same framework as CornerNet (Law and Deng [26]) to introduce a new key-point based anchor-less single-stage object detection technique called as ExtremeNet. However, all the outputs of ExtremeNet are heat-maps. For each image, the network predicts a total of five C-channel heat-maps (i.e., one C-channel heat map for top, bottom, right, left, and center of the objects where C is the number of classes). It also predicts four class agnostic offset maps to refine the bounding boxes. With this arrangement, the need for matching of corners using associative embedding is avoided. The network architecture and loss is similar to that of CornerNet (Law and Deng [26]). Figure 2.24 represents the architecture of the ExtremeNet.

Zhou et al. [70] claims that anchor-based techniques are highly inefficient and wasteful because they densely sample potential object positions and then classify each of them. They also point out that most of the object detection techniques are not end-to-end learnable as they usually employ some form of post-processing like Non Maximum Suppression (NMS), which is not differentiable. To overcome these drawbacks, the authors propose a new key-point based anchor-less object detection technique that models each object instance by a single point (i.e., the center of the bounding box). The network uses a key-point estimation network to generate C channel heat-maps for the centers of the bounding boxes (where C is...
2.3. Instance Segmentation

Deep-learning based instance segmentation techniques typically formulate instance segmentation as a binary multi-label pixel-wise classification task. In recent years instance segmentation has been envisioned as a fine-grained object detection task. This section deals with the prior problem formulation. The latter is discussed in the Related Work section.

Pinheiro et al. [42] propose DeepMask network for fine-grained object proposal prediction. For each input patch, the DeepMask network, predicts a segmentation mask and a score which determines the likelihood of the input patch to contain a relevant object. The network is as shown in Figure 2.25.

![DeepMask architecture](image)

**Figure 2.25:** Top: DeepMask architecture proposed by Pinheiro et al. [42]. Bottom: Positive and Negative image patches and their associated masks.

Each sample of the training data contains 3 components, RGB input image patch, a binary mask corresponding to the input patch, and a label that determines if the patch contains an object or not. This label value is set to 1 if the following criteria are met,

- The input patch contains an object which is roughly centered in the patch.
- The object is fully contained in the input patch.

If any of the above-mentioned criteria are not met, the label value is -1. Only masks with label value equal to 1 are used for training. These are represented by the green bordered image patches and masks in Figure 2.25. The network consists of a VGG-A (Simonyan and Zisserman [54]) architecture pre-trained
on ImageNet (Deng et al. [9]) for the classification task. All the fully connected layers and the last max-pooling layer from the original architecture are removed. The resulting network produces a feature map down-sampled by 16. The segmentation branch contains a $1 \times 1$ convolution with ReLU. The output is then passed through two linear layers without any non-linearity. This arrangement reduces the number of parameters and at the same time enables the classification to be made by considering the entire input feature map. To reduce the model capacity further, the segmentation branch produces an output segmentation mask of resolution smaller than the input resolution and then up-samples it to the original input resolution.

Pinheiro et al. [43] introduced SharpMask network as an attempt to improve the run-time and accuracy of the DeepMask (Pinheiro et al. [42]) network. The network architecture is based on DeepMask with an additional refinement module to efficiently merge the spatially rich information in the feature maps from the lower layers with the semantically rich information in the feature maps from the higher layers. The comparison of the different variants of instance segmentation networks are as shown in the Figure 2.26. The architecture (a) represents the DeepMask (Pinheiro et al. [42]) type network (only mask head shown for convenience). Here the predictions are made using only the semantically rich information from the deeper layers of the network which have lower resolutions and hence the resulting masks are coarser. The architecture (b) makes independent mask predictions along the network and the final mask prediction is obtained by averaging the different mask predictions. The resolutions of the mask might be different so they might be up-sampled before averaging. The architecture (c) is the SharpMask (Pinheiro et al. [43]) network, which uses the same feed-forward-network from the DeepMask and augments it with a top-down refinement pathway. The refinement pathway contains a number of stacked refinement modules (shown in (d)). Each refinement module first performs a $3 \times 3$ convolution with ReLU on the feature maps.
obtained via skip connections from the feed-forward network. The $3 \times 3$ convolution reduces the channel dimensionality of the feature maps to reduce the computational resource requirements. The resulting feature maps are concatenated with the feature maps produced by the previous refinement module. The combined features are then processed by a $3 \times 3$ convolution with ReLU and then upsampled by 2 to generate the output from the refinement module. Based on the observations made by Pinheiro et al. [43], the DeepMask spends about 40% time in feature extraction, 40% time in mask prediction, and 20% time in score prediction. To reduce the network latency, the Pinheiro et al. [43] propose the following improvements,

- **Replace VGG-A (Simonyan and Zisserman [54]) backbone with ResNet-50 (He et al. [17]).** The ResNet-50 achieves the accuracy of VGG-D (deeper VGG model with twice the run-time as VGG-A) with the run-time of VGG-A.
- **Restructuring the prediction head of the network.** The network head architectures which generate the final predictions are as shown in Figure 2.27.

The SharpMask network outperformed DeepMask (Pinheiro et al. [42]) on the object proposal generation task whilst being 50% faster.

He et al. [18] extended the Faster R-CNN (Ren et al. [48]) network by adding a segmentation mask predicting branch in parallel with the existing classification and bounding box regression branches of the Faster R-CNN. The authors claim that the mask predicting branch is a small FCN network and hence adds only a small computational overhead. The ROIPool operation used in Faster R-CNN (Ren et al. [48]) is not designed to maintain pixel-to-pixel alignment between the network input and output.
output. For predicting high-quality instance masks, this pixel-to-pixel alignment is of utmost importance. Hence Mask R-CNN replaces the ROI Pool by ROIAlign layer. This layer is critical to the high-mask accuracy achieved by Mask R-CNN. The multi-task loss used in Faster R-CNN now includes a new component for the mask prediction i.e., $L_{mask}$. The combined multi-task loss is given as $L = L_{cls} + L_{box} + L_{mask}$. The mask branch has a $K \times m \times m$ output. Where $K$ is the number of classes and $m$ is the resolution of the generated mask. To this output, a pixel-wise sigmoid activation is applied. The mask loss is now defined as the average binary cross-entropy between the ground-truth mask and the predicted masks. The architecture is as shown in Figure 2.28.

Chen et al. [6] claim that most of the instance segmentation based techniques rely on object detection to generate ROI crops and then process these crops to generate instance masks. They also claim that dense sliding-window-based instance segmentation (analogous to dense sliding-window based object detection proposed by SSD (Liu et al. [33]) or RetinaNet (Lin2017)) has not been adequately investigated. Bounding boxes used by the dense sliding-window-based single-stage object detectors have a fixed low-dimensional representation. However, according to the authors instance segmentation requires richer high dimensional structured representation. Consider a sliding window of size $V \times U$ sliding on a feature map of size $W \times H$. In DeepMask (Pinheiro et al. [42]), all the masks in all the sliding windows are represented using a tensor of shape $(C, H, W)$ where $C$ has $V \cdot U$ parameters/pixels. Chen et al. [6] argue that a high dimensional representation might be beneficial for instance segmentation. Hence they propose a 4D tensor representation with shape $(V, U, H, W)$. The sub-tensor $(V, U)$ in the 4D representation, represents a mask as a spatial entity. The single-stage object detectors like RetinaNet (Lin2017) use multi-scale detections. Bounding boxes are always represented by 4 scalars irrespective of the scale. However, the situation is different for masks. The pixel size needs to be scaled based on the object size. With this idea at the core, the authors built a Feature pyramid that reduces the input resolution from $(H, W)$ and at the same time increases the mask resolution $(V, U)$. This is called a tensor bipyramid. The TensorMask network consists of a new mask prediction branch inspired by the above idea. It is attached to a convolutional FPN backbone. The outputs from each level are used as inputs for each of the prediction heads i.e., mask, class, and box. (Similar to Mask R-CNN (He et al. [18])) The weights of the prediction heads are task-specific and are shared across different levels.

Liu et al. [32], motivated by effectively extracting and aggregating multi-scale features, introduced a new instance aware segmentation network called as PANet. The architecture is as shown in Figure 2.29. They augments an FPN based backbone network with a bottom-up pathway to fortify the hierarchy of features with context information. The feature map from a particular level of the FPN is down-sampled by 2 using a $3 \times 3$ convolution of stride 2. The down-scaled feature map is merged with lower resolution feature map from the FPN using lateral connections and element-wise addition. The fused feature map is then passed through a $3 \times 3$ convolution to get the output feature map. Liu et al. [32] argue that high-level features are generated using larger receptive fields and hence the smaller proposals should also have access to them. On the same lines, the low-level features have fine details and have high localization information and hence the large proposals should also have access to them. Hence they propose that the feature maps from all the levels are pooled together using a new pooling operation called Adaptive
2.3. Instance Segmentation

Feature Pooling and these pooled features are used for all the proposals. The rest of the network is more or less similar to Mask R-CNN (He et al. [18]) except that PANet uses a fully connected layer to generate a class-agnostic background/foreground mask for each proposal. The class-specific mask output is added to the class-agnostic mask to generate the final instance mask.

As seen before, Mask R-CNN (He et al. [18]) adds a parallel mask branch to the Faster R-CNN (Ren et al. [48]) object detection network. YOLACT introduced by Bolya et al. [2] intends to emulate that by adding a mask branch to a single-stage object detection network without an explicit localization step (like the ROIAlign used in Mask R-CNN). This enables YOLACT to achieve real-time performance on the MS COCO (Lin et al. [29]) dataset. To achieve this, the authors break the instance segmentation task into two parallel sub-tasks which can be performed by two parallel branches.

1. The first branch is an FCN network which generates a set of prototype mask which have the same size as the input image. These masks are independent of any instance of any class.

2. The second branch in addition to the classification and bounding box regression heads also contains
an extra head to predict mask coefficients for each anchor. These masks coefficients are used as weights while combining the different prototype masks.

Finally, for each bounding box prediction that survives the Non Maximum Suppression (NMS), an instance mask is generated by linearly combining the prototype masks using the mask coefficients. The architecture is as shown in Figure 2.30.

2.4 Related Work

This work is not the first to consider the instance segmentation task as a regression problem, ESE-Seg by Xu et al. [64] tries to encode the shape of the mask by,

- First finding the inner center, which is a point on the mask which is farthest from the contour of the mask. This point becomes the center of a polar coordinate system.

- The contour of the mask is now sampled to record the distance from the center, at the rate of \( \tau = \pi/180 \), which corresponds to \( N = 360 \) samples (where, \( N = 2\pi/\tau \)).

- The size of the vector containing the center and the sampled distances is too large for direct regression so ESE-Seg first uses Chebyshev polynomials to find a near-optimal but a smaller approximation of the vector. The parameters in this vector, in addition to the bounding box parameters, are jointly predicted by the ESE-Seg network.

Xu et al. [64] use the YOLOv3 (Redmon and Farhadi [46]) as the based detector. An additional detection head is used to predict the shape vector in addition to the bounding box and the class.
Elimination of the mask predicting layer and the use of a single-stage detector enables ESE-Seg to be around 7 times faster than Mask R-CNN (He et al. [18]). The architecture of the network is as shown in Figure 2.32.

**Limitations:**

Figure 2.31 illustrates the contour predictions of ESE-Seg (Xu et al. [64]) network. Qualitatively it can be seen that the predictions leave out a significant number of true positive pixels. This can have catastrophic consequences especially in use-cases where critical decisions are taken based on the predictions of the perception sub-system like Autonomous driving and Collaborative robots. The following factors limit the performance of the ESE-Seg as compared to other instance segmentation techniques.

- ESE-Seg first finds the inner center and then samples the contour using a polar coordinate system with the origin at this inner center. This process limits the network’s capability to effectively encode concave shapes.

- The network uses Chebyshev polynomials to get a near-optimal approximation of the actual contour points. The number of coefficients used to represent the shape signature is then limited. This effectively increases the reconstruction error which in-turn affects the quality of the instance masks.

- The CNN prediction is rarely equal to the exact ground-truth. Hence the added noise by CNN regression also deteriorates the quality of the instance masks.

Prior to Xu et al. [64], Jetley et al. [21] also proposed another technique that considered instance segmentation as a regression problem. The network makes use of the YOLO object detector introduced by Redmon et al. [47] as the base detector. This base detector is augmented with an additional head for predicting the object shape encoding vector. YOLO divides the input image into a grid of $S \times S$. Each grid cell predicts $B$ boxes. For each box/instance, the network predicts 1 confidence score, 4 bounding box parameters, and $k$ dimensional object shape encoding vector. Hence for each box, the network predicts $1 + 4 + k$ predictions. For each grid cell, the network makes $(1 + 4 + k) \times B + |C|$ predictions. Hence the output volume of the network is $S \times S \times ((1 + 4 + k) \times B + |C|)$, where $|C|$ represents the number of classes in the class probability distribution.

The network architecture is as shown in Figure 2.33. The shape encoding vectors used to train the augmented detector vector are themselves learned in an unsupervised manner using a convolutional denoising auto-encoder. Once trained the decoder is capable of regenerating the binary segmentation masks from the object shape encoding, predicted by the augmented YOLO object detector. In the experiments, the shape embedding size of 20 and 50 are used. The network is capable of generating instance masks at 35 fps on NVIDIA Titan X GPU.

**Limitations:**

- The technique proposed by Jetley et al. [21] first requires training of the auto-encoder on the dataset using the binary instance masks. The augmented YOLO detector is now trained using the shape embedding vector generated by the Encoder part of the auto-encoder as the ground-truth annotation.
The shape embedding learning proposed by Jetley et al. [21] might not be scalable when a dataset with a large variety of shapes and classes is used.

Zhou et al. [71] introduced ExtremeNet which is a key-point estimation based object detection network (discussed earlier). The authors propose a way to post-process the network predictions to construct an octagonal instance mask approximation. Though this resembles the idea proposed in our work, where an instance mask is approximated by an irregular octagonal representation, there are certain nuances.

1. The octagonal mask construction proposed by Zhou et al. [71] is strictly done as a post-processing step, using the extreme points of the object instance predicted by the network. In contrast, in our work, all the parameters of the irregular octagon are explicitly predicted by the network.

2. Zhou et al. [71] uses a heuristic value to construct the octagon. Specifically, for each of the 4 extreme points, a straight line is extended in both directions on its edge to $\frac{1}{4}$ the edge length. The line segment is truncated if it meets the corner of the bounding box. The endpoints are now connected to form an octagon. In our work, all the parameters needed to completely describe an octagonal approximation of an instance mask are predicted by the network.

**Limitations:**

The heuristic value ($1/4$) employed by Zhou et al. [71] to construct the octagonal mask might not ensure that the mask will enclose the object instance completely. For instance consider the predictions made by ExtremeNet in Figure 2.34.

In our work, the supervision during network training incorporating all the eight octagonal mask parameters ensures a better fitting octagonal approximation of the instance mask. Hence the likelihood of true-positive pixels lying outside the octagonal mask is less.
2.5 Summary

Following are the key take away points from this chapter,

- Most of the research in semantic segmentation is trying to address a common problem that the features extracted by backbone feature extractors (which are usually trained on ImageNet (Deng et al. [9]) for image classification task) lacked the global context. In other words feature maps from deeper layers of the networks had high semantic information where as the feature maps from the shallower layers had high location information. Most of the research was oriented towards exploring ways to fuse the location rich information from lower layers with the high semantic information from the deeper layers. The fused feature maps are then used for dense pixel classification.

- Object detection has seen extensive research. Early object detection techniques first generate multiple equal sized crops of the input image and then classify each of these crops by passing them through an image classification network. The fixed size restriction was mainly due to the fully connected layers used in early image classification networks. One of the most important contribution was to replace the fully-connected layers by 1x1 convolutions. This enabled the network to accept variable sized inputs. To improve the inference speed and accuracy, an alternate formulation was proposed that treated object detection as a combination of classification and bounding box regression. Early work under this paradigm, approached this as a two stage process. The first stage generated class agnostic region proposals and the second stage classified these region proposals and refined them. To improve the inference speed even further, later works replaced the region proposal stage with a dense grid of pre-defined bounding boxes called anchors. The networks then learn to classify these anchors and refine them. Further research was mainly dedicated towards improving the quality of feature maps which would be used to make the final predictions. Noteworthy contributions are, using multi-scale features using FPN Lin et al. [30] and its variants, using deconvolution to upsample the feature maps from deeper layers and fusing them with the feature maps from shallower layers. Object detection landscape saw a major landscape change with the introduction of new anchors-less networks. These networks usually are made of an encoder-decoder style architecture which predicts a multi-channel heat-map. The number of channels depends on how the bounding box is parameterized.

- Instance segmentation is traditionally formulated as a binary multi-label pixel wise classification task. Early attempts at instance segmentation were motivated towards generating fine-grained object proposals. Further work was again mainly aimed towards improving quality of feature maps which will be used for making predictions. Newer instance segmentation techniques extended object detection networks by adding a parallel mask generation branch. Recent work have formulated instance segmentation as regression problem. This can serve as an ideal solution in applications which require quick inference at the permissible expense of the accuracy of the generated instance masks.
This chapter elaborates on the major design decisions taken while constructing the experiments for this work. It also provides the rationale behind each of those decisions. This chapter is structured as follows, the first section deals with the shortlisting of a suitable dataset for the experiments, the second section elaborates on the process of shortlisting a suitable object detector network, which will serve as the base for the proof-of-concept (POC), the third section describes the performance metrics that will be used in this work, the fourth section provides details on the various pre-processing operations performed on the data and the final section deals with the experimental design.

Note:

- For the rest of this chapter,

1. SqueezeDet or vanilla SqueezeDet refers to the original bounding box predicting object detector network proposed by Wu et al. [62]. For the rest of this work, all bounding box predicting networks are vanilla SqueezeDet networks.

2. SqueezeDetOcta refers to the modified SqueezeDet network proposed by this work, which is capable of predicting octagonal approximations of the instance masks. Refer the section 4.4 for more details regarding the architectural modifications. For the rest of this work, all octagonal mask predicting networks are SqueezeDetOcta networks.

- All the design decisions (DD) are tagged according to the following convention,

\[ \text{StageTag}_{DD} \_N \]

where,

- \text{StageTag} represents the section of this chapter in which the design decision is taken,
- \_N represents the design decision count for the chosen StageTag.

For instance, the StageTag for the section Shortlisting of the dataset is Dataset and hence the first design decision taken in this section will have the tag of Dataset\_DD\_1.
3.1 Shortlisting of the dataset

Note: StageTag for the design decisions taken in this section is Dataset

The main motivation behind this work is to demonstrate that, given the same amount of fine instance annotated data, a finer object segmentation model (compared to a coarser bounding box segmentation) can be obtained, which performs not worse than the coarser bounding box segmentation model. This places three requirements on the datasets, for them to be considered suitable for this work,

1. The dataset should provide fine-grained instance mask annotations. These can then be used to derive both bounding boxes (for coarser bounding box segmentation) and irregular octagons (for finer segmentation).

2. The dataset should contain objects which can reasonably be approximated by their convex hull.

3. Dataset size determines the time needed to train a segmentation network. Using a very large scale dataset effectively decreases the number of experiments that can be conducted as a part of this work. Using a very small dataset might not reliably prove that the proposed concept works. Hence a medium-sized (< 10,000 samples) dataset is desirable for this work.

It was observed that the autonomous driving datasets contain objects (for instance, cars and other vehicles) which are roughly convex in shape. The objects which are not convex (for instance, a pedestrian or a cyclist), can be reasonably approximated by convex polygons as seen in Figure 3.1.

Figure 3.1: Ground-truth extraction examples. Left: Cityscape (Cordts et al. [7]) fine pixel-wise annotation, Right top row: Bounding box extraction for “CAR”, Right middle row: Octagonal mask extraction for “CAR”, Right bottom row left: Bounding box extraction for “PERSON” and Right bottom row right: Octagonal mask extraction for “PERSON”. Significant amount of false positive pixels are avoided by using octagon as compared to bounding box.
These datasets are collected in the wild which makes them quite challenging. The objects have diverse scales. There are objects larger than the receptive field of most object detection networks, this tests the capability of the network to predict the size of an object just by seeing a small patch of the object. There are objects which are a lot smaller than the receptive field of the network and this tests the network’s capability to predict the size of the object by considering only meaningful pixels and discarding a large number of unnecessary background pixels.

These arguments back the first design decision for dataset selection.

**Table 3.1**: Comparison of prominent autonomous driving datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Instance annotations available?</th>
<th>Number of pixel-wise annotated samples</th>
<th>Remarks</th>
<th>Number of instance annotated classes</th>
<th>Suitable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>KITTI Vision Benchmark Suite (Geiger et al. [12])</td>
<td>Yes</td>
<td>400 (200 training and 200 test)</td>
<td>The pixel-wise annotations are not official and are provided by third party groups.</td>
<td>10</td>
<td>No</td>
</tr>
<tr>
<td>CamVid (Brostow et al. [3])</td>
<td>No</td>
<td>701</td>
<td>701 pixel-wise annotated images are available but without instance specific information.</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>Daimler Urban Segmentation (Scharwächter et al. [51])</td>
<td>No</td>
<td>500</td>
<td>It provides 5000 images out of which only 500 have pixel-wise annotations.</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>Waymo open dataset (Sun et al. [58])</td>
<td>No</td>
<td>0</td>
<td>Dataset does not provide pixel-wise annotations and hence not suitable for this work.</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>nuScenes dataset (Caesar et al. [4])</td>
<td>No</td>
<td>0</td>
<td>Dataset does not provide pixel-wise annotations and hence not suitable for this work.</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>Oxford RobotCar dataset (Maddern et al. [36])</td>
<td>No</td>
<td>0</td>
<td>Dataset does not provide pixel-wise annotations and hence not suitable for this work.</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>The Mapillary Vistas Dataset (Neuhold et al. [39])</td>
<td>Yes</td>
<td>25,000</td>
<td>Dataset collected over different parts of the world. It contains instance-specific annotations for 37 object classes.</td>
<td>37</td>
<td>Yes</td>
</tr>
<tr>
<td>Cityscape dataset (Cordts et al. [7])</td>
<td>Yes</td>
<td>5000</td>
<td>Dataset collected from 50 different cities in Germany. It provides 5000 fine and 20,000 coarse pixel-wise annotations. It contains instance-specific annotations for around 10 object classes.</td>
<td>10</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 3.1 provides a comparison of the important datasets in the autonomous driving domain based on the selection criteria defined earlier. From the table, it can be seen that only The Mapillary Vistas Dataset
(Neuhold et al. [39]) and the Cityscape dataset (Cordts et al. [7]) are suitable for this work. Based on the number of pixel-wise annotated samples in the dataset, Cityscape dataset has a reasonable number of data samples and detection classes. This observation backs the final design decision for dataset selection.

**Dataset_DD_2:**

Cityscape dataset is the chosen dataset for the entirety of this work.

Depending on the results of the POC on the Cityscape dataset (Cordts et al. [7]), its extension to the Mapillary Vistas Dataset (Neuhold et al. [39]) can be considered for future work.

### 3.2 Shortlisting of the object detection network

**Note:** StageTag for the design decisions taken in this section is Network

As seen in section 2.2, there is a wide abundance of CNN based object detection techniques, many of which can serve as a good baseline. These CNN based object detection techniques can be predominantly grouped into two categories i.e., anchor-based and anchor-less (key-point extraction based). Recently, anchor-less techniques have developed a great following. However, anchor-based techniques have a unique advantage, which makes them appealing. The anchor-based techniques enable the incorporation of prior knowledge from the data distribution into the design of the architecture. This has the following advantages,

1. It enables a way to customize the network to a particular application with minimal changes to the network architecture.
2. Training convergence is accelerated because of the bootstrapped prior knowledge.

The above-mentioned advantages justify the first design decision for the object detector selection.

**Network_DD_1:**

Restrict the object detector search to anchor-based architectures.

Recalling from section 2.2, anchor-based object detectors are divided into two-stage detectors and single-stage detectors. Two-stage object detectors, though highly accurate, have high latency. Two-stage networks like Faster R-CNN by Ren et al. [48] employ alternating training strategy for the training of RPN and detector networks which can be tricky. Single-stage object detectors have tried to bridge the accuracy gap between them and their two-stage counterparts, whilst having considerably low latency. Single-stage architectures support end-to-end training. This work intends to find a good precision-run-time trade-off for object segmentation. The above argument justifies the second design decision for the object detector selection.

**Network_DD_2:**

Restrict the object detector search to anchor-based single-stage architectures.
Chapter 3. Methodology

MS COCO (Lin et al. [29]) object detection dataset has been instrumental in accelerating the development of object detection networks over the years. The lower latency of the single-stage architectures as compared to their two-stage counterparts can be attributed majorly to two factors,

1. Reduction in the resolution of the input images. Faster R-CNN (Ren et al. [48]) a well known two-stage detector resizes the input images such that the shorter side is 600 pixels. SSD (Liu et al. [33]) which is a single-stage anchor-based detector resizes the inputs to a size of 300x300 or 512x512 in its experiments. The maximum, input image resolution used by the YOLO (Redmon et al. [47], Redmon and Farhadi [45], Redmon and Farhadi [46]) series of architectures is 608x608.

2. Elimination of Region Proposal Network (Ren et al. [48]).

The Cityscape (Cordts et al. [7]) dataset chosen for evaluation uses an input image size of 2048x1024. Most of the general-purpose single-stage object detection networks have been developed precisely with the MS COCO dataset in mind. To train the SSD or YOLO series of architectures on Cityscape (Cordts et al. [7]) dataset, one of the following adjustments is essential,

1. Input images have to be scaled to match the input resolution of the network architecture. Resizing the images to match the input sizes specified by YOLO or SSD has the following consequences,
   - Unwanted geometric distortion of the aspect ratio and its associated side-effects on the features learned by the network.
   - Cityscape (Cordts et al. [7]) dataset is collected in the wild and hence contains an abundance of objects across a variety of different scales. Hence scaling the images by a factor close to 4 (since 2048/512 = 4), effectively squashes the miniature objects in the images. This makes it hard for the network to learn to segment them.

2. The input size of the network needs to be increased. By increasing the input size of the single-stage detectors like SSD or YOLO, the real-time performance quoted by these techniques is not guaranteed. There are other multi-scale object detection techniques like RetinaNet (Lin et al. [31]) which is capable of detecting small objects by processing the input images at multiple scales using FPN backbones. However, the input resolution used by RetinaNet which comes closest to Cityscape (Cordts et al. [7]) input resolution is 800x800 and the latency of this network varies between 153ms (RetinaNet-50-800) and 198ms (RetinaNet-101-800) which is not close to real-time.

SqueezeDet by Wu et al. [62] is a miniature single-stage anchor-based object detection architecture. It is often overlooked in the research domain because it borrows a lot of features from the YOLO architecture. However, one of the stand-out features of SqueezeDet is that it achieves an impressive 57.3 fps frame-rate at an input resolution of 1248x384 whilst maintaining a small memory footprint and decent performance. It was designed for KITTI (Geiger et al. [12]) object detection dataset which contains images with resolution of 1224x370. With minimal changes, the network can be adapted to work on Cityscape (Cordts et al. [7]) dataset with a marginal reduction in run-time performance. This makes it a prime candidate for the experiments performed as a part of this work and hence justifies the final design decision for the object
3.3 Evaluation metrics

The following section extensively elaborates on the performance metrics which are used throughout this work.

### 3.3.1 Mean Average Precision (mAP)

Mean average precision (mAP) is the most widely used metric for assessing the performance of object segmentation algorithms. Variants of mAP are employed by a vast number of object segmentation challenges like PASCAL VOC (Everingham et al. [10]), MS COCO (Lin et al. [29]), etc. This sub-section takes a bottom up approach by first building up on the necessary low level concepts which are essential to effectively define mAP. The most fundamental concepts towards defining mAP are Precision and Recall.

- **Precision**: Precision is the metric used to measure the capability of an algorithm to identify or classify the relevant objects. Informally, it determines what percentage of all the predictions made by the algorithm are correct. Mathematically it is given by,

  \[
  \text{Precision} = \frac{TP}{TP + FP} \quad (3.1)
  \]

- **Recall**: Recall is the metric used to measure the capability of an algorithm to identify or classify all the relevant objects. Informally, it determines what percentage of all the known relevant cases are correctly identified by the algorithm. Mathematically it is given by,

  \[
  \text{Recall} = \frac{TP}{TP + FN} \quad (3.2)
  \]

where,

![Figure 3.2: Precision and Recall illustration by Walber [61]](image_url)
Chapter 3. Methodology

\[ TP = \text{True Positive} \]
\[ FP = \text{False Positive} \]
\[ FN = \text{False Negative} \]

Generally, these metrics are used for classification or information retrieval systems where the definitions of True Positive, False Positive and False Negative are fairly straightforward. The visual illustrations of these definitions are provided by the well known Figure 3.2.

For object segmentation, these definitions are not that obvious. **How does one determine if a bounding box/mask prediction made by an algorithm is a True Positive, False Positive, or False Negative?** There is a need to introduce an additional concept that will aid in defining what is a True Positive, False Positive, or False Negative bounding box/mask prediction. This concept is known as Intersection Over Union (IoU).

- **Intersection Over Union (IoU):** The IoU also termed as Jaccard index is a metric based on the Jaccard similarity coefficient which is commonly used to measure the similarity between two finite sets (Jaccard [20]). For segmentation algorithms, all the pixel locations lying within a bounding box/mask can be considered as belonging to a set.

If \( S \) represents the ground-truth region, then the ground-truth set can be formally represented as,

\[ A = \{ x | x \in S, S \subset \mathbb{N} \times \mathbb{N} \} \] (3.3)

If \( P \) represents the prediction region, then the prediction set can be formally represented as,

\[ B = \{ x | x \in P, P \subset \mathbb{N} \times \mathbb{N} \} \] (3.4)

\( \mathbb{N} \) the set of natural numbers. It is used because of the digital representation of images. It is assumed that \( \mathbb{N} \) also includes 0.

Then IoU between the prediction and the ground-truth is give by,

\[ \text{IoU} = \frac{|A \cap B|}{|A \cup B|} \] (3.5)

For the ease of representation, the cardinality notation can be replaced by area and the IOU can hence be defined as,

IoU varies between 0 (no overlap) and 1 (complete overlap).

Armed with this definition of IoU, it is now possible to define True Positive, False Positive, and False Negative for object segmentation.

**True Positive:** A prediction is considered as True Positive if the IoU (bounding box/mask) is greater than or equal to a particular threshold.
3.3. Evaluation metrics

Figure 3.3: IoU illustration inspired by Rosebrock [50]. The figure shows how IoU can be calculated when the ground truth type varies between \{bounding box, instance mask\} and detection type varies between \{bounding box, octagonal mask\}

**False Positive:** A prediction is considered as False Positive if the IOU (bounding box/mask) is less than a particular threshold

**False Negative:** A ground-truth (bounding box/mask) is considered as a False Negative if it is not detected.

Different object segmentation challenges use different threshold values for their metrics. The typical values vary between 30% to 95%. A smaller threshold corresponds to a lenient metric and a larger threshold corresponds to a much stricter metric. In situations where more than one prediction overlaps with a particular ground-truth, the PASCAL VOC 2012 (Everingham et al. [10]) metric states that the prediction with the largest IoU is considered as True Positive and the rest are termed as False Positive. With these definitions, it is possible to calculate the Precision and Recall values for an object segmentation algorithm.

One of the most critical concepts in the journey towards defining the mAP is the concept of the Precision vs Recall curve.

- **Precision vs Recall curve:** Precision vs Recall curves are usually class-specific. In other words, if the segmentation algorithm is designed to distinguish between n-classes of objects, then n precision vs recall curves are plotted for that object segmentation algorithm. The precision vs recall curve is plotted by calculating the precision and recall values of the accumulated True Positive and False Positive predictions (Rafael Padilla and da Silva [44]). The steps involved in the calculation of the precision vs recall curve for a particular class are as follows,

1. Accumulate all the predictions and ground-truths belonging to the class under investigation.
2. Label all the predictions as True Positive or False Positive based on a set IoU threshold.
3. In addition to each bounding box/mask prediction, an object segmentation algorithm also
predicts a confidence score. This score quantifies how confident the algorithm is about the specific prediction. Arrange all the predictions in the decreasing order of their confidence scores.

4. Starting from the most confident prediction of the segmentation algorithm, at each prediction step, accumulate the number of True Positive and False Positive predictions seen until that prediction step.

5. Use the accumulated number of True Positive and False Positive predictions at each prediction step to calculate the Precision (formula) and Recall (formula) values at that step using their respective formulas.

6. Plot the precision values vs their corresponding recall values to get the Precision vs Recall curve for the specific class of predictions.

Rafael Padilla and da Silva [44] provides an intuitive example\(^1\) that goes through this process of obtaining the precision vs recall curve manually.

The next step is to calculate Average Precision, which quantifies how well a segmentation algorithm segments a particular class of objects.

- **Average Precision (AP):** Theoretically average precision is calculated as the Area Under Curve (AUC) of the precision vs recall curve. Mathematically there are two ways in which this can be done.

  1. **11-point interpolation:** In this interpolation technique, the precision values at 11 equally spaced recall values \([0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]\) are averaged to calculate the AP (Rafael Padilla and da Silva [44]). Mathematically,

\[
AP = \frac{1}{11} \sum_{r \in \{0, 0.1, 0.2, \ldots, 1.0\}} \rho_{\text{interpolation}}(r)
\]  

\[
\rho_{\text{interpolation}}(r) = \max_{\tilde{r} \geq r} (\rho(\tilde{r}))
\]

where,

\(\rho(\tilde{r})\) is the precision value at recall value of \(\tilde{r}\).

Until 2010, the PASCAL VOC (Everingham et al. [10]) challenge, used 11-point interpolation for AP calculation. Figure 3.4a illustrates 11 point interpolation.

2. **All point interpolation:** In this interpolation technique, the precision values are interpolated at all the recall values instead of equally spaced 11 recall values (Rafael Padilla and da Silva [44]). Mathematically,

\[
AP = \sum_{n=0}^{K-1} (r_{n+1} - r_n) \rho_{\text{interpolation}}(r_{n+1})
\]

\(^1\)https://github.com/rafaelpadilla/Object-Detection-Metrics
3.3. Evaluation metrics

(a) 11 point interpolation.

(b) All point interpolation.

Figure 3.4: Illustrations by Rafael Padilla and da Silva [44]

\[ \rho_{\text{interpolation}}(r_{n+1}) = \max_{\tilde{r} : \tilde{r} \geq r_{n+1}}(\rho(\tilde{r})) \]  

(3.9)

where,

\( \rho(\tilde{r}) \) is the precision value at recall value of \( \tilde{r} \).

\( K \) is the total number of points.

From 2010, the PASCAL VOC (Everingham et al. [10]) challenge, uses this type of interpolation for AP calculation. Figure 3.4b illustrates all point interpolation.

In both cases, the interpolated precision at a recall step of \( r \) is taken to be the maximum value of all the precision values which have recall values greater than or equal to \( r \).

With all the necessary concepts covered, it is now possible to finally define the mAP.

- **mAP**: Since the precision vs recall curves are class-specific, there will be one AP value for every class the object segmentation algorithm is designed to segment. Calculating mAP is relatively straightforward and is obtained by taking the mean of all the class-specific AP values. It quantifies how well a segmentation algorithm is capable of segmenting all the classes it was designed to handle.

Mathematically,

\[ mAP = \frac{1}{N} \sum_{i=0}^{N-1} AP(c_i) \]  

(3.10)

where,

\( N \) is the number of classes.

\( AP(c_i) \) is the average precision value for a specific class \( c_i \).
Note:

The mAP value highly depends on the IoU threshold used to make the distinction between True Positive and False Positive. Hence corresponding to different IoU threshold values, an object segmentation algorithm can have different mAP values.

For all the experiments performed as a part of this work the metric used is an extension of the mAP metric defined above. The specifications are as follows,

1. Similar to PASCAL VOC 2012(Everingham et al. [10]) metrics, All point interpolation is used to calculate the class-specific AP values.

2. The mAP values are calculated using IoU thresholds in the range of 0.30 to 0.95 in increments of 0.05. The actual metric will be the average of all those mAP values. This bears similarity with MS COCO (Lin et al. [29]) and Cityscape (Cordts et al. [7]) evaluation metric with one exception that these competitions use the IoU values in the range of 0.50 to 0.95 in increments of 0.05. Using 0.3 as the start of the range helps to increase the dynamic range of the final average mAP value and this provides a better ground for comparing the bounding box and octagonal mask predicting networks.

3.3.2 mAP calculation scenarios

As a part of this work, it is necessary to compare the performances of a fine octagonal mask regression network to that of a coarse bounding box regression network. Towards this end, the following mAP calculation scenarios are formulated. Depending on the type of ground-truth (instance mask or bounding box) and the type of prediction (octagonal mask or bounding box), different calculation scenarios will be used.

1. mAP - (Box)x(Box):

In this scenario, the ground-truth and prediction segmentation types are both bounding boxes. Cityscape (Cordts et al. [7]) dataset provides fine instance segmentation masks. Obtaining the bounding boxes for these masks is relatively straightforward. Vanilla SqueezeDet (Wu et al. [62]) is designed to predict bounding box segmentations, hence no post-processing of the predictions is necessary. The SqueezeDetOcta is designed to predict octagonal mask segmentations. There is a need to post-process these predictions to get bounding boxes from the octagonal masks. Figure 3.5 illustrates this scenario.
3.3. Evaluation metrics

This mAP calculation scenario provides a common ground for the unbiased comparison of vanilla SqueezeDet (Wu et al. [62]) and SqueezeDetOcta. $\bigoplus$ represents OR operation and $\bigotimes$ is COMPARE operation.

2. mAP - (Mask)x(Box or Octagon):

In this scenario, the ground-truth segmentation type is always instance mask.

- If the network is a bounding box predicting network, then the prediction segmentation type is a rectangular mask. Hence the mAP calculation scenario is mAP-(Mask)x(Box). Vanilla SqueezeDet (Wu et al. [62]) is designed to predict bounding box segmentations, hence the only post-processing needed is to convert the rectangular bounding boxes to rectangular masks.

- If the network is an octagonal mask predicting network then there are two cases possible,

  (a) The prediction segmentation type is an octagonal mask. Hence the mAP calculation scenario is mAP-(Mask)x(Octagon). The SqueezeDetOcta is designed to predict octagonal mask segmentations and hence there is no post-processing of predictions needed.

  (b) The prediction segmentation type is a bounding box. Hence the mAP calculation scenario is mAP-(Mask)x(Box). The SqueezeDetOcta is designed to predict octagonal mask segmentations and hence post-processing is needed to convert the octagonal mask to rectangular mask.

Figure 3.6 illustrates this scenario.
Chapter 3. Methodology

Figure 3.6: mAP-(Mask) x (Box or Octagon) calculation scenario.

mAP-(Mask)x(Box) provides a common metric on which vanilla SqueezeDet and SqueezeDetOcta can be compared. Comparing the mAP score of vanilla SqueezeDet calculated using the mAP-(Mask)x(Box) with the mAP score of SqueezeDetOcta calculated using the mAP-(Mask)x(Octagon) provides a way to quantify the benefits of using finer octagonal masks segmentations produced by SqueezeDetOcta over the bounding box segmentations produced by vanilla SqueezeDet (Wu et al. [62]). ⊕ represents OR operation and ⊗ is COMPARE operation.

3.3.3 Run-time

In addition to comparing segmentation performance, run-time is also another metric which is vital for the scope of this work. Figure 3.7 illustrates the various stages in the inference pipeline of the object segmentation network.

The probe points (T0 to T4) indicate the points at which the time measurements are recorded. This enables us to capture the latency of each block in the inference pipeline. This information can then be utilized to improve the overall run-time of the inference pipeline. For the scope of this work however, the latency of forward pass through the segmentation network has the most relevance. It can be reasonably stated that with only a minor increase in run-time, finer masks can be generated by SqueezeDetOcta as compared to vanilla SqueezeDet (Wu et al. [62]). The forward pass time in milliseconds (ms) will help to clarify this hypothesis.
3.4 Data pre-processing

This section elaborates on the pre-processing steps performed on the dataset before it is fed into the network for training. The following pre-processing operations are performed,

1. **Reject classes with insignificant sample count:** Figure 3.8 provides a detailed frequency of occurrence (sample count) breakdown of the different classes in both the training and the validation sets of Cityscape (Cordts et al. [7]) dataset. Here occurrence refers to an instance of that class object appearing in the images in the dataset. As a part of this pre-processing step, all the classes having a sample size less than 300 in the training set are rejected.

2. **Rejection of group examples:** The Cityscape (Cordts et al. [7]) annotations do not always account for every individual instance of an object, especially when the objects are clustered together. These clustered objects are grouped and each group is labeled together as a single entity with the suffix “group”. For example, if there is a cluster of people in an image, they will be labeled as “person\_group”. Figure 3.9 provides the occurrences of such grouped labels in the training and the validation sets of the Cityscape (Cordts et al. [7]) dataset.

Including such annotations introduces noise in the learning process. The low percentage of such examples indicate that excluding them would not significantly affect the number of data samples present in the dataset. The only downside would be during the evaluation of the model. Since the network might still identify individual instances of the objects in the clustered group of objects, they will be deemed as False Positives by the evaluation scripts, as there are no ground-truth instances in the clustered object groups. However, this is not a pressing issue, since we are interested in relative performance evaluation of the SqueezeDet (Wu et al. [62]) vs SqueezeDetOcta and this pre-processing step is invariably performed on both of them.

Figure 3.7: Inference pipeline of a typical anchor based object detection network.
Figure 3.8: Class instance distribution in the Cityscape dataset

Figure 3.9: Grouped instance distribution as a percentage of the total sample size.
3. Resizing and standardization of the images: The Cityscape dataset has an image resolution of $2048 \times 1024$. The SqueezeDet and its larger variant, SqueezeDet+ introduced by Wu et al. [62] use input sizes of $1248 \times 384$ and $1242 \times 375$ respectively. To adapt these networks to Cityscape (Cordts et al. [7]) dataset, their input sizes are changed to $1024 \times 512$. The input sizes are not increased to full resolution $2048 \times 1024$ used by the dataset, to prevent the degradation of the run-time performances of the networks. As a part of this pre-processing step, the images are down-scaled by 2 to match the input sizes of the modified networks. SqueezeDet/SqueezeDet+ (Wu et al. [62]) use SqueezeNet introduced by Iandola et al. [19] and pre-trained on ISLVRC2012 dataset as their backbones. Hence they employ RGB mean (calculated over ISLVRC2012 (Deng et al. [9]) training set) subtraction to standardize the input images. This pre-processing is also performed in all the experiments performed as a part of this work.

4. Rejection of extremely minute samples: SqueezeDet/SqueezeDet+ (Wu et al. [62]) are small yet performant object detectors operating on single input image scale. The larger input image resolution helps but they still struggle to detect small scale objects. The resizing mentioned in the previous step will squash the small objects in the dataset making them even harder to detect. Hence as a part of this pre-processing step, the objects that have both width and height values less than a particular threshold value are rejected. For all our experiments we use a threshold value of 10 pixels.

3.5 Experimental Design

This section first provides a brief overview of the placeholders which define an experimental network specimen. Each placeholder is associated with discrete set of possible values. By assigning these values to the placeholders, different candidate network specimens can be constructed. As a convention, the placeholders are represented in camelCase and the values associated with them are either numeric or represented in snake_case. The process of designing the experiments involves first identifying the objective of the experiment, this is then followed by constructing candidate specimens to achieve this objective. Care is taken to ensure an unbiased design of the experiments.

3.5.1 Experimental placeholders

The following are the placeholders that are used to define the network candidates for the different experiments.

1. Mask parameterization size:

   Mask parameterization size determines the number of parameters used to represent the instance mask annotation of an object in the image. Refer Figure 3.1 for a visual illustration of the bounding box and octagonal approximations of the instance masks. maskParams is used as an identifier for this placeholder for the rest of the discussion.

   Possible values: \{4 (bounding box regression), 8 (octagonal masks regression)\}. 

48
2. Anchor type:

Section 1.1 pointed out the inconsistency issue between the ground-truth encoding and the type of clustering used to generate the prior anchors. This work proposes clustering in a space obtained by performing the natural log transformation of the widths and heights of the ground-truth annotations. Refer section 4.4.1 for more details on the linearly extracted anchors vs the logarithmically extracted anchors. anchorType is used as an identifier for this placeholder for the rest of the discussion.

Possible values: \{linear, log\}.

3. Fine-tuning modes:

In a bid to boost the performance of the octagonal mask predicting network, the network is initialized with the trained weights of the bounding box predicting network and then fine-tuned. As a part of this fine-tuning process, either all the layer parameters are fine-tuned or only the last layer parameters are fine-tuned. No fine-tuning implies that the network is trained from scratch. fineTuneMode is used as an identifier for this placeholder for the rest of the discussion.

Possible values: \{fine\_disable (No fine-tuning), fine\_all (Fine-tune all layers), fine\_last (Fine-tune only last layer)\}.

4. Ground-truth encoding:

Section 1.1 provided insights into why boundary adhering object instances might inversely affect the regression-based object segmentation performance. Detecting and dealing with these problematic object instances might improve the performance of such models. One of the problems with the current object detection framework that limits their ability to handle these problematic object instances is the way the ground-truth bounding boxes are parameterized. Most of the object detection techniques parameterize a bounding box using the typical quartet \([c_x, c_y, w, h]\). However, a decoupled parameterization would be more suitable for handling the boundary adhering object instances. Such a decoupled parameterization uses the quartet of \([x_{min}, y_{min}, x_{max} and y_{max}]\) to represent the bounding box. The typical bounding box parameters are encoded using the center-offset encoding/decoding scheme, however, this scheme is not suitable for the decoupled parameterization. To this end, two alternate encoding/decoding schemes (i.e., anchor-offset linear encoding/decoding scheme and anchor-offset non-linear encoding/decoding scheme) are proposed in the section 4.3.2. gtEncode is used as an identifier for this placeholder for the rest of the discussion.

Possible values: \{normal (usual center-offset encoding/decoding), asymmetric\_linear (anchor-offset linear encoding/decoding), asymmetric\_log (anchor-offset non-linear encoding/decoding)\}.

3.5.2 Experiment design process

A large number of networks with different settings will be evaluated against each other in the experiments performed as a part of this work. Hence there is a need to define a naming convention to uniquely
identify each network. The naming convention is defined below.

\[
\text{PredType}_\text{AnchorType}_\text{GtEncodeType}_\text{FineTuneType}
\]

where,

\[
\begin{align*}
\text{PredType} &= \begin{cases} 
  \text{bbox} & \text{if network predicts bounding boxes} \\
  \text{octa} & \text{if network predicts octagonal masks} 
\end{cases} \\
\text{AnchorType} &= \begin{cases} 
  \text{log} & \text{if network is paired with logarithmically extracted anchors} \\
  \text{lin} & \text{if network is paired with linearly extracted anchors} 
\end{cases} \\
\text{GtEncodeType} &= \begin{cases} 
  \text{nor} & \text{if network uses center-offset encoding/decoding scheme} \\
  \text{log} & \text{if network uses anchor-offset non-linear encoding/decoding scheme} \\
  \text{lin} & \text{if network uses anchor-offset linear encoding/decoding scheme} 
\end{cases} \\
\text{FineTuneType} &= \begin{cases} 
  \text{no} & \text{if network is trained from scratch} \\
  \text{all} & \text{if network is fine-tuned and all the layers weights are updated} \\
  \text{lst} & \text{if network is fine-tuned and only the last layer weights are updated} 
\end{cases}
\]

**Experiment 1:**

**Objective:**

The objective of this experiment is to assess the quality of the linearly extracted anchors proposed by Wu et al. [62] and the logarithmically extracted anchors introduced as a part of this work.

**Network candidates:**

Two bounding box predicting networks, one with linearly extracted anchors and another with logarithmically extracted anchors are trained. The mAP metric defined in section 3.3.1 is used to compare the performances of the two networks. The two networks being compared are identical except for the type of anchors used by them. Hence the network which achieves a higher value of mAP is the one with better quality anchors. The following table describes the network candidates which will be used for this experiment.

<table>
<thead>
<tr>
<th>id</th>
<th>maskParams</th>
<th>anchorType</th>
<th>gtEncode</th>
<th>fineTuneType</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbox_lin_nor_no</td>
<td>4</td>
<td>linear</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>bbox_log_nor_no</td>
<td>4</td>
<td>log</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
</tbody>
</table>

Table 3.2: Candidate networks which will be used for the experiment 1.

**Experiment 2:**

**Objective:**

50
Chapter 3. Methodology

The objective of this experiment is to compare the performance of an octagonal mask predicting network with the performance of a bounding box predicting network.

**Network candidates:**

Both the candidate networks used in experiment 1 are also considered for this work. The type of anchors used for the octagonal mask predicting network candidate is determined based on the candidate network with the better mAP score (hence better anchors) in experiment 1. In this experiment, the candidate networks are compared by considering both the mAP and the run-time metrics defined in section 3.3. The following table describes the network candidates which will be used for this experiment.

<table>
<thead>
<tr>
<th>id</th>
<th>maskParams</th>
<th>anchorType</th>
<th>gtEncode</th>
<th>fineTuneMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbox/log/nor/no</td>
<td>4</td>
<td>log</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>bbox/lin/nor/no</td>
<td>4</td>
<td>linear</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa(log/lin)/nor/no</td>
<td>8</td>
<td>log/linear</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
</tbody>
</table>

Table 3.3: Candidate networks which will be used for the experiment 2. octa(log/lin)/nor/no represents a single octagonal mask predicting network which is paired with the best anchors that were found in experiment 1.

**Experiment 3:**

**Objective:**

The objective of this experiment is to investigate if the performance of an octagonal mask predicting network can be improved by first initializing it with the trained weights of the bounding box predicting network and then fine-tuning it.

**Network candidates:**

All the candidate networks used in this experiment are paired with the best anchor type that was found in experiment 1. As a part of this experiment, two variants of fine-tuning of the octagonal mask predicting network are considered. In the first variant, all the layers of the network are fine-tuned. In the second variant, only the last layer is fine-tuned. The fine-tuned network variants are compared with a bounding box predicting network and an octagonal mask predicting network, both of which are trained from scratch. All the networks are compared by considering both the mAP and the run-time metrics defined in section 3.3. The following table describes the network candidates which will be used for this experiment.

<table>
<thead>
<tr>
<th>id</th>
<th>maskParams</th>
<th>anchorType</th>
<th>gtEncode</th>
<th>fineTuneMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbox_{log/lin}/nor/no</td>
<td>4</td>
<td>linear/log</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_{log/lin}/nor/no</td>
<td>8</td>
<td>linear/log</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_{log/lin}/nor/all</td>
<td>8</td>
<td>linear/log</td>
<td>normal</td>
<td>fine_all</td>
</tr>
<tr>
<td>octa_{log/lin}/nor_lst</td>
<td>8</td>
<td>linear/log</td>
<td>normal</td>
<td>fine_last</td>
</tr>
</tbody>
</table>

Table 3.4: Candidate networks which will be used for the experiment 3.
3.5. Experimental Design

Experiment 4:

**Objective:**
The primary objective of this experiment is to compare the performance of object segmentation networks (bounding box and octagonal mask predicting networks) trained from scratch with/without boundary adhesion considerations. The secondary objective is to compare the performance of the octagonal mask predicting network vs that of the bounding box predicting network when both are trained while being conscious of the boundary adhering object instances mentioned in section 4.2.

**Network candidates:**
For this experiment, all the candidate networks used in experiment 2 are chosen as the candidates trained from scratch without any boundary adhesion considerations. Two bounding box predicting networks and two octagonal mask predicting networks are trained from scratch while being conscious of the problematic boundary adhering object instances mentioned in the section 4.2. Out of these, one bounding box predicting network and one octagonal mask predicting network use the anchor-offset linear encoding/decoding scheme and the other bounding box predicting network and octagonal mask predicting network use the anchor-offset non-linear encoding/decoding scheme mentioned in section 4.3.2. Both the mAP and the run-time metrics defined in section 3.3 are used for the network comparisons in this experiment. The following table describes the network candidates which will be used for this experiment.

<table>
<thead>
<tr>
<th>id</th>
<th>maskParams</th>
<th>anchorType</th>
<th>gtEncode</th>
<th>fineTuneMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbox_lin_nor_no</td>
<td>4</td>
<td>linear</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>bbox_log_nor_no</td>
<td>4</td>
<td>log</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>bbox_lin_lin_no</td>
<td>4</td>
<td>linear</td>
<td>asymmetric_linear</td>
<td>fine_disable</td>
</tr>
<tr>
<td>bbox_log_log_no</td>
<td>4</td>
<td>log</td>
<td>asymmetric_log</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_{log/lin}_nor_no</td>
<td>8</td>
<td>log/linear</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_lin_lin_no</td>
<td>8</td>
<td>linear</td>
<td>asymmetric_linear</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_log_log_no</td>
<td>8</td>
<td>log</td>
<td>asymmetric_log</td>
<td>fine_disable</td>
</tr>
</tbody>
</table>

Table 3.5: Candidate networks which will be used for the experiment 4. octa_{log/lin}_nor_no represents a single octagonal mask predicting network which is paired with the best anchors that were found in experiment 1.

Experiment 5:

**Objective:**
The objective of this experiment is to investigate if the performance of an octagonal mask predicting network can be improved by first initializing it with the trained weights of the bounding box predicting network and then fine-tuning it. This experiment is similar to experiment 3 except for the fact that in this experiment, the candidate networks are trained considering the boundary adhering object instances.

**Network candidates:**
Similar to experiment 3, two variants of fine-tuning are considered in this experiment. In the first variant, all the layers of the network are fine-tuned. In the second variant, only the last layer is fine-tuned.
Chapter 3. Methodology

The different fine-tuned variants are compared with the octagonal mask predicting networks and the bounding box predicting networks used in experiment 4 (which were trained from scratch considering the boundary adhering object instances). All the networks are compared by considering both the mAP and the run-time metrics defined in section 3.3. Table 3.6 describes the network candidates which will be used for this experiment.

<table>
<thead>
<tr>
<th>id</th>
<th>maskParams</th>
<th>anchorType</th>
<th>gtEncode</th>
<th>fineTuneMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbox_lin_lin_no</td>
<td>4</td>
<td>linear</td>
<td>asymmetric_linear</td>
<td>fine_disable</td>
</tr>
<tr>
<td>bbox_log_log_no</td>
<td>4</td>
<td>log</td>
<td>asymmetric_log</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_lin_lin_no</td>
<td>8</td>
<td>linear</td>
<td>asymmetric_linear</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_log_log_no</td>
<td>8</td>
<td>log</td>
<td>asymmetric_log</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_lin_lin_all</td>
<td>8</td>
<td>linear</td>
<td>asymmetric_linear</td>
<td>fine_all</td>
</tr>
<tr>
<td>octa_log_log_all</td>
<td>8</td>
<td>log</td>
<td>asymmetric_log</td>
<td>fine_all</td>
</tr>
<tr>
<td>octa_lin_lin_lst</td>
<td>8</td>
<td>linear</td>
<td>asymmetric_linear</td>
<td>fine_last</td>
</tr>
<tr>
<td>octa_log_log_lst</td>
<td>8</td>
<td>log</td>
<td>asymmetric_log</td>
<td>fine_last</td>
</tr>
</tbody>
</table>

Table 3.6: Candidate networks which will be used for the experiment 5.

Note:

- All the experiments are repeated for three trials and the mean and standard deviation values of the metrics defined in section 3.3 are reported in chapter 5.

3.5.3 Training and evaluation strategies

The following training schedule is used for training all the networks developed as a part of this work.

- Cityscape dataset (Cordts et al. [7]) provides explicit training and validation sets. These will be used for training and validating all the networks across all the experiments.
- Stochastic Gradient Descent with momentum is used as the optimizer.
- Initial learning rate is set to 0.01.
- Learning rate decay factor of 0.5 is used.
- Decay steps size is set to 10000.
- The training is performed for 200000 steps with a batch size of 10.
- Training loss is recorded every 10 steps.
- Every 2000 steps, the loss of the network on the validation set is recorded as a means to detect network over-fitting.
3.5. Experimental Design

- All the image and ground-truth pre-processing steps mentioned in section 3.4 are performed during both training and validation.
- The following data augmentation techniques are used during training throughout all the experiments.
  1. Random cropping
  2. Random horizontal flipping
- Data augmentation is disabled during network validation.
- During training, a fixed drop-out rate of 0.5 is used for all the experiments.
- Dropout is disabled during network validation.

Evaluation and comparison of the different networks are done using the metrics defined in the section 3.3. The specifications of the inference procedure are specified as follows,

- All the inference is done on Alienware 13 R3 laptop which has the following specifications,
  1. CPU: Intel Core i7-7700HQ
  2. GPU: NVIDIA GeForce GTX 1060 Mobile with 6144 MB of GDDR5.
  3. RAM: 16GB (15.9GB usable)
  5. Tensorflow version: TensorFlow (GPU) 1.9.0.

The system is disconnected from the internet and all the back-ground programs are closed while running the bench-marking evaluation code.

- Since the ground-truth is not provided for the test-set of the cityscape dataset, the evaluation of the trained networks is done against the validation set.
- Dropout is disabled.
- No data augmentation is used.
- All the image and ground-truth pre-processing steps mentioned in section 3.4 are performed.
- The ground-truth annotations for every image is written into a separate file, for documentation purpose.
- The time in milliseconds (ms) is recorded at different stages of the inference pipeline specified in the section 3.3.3.
- The detections from the network are filtered use Non-Maximum-Suppression (with IoU threshold of 0.4).
• All the resulting detections with class-scores lesser than 0.4 are further filtered out.

• The final resulting detections are written into a file, for documentation purposes.
This chapter provides the implementation details of the various ideas proposed by this work. The first section describes the process of anchor design, the second section deals with the handling of the boundary adhering object instances, the third section elaborates on the ground-truth encoding/decoding schemes, this is followed by the architectural modification and finally the loss modifications. A number of hypotheses are defined in this chapter which will be evaluated in the experiments conducted by this work. **Note:** All the hypotheses proposed by this work are tagged according to the following convention,

\[ Hypothesis_N \]

where,  
\( N \) represents the hypothesis count.

### 4.1 The quest for the finest anchors

The quality of prior anchors used by an anchor-based object detection network not only accelerates the learning process of the network but also has a substantial influence on the performance of the network. The first part of this section elaborates on the traditional process of designing custom anchors for a particular dataset. The next section provides insight as to why the traditional approach is not appropriate and how it can be improved.

#### 4.1.1 Anchor design as a clustering problem

As justified earlier in the network design decision **Network-DD-2**, a key positive side-effect of using anchor-based object detection techniques is the ability to incorporate knowledge extracted from the dataset distribution. Specifically, the knowledge extracted from the dataset distribution can be used to design custom prior anchor shapes for the object segmentation network. The authors in **SqueezeDet** (Wu et al. [62]) and **YOLOv2** (Redmon and Farhadi [45]) formulate the process of generating prior anchor shapes from the dataset distribution as a clustering problem (k-means). There is a key difference in the way these techniques formulate the distance metric for the clustering.
4.1. The quest for the finest anchors

Figure 4.1: Linearly extracted anchors

Figure 4.2: Logarithmically extracted anchors.
Suppose the dataset contains $N$ object instances and hence contains $N$ bounding boxes. Let $b_i = (w_i, h_i)$ represent the $i^{th}$ bounding box. If $K$ represent the desired number of anchor boxes, then let $a_k = (a_w^k, a_h^k)$ represent the $k^{th}$ anchor box. Then the different distance metric formulations are as follows,

1. **YOLOv2 (Redmon and Farhadi [45]):**

   \[
   d = \sum_{i=0}^{N-1} \sum_{k=0}^{K-1} u_{ik} (1 - IoU(b_i, a_k))
   \]  

   \[
   \]  

   where, \[
   u_{ik} = \begin{cases} 
   1 & \text{if } b_i \text{ belongs to } a_k\text{'s cluster.} \\
   0 & \text{otherwise} 
   \end{cases} 
   \]

   The objective is to minimize this distance metric by choosing appropriate anchors. For the scope of this work, we employ the SqueezeDet distance metric. The anchor shapes extracted using this SqueezeDet formulation are termed as linearly extracted anchors or just linear anchors throughout the scope of this work. The number of anchors is fixed to 9 for the entirety of this work. The corresponding linearly extracted anchors for the Cityscapes dataset (Cordts et al. [7]) are as defined in Table 4.1. The clustering used to generate these anchors is illustrated in Figure 4.1.

2. **SqueezeDet (Wu et al. [62]):**

   \[
   d = \sum_{i=0}^{N-1} \sum_{k=0}^{K-1} u_{ik} \sqrt{(w_i - a_w^k)^2 + (h_i - a_h^k)^2}
   \]

   \[
   \]  

   Table 4.1: Linear anchors.

   4.1.2 Non-linearly extracted anchors

   As seen above, SqueezeDet performs clustering in cartesian space of width and height. In other words each data-point is given by the pair $(w_i, h_i)$. The objective of the clustering is to reduce the distance between pre-defined number of centroids (anchors) and all the other data-points. The SqueezeDet (Wu et al. [62]) uses center offset encoding/decoding scheme described elaborately in the section 4.3.1. The equations which represent the ground-truth width and height encoding relative to the $k^{th}$ anchor’s width and height are restated below.

   \[
   \Delta_w = \ln \left( \frac{w}{a_w^k} \right)
   \]

   \[
   \Delta_h = \ln \left( \frac{h}{a_h^k} \right)
   \]

   Table 4.2: Log anchors.
where,

\[ w \text{ and } h \text{ are ground-truth width and height respectively.} \]

\[ a_{w_k} \text{ and } a_{h_k} \text{ are } k^{th} \text{ anchor’s width and anchor height respectively.} \]

This can be reformulated using the logarithmic identity \( \ln \left( \frac{nr}{dr} \right) = \ln (nr) - \ln (dr) \) as follows,

\[
\Delta_w = \ln (w) - \ln (a_{w_k}) \quad (4.5)
\]

\[
\Delta_h = \ln (h) - \log_e (a_{h_k}) \quad (4.6)
\]

\( \Delta_w \) and \( \Delta_h \) are the values the network is designed to predict. From the above equations, a key observation is that the network tries to predict the difference between the logarithmic transformations of the widths and heights. The natural logarithm is a non-linear transformation and hence there is a discrepancy between the standard cartesian space (the coordinate basis is defined by the width and height of the ground-truth bounding boxes) used for clustering by SqueezeDet (Wu et al. [62]) and the center offset encoding/decoding scheme.

This work proposes to perform clustering in the cartesian space defined by \( \ln(\text{width}) \) and \( \ln(\text{height}) \) where each data-point is given by the pair \( (\ln(w_i), \ln(h_i)) \) with the objective being to reduce the distance between a pre-defined number of centroids (anchors) and all the other data-points in this space. The distance metric can be stated as below,

\[
d = \sum_{i=0}^{N-1} \sum_{k=0}^{K-1} u_{ik} \sqrt{(\ln(w_i) - \ln(a_{w_k}))^2 + (\ln(h_i) - \ln(a_{h_k}))^2} \quad (4.7)
\]

The anchor shapes extracted using this non-linear formulation are termed as logarithmically extracted anchors or just log anchors for the entire scope of this work. The corresponding logarithmically extracted anchors for the Cityscape dataset (Cordts et al. [7]) are as defined in Table 4.2. The clustering used to generate these anchors is illustrated in Figure 4.2.

This provides the basis for the first hypothesis proposed by this work.

\[ \text{Hypothesis 1:} \]

\emph{Logarithmically extracted anchors are better than linearly extracted anchors and hence produce better object segmentation networks.}

\textbf{Note:} Centroids for both the clustering methods are initialized by naive sharding technique\(^1\) instead of random initialization. This has shown to accelerate convergence of k-means.

\(^1\)The steps involved are as mentioned below, 1. For all points say \( (x_i, y_i) \), calculate their sum i.e. \( x_i + y_i \). 2. Arrange the points in ascending order according to the sum calculated in Step 1. 3. Split all these points in the dataset into \( k \) equal sized shards. In our case \( k = 9 \). 4. Find the mean \( x \) and \( y \) co-ordinates of the points in each of the \( k \) shards and these are the \( k \) initial centroids.
4.2 Boundary adhesion considerations

Section 1.1 elaborately described the rationale behind the problematic object instances at the boundaries of the image. These problematic object instances are termed boundary adhering object instances or object instances with boundary adhesion.

Section 1.1 provided two courses of actions to address these boundary adhering object instances. Ignoring these object instances is not feasible since well-annotated data is hard to come by. Hence the chosen course of action was to selectively filter out tainted information in the annotation and use only the untainted information to train the network. This section is structured into two parts. The first part deals with the automatic detection of these boundary adhering object instances. The second part provides details on how to extract untainted information from these problematic object instance annotations to train the network.

4.2.1 Detecting boundary adhering object instances

An image is two dimensional and it has four boundaries namely, left, top, right, and bottom. Detecting object instances adhering to any of these boundaries is relatively straightforward. Let P represent the set of points in a 2D image plane representing the instance mask or instance polygon of the object. Formally, it is given by,

\[ P = \{(x_i, y_i) | i = 0, 1, 2, ..., K\} \quad (4.8) \]

where,

\[ K = \text{number of points in the annotation (mask/polygon).} \]

The line segments bounding the instance annotation (mask/polygon) can be obtained as follows,

- **Left boundary indicator:**

  \[ B_{\text{left}} = \{\min(X) | X = \{x_i | (x_i, y_i) \in P\}\} \quad (4.9) \]

- **Top boundary indicator:**

  \[ B_{\text{top}} = \{\min(Y) | Y = \{y_i | (x_i, y_i) \in P\}\} \quad (4.10) \]

- **Right boundary indicator:**

  \[ B_{\text{right}} = \{\max(X) | X = \{x_i | (x_i, y_i) \in P\}\} \quad (4.11) \]

<table>
<thead>
<tr>
<th></th>
<th>margin thickness</th>
</tr>
</thead>
<tbody>
<tr>
<td>left_margin</td>
<td>6 pixels</td>
</tr>
<tr>
<td>top_margin</td>
<td>5 pixels</td>
</tr>
<tr>
<td>right_margin</td>
<td>5 pixels</td>
</tr>
<tr>
<td>bottom_margin</td>
<td>5 pixels</td>
</tr>
</tbody>
</table>

Table 4.3: Cityscape dataset (Cordts et al. [7]) image margins.
4.2. Boundary adhesion considerations

- **Bottom boundary indicator**:

\[ B_{\text{bottom}} = \max(Y) | Y = \{y_i | (x_i, y_i) \in P \} \]  \hspace{1cm} (4.12)

The images in the Cityscape (Cordts et al. [7]) have zero padded margins at the image boundaries. The thickness of these margins are given by the Table 4.3. Refer appendix chapter A for more details on how these margins were determined. If W×H (for cityscape W=2048 and H=1024) is the resolution of the image in the dataset, then the boundary adhesion conditions can be defined as follows,

- **Left boundary adhesion condition**:

\[ C_{\text{left}} = \begin{cases} 
    \text{True} & \text{if } B_{\text{left}} \leq \text{left\_margin} \\
    \text{False} & \text{otherwise} 
\end{cases} \]

- **Top boundary adhesion condition**:

\[ C_{\text{top}} = \begin{cases} 
    \text{True} & \text{if } B_{\text{top}} \leq \text{top\_margin} \\
    \text{False} & \text{otherwise} 
\end{cases} \]

- **Right boundary adhesion condition**:

\[ C_{\text{right}} = \begin{cases} 
    \text{True} & \text{if } B_{\text{right}} \geq W - \text{right\_margin} \\
    \text{False} & \text{otherwise} 
\end{cases} \]

- **Bottom boundary adhesion condition**:

\[ C_{\text{bottom}} = \begin{cases} 
    \text{True} & \text{if } B_{\text{bottom}} \leq H - \text{bottom\_margin} \\
    \text{False} & \text{otherwise} 
\end{cases} \]

**Note:**

1. The conditions defined above are not mutually exclusive.

2. For bounding-box prediction, each ground-truth bounding box is associated with a boundary adhesion condition vector which is defined as follows,

\[ C_{\text{vector}} = \{C_{\text{left}}, C_{\text{top}}, C_{\text{right}}, C_{\text{bottom}}\} \]  \hspace{1cm} (4.13)

3. For octagonal predictions, each ground-truth parameterization is associated with a boundary adhesion condition vector of size 8 and it is defined as follows,

\[ C_{\text{vector}} = \{C_{\text{left}}, C_{\text{left\_top}}, C_{\text{top}}, C_{\text{top\_right}}, C_{\text{right}}, C_{\text{right\_bottom}}, C_{\text{bottom}}, C_{\text{bottom\_left}}\} \]  \hspace{1cm} (4.14)
The new condition values in the vector are deduced from the values of $C_{left}$, $C_{top}$, $C_{right}$, $C_{bottom}$ using the logical AND ($\land$) operator. Their definitions are as follows,

$$C_{left\_top} = C_{left} \land C_{top}$$

$$C_{top\_right} = C_{top} \land C_{right}$$

$$C_{right\_bottom} = C_{right} \land C_{bottom}$$

$$C_{bottom\_left} = C_{bottom} \land C_{left}$$

### 4.2.2 Untainted information extraction from boundary adhering object annotations

The condition vector $C_{vector}$ defined in the previous sub-section provides all the information necessary to extract untainted information from the ground-truth annotation. However, using this information for training the network is limited by the ground-truth parameterization employed by the network.

For bounding boxes, each ground-truth annotation is associated with a condition vector $C_{vector} = \{C_{left}, C_{top}, C_{right}, C_{bottom}\}$. Consider an example where the object instance touches the left and the top boundaries of the image. Then the corresponding condition vector is given by, $C_{vector} = \{True, True, False, False\}$. In this situation, the parameters associated with the top and left boundary of the bounding box are tainted and should be ignored. The parameters associated with the right and bottom boundaries are untainted and can be used.

Most object detectors, parameterize the bounding boxes using four parameters, i.e., x coordinate of the center, y coordinate of the center, width of the bounding box and height of the bounding box. This
is illustrated by the “Legacy bounding box parameterization” in the Figure 4.3. This parameterization does not provide distinct parameters for each boundary of the ground-truth bounding box and hence is not suitable. To effectively utilize the information provided by the condition vector an alternate parameterization is needed. If the bounding box is parameterized by xmin, ymin, xmax, ymax then this parameterization decouples the distinct boundaries of the bounding box from each other. This is indicated by the “Decoupled bounding box parameterization” in the Figure 4.3. Referring to the above example where \( C_{vector} = \{True, True, False, False\} \), it is now possible to ignore the tainted xmin and ymin parameters and consider only xmax and ymax for training.

This provides the basis for the second hypothesis proposed by this work.

**Hypothesis 2:**

Selectively filtering the boundary adhering object instances and utilizing only the untainted partial information of these instances for learning, should result in the overall performance improvement of a regression based object segmentation network.

### 4.3 Ground-truth transformation design

Ground-truth transformation is one of the most critical components of any regression-based object segmentation network. The idea behind the transformation is to have an intermediate representation of the ground-truth annotation which makes it easier for the network to learn it. This representation is created relative to the prior anchor boxes, which eases the learning process. During inference, the encoded vector obtained at the output of the network is subjected to an inverse transformation to make sense of the network’s predictions. This transformation and its associated inverse transformation are termed as ground-truth encoding/decoding for the rest of this work. The key objectives of ground-truth encoding/decoding are,

- The encoded vector size determines the number of values the network needs to predict. Hence it should be minimal.
- The decoding of the encoded ground-truth vector should be fast. This is essential to minimize network latency during inference.

Below sub-sections elaborate on the different ground-truth encoding/decoding schemes employed by this work. Since the parameterization of the ground-truth has a direct implication on the encoding and decoding formulation, the first part of each sub-section describes the parameterizations used for the bounding boxes and the octagonal masks. This is then followed by an explanation of how these parameterizations are encoded and decoded by the scheme.

#### 4.3.1 Center-offset encoding/decoding scheme

Figures 4.4a and 4.4b illustrate the bounding box and octagonal mask parameterizations used by this scheme.
Chapter 4. Solution

(a) Bounding box parameterization.
(b) Octagonal mask parameterization.

Figure 4.4: Parameterizations used by center-offset encoding/decoding scheme. The green bounding-box/polygon represents the ground-truth and the red bounding-box represents the anchor box.

Parameterization specifications:

1. Bounding box:

   In this encoding scheme, bounding box parameterization needs **four** degrees of freedom to completely describe each individual instance of an object. Referring to the Figure 4.4a, **two** degrees of freedom [the object center’s x-coordinate (c_x) and y-coordinate (c_y)] are used for localization and **two** other degrees of freedom [width (w) and height (h)] are used for size estimation.

2. Irregular octagonal masks:

   Using irregular octagonal polygons to parameterize the instance masks increases the number of degrees of freedom from **four** to **eight**. Referring to the Figure 4.4b, the first **four** degrees of freedom are similar to the ones used for bounding box parameterization above i.e., c_x and c_y are used for localization and w and h are used for size estimation. The additional **four** degrees of freedom i.e., o_1, o_2, o_3 and o_4 now help in capturing the approximate shape information of the objects.

Encoding-Decoding specifications:

The table 4.4 provides the encoding and decoding transformations used by this scheme. Only the **first four** transformations are used for bounding box regression (since the number of degrees of freedom is four). All the **eight** transformations are used for octagonal mask regression.
4.3. Ground-truth transformation design

4.3.2 Anchor-offset encoding/decoding schemes

In anchor-based object detection techniques, the availability of dense grid of anchors eliminates the need to explicitly localize the center of the objects. In these encoding/decoding schemes, bounding box parameterization uses four degrees of freedom to completely describe each individual instance of an object (similar to center offset encoding/decoding scheme). However, all of those four degrees of freedom are used for localization of the boundaries of the object. Since there are scenarios where the anchor centers might lie outside the objects, the offsets i.e., \( o_1, o_2, o_3 \) and \( o_4 \) are measured relative to the object centers for the irregular octagonal parameterization. A direct implication of this is that the encoding and decoding transformations used for \( o_1, o_2, o_3 \) and \( o_4 \) are exactly the same as the ones defined in the

<table>
<thead>
<tr>
<th></th>
<th>Encoding transformation</th>
<th>Decoding transformation</th>
<th>Bounding box regression</th>
<th>Octagonal mask regression</th>
</tr>
</thead>
<tbody>
<tr>
<td>center x-coordinate</td>
<td>( \Delta x = \frac{c_x - a_x}{\hat{a}_w} )</td>
<td>( \hat{c}_x = a_x + (a_w \cdot \Delta x) )</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>center y-coordinate</td>
<td>( \Delta y = \frac{c_y - a_y}{\hat{a}_h} )</td>
<td>( \hat{c}_y = a_y + (a_h \cdot \Delta y) )</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>width</td>
<td>( \Delta w = \ln \left( \frac{w}{\hat{a}_w} \right) )</td>
<td>( \hat{w} = a_w \cdot e^{\Delta w} )</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>height</td>
<td>( \Delta h = \ln \left( \frac{h}{\hat{a}_h} \right) )</td>
<td>( \hat{h} = a_h \cdot e^{\Delta h} )</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>offset-1</td>
<td>( \Delta o_1 = \ln \left( \frac{a_1 + \epsilon}{\hat{a}_d} \right) )</td>
<td>( \hat{o}_1 = (a_d \cdot e^{\Delta o_1}) - \epsilon )</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>offset-2</td>
<td>( \Delta o_2 = \ln \left( \frac{a_2 + \epsilon}{\hat{a}_d} \right) )</td>
<td>( \hat{o}_2 = (a_d \cdot e^{\Delta o_2}) - \epsilon )</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>offset-3</td>
<td>( \Delta o_3 = \ln \left( \frac{a_3 + \epsilon}{\hat{a}_d} \right) )</td>
<td>( \hat{o}_3 = (a_d \cdot e^{\Delta o_3}) - \epsilon )</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>offset-4</td>
<td>( \Delta o_4 = \ln \left( \frac{a_4 + \epsilon}{\hat{a}_d} \right) )</td>
<td>( \hat{o}_4 = (a_d \cdot e^{\Delta o_4}) - \epsilon )</td>
<td>×</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4.4: Encoding and decoding transformations employed by center-offset encoding/decoding scheme. \( \epsilon = 10^{-8} \) is a small value added to the offsets to prevent invalid values inside logarithm.

66
Chapter 4. Solution

Table 4.4. There are two variants of this encoding/decoding scheme proposed in this work.

1. **Anchor-offset linear encoding/decoding scheme:**

   Figures 4.5a and 4.5b illustrate the bounding box and octagonal mask parameterizations used by this scheme.

   ![Figure 4.5a](image)

   ![Figure 4.5b](image)

   (a) Bounding box parameterization. (b) Octagonal mask parameterization.

   **Figure 4.5:** Parameterizations used by anchor-offset linear encoding/decoding scheme. The green bounding-box/polygon represents the ground-truth and the red bounding-box represents the anchor box.

   **Parameterization specifications:**

   (a) **Bounding box:** In this encoding scheme, all the **four** degrees of freedom are used for localization of the boundary of the object with respect to the anchor boundaries. Referring to the Figure 4.5a,
   
   - $\delta x -$ represents the perpendicular distance of the object’s left boundary from the left anchor boundary.
   - $\delta x +$ represents the perpendicular distance of the object’s right boundary from the right anchor boundary.
   - $\delta y -$ represents the perpendicular distance of the object’s top boundary from the top anchor boundary.
   - $\delta y +$ represents the perpendicular distance of the object’s bottom boundary from the bottom anchor boundary.

   (b) **Irregular octagonal masks:** For the irregular octagonal parameterization, the first **four** degrees of freedom are same as the ones used for bounding box encoding i.e., $\delta x -$, $\delta x +$, $\delta y -$ and $\delta y +$ relative to the anchor boundaries. The additional **four** degrees of freedom i.e., $o_1$, $o_2$, $o_3$ and $o_4$ are relative to the object centers as seen in the Figure 4.5b.
4.3. Ground-truth transformation design

Encoding-Decoding specifications:

The Table 4.5 provides the encoding and decoding transformations used by this scheme. Please refer to the Table 4.4 for the encoding/decoding transformations of the offsets.

<table>
<thead>
<tr>
<th>Boundary</th>
<th>Encoding transformation</th>
<th>Decoding transformation</th>
<th>Bounding box regression</th>
<th>Octagonal mask regression</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>( \Delta_{x_{\text{min}}} = \frac{(c_{x} - 0.5w) - (a_{x} - 0.5a_{w})}{a_{w}} )</td>
<td>( x_{\text{min}} = (a_{x} - 0.5a_{w}) + a_{w} \cdot \Delta_{x_{\text{min}}} )</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>right</td>
<td>( \Delta_{x_{\text{max}}} = \frac{(c_{x} + 0.5w) - (a_{x} + 0.5a_{w})}{a_{w}} )</td>
<td>( x_{\text{max}} = (a_{x} + 0.5a_{w}) + a_{w} \cdot \Delta_{x_{\text{max}}} )</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>top</td>
<td>( \Delta_{y_{\text{min}}} = \frac{(c_{y} - 0.5h) - (a_{y} - 0.5a_{h})}{a_{h}} )</td>
<td>( y_{\text{min}} = (a_{y} - 0.5a_{h}) + a_{h} \cdot \Delta_{y_{\text{min}}} )</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>bottom</td>
<td>( \Delta_{y_{\text{max}}} = \frac{(c_{y} + 0.5h) - (a_{y} + 0.5a_{h})}{a_{h}} )</td>
<td>( y_{\text{max}} = (a_{y} + 0.5a_{h}) + a_{h} \cdot \Delta_{y_{\text{max}}} )</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4.5: Encoding and decoding transformations employed by anchor offset linear encoding/decoding scheme

2. Anchor-offset non-linear encoding/decoding scheme:

Figures 4.6a and 4.6b illustrate the bounding box and octagonal mask parameterizations used by this scheme.

![Parameterizations](image)

(a) Bounding box parameterization.  
(b) Octagonal mask parameterization.

Figure 4.6: Parameterizations used by anchor-offset non-linear encoding/decoding scheme. The green bounding-box/polygon represents the ground-truth and the red bounding-box represents the anchor box.

Parameterization specifications:
(a) **Bounding box:** In this encoding scheme, all the four degrees of freedom are used for localization of the boundary of the object with respect to the anchor center. Referring to the Figure 4.6a,

- $\delta x -$ represents the perpendicular distance of the object’s left boundary from the anchor center.
- $\delta x +$ represents the perpendicular distance of the object’s right boundary from the anchor center.
- $\delta y -$ represents the perpendicular distance of the object’s top boundary from the anchor center.
- $\delta y +$ represents the perpendicular distance of the object’s bottom boundary from the anchor center.

(b) **Irregular octagonal masks:** For the irregular octagonal parameterization, the first four degrees of freedom are same as the ones used for bounding box encoding i.e., $\delta x -, \delta x +, \delta y -$ and $\delta y +$ relative to the anchor center. The additional four degrees of freedom i.e., $\delta_1, \delta_2, \delta_3$ and $\delta_4$ are relative to the object centers as seen in the Figure 4.6b.

**Encoding-Decoding specifications:**

The Table 4.6 provides the encoding and decoding transformations used by this scheme. Please refer the Table 4.4 for the encoding/decoding transformations of the offsets.

<table>
<thead>
<tr>
<th></th>
<th>Encoding transformation</th>
<th>Decoding transformation</th>
<th>Bounding box regression</th>
<th>Octagonal mask regression</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>left boundary</strong></td>
<td>$\Delta_{xmin} = \ln \left( \max \left( \frac{a_x - (c_x - 0.5w)}{a_x}, 0 \right) + \rho \right)$</td>
<td>$xmin = a_x - a_w \cdot (e^{\Delta_{xmin}} - \rho)$</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>right boundary</strong></td>
<td>$\Delta_{xmax} = \ln \left( \max \left( \frac{a_x + (0.5w - c_x)}{a_x}, 0 \right) + \rho \right)$</td>
<td>$xmax = a_x + a_w \cdot (e^{\Delta_{xmax}} - \rho)$</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>top boundary</strong></td>
<td>$\Delta_{ymin} = \ln \left( \max \left( \frac{a_y - (c_y - 0.5h)}{a_y}, 0 \right) + \rho \right)$</td>
<td>$ymin = a_y - a_h \cdot (e^{\Delta_{ymin}} - \rho)$</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>bottom boundary</strong></td>
<td>$\Delta_{ymax} = \ln \left( \max \left( \frac{a_y + (0.5h - c_y)}{a_y}, 0 \right) + \rho \right)$</td>
<td>$ymax = a_y + a_h \cdot (e^{\Delta_{ymax}} - \rho)$</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4.6: Encoding and decoding transformations employed by anchor offset non-linear encoding/decoding scheme. Based on experiments $\rho = 0.5$ produced stable training and hence is used for all the experiments.

To relatively compare the run-times of the networks using anchor-offset encoding/decoding schemes against the networks using center-offset encoding/decoding scheme, the number of operations can be counted for one bounding box decoding (since encoding is used only during training and not during inference). The additional parameters used for defining the octagonal masks have same encoding/decoding
operations in the center-offset and the anchor-offset schemes and hence just considering the bounding box encoding/decoding should provide a good estimate for the relative run-time performance.

1. **Center-offset encoding/decoding scheme:** Referring to the Table 4.4, decoding of x and y co-ordinate of the center needs 1 multiplication operation and 1 addition operation each. The decoding of the width and height of the bounding box needs, 1 exponential operation and 1 multiplication operation each. Hence in total, decoding of one bounding box needs 4 multiplication, 2 addition and 2 exponential operations.

2. **Anchor-offset linear encoding/decoding scheme:** Referring to the Table 4.5, decoding of each boundary needs 2 addition operations (since subtracting a number is the same as adding negative of the number) and 2 multiplication operations. Hence in total, decoding of one bounding box needs 8 multiplication, 8 addition operations.

3. **Anchor-offset non-linear encoding/decoding scheme:** Referring to the Table 4.6, decoding of each boundary needs 2 addition operations (since subtracting a number is the same as adding negative of the number), 1 multiplication operation and 1 exponential operation. Hence in total, decoding of one bounding box needs 4 multiplication, 8 addition and 4 exponential operations.

From the above discussion, it is clear that the anchor-offset encoding/decoding scheme has more number of operations as compared to the center-offset encoding/decoding scheme and this provides the basis for the third hypotheses proposed by this work.

<table>
<thead>
<tr>
<th>Hypothesis 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Boundary adhesion considerations should result in only minimal degradation of the run-time performance of the regression based object segmentation networks.</strong></td>
</tr>
</tbody>
</table>

The log anchors coupled with anchor-offset non-linear encoding/decoding eases the learning process of the network. Hence the fourth hypothesis of this work can be proposed as follows,

<table>
<thead>
<tr>
<th>Hypothesis 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A regression based object segmentation network with anchor-offset non-linear encoding/decoding scheme, paired with logarithmically extracted anchors, should perform better than a similar network with anchor-offset linear encoding/decoding scheme, paired with linearly extracted anchors.</strong></td>
</tr>
</tbody>
</table>

### 4.4 Architectural modifications

According to the design decision **Network DD 3**, the SqueezeDet/SqueezeDet+ are chosen as the base for all our experiments. SqueezeDet/SqueezeDet+ introduced by Wu et al. [62] segment the objects using bounding boxes. This section provides details on the modifications to be made to the vanilla
SqueezeDet/SqueezeDet+ for them to be able to produce finer octagonal segmentations. The Figure 4.7 provides a restructured view of the network by separating the feature extracting back-bone network and detection head.

The interpretations of the basic building blocks of the network presented in Figure 4.7 are as follows,

- **CONV-N**: This represents a 2D convolutional layer. The specifications of the convolutional layer are provided in the parenthesis. The template for the specifications for the convolutional layer is \((k, s, ch)\). \(k\) is the size of the square kernel, \(s\) is the stride, \(ch\) is the number of output channels.

- **POOL-N**: This represents a 2D max-pooling layer and its specifications are provided in the parenthesis. The template for the specifications is \((k, s)\). \(k\) is the size of the kernel and \(s\) is the stride. The number of output channels is the same as the number of input channels.

- **FIRE-N**: This represents a FIRE module which was introduced by Iandola et al. [19] in SqueezeNet. An example of a FIRE module is as shown in Figure 4.8. The template for the specifications of the
4.4. Architectural modifications

Figure 4.8: Fire module introduced by Iandola et al. [19]

Fire module is \((s_{1\times1}, e_{1\times1}, e_{3\times3})\). \(s_{1\times1}\) is the number of output channels for the \(1\times1\) convolution used in the squeeze layer of the fire module. \(e_{1\times1}\) and \(e_{3\times3}\) represent the number of output channels from the \(1\times1\) and \(3\times3\) convolutions respectively in the expand layer of the fire module.

- **DROP OUT-N**: This represents the drop-out layer introduced by Srivastava et al. [55] for the regularization of the networks. The value in the parenthesis represents the keep-probability of the drop-out layer.

**Note**: In the above discussion, \(N\) is just an incremental counter to keep track of the trainable layers in the network.

In Figure 4.7, both the backbone feature extractor and the detection heads are represented using two separate networks. The left sub-network represents the components for SqueezeDet and the right sub-network represents the components for SqueezeDet+. From the diagram, it is clear that the back-bone network is common for both bounding box prediction and octagonal mask prediction. Detection head (A) represents the ConvDet sub-network used by Wu et al. [62] in SqueezeDet/SqueezeDet+ to make bounding box predictions. Hence the combination of the back-bone feature extractor and the detection head (A) constitutes the vanilla SqueezeDet/SqueezeDet+ network.

By connecting the detection head (B) to the back-bone feature extractor, the resulting network will become capable of predicting octagonal masks segmentations. The detection head (B) is almost identical to the detection head (A) except for the last convolutional layer. The number of output channels from the last convolutional layer is given by the following formula,

\[
n = b \times (c + 1 + p)
\]  

(4.15)

where,

\(n\) = number of output channels from the last convolutional layer.

\(b\) = number of anchor boxes per grid center. \((b = 9\) in all our experiments\)

\(c\) = number of class scores. \((c = 7\) after filtering the classes for Cityscape (Cordts et al. [7]) dataset)
Chapter 4. Solution

\[ p = \text{number of parameters needed for segmentation.} \ (p = 4 \text{ for bounding boxes and for octagonal mask predictions } p = 8) \]

Using the above equation, the number of output channels for the detection head (B) is 144. For the detection head (A) it is 108. The increase in the number of output channels in the detection head (B) as compared to the detection head (A) is due to the increase in the size of the ground-truth encoding.

This provides the basis for the fifth and sixth hypotheses proposed by this work.

**Hypothesis 5:**

The object segmentation performance of an octagonal masks predicting network is at least as good as that of the bounding box predicting network.

**Hypothesis 6:**

The latency of one forward pass through an octagonal mask predicting network is equal to or marginally more than that of the bounding box predicting network.

As discussed before, the octagonal mask predicting network in Figure 4.7 is similar to the bounding box predicting network except for the last layer CONV-12 layer. This makes the setting well-suited for transfer-learning. The trained weights of the bounding box predicting network (except CONV-12) can be loaded into the octagonal mask predicting network as a good initialization step for further fine-tuning. This would be especially beneficial when the dataset is sparse. This provides the basis for the following hypotheses proposed by this work.

**Hypothesis 7:**

Loading the trained weights of a bounding box predicting network into an octagonal mask predicting network and fine-tuning it, helps improve the performance of the octagonal mask predicting network.

**Hypothesis 8:**

Fine-tuning all the layers of the octagonal mask predicting network should provided marginally better performance than fine-tuning only the last layer of the network.

**Note:** Original SqueezeDet by Wu et al. [62] used ‘SAME’ padding for the POOL-1, POOL-3, and POOL-5 layers. In this implementation, the padding is updated to ‘VALID’ for the POOL-1, POOL-3, and POOL-5 layers to maintain the network stride to 16 when the input resolution is increased for the Cityscape dataset.
4.5 Loss modifications

The SqueezeDet by Wu et al. [62] predicts an output volume of \( W \times H \times K \times (4 + 1 + C) \). Here, \( W \times H \) represents number of grid centers and \( K \) is the number of anchors associated with each grid center. \( C \) is the total number of classes. Each anchor box is associated with 4 bounding box parameters, 1 confidence score and \( C \) conditional class probabilities. Let,

(a) the quartet \((\delta x_{ijk}, \delta y_{ijk}, \delta w_{ijk}, \delta h_{ijk})\) represent the encoded bounding box parameters predicted by the network for the \( k^{th} \) anchor at the grid center located at \((i,j)\).

(b) the quartet \((\delta x_{ijk}^G, \delta y_{ijk}^G, \delta w_{ijk}^G, \delta h_{ijk}^G)\) represent the ground-truth encoding of the bounding box parameters. The ground-truth bounding box is matched to the \( k^{th} \) anchor at the grid center located at \((i,j)\).

(c) \( \gamma_{ijk} \) represent the confidence score predicted by the network for the \( k^{th} \) anchor at the grid center located at \((i,j)\).

(d) \( \gamma_{ijk}^G \) represent the ground-truth confidence score for the \( k^{th} \) anchor at the grid center located at \((i,j)\). It is obtained by computing the IOU of the predicted bounding box with the ground-truth bounding box.

(e) \( L_c^G \in [0, 1] \) represent the ground-truth label encoding for class \( c \in [0, C] \).

(f) \( p_c \in [0, 1] \) represent the probability value predicted by the network for class \( c \in [0, C] \).

(g)

\[
I_{ijk} = \begin{cases} 
1 & \text{if } k^{th} \text{ anchor at grid center located at } (i,j) \text{ has the largest overlap with a ground-truth bounding box.} \\
0 & \text{otherwise} 
\end{cases}
\]

(h) \( T_{ijk} = 1 - I_{ijk} \)

(i) \( \lambda_{\text{bbox}} = 5, \lambda_{\text{conf}}^+ = 75, \lambda_{\text{conf}}^- = 100 \) are all constants. These values are borrowed from Wu et al. [62] and they specify the weights assigned to the different loss components in the multi-task loss equation.

(j) \( \# \text{obj} \) is the number of object instances.

The multi-task loss function employed by the network is defined as shown below,

\[
L_{\text{total}} = L_{\text{reg}}^{\text{bbox}} + L_{\text{conf}} + L_{\text{class}} \tag{4.16}
\]

where,

\( L_{\text{reg}}^{\text{bbox}} \) is the bounding-box regression loss and it is given by,
\[ L_{bbox}^{\text{reg}} = \frac{\lambda_{bbox}}{N_{\text{obj}}} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} I_{ijk} \left( (\delta x_{ijk} - \delta x_{ijk}^G)^2 + (\delta y_{ijk} - \delta y_{ijk}^G)^2 + (\delta w_{ijk} - \delta w_{ijk}^G)^2 + (\delta h_{ijk} - \delta h_{ijk}^G)^2 \right) \]  

\[ L_{conf} \text{ is the confidence score regression loss and it is given by,} \]

\[ L_{conf} = \frac{\lambda_{conf}}{N_{\text{obj}}} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} I_{ijk} \left( (\gamma_{ijk} - \gamma_{ijk}^G)^2 \right) \]  

\[ L_{class} \text{ is the cross-entropy classification loss and it is given by,} \]

\[ L_{class} = -\frac{1}{N_{\text{obj}}} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} \sum_{c=1}^{C} \left( I_{ijk} l_c^G \log(p_c) \right) \]

4.5.1 Loss modification for octagonal mask prediction

According to the architectural changes described in section 4.4, to enable the SqueezeDet network to predict octagonal masks, the size of the encoding is increased from 4 to 8. Hence to train these modified variants of the SqueezeDet (i.e., SqueezeDetOcta), the bounding-box regression loss needs to be modified. All the other losses remain unchanged. The modification is as described below.

\[ L_{\text{octa}}^{\text{reg}} = \frac{\lambda_{bbox}}{N_{\text{obj}}} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} I_{ijk} \left( (\delta x_{ijk} - \delta x_{ijk}^G)^2 + (\delta y_{ijk} - \delta y_{ijk}^G)^2 + (\delta w_{ijk} - \delta w_{ijk}^G)^2 + (\delta h_{ijk} - \delta h_{ijk}^G)^2 \right. \]

\[ \left. + (\delta o_1 \delta_{ijk} - \delta o_1 G_{ijk}^G)^2 + (\delta o_2 \delta_{ijk} - \delta o_2 G_{ijk}^G)^2 + (\delta o_3 \delta_{ijk} - \delta o_3 G_{ijk}^G)^2 + (\delta o_4 \delta_{ijk} - \delta o_4 G_{ijk}^G)^2 \right) \]  

where,

(a) \( \delta o_1 \delta_{ijk}, \delta o_2 \delta_{ijk}, \delta o_3 \delta_{ijk} \) and \( \delta o_4 \delta_{ijk} \) represent the encodings of the additional offsets needed for the octagonal approximation of the instance mask, predicted by the network for the \( k^{th} \) anchor at the grid center located at \( (i,j) \).

(b) \( \delta o_1 G_{ijk}^G, \delta o_2 G_{ijk}^G, \delta o_3 G_{ijk}^G \) and \( \delta o_4 G_{ijk}^G \) represent the additional ground-truth encodings of the offsets needed for the octagonal approximation of the instance mask. The ground-truth mask is matched to the \( k^{th} \) anchor at the grid center located at \( (i,j) \).

Refer section 4.3 for more information on these offsets. The multi-task loss is now given by,

\[ L_{\text{total}} = L_{\text{reg}}^{\text{octa}} + L_{\text{conf}} + L_{\text{class}} \]  

75
4.5. Loss modifications

4.5.2 Loss modification for boundary adhesion considerations

According to the discussion in section 4.2, each ground-truth object instance is associated with a condition vector. The size of the condition vector is 4 if the bounding boxes are used to approximate the instance masks and the size is 8 if the octagonal masks are used to approximate the instance masks. The condition vector determines which parameters of the ground-truth encoding are tainted (hence need to be ignored) and which parameters are clean (hence can be used to train the network). These vectors are as shown below,

(a) For bounding box approximation of the instance mask (refer equation 4.13),

\[ C_{\text{vector}} = \{C_{\text{left}}, C_{\text{top}}, C_{\text{right}}, C_{\text{bottom}} \} \]

(b) For octagonal approximation of the instance mask (refer equation 4.14),

\[ C_{\text{vector}} = \{C_{\text{left}}, C_{\text{left-top}}, C_{\text{top}}, C_{\text{top-right}}, C_{\text{right}}, C_{\text{right-bottom}}, C_{\text{bottom}}, C_{\text{bottom-left}} \} \]

Refer section 4.2 for more information on these condition vectors. In addition to this, for boundary adhesion considerations, the bounding boxes are parameterized using the decoupled parameterization where each bounding box is represented using the quarter \([x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}}]\). To incorporate these changes, the following modifications are made to the regression loss functions,

\[
L_{\text{bbox-bound}}^{\text{reg}} = \frac{\lambda_{\text{bbox}}}{N_{\text{obj}}} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} I_{ijk} \left( s_l (\delta x_{\text{min}}_{ijk} - \delta x_{\text{min}}_{G_{ijk}})^2 + s_t (\delta y_{\text{min}}_{ijk} - \delta y_{\text{min}}_{G_{ijk}})^2 + \\
+ s_r (\delta x_{\text{max}}_{ijk} - \delta x_{\text{max}}_{G_{ijk}})^2 + s_b (\delta y_{\text{max}}_{ijk} - \delta y_{\text{max}}_{G_{ijk}})^2 \right) 
\]

\[
L_{\text{octa-bound}}^{\text{reg}} = \frac{\lambda_{\text{bbox}}}{N_{\text{obj}}} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} I_{ijk} \left( s_l (\delta x_{\text{min}}_{ijk} - \delta x_{\text{min}}_{G_{ijk}})^2 + s_t (\delta y_{\text{min}}_{ijk} - \delta y_{\text{min}}_{G_{ijk}})^2 + \\
+ s_r (\delta x_{\text{max}}_{ijk} - \delta x_{\text{max}}_{G_{ijk}})^2 + s_b (\delta y_{\text{max}}_{ijk} - \delta y_{\text{max}}_{G_{ijk}})^2 + \\
+ s_{lt} (\delta o_{1_{ijk}} - \delta o_{1_{G_{ijk}}})^2 + s_{lb} (\delta o_{2_{ijk}} - \delta o_{2_{G_{ijk}}})^2 + \\
+ s_{rb} (\delta o_{3_{ijk}} - \delta o_{3_{G_{ijk}}})^2 + s_{tr} (\delta o_{4_{ijk}} - \delta o_{4_{G_{ijk}}})^2 \right) 
\]

where,

(a) \( \delta x_{\text{min}}_{ijk}, \delta y_{\text{min}}_{ijk}, \delta x_{\text{max}}_{ijk} \) and \( \delta y_{\text{max}}_{ijk} \) represent the decoupled bounding box parameterization encoding predicted by the network for the \( k^{\text{th}} \) anchor at the grid center located at \((i,j)\).
(b) $\delta x_{\min}^{ijk}, \delta y_{\min}^{ijk}, \delta x_{\max}^{ijk}$ and $\delta y_{\max}^{ijk}$ represent the decoupled bounding box parameterization ground-truth encoding. The ground-truth mask is matched to the $k^{th}$ anchor at the grid center located at $(i,j)$.

(c) $s_l$, $s_t$, $s_r$, $s_b$ represent the weights associated with the loss components of the left, top, right and bottom parameters of the bounding box/octagonal mask respectively.

(d) $s_{lt}$, $s_{bl}$, $s_{rb}$, $s_{tr}$ represent the weights associated with the loss components of the left-top offset ($o_1$), bottom-left offset ($o_2$), right-bottom offset ($o_3$) and top-right offset ($o_4$) parameters of the octagonal mask respectively.

The following function is used to derive the weight for a particular condition vector value.

$$getWeight(c) = \begin{cases} 0 & \text{if } c = True \\ 1 & \text{otherwise} \end{cases}$$

The derived weights are as described below,

- **Left loss component weight:**
  
  $s_l = getWeight(C_{left})$

- **Top loss component weight:**
  
  $s_t = getWeight(C_{top})$

- **Right loss component weight:**
  
  $s_r = getWeight(C_{right})$

- **Bottom loss component weight:**
  
  $s_b = getWeight(C_{bottom})$

- **Left-top offset loss component weight:**
  
  $s_{lt} = getWeight(C_{left\_top})$

- **Bottom-left offset loss component weight:**
  
  $s_{bl} = getWeight(C_{bottom\_left})$

- **Right-bottom offset loss component weight:**
  
  $s_{rb} = getWeight(C_{right\_bottom})$

- **Top-right offset loss component weight:**
  
  $s_{tr} = getWeight(C_{top\_right})$
All the other losses are unchanged. Hence the multi-task loss for a network trained while being conscious of the boundary adhering object instances is given as follows.

If the network is bounding box predicting network then,

\[ L_{\text{total}} = L_{\text{bbox}}^{\text{bound}} + L_{\text{conf}} + L_{\text{class}} \]  \hspace{1cm} (4.24)

If the network is octagonal mask predicting network then,

\[ L_{\text{total}} = L_{\text{octa}}^{\text{bound}} + L_{\text{conf}} + L_{\text{class}} \]  \hspace{1cm} (4.25)
Results and Evaluation

This chapter presents the results of the various experiments defined in section 3.5. An in-depth analysis of the results is conducted with the intention of finding evidence to justify if the various hypotheses defined in chapter 4 are true or not. Multiple experiments might test the same hypothesis for robustness.

Note:

- It is highly recommended to go through the appendix section B.1 to get acquainted with the parallel categorical plots which are extensively used in this chapter.
- In the Run-time analysis plots (for instance Figure 5.4),
  1. **FP**: Forward-pass time, **FT**: Filtration time, **IT**: Interpretation time, **NMS**: Non Maximum Suppression time. These times represent the latencies of the different stages in the inference pipeline defined in Figure 3.7.
  2. The sectors of some of the very minute (like FT: Filtration time) outer slices of time are increased in size so that the description is visible in the figures. However, the overall colors are assigned according to the actual time values.
  3. For each network, as each image in the validation set is passed through it, the latency of each stage in the inference pipeline is recorded. The latency values of the network for all the examples (≈ 492) in the validation set are recorded. This is repeated thrice (since each network variant was trained thrice). The mean ($\mu_t$) and standard deviations ($\sigma_t$) of all these recorded latencies, for all the stages is calculated and then reported in the format $\mu_t \pm \sigma_t$ in the Run-time Analysis.
- In the mAP Analysis for all the experiments, the mAP values are reported in the format $\mu_{mAP} \pm \sigma_{mAP}$ where $\mu_{mAP}$ and $\sigma_{mAP}$ represent the mean and standard deviations across three trials.

5.1 Experiment 1

This experiment is designed to compare the quality of linearly extracted anchors proposed by Wu et al. [62] and the logarithmically extracted anchors introduced as a part of this work. Table 5.1 are the candidate networks that were used for this experiment.

The hypothesis which will be influenced by the results of this experiment is Hypothesis 1.
5.1. Experiment 1

### Table 5.1: Candidate networks that were used for experiment 1.

<table>
<thead>
<tr>
<th>id</th>
<th>maskParams</th>
<th>anchorType</th>
<th>gtEncode</th>
<th>fineTuneMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbox_lin_nor_no</td>
<td>4</td>
<td>linear</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>bbox_log_nor_no</td>
<td>4</td>
<td>log</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
</tbody>
</table>

5.1.1 mAP Analysis

Since both the candidate networks are bounding box predicting networks, the values for the placeholders `predictionType` and `maskParams` are bounding box and 4 respectively for both these networks. Hence, in Figure 5.1, only the `groundTruthType` field determines which mAP calculation scenario is being used. From the figure, it is clear that consistently across different mAP calculation scenarios (refer section 3.3.2 for more details), the network with logarithmically extracted anchors outperform the network with linearly extracted anchors.

On the mAP calculation scenario where the `groundTruthType` and the `predictionType` are both bounding boxes, the `bbox_log_nor_no` network achieves an mAP value of $21.17 \pm 0.13$ which is significantly higher than the mAP value of $19.32 \pm 0.50$ achieved by `bbox_lin_nor_no` network.

On the mAP calculation scenario where the `groundTruthType` is instance mask and the `predictionType` is a bounding box, the `bbox_log_nor_no` network achieves an mAP value of $13.20 \pm 0.10$ which is also considerably larger than the mAP value of $11.60 \pm 0.41$ achieved by `bbox_lin_nor_no` network.

The effectiveness of logarithmically extracted anchors can be attributed to the resolution of the discrepancy between the ground-truth encoding/decoding scheme and the clustering space used for the
anchor generation. In-spite of training both the candidate networks until their training/validation losses settle down, the vast difference in performance between the two networks is evident. This fortifies the argument that the prior anchors play an extremely important role in the anchor-based object detector performance. Anchors don’t just serve as a good starting point for the network, they dictate the final performance achieved by the network. These observations provide evidence to support the argument that the first hypothesis is True.

**Hypothesis 1:**
Logarithmically extracted anchors are better than linearly extracted anchors and hence produce better object segmentation networks.

5.2 Experiment 2

This experiment is designed to compare the performance of an octagonal mask predicting network with the performance of a bounding box predicting network. The type of anchors used for the octagonal mask predicting network candidate depends on the results of the Experiment 1 discussed in the previous section. According to the discussion in the previous section, the candidate network with logarithmically extracted anchors performed better than the network with linearly extracted anchors. Hence the octagonal mask predicting network is paired with logarithmically extracted anchors. The following table provides the candidate network specifications used by this experiment.

<table>
<thead>
<tr>
<th>id</th>
<th>maskParams</th>
<th>anchorType</th>
<th>gtEncode</th>
<th>fineTuneMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbox_log_nor_no</td>
<td>4</td>
<td>log</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>bbox_lin_nor_no</td>
<td>4</td>
<td>linear</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_log_nor_no</td>
<td>8</td>
<td>log</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
</tbody>
</table>

Table 5.2: Candidate networks that were used for experiment 2.

The hypotheses which will be influenced by the results of this experiment are Hypothesis 5 and Hypothesis 6.

5.2.1 mAP Analysis

In Figure 5.2, the groundTruthType and predictionType fields jointly determine which mAP calculation scenario is being used.

Consider the mAP calculation scenario where the groundTruthType and the predictionType are both bounding boxes. In this scenario, from Figure 5.2, bbox_log_nor_no (bounding box predicting network with log anchors) achieves the highest mAP of 21.17 ± 0.13. This is followed by bbox_lin_nor_no (bounding box predicting network with linear anchors) with 19.32 ± 0.50. octa_log_nor_no (octagonal mask predicting...
5.2. Experiment 2

Figure 5.2: Flow-diagram illustrating the dependency of mAP on the candidate network placeholder values for experiment 2. Only placeholders with unique values are indicated in the figure to reduce clutter.

network with log anchors), lags behind both the variants of the bounding box predicting network by a large margin, as it achieves a mere $14.98 \pm 0.53$ mAP.

Consider the mAP calculation scenario where the $\text{groundTruthType}$ is instance mask and the $\text{predictionType}$ can be a bounding box or an octagonal mask. Even in this scenario, bbox$_{\text{log, nor, no}}$ achieves the highest mAP of $13.20 \pm 0.10$. This is followed by bbox$_{\text{lin, nor, no}}$ with mAP of $11.60 \pm 0.41$. octa$_{\text{log, nor, no}}$, when using the bounding box extracted from the octagonal mask predictions, achieves an mAP of $8.74 \pm 0.32$. As expected, the mAP of the octa$_{\text{log, nor, no}}$ improves to $10.31 \pm 0.37$, when the finer octagonal mask predictions are directly used for mAP calculation. However it still under-performs the corresponding bounding box predicting networks. This observation contradicts the fifth hypothesis.

**Hypothesis 5:**

The object segmentation performance of an octagonal masks predicting network is at least as good as that of the bounding box predicting network.

5.2.2 Class-wise AP analysis

To investigate the reason behind the poor performance of the octagonal mask predicting network, the class-wise average precision (AP) values are analyzed for the bounding box predicting network, and the octagonal mask predicting network. For this investigation, the class-wise AP values of the octagonal mask predicting network are subtracted from the corresponding class-wise AP values of the bounding box predicting network. This difference in the class-wise AP values is then plotted against the sample size of the specific class in the Cityscape dataset. Each subplot in Figure 5.3 represents one such plot. The left
Chapter 5. Results and Evaluation

Figure 5.3: Plot of difference in the average precision values vs the sample size for the two mAP calculation scenarios (specified in the individual titles). The size of the markers is proportional to the sample size.

An extremely clear and common overall trend can be seen across the two subplots in Figure 5.3. The difference in APs is small for classes having large sample sizes and it increases with the reduction in the sample size. This trend makes logical sense. Since the octagonal mask predicting network is trying to learn a more detailed version of the object instance, it needs more examples to learn from. From this observation, it can be reasonably stated that the octagonal mask predicting network with performance similar to the bounding box prediction network can be achieved given that there are enough samples in the dataset.

The octagonal mask predicting network and the bounding box predicting network have a lot of layers in common. This setting provides an excellent opportunity to experiment with transfer-learning. Transfer-learning is a proven method to deal with situations where the dataset is sparse. Often a network is first trained on an exhaustive dataset like ImageNet (Deng et al. [9]) or MS COCO (Lin et al. [29]) and then it is fine-tuned on the sparse dataset, starting from the previously trained weights. For the current case, the pre-trained weights of the bounding box predicting network are already available. The octagonal mask predicting network can be loaded with the trained weights of the bounding box predicting network and then it can be further fine-tuned. This should result in significant performance improvement of the octagonal mask predicting network. This provides the basis for the ninth hypotheses proposed by this work which is the reformulation of the Hypothesis 5.
5.2. Experiment 2

Hypothesis 9:

The object segmentation performance of an octagonal masks predicting network, which has been fine-tuned starting from the trained weights of a bounding box predicting network, is at least as good as that of the bounding box predicting network.

5.2.3 Run-time Analysis

Referring to Figure 5.4, it can be seen that the latency of one forward pass through all the candidate networks are close to each other. The \( \text{bbox\_log\_nor\_no} \) and \( \text{bbox\_lin\_nor\_no} \) have a forward latency of 18.62 ± 1.76 ms and 18.40 ± 0.59 ms respectively. \( \text{octa\_log\_nor\_no} \) has a forward latency of 18.73 ± 3.72 ms. This observation provides evidence to support the argument that the sixth hypothesis is True.

Hypothesis 6:

The latency of one forward pass through an octagonal mask predicting network is equal to or marginally more than that of the bounding box predicting network.

bbox\_lin\_nor\_no has the largest interpretation time of 16.77 ± 38.10 ms. This is followed by \( \text{octa\_log\_nor\_no} \) with an interpretation time of 13.61 ± 14.14 ms. \( \text{bbox\_log\_nor\_no} \) has the least interpretation time of 10.84 ± 2.00 ms. Filtration time is negligible in all cases. There are huge variations in the Non Maximum Suppression (NMS) time values between the different networks. Before the outputs of the network are sent for NMS (refer Figure 3.7), they are filtered based on their confidence scores. Only those predictions with confidence scores higher than a particular cut-off threshold are sent for NMS. This justifies the high standard deviation for the latency of the NMS stage as compared to the other stages in the inference pipeline. A large portion of the total run-time is due to the latency of the NMS stage. Highly optimized NMS implementations are available which can be used to reduce the overall latency of these networks.
Chapter 5. Results and Evaluation

5.3 Experiment 3

This experiment was designed to investigate if the performance of the octagonal mask predicting network could be improved by transfer-learning. Specifically, the octagonal mask predicting network is first loaded with the trained weights of the bounding box predicting network. It is then fine-tuned on the same dataset till the training/validation loss of the network settles. In this experiment, two variants of fine-tuning are considered. In the first variant all the layers of octagonal mask predicting network are fine-tuned. In the second variant only the last layer of the octagonal mask predicting network is fine-tuned keeping all the other layers fixed. The specifications of the candidate networks used by this experiment are as follows,

<table>
<thead>
<tr>
<th>id</th>
<th>maskParams</th>
<th>anchorType</th>
<th>gtEncode</th>
<th>fineTuneMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbox_log_nor_no</td>
<td>4</td>
<td>log</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_log_nor_no</td>
<td>8</td>
<td>log</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_log_nor_all</td>
<td>8</td>
<td>log</td>
<td>normal</td>
<td>fine_all</td>
</tr>
<tr>
<td>octa_log_nor_lst</td>
<td>8</td>
<td>log</td>
<td>normal</td>
<td>fine_last</td>
</tr>
</tbody>
</table>

Table 5.3: Candidate networks that were used for experiment 3.

The hypotheses which will be influenced by the results of this experiment are Hypothesis 7, Hypothesis 8 and Hypothesis 9.

5.3.1 mAP Analysis

Since all the candidate networks used in this experiment are equipped with logarithmically extracted anchors, they are not explicitly shown in Figure 5.5. In the figure, the groundTruthType and predictionType fields jointly determine which mAP calculation scenario is being used.

Consider the mAP calculation scenario where both the groundTruthType and the predictionType are bounding boxes. In this scenario, the octa_log_nor_all (octagonal mask predicting network with all the layers fine-tuned) achieves the best mAP score of 21.94 ± 0.14. This is followed by octa_log_nor_lst (octagonal mask predicting network with only the last layer fine-tuned) by achieving an mAP score of 21.21 ± 0.01. Next comes the bbox_log_nor_no (bounding box predicting network with log anchors) by achieving an mAP score of 21.17 ± 0.13. The octa_log_nor_no (octagonal mask predicting network with no fine-tuning) has the least mAP value of 14.98 ± 0.53.

Consider the mAP calculation scenario where the groundTruthType is instance mask and the predictionType can be a bounding box or an octagonal mask. In this scenario, The highest mAP of 15.88 ± 0.20 is achieved by octa_log_nor_all (with octagonal predictions). This is followed by octa_log_nor_lst with an mAP of 15.15 ± 0.03 (with octagonal predictions). Next comes octa_log_nor_all (with bounding boxes extracted from the octagonal masks). It achieves an mAP score of 13.77 ± 0.18. This is followed by octa_log_nor_lst (with bounding boxes extracted from the octagonal masks). It achieves an mAP of 13.26 ±
0.00. This is then followed by \texttt{bbox\_log\_nor\_no} with an mAP of 13.20 ± 0.10. Next comes the \texttt{octa\_log\_nor\_no} (with octagonal predictions). It achieves an mAP score of 10.31 ± 0.37. \texttt{octa\_log\_nor\_no} gets the least mAP score (with bounding boxes extracted from the octagonal masks). It achieves an mAP of 8.74 ± 0.32.

In summary, the fine-tuning considerably improves the performance of the octagonal mask predicting networks.

- In the mAP calculation scenario where the \texttt{groundTruthType} and \texttt{predictionType} are both bounding boxes, \texttt{octa\_log\_nor\_all} and \texttt{octa\_log\_nor\_lst} result in improvements of \(\approx 6.96\) mAP and \(\approx 6.23\) mAP over the \texttt{octa\_log\_nor\_no}.

- In the mAP calculation scenario where the \texttt{groundTruthType} is instance mask and \texttt{predictionType} is octagonal mask, \texttt{octa\_log\_nor\_all} and \texttt{octa\_log\_nor\_lst} result in improvements of \(\approx 5.57\) mAP and \(\approx 4.84\) mAP over the \texttt{octa\_log\_nor\_no}.

- In the mAP calculation scenario where the \texttt{groundTruthType} is instance mask and \texttt{predictionType} is bounding box, \texttt{octa\_log\_nor\_all} and \texttt{octa\_log\_nor\_lst} result in improvements of \(\approx 5.03\) mAP and \(\approx 4.52\) mAP over the \texttt{octa\_log\_nor\_no}.

These observations provide evidence to support the argument that the seventh hypothesis is \textbf{True}. 
Chapter 5. Results and Evaluation

**Hypothesis 7:**

*Loading the trained weights of a bounding box predicting network into an octagonal mask predicting network and fine-tuning it, helps improve the performance of the octagonal mask predicting network.*

In all the mAP calculation scenarios, the octa\_log\_nor\_all has marginally higher mAP values than the octa\_log\_nor\_lst.

- In the mAP calculation scenario where the `groundTruthType` and `predictionType` are both bounding boxes, octa\_log\_nor\_all has an mAP value 0.73 point higher than that of the octa\_log\_nor\_lst.

- In the mAP calculation scenario where the `groundTruthType` is instance mask and `predictionType` is octagonal mask, octa\_log\_nor\_all has an mAP value 0.73 point higher than that of the octa\_log\_nor\_lst.

- In the mAP calculation scenario where the `groundTruthType` is instance mask and `predictionType` is bounding box, octa\_log\_nor\_all has an mAP value 0.51 point higher than that of the octa\_log\_nor\_lst.

These observations provide evidence to support the argument that the eighth hypothesis is **True**.

**Hypothesis 8:**

*Fine-tuning all the layers of the octagonal mask predicting network should provided marginally better performance than fine-tuning only the last layer of the network.*

In all the mAP calculation scenarios, the fine-tuned octagonal mask predicting networks seem to outperform the bounding box predicting networks.

- In the mAP calculation scenario where the `groundTruthType` and `predictionType` are both bounding boxes, the octa\_log\_nor\_all and the octa\_log\_nor\_lst outperform the bbox\_log\_nor\_no by $\approx 0.77$ mAP and $\approx 0.04$ mAP respectively.

- In the mAP calculation scenario where the `groundTruthType` is instance mask and `predictionType` is bounding box, the octa\_log\_nor\_all and the octa\_log\_nor\_lst outperform the bbox\_log\_nor\_no by $\approx 0.57$ mAP and $\approx 0.06$ mAP respectively.

These observations provide evidence to support the argument that the ninth hypothesis is **True**.

**Hypothesis 9:**

*The object segmentation performance of an octagonal masks predicting network, which has been fine-tuned starting from the trained weights of a bounding box predicting network, is at least as good as that of the bounding box predicting network.*

87
5.3.2 Run-time Analysis

Since the network architectures of octa_log_nor_all and octa_log_nor_lst are the same as that of the octa_log_nor_no, the forward pass latencies of these networks are not much different from each other.

5.4 Experiment 4

This experiment was designed to investigate the influence of detecting and handling the boundary adhering object instances on the performance of an anchor-based object segmentation network. This objective can be divided into three action points,

- Compare the performance of a bounding box predicting candidate network trained from scratch with/without boundary adhesion considerations.
- Compare the performance of an octagonal mask predicting candidate network trained from scratch with/without boundary adhesion considerations.
- Compare the performance of an octagonal mask predicting network with the performance of a bounding box predicting network when both are trained while being conscious of the boundary adhering object instances.

The candidate networks used for this experiment have the following specifications,

<table>
<thead>
<tr>
<th>id</th>
<th>maskParams</th>
<th>anchorType</th>
<th>gtEncode</th>
<th>fineTuneMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbox_lin_nor_no</td>
<td>4</td>
<td>linear</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>bbox_log_nor_no</td>
<td>4</td>
<td>log</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>bbox_lin_lin_no</td>
<td>4</td>
<td>linear</td>
<td>asymmetric_linear</td>
<td>fine_disable</td>
</tr>
<tr>
<td>bbox_log_log_no</td>
<td>4</td>
<td>log</td>
<td>asymmetric_log</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_lin_nor_no</td>
<td>8</td>
<td>log</td>
<td>normal</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_log_nor_no</td>
<td>8</td>
<td>log</td>
<td>asymmetric_log</td>
<td>fine_disable</td>
</tr>
</tbody>
</table>

Table 5.4: Candidate networks that were used for experiment 4.

The hypotheses which will be influenced by the results of this experiment are Hypothesis_2, Hypothesis_3, Hypothesis_4, Hypothesis_5 and Hypothesis_6.

5.4.1 mAP Analysis

The overall mAP analysis is split into three parts. In the first part, the influence of boundary adhesion considerations on the performance of the bounding box predictor is investigated. In the second part, the influence of boundary adhesion considerations on the performance of the octagonal mask predictor
is evaluated. The final part compares the performances of the bounding box predictor vs that of the octagonal mask predictor when both are trained by being conscious of the boundary adhering object instances.

**Bounding box predictor analysis**

![Flow-diagram illustrating the dependency of mAP on the candidate network (bounding box predicting only) placeholder values for experiment 4. Only placeholders with unique values are indicated in the figure to reduce clutter.]

Since all the candidate networks for this analysis are bounding box predicting networks, the `maskParams` and `predictionType` fields are not explicitly shown in Figure 5.6. In the figure, only the field, `groundTruthType` determines which mAP calculation scenario is being used.

1. **Influence of boundary adhesion:**

   Consider the mAP calculation scenario where both the `groundTruthType` and the `predictionType` are bounding boxes. In this scenario, the bbox\_lin\_lin\_no (bounding box predicting network with linear anchors and anchor-offset linear encoding/decoding scheme) achieves the best mAP score of 21.94 ± 0.34. This is closely followed by bbox\_log\_log\_no (bounding box predicting network with log anchors and anchor-offset non-linear encoding/decoding scheme) by achieving an mAP score of 21.79 ± 0.24. Next comes bbox\_log\_nor\_no (bounding box predicting network with log anchors) by achieving an mAP score of 21.17 ± 0.13. The bbox\_lin\_nor\_no (bounding box predicting network with linear anchors) has the least mAP score of 19.32 ± 0.50. In summary, boundary adhesion considerations in this mAP calculation scenario results in,

   - improvement of $\approx 0.62$ in the mAP of the bounding box predicting network, when log anchors are used.
5.4. Experiment 4

- improvement of $\approx 2.62$ in the mAP of the bounding box predicting network when linear anchors are used.

Consider the mAP calculation scenario where the `groundTruthType` is instance mask and the `predictionType` can be a bounding box. In this scenario, bbox\_log\_log\_no achieves the highest mAP of 13.70 ± 0.12. This is closely followed by bbox\_lin\_lin\_no with an mAP score of 13.52 ± 0.22. Next comes the bbox\_log\_nor\_no which gets an mAP score of 13.20 ± 0.10. bbox\_lin\_nor\_no gets the least mAP score of 11.60 ± 0.41. In summary, boundary adhesion considerations in this mAP calculation scenario results in,

- improvement of $\approx 0.50$ in the mAP of the bounding box predicting network, when log anchors are used.
- improvement of $\approx 1.92$ in the mAP of the bounding box predicting network when linear anchors are used.

2. Influence of ground-truth encoding/decoding schemes:

Comparing the bbox\_log\_log\_no and bbox\_lin\_lin\_no networks,

- in the mAP calculation scenario where the `groundTruthType` and `predictionType` are both bounding boxes, the bbox\_lin\_lin\_no is $\approx 0.15$ point better than the bbox\_log\_log\_no.
- in the mAP calculation scenario where the `groundTruthType` is instance mask and the `predictionType` is bounding box, the bbox\_log\_log\_no is $\approx 0.18$ point better than the bbox\_lin\_lin\_no.

In summary, bbox\_log\_log\_no is better than bbox\_lin\_lin\_no but only marginally.

Octagonal mask predictor analysis

Since all the candidate networks for this analysis are octagonal mask predicting networks, the `maskParams` field is not explicitly shown in Figure 5.7. In the figure, the `groundTruthType` and `predictionType` fields jointly determines which mAP calculation scenario is being used.

1. Influence of boundary adhesion:

Consider the mAP calculation scenario where both the `groundTruthType` and the `predictionType` are bounding boxes. In this scenario, the octa\_log\_log\_no (octagonal mask predicting network with log anchors and anchor-offset non-linear encoding/decoding scheme) achieves the best mAP score of 17.23 ± 0.31. This is followed by octa\_lin\_lin\_no (octagonal mask predicting network with linear anchors and anchor-offset linear encoding/decoding scheme) by achieving an mAP score of 15.49 ± 0.23. Next comes the octa\_log\_nor\_no (octagonal mask predicting network with log anchors) with an mAP score of 14.98 ± 0.53. In summary, boundary adhesion considerations in this mAP calculation scenario result in an improvement of $\approx 2.25$ in the mAP of the octagonal mask predicting network, when log anchors are used.
Chapter 5. Results and Evaluation

Figure 5.7: Flow-diagram illustrating the dependency of mAP on the candidate network (octagonal mask predicting only) placeholder values for experiment 4. Only placeholders with unique values are indicated in the figure to reduce clutter.

Consider the mAP calculation scenario where the \textit{groundTruthType} is instance mask and the \textit{predictionType} is an octagonal mask. In this scenario, octa\_log\_log\_no achieves the highest mAP of $12.13 \pm 0.30$. This is followed by octa\_lin\_lin\_no with an mAP score of $10.50 \pm 0.12$. Next comes the bbox\_log\_nor\_no which gets an mAP score of $10.31 \pm 0.37$. Hence in this mAP calculation scenario, boundary adhesion considerations result in an improvement of $\approx 1.82$ in the mAP of the octagonal mask predicting network, when log anchors are used.

Consider the mAP calculation scenario where the \textit{groundTruthType} is instance mask and the \textit{predictionType} is a bounding box. In this scenario, octa\_log\_log\_no achieves the highest mAP of $10.41 \pm 0.31$. This is followed by octa\_lin\_lin\_no with an mAP score of $8.90 \pm 0.11$. Next comes the bbox\_log\_nor\_no which gets an mAP score of $8.74 \pm 0.32$. Hence in this mAP calculation scenario, boundary adhesion considerations result in an improvement of $\approx 1.67$ in the mAP of the octagonal mask predicting network, when log anchors are used.

2. \textbf{Influence of ground-truth encoding/decoding schemes:}

Comparing the octa\_log\_log\_no and octa\_lin\_lin\_no networks,

- in the mAP calculation scenario where the \textit{groundTruthType} and \textit{predictionType} are both bounding boxes, the octa\_log\_log\_no is $\approx 1.74$ point better than the octa\_lin\_lin\_no.

- in the mAP calculation scenario where the \textit{groundTruthType} is instance mask and the \textit{predictionType} is a bounding box, the octa\_log\_log\_no is $\approx 1.51$ point better than the octa\_lin\_lin\_no.
• in the mAP calculation scenario where the \texttt{groundTruthType} is instance mask and the \texttt{predictionType} is an octagonal mask, the \texttt{octa\_log\_log\_no} is \( \approx 1.63 \) point better than the \texttt{octa\_lin\_lin\_no}.

In summary, \texttt{octa\_log\_log\_no} is better than \texttt{octa\_lin\_lin\_no} in all the mAP calculation scenarios.

The observations presented in the Influence of boundary adhesion analysis for bounding box predictors and octagonal mask predictors provide evidence to support the argument that the second hypothesis is True.

\textbf{Hypothesis 2:}

\begin{quote}
Selectively filtering the boundary adhering object instances and utilizing only the untainted partial information of these instances for learning, should result in the overall performance improvement of a regression based object segmentation network.
\end{quote}

The observations presented in the Influence of ground-truth encoding/decoding schemes analysis for bounding box predictors and octagonal mask predictors provide evidence to support the argument that the fourth hypothesis is True.

\textbf{Hypothesis 4:}

\begin{quote}
A regression based object segmentation network with anchor-offset non-linear encoding/decoding scheme, paired with logarithmically extracted anchors, should perform better than a similar network with anchor-offset linear encoding/decoding scheme, paired with linearly extracted anchors.
\end{quote}

\noindent\textbf{Bounding box predictor vs Octagonal mask predictor analysis}

From the 5.8, it can be seen that across all the mAP calculation scenarios, the performance of an octagonal mask predicting network consistently lags behind the performance of the bounding box predicting network.

Consider the mAP calculation scenario where both the \texttt{groundTruthType} and the \texttt{predictionType} are bounding boxes. In this scenario, the \texttt{octa\_log\_log\_no} (octagonal mask predicting network with log anchors and anchor-offset non-linear encoding/decoding scheme) and the \texttt{bbox\_log\_log\_no} (bounding box predicting network with log anchors and anchor-offset non-linear encoding/decoding scheme) achieve mAP scores of \( 17.23 \pm 0.31 \) and \( 21.79 \pm 0.24 \) respectively. The \texttt{octa\_lin\_lin\_no} (octagonal mask predicting network with linear anchors and anchor-offset linear encoding/decoding scheme) and \texttt{bbox\_lin\_lin\_no} (bounding box predicting network with linear anchors and anchor-offset linear encoding/decoding scheme) achieve mAP scores of \( 15.49 \pm 0.23 \) and \( 21.94 \pm 0.34 \) respectively. In summary, in this mAP calculation scenario,

• \texttt{octa\_log\_log\_no} lags behind \texttt{bbox\_log\_log\_no} by \( \approx 4.56 \) mAP points.

• \texttt{octa\_lin\_lin\_no} lags behind \texttt{bbox\_lin\_lin\_no} by \( \approx 6.45 \) mAP points.
Consider the mAP calculation scenario where the *groundTruthType* is instance mask and the *predictionType* is a bounding box. In this scenario, octa\_log\_log\_no and bbox\_log\_log\_no achieve mAP scores of 10.41 ± 0.31 and 13.70 ± 0.12 respectively. The octa\_lin\_lin\_no and bbox\_lin\_lin\_no achieve mAP scores of 8.90 ± 0.11 and 13.52 ± 0.22 respectively. In summary, in this mAP calculation scenario,

- octa\_log\_log\_no lags behind bbox\_log\_log\_no by ≈ 3.29 mAP points.
- octa\_lin\_lin\_no lags behind bbox\_lin\_lin\_no by ≈ 4.62 mAP points.

Similar to **Experiment 2**, these above observations also *contradict* the fifth hypothesis.

*Hypothesis 5:*

The object segmentation performance of an octagonal masks predicting network is at least as good as that of the bounding box predicting network.

To investigate the reason behind the poor performance of the octagonal mask predicting network as compared to the bounding box predicting network, a class-wise average precision (AP) analysis (similar to the one performed in **Experiment 2**) is undertaken. The analysis is performed separately for the linear candidate pair (octa\_lin\_lin\_no - bbox\_lin\_lin\_no) and non-linear candidate pair (octa\_log\_log\_no - bbox\_log\_log\_no).

For this analysis, the class-wise AP values of the octagonal mask predicting network are subtracted from the corresponding class-wise AP values of the bounding box predicting network. This difference in the class-wise AP values is plotted against the sample size of the specific class in the Cityscape dataset.
5.4. Experiment 4

Figure 5.9: Plot of difference in the average precision values vs the sample size for different mAP calculation scenarios. **Top row:** Plots for linear candidate pair. **Bottom row:** Plots for non-linear candidate pair. The size of the markers is proportional to the sample size.

Each subplot in figure 5.9 represents one such plot. The left subplot in each row is for the mAP calculation scenario where both, the ground-truth annotations and the predictions are bounding boxes. The right subplot in each row is for the mAP calculation scenario where the ground-truth annotations are instance masks and the predictions are bounding boxes.

A trend similar to the one seen in **Experiment 2** can be observed across the four subplots in Figure 5.9. The difference in APs is small for classes having large sample sizes and it increases with the reduction in the sample size.

For the linear candidate pairs (plots in the top row of Figure 5.9),

- in the mAP calculation scenario where the ground-truth annotations and the predictions are both bounding boxes, the AP gap varies between **1.737** (car) to **13.336** (bus).
• in the mAP calculation scenario where the ground-truth annotations are instance masks and the predictions are bounding boxes, the AP gap varies between 0.925 (rider) to 12.087 (bus).

For Experiment 2 in Figure 5.3,

• in the mAP calculation scenario where the ground-truth annotations and the predictions are both bounding boxes, the AP gap varies between 1.0 (car) to 16.43 (bus).

• in the mAP calculation scenario where the ground-truth annotations are stance masks and the predictions are bounding boxes, the AP gap varies between 0.451 (person) to 14.397 (bus).

For the non-linear candidate pairs (plots in the bottom row of Figure 5.9),

• in the mAP calculation scenario where the ground-truth annotations and the predictions are both bounding boxes, the AP gap varies between 0.758 (car) to 9.178 (truck).

• in the mAP calculation scenario where the ground-truth annotations are instance masks and the predictions are bounding boxes, the AP gap varies between 0.538 (car) to 8.079 (bus).

Based on the above discussion, the following remarks can be made,

1. The candidate networks used in Experiment 2 have log anchors and they do not explicitly handle boundary adhering object instances. These networks have the largest AP difference ranges.

2. By explicitly handling the boundary adhering object instances, the candidate networks used in this experiment are able to reduce the AP difference ranges.

3. The non-linear candidates have the smallest difference ranges.

4. The difference range of the linear candidates seems to fall between the difference range of the candidates used in Experiment 2 and that of the non-linear candidates used in this experiment.

**AP difference range is an indicator of the extent of degradation of the object segmentation performance as the parameterization of the instance mask is increased from 4 (bounding boxes) to 8 (octagonal masks).** From the above remarks, it is clear that explicitly handling the boundary adhering object instances, enhances the robustness of the object segmentation network, to the increase in the parameterization size of the instance mask.

### 5.4.2 Run-time Analysis

Similar to the run-time analysis in Experiment 2 (Figure 5.4), it can be seen in Figure 5.10 that the latency of one forward pass through all the candidate networks are close to each other. The bounding box predicting candidates, i.e., bbox\_log\_log\_no, bbox\_lin\_lin\_no, bbox\_lin\_nor\_no and bbox\_log\_nor\_no have forward latencies of 18.30 ± 0.52 ms, 18.31 ± 0.52 ms, 18.40 ± 0.59 ms and 18.62 ± 1.76 ms respectively.

The octagonal mask predicting candidates, i.e., octa\_log\_nor\_no, octa\_lin\_lin\_no and octa\_log\_log\_no have forward latencies of 18.73 ± 3.72 ms, 18.45 ± 0.56 ms and 18.55 ± 0.56 ms.
Since the encoding/decoding schemes vary across the different candidate networks, the interpretation times might vary between them. In Figure 5.10, bbox\_lin\_lin\_no has the highest interpretation time of $18.51 \pm 48.66$ ms. The interpretation times of bbox\_log\_log\_no, bbox\_lin\_nor\_no and bbox\_log\_nor\_no are $17.54 \pm 40.54$ ms, $16.77 \pm 38.10$ ms and $10.84 \pm 2.00$ ms respectively. The octagonal mask predicting networks, i.e., octa\_log\_nor\_no, octa\_lin\_lin\_no and octa\_log\_log\_no have interpretation times of $13.61 \pm 14.14$ ms, $17.54 \pm 37.54$ ms and $17.27 \pm 35.34$ ms respectively. The Filtration times are negligible and the NMS times are highly dependent on the number of predictions passed into the NMS stage.

**Bounding box predictor analysis:** The bbox\_lin\_lin\_no and bbox\_log\_log\_no have forward pass latencies similar to that of the bbox\_log\_nor\_no. However, they have slightly higher interpretation times than bbox\_log\_nor\_no network. bbox\_lin\_lin\_no and bbox\_log\_log\_no have larger mAP scores than the bbox\_log\_nor\_no network and hence the slightly higher run-time comes with a welcome improvement in performance.

**Octagonal mask predictor analysis:** Similar to the bounding box analysis performed above, the octa\_lin\_lin\_no and octa\_log\_log\_no have interpretation time slightly higher than that of the octa\_log\_nor\_no while having similar forward pass latencies. The cost of added interpretation time is mitigated to some extent by the improvement in the segmentation performance, hence providing a good trade-off.

The observations made in the bounding box and octagonal mask predictor analysis present evidence to support the argument that the third hypothesis is True.
Chapter 5. Results and Evaluation

**Hypothesis 3:**

Boundary adhesion considerations should result in only minimal degradation of the run-time performance of the regression based object segmentation networks.

**Bounding box predictor vs Octagonal mask predictor analysis:** The forward pass latencies of the octagonal mask predicting candidate networks i.e., octa_lin_lin_no and octa_log_log_no are similar to that of the bounding box predicting networks i.e., bbox_lin_lin_no and bbox_log_log_no. These observations present evidence to support the argument that the sixth hypothesis is True.

**Hypothesis 6:**

The latency of one forward pass through an octagonal mask predicting network is equal to or marginally more than that of the bounding box predicting network.

### 5.5 Experiment 5

The main motivation behind this experiment is to improve the segmentation performance of an octagonal mask predicting network. In other words, this experiment was devised to investigate if the segmentation performance of an octagonal mask predicting network can be improved by first initializing it with the trained weights of the bounding box predicting network and then fine-tuning it on the same dataset. The objective of this experiment is similar to Experiment 3 except for the fact that in this experiment, the candidate networks are trained while being aware of the boundary adhering object instances mentioned in the section 4.2. The specifications of the candidate networks used for this experiment are as follows,

<table>
<thead>
<tr>
<th>id</th>
<th>maskParams</th>
<th>anchorType</th>
<th>gtEncode</th>
<th>fineTuneMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbox_lin_lin_no</td>
<td>4</td>
<td>linear</td>
<td>asymmetric_linear</td>
<td>fine_disable</td>
</tr>
<tr>
<td>bbox_log_log_no</td>
<td>4</td>
<td>log</td>
<td>asymmetric_log</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_lin_lin_no</td>
<td>8</td>
<td>linear</td>
<td>asymmetric_linear</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_log_log_no</td>
<td>8</td>
<td>log</td>
<td>asymmetric_log</td>
<td>fine_disable</td>
</tr>
<tr>
<td>octa_lin_lin_all</td>
<td>8</td>
<td>linear</td>
<td>asymmetric_linear</td>
<td>fine_all</td>
</tr>
<tr>
<td>octa_log_log_all</td>
<td>8</td>
<td>log</td>
<td>asymmetric_log</td>
<td>fine_all</td>
</tr>
<tr>
<td>octa_lin_lin_lst</td>
<td>8</td>
<td>linear</td>
<td>asymmetric_linear</td>
<td>fine_last</td>
</tr>
<tr>
<td>octa_log_log_lst</td>
<td>8</td>
<td>log</td>
<td>asymmetric_log</td>
<td>fine_last</td>
</tr>
</tbody>
</table>

Table 5.5: Candidate networks that were used for experiment 5.

The hypotheses which will be influenced by the results of this experiment are Hypothesis_7, Hypothesis_8 and Hypothesis_9.
5.5.1 mAP Analysis

The overall mAP analysis is split into two parts. In the first part, the mAP analysis for the linear candidate networks (linear anchors and anchor offset linear encoding/decoding scheme) is performed. The second part deals with the mAP analysis for the non-linear candidate networks (log anchors and anchor offset non-linear encoding/decoding scheme).

Linear candidate mAP analysis

Since only linear candidates are considered in this analysis, the anchorType and the gtEncode fields are not explicitly shown in Figure 5.11. For the rest of this discussion, anchorType is linear and the gtEncode is asymmetric_linear. In the figure, the groundTruthType and predictionType fields jointly determine which mAP calculation scenario is being used.

Consider the mAP calculation scenario where both the groundTruthType and the predictionType are bounding boxes. In this scenario, the octa_lin_lin_lst (octagonal mask predicting network with only the last layer fine-tuned) achieves the best mAP score of 22.09 ± 0.03. This is closely followed by bbox_lin_lin_no (bounding box predicting network) by achieving an mAP score of 21.94 ± 0.34. Next comes the octa_lin_lin_all (octagonal mask predicting network with all the layers fine-tuned) by achieving an mAP score of 21.73 ± 0.11. The octa_lin_lin_no (octagonal mask predicting network with no fine-tuning) comes in last with an mAP value of 15.49 ± 0.23.

Consider the mAP calculation scenario where the groundTruthType is instance mask and the predictionType can be a bounding box or an octagonal mask. In this scenario, the highest mAP of 15.64 ± 0.07
is achieved by octa_lin_lin_lst (with octagonal predictions). This is closely followed by octa_lin_lin_all with an mAP of 15.41 ± 0.03 (with octagonal predictions). Next comes octa_lin_lin_lst (with bounding boxes extracted from the octagonal masks). It achieves an mAP score of 13.65 ± 0.04. This is then followed by bbox_lin_lin_no with an mAP score of 13.52 ± 0.22. Next comes, octa_lin_lin_all (with bounding boxes extracted from the octagonal masks). It achieves an mAP of 13.31 ± 0.05. This is then followed by octa_lin_lin_no (with octagonal predictions). It achieves an mAP of 10.50 ± 0.12. octa_lin_lin_no (with bounding boxes extracted from the octagonal masks) has the least mAP score. It achieves an mAP score of 8.90 ± 0.11.

**Non-Linear candidate mAP analysis**

Since only non-linear candidates are considered in this analysis, the anchorType and the gtEncode fields are not explicitly shown in Figure 5.12. For the rest of this discussion, anchorType is log and the gtEncode is asymmetric_log. In the figure, the groundTruthType and predictionType fields jointly determine which mAP calculation scenario is being used.

Figure 5.12: Flow-diagram illustrating the dependency of mAP on the non-linear candidate network placeholder values for experiment 5. Only placeholders with unique values are indicated in the figure to reduce clutter.

Consider the mAP calculation scenario where both the groundTruthType and the predictionType are bounding boxes. In this scenario, the octa_log_log_all (octagonal mask predicting network with all the layers fine-tuned) achieves the best mAP score of 23.16 ± 0.10. Next comes the bbox_log_log_no (bounding box predicting network) by achieving an mAP score of 21.79 ± 0.24. This is followed by the octa_log_log_lst (octagonal mask predicting network with only the last layer fine-tuned) with an mAP score of 21.65 ± 0.03. The octa_log_log_no (octagonal mask predicting network with no fine-tuning) has the least mAP value of 17.23 ± 0.31.
Consider the mAP calculation scenario where the \textit{groundTruthType} is instance mask and the \textit{predictionType} can be a bounding box or an octagonal mask. In this scenario, The highest mAP of 16.81 ± 0.18 is achieved by octa\_log\_log\_all (with octagonal predictions). This is followed by octa\_log\_log\_lst with an mAP of 15.57 ± 0.01 (with octagonal predictions). Next comes octa\_lin\_lin\_all (with bounding boxes extracted from the octagonal masks). It achieves an mAP score of 14.66 ± 0.19. This is then followed by bbox\_log\_log\_no with an mAP score of 13.70 ± 0.12. Next comes, octa\_log\_log\_lst (with bounding boxes extracted from the octagonal masks). It achieves an mAP of 13.63 ± 0.03. This is then followed by octa\_log\_log\_no (with octagonal predictions). It achieves an mAP of 12.13 ± 0.30. octa\_log\_log\_no (with bounding boxes extracted from the octagonal masks) has the least mAP score. It achieves an mAP score of 10.41 ± 0.31.

In summary, transfer-learning considerably improves the performance of the octagonal mask predicting network even when the boundary adhering object instances are considered during the training. This provides evidence to support the argument that the seventh hypothesis is True.

\textbf{Hypothesis 7:}

\textit{Loading the trained weights of a bounding box predicting network into an octagonal mask predicting network and fine-tuning it, helps improve the performance of the octagonal mask predicting network.}

In the linear candidate mAP analysis, the octa\_lin\_lin\_lst consistently but marginally outperforms octa\_lin\_lin\_all in all the mAP calculation scenarios.

- In the mAP calculation scenario where the \textit{groundTruthType} and \textit{predictionType} are both bounding boxes, octa\_lin\_lin\_lst has an mAP value 0.36 point higher than that of the octa\_lin\_lin\_all.
- In the mAP calculation scenario where the \textit{groundTruthType} is instance mask and \textit{predictionType} is octagonal mask, octa\_lin\_lin\_lst has an mAP value 0.23 point higher than that of the octa\_lin\_lin\_all.
- In the mAP calculation scenario where the \textit{groundTruthType} is instance mask and \textit{predictionType} is bounding box, octa\_lin\_lin\_lst has an mAP value 0.34 point higher than that of the octa\_lin\_lin\_all.

So on an average, octa\_lin\_lin\_lst leads the octa\_lin\_lin\_all by ≈ 0.31 mAP.

In the non-linear candidate mAP analysis, the octa\_log\_log\_all consistently outperforms octa\_log\_log\_lst in all the mAP calculation scenarios.

- In the mAP calculation scenario where the \textit{groundTruthType} and \textit{predictionType} are both bounding boxes, octa\_log\_log\_all has an mAP value 1.51 point higher than that of the octa\_log\_log\_lst.
- In the mAP calculation scenario where the \textit{groundTruthType} is instance mask and \textit{predictionType} is octagonal mask, octa\_log\_log\_all has an mAP value 1.24 point higher than that of the octa\_log\_log\_lst.
- In the mAP calculation scenario where the \textit{groundTruthType} is instance mask and \textit{predictionType} is bounding box, octa\_log\_log\_all has an mAP value 1.03 point higher than that of the octa\_log\_log\_lst.
Chapter 5. Results and Evaluation

So on an average, octa\_log\_log\_all leads the octa\_log\_log\_all by \( \approx 1.26 \) mAP.

When using log anchors, fine-tuning all the layers results in higher performance improvement as compared to fine-tuning just the last layer. On the other hand, when using linear anchors, fine-tuning only the last layer results in higher performance improvement as compared to fine-tuning all the layers of the network.

Hence the eighth hypothesis holds good when using log anchors and does not hold good when using linear anchors.

**Hypothesis 8:**

Fine-tuning all the layers of the octagonal mask predicting network should provided marginally better performance than fine-tuning only the last layer of the network.

In all the mAP calculation scenarios, the fine-tuned octagonal mask predicting networks seem to outperform the bounding box predicting networks. This provides evidence to support the argument that the ninth hypothesis is True.

**Hypothesis 9:**

The object segmentation performance of an octagonal masks predicting network, which has been fine-tuned starting from the trained weights of a bounding box predicting network, is at least as good as that of the bounding box predicting network.

5.5.2 Run-time Analysis

From the subplots in Figure 5.13, it can be seen that the latency of one forward pass through all the candidate networks are close to each other. The networks octa\_log\_log\_all, octa\_log\_log\_lst, octa\_lin\_lin\_all, octa\_lin\_lin\_lst, octa\_log\_log\_no and octa\_lin\_lin\_no have forward pass latencies of 19.21 ± 7.15 ms, 19.01 ± 6.94 ms, 18.62 ± 0.59 ms, 18.48 ± 0.57 ms, 18.55 ± 0.56 ms and 18.45 ± 0.56 ms respectively.

Since the encoding/decoding schemes vary across the different candidate networks, the interpretation times might vary between them. In the subplots in Figure 5.13, octa\_lin\_lin\_lst has the highest interpretation time of 19.49 ± 42.79 ms. The interpretation times of octa\_log\_log\_all, octa\_lin\_lin\_all, octa\_log\_log\_lst, octa\_log\_log\_no and octa\_lin\_lin\_no are 15.98 ± 31.15 ms, 15.45 ± 28.80 ms and 12.75 ± 2.26 ms, 17.27 ± 35.34 ms and 17.54 ± 37.54 ms respectively. The Filtration times are negligible and the NMS times are highly dependent on the number of predictions passed into the NMS stage.
Figure 5.13: Sun-burst plots representing the run-time comparison of the different candidate networks.
Conclusions

This work successfully demonstrated that if given sufficient fine instance annotated data (i.e., instance masks), an existing object detection network can be modified to predict much finer approximations (i.e., irregular octagons) of the instance annotations. The resulting network performs (metric is mAP) not worse than the bounding box predicting network, with only a marginally increase in the run-time (metric is time in ms). For reliable and unbiased performance comparison between the bounding box predicting network and the octagonal mask predicting network, different mAP calculation scenarios were devised. Initial experiments with the octagonal mask predicting network indicated that its performance lagged behind that of the bounding box predicting network by a significant amount. Further analysis revealed that the reason for this lackluster performance was the sparsity of the dataset used for the evaluation. To mitigate the issue of data-shortage, this work proposed a transfer-learning inspired simple-to-complex training regime for the octagonal mask predicting network. In this regime, the octagonal mask predicting network is first loaded with the trained weights of the bounding box predicting network and then fine-tuned on the same dataset. This resulted in an octagonal mask predicting network (all layers fine-tuned) whose performance not only matched that of the bounding box predicting network but exceeded it by,

1. $\approx 0.77$ mAP, on the mAP calculation scenario where both the ground-truth annotations and the predictions are bounding boxes.

2. $\approx 0.57$ mAP, on the mAP calculation scenario where the ground-truth annotations are instance masks and the predictions, are bounding boxes.

This justifies the inclusion of the underlined word “sufficient” in the opening statement.

This work is the first to identify the discrepancy between the way the prior anchors are generated and the center-offset ground-truth encoding/decoding employed by most of the anchor-based object detection networks. The work also proposed a way to settle this discrepancy by performing clustering in a space defined by the coordinate basis, obtained by taking the natural log transformation of the width and the height of the ground-truth bounding boxes. The advantage of this was validated empirically by,

1. $\approx 1.85$ mAP improvement in performance, on the mAP calculation scenario where both the ground-truth annotations and the predictions are bounding boxes.
2. ≈ 1.60 mAP improvement in performance, on the mAP calculation scenario where the ground-truth annotations are instance masks and the predictions are bounding boxes.

This work is also the first to shed light on the issues posed by the image boundary adhering object instances. Towards this end, an adaptive mechanism for automatic detection and handling of these problematic object instances was proposed in this work. As a part of this mechanism, first, a decoupled bounding box parameterization was used to ensure that all the boundaries of the bounding box are independently represented by a single parameter. Next, two new ground-truth encoding/decoding schemes were introduced (refer section 4.3.2 for more details) which are more suitable for the boundary adhesion considerations. Finally, a modified loss function was proposed which facilitated selective learning of, only partial untainted parameters of the problematic boundary adhering object instances. The merits of boundary adhesion considerations were validated empirically by the following improvements in the performance.

- For the bounding box predicting network, boundary adhesion consideration resulted in improvements of,
  1. ≈ 0.62 mAP (log anchors) and ≈ 2.62 mAP (linear anchors), on the mAP calculation scenario where both the ground-truth annotations and the predictions are bounding boxes.
  2. ≈ 0.50 mAP (log anchors) and ≈ 1.92 mAP (linear anchors), on the mAP calculation scenario where the ground-truth annotations are instance masks and the predictions are bounding boxes.

- For the octagonal mask predicting network (log anchors), boundary adhesion consideration resulted in improvements of,
  1. ≈ 2.25 mAP, on the mAP calculation scenario where both the ground-truth annotations and the predictions are bounding boxes.
  2. ≈ 1.67 mAP, on the mAP calculation scenario where the ground-truth annotations are instance masks and the predictions are bounding boxes.
  3. ≈ 1.82 mAP, on the mAP calculation scenario where the ground-truth annotations are instance masks and the predictions are octagonal masks.

A birds-eye view of all the experimental results (mAP) is presented in the Figure 6.1. The forward latencies for all the experiments are all close to 19 ms.
Chapter 6. Conclusions

Figure 6.1: Summary of all experiment conducted as a part of this work.
6.1 Contributions

The following contributions were made as a part of this work,

1. Clustering in a coordinate space where the basis represents the natural log transformations of the width and the height of the ground-truth bounding boxes was proposed in this work as a means to find superior prior anchor boxes for anchor-based object detectors.

2. A proof-of-concept was developed by augmenting an existing anchor-based object detection network (SqueezeDet by Wu et al. [62]) to predict the parameters of irregular octagonal approximations of the instance masks.

3. A transfer-learning inspired simple-to-complex training regime was introduced as a means to boost the learning capability of the octagonal mask predicting network. This can be particularly helpful in use-cases where the availability of annotated data is sparse.

4. This work uncovered the issues posed by boundary adhering object instances and why they warrant separate handling. Towards this end, a mechanism for automatic handling of these problematic object instances during network training was proposed. This involves,
   - Introduction of a robust mechanism for automatic identification of problematic image border adhering object instances, compatible with the data-augmentation strategies like random horizontal flipping and horizontal and vertical image translation and image cropping.
   - Introduction of alternate ground-truth encoding/decoding schemes which are better suited for encoding/decoding the decoupled bounding box parameterization. In decoupled bounding box parameterization, each border of the bounding box is independently parameterized i.e., each bounding box is represented by the quartet of $x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}}$ instead of the usual center-coordinates $(c_x, c_y)$ and width (w) and height (h).
   - Introduction of a modified L2 loss function for regression. This loss acts like a normal L2 loss for the object instances which are not in contact with the image boundaries. However, for the problematic image border adhering object instances, it enables only selective learning of the partial untainted parameters.

5. The entire code-base and the trained checkpoints are made openly available\(^1\).

6.2 Lessons learned

The following are some of the important lessons learned during this work.

1. **Importance of checking dataset for errors:** It is not uncommon to find corrupt images in standard and widely adopted datasets. Many times the training is done by randomly sampling a batch from the dataset and passing through the network. There is a high possibility that a corrupt image might be batched in, after many days of training has already elapsed. If there is no checking

\(^1\)https://github.com/DRealArun/squeezeDet
and handling of corrupt images in the code then it might result in a lot of wasted computation. Hence it is always a good practice to do a preliminary check to find if there are corrupt data samples. If they are present then they can be discarded physically or they can be ignored in the code.

2. **Importance of local testing before deploying to cluster:** The computing time on the cluster is extremely valuable. Sometimes minor bugs in the code can cause the scheduled jobs to fail thus wasting days of computation. It is always a good practice to test the implementation locally (possibly on a small partition of the dataset, when the entire dataset is too big to be run locally) before scheduling the job on the cluster.

### 6.3 Limitation

The following are the limitations of this work,

1. **Octagonal masks and its associated limitations:**
   - **Overlapping octagonal masks:** The intention behind using the bounding octagons in place of bounding boxes was to better delineate between different object instances especially in cluttered environments. Though the use of octagonal masks reduced the number of pixels being labeled as belonging to multiple object instances, some overlap between the octagonal masks still exists. This is illustrated in the Figure 6.2.

   ![Figure 6.2](image)

   **Figure 6.2:** Octagonal mask predictions overlapping each other in a cluttered environment.

   - **Concave objects are approximated by convex shapes:** Using an octagonal approximation of the instance masks inherently converts the concave object instances in the scene to convex objects, which might lead to further overlapping masks.

   - **Multi-segment masks are combined into a single whole mask:** Occluded objects (e.g., a car partially occluded by a tree trunk) can have masks that have multiple segments. The choice
of using an octagon to approximate instance masks limits the ability to represent multi-segment masks.

2. **Boundary adhesion considerations and its associated limitations:**

   - **Computational overhead for boundary adhesion considerations:** The decoding involved in the anchor-offset linear/non-linear encoding/decoding schemes proposed as a part of this work involves more number of operations than that in the traditional center-offset encoding/decoding scheme. As a result, the network with boundary adhesion considerations will have marginally larger run-time than the network without boundary adhesion considerations. This limitation is in some way balanced by the better performance of the network with boundary adhesion considerations than the one without.

   - **Distorted masks at image boundaries:** Boundary adhesion considerations particularly for the octagonal mask predicting networks might lead to distorted predictions at the image boundaries. Consider Figure 6.3 for an illustration of the distorted masks. The probable reason for this distortion is the zero-padding of the image performed by TensorFlow, when it is fed into the network during training. However, due to limited time, this could not be verified. This can be considered for future work.

![Figure 6.3: Left: Distorted mask predictions. Right: Post-process correction. Notice the knot and the associated correction indicated by ellipse](image)

### 6.4 Future work

The following are the possible directions worth exploring in the future.

1. Extend the boundary adhesion conscious training for state-of-the-art object-detection and instance segmentation techniques, on challenging datasets like MS COCO (Lin et al. [29]), The Mapillary Vistas Dataset (Neuhold et al. [39]), etc. This can validate if the merits of boundary adhesion consideration, highlighted in this work, are also seen when applied to other datasets.
2. Exploring alternate mask representations which are suitable to be predicted by the network. To some extent, wavelet transforms for contour encoding were visited however due to time constraints, it could not be investigated in detail.

3. Active contour modeling is another attractive avenue worth visiting, for regression-based instance segmentation.

4. Different variations of the loss functions can be explored for selectively learning only untainted parameters of the boundary adhering object instances.

5. Investigating alternative padding strategies to mitigate the distorted predictions at the image boundaries, when the network is trained under boundary adhesion considerations.
6.4. Future work
Boundary analysis

Figure A.1 provides the boundary margin statistics of the Cityscape dataset.

Figure A.1: Margin vs image sample number in Cityscape dataset.
Figure A.2: Illustrations for the bottom margin values.

From Figure A.1, it is clear that except for the bottom margin, all the other margins are constant. The bottom margin switches between 5 pixels and 45 pixels (with one occurrence of 46 pixels).

For the handling of the boundary adhering object instances, there is a need to decide on one single value for the bottom margin. Towards this end,

1. We find the number of images that have bottom margins greater than 5.

2. In the images which have bottom margins > 5, we extract a bounding box for each ground-truth instance mask and then try to find how many ground-truth bounding-boxes have their $y_{\text{max}}$ (i.e., $y$ coordinate of the bottom boundary) in the range $(H-5, H-46)$ (where $H$ is the height of the image in Cityscape dataset, which is 1024). Specifically, we try to find the number of ground-truth bounding-boxes having their $y_{\text{max}}$ in the range $[1018, 977]$.

From the analysis of the Cityscape dataset, it was found that,

1. There are 1098 images in the training set with bottom margins greater than 5. In these 1098 images, there are 6 ground-truth annotations with their $y_{\text{max}}$ values in the range $[1018, 977]$.

2. Similarly, there are 267 images in the validation set with bottom margins greater than 5. In these 267 images, there are 3 ground-truth annotations with their $y_{\text{max}}$ values in the range $[1018, 977]$.

From the above observations, it can be concluded that setting the bottom margin to 5 would result in 9 ground-truth annotations to be falsely classified as non-boundary adhering instances. This is an extremely small number and hence should not have a significant impact. Hence we reasonably set the bottom margin to 5 for all the experiments.
Interpreting the plots

B.1 Parallel categorical plots

For the experiments performed as a part of this work, it was necessary to illustrate the dependency of the mAP metric value on,

1. the mAP calculation scenario

2. the placeholder values defining the candidate networks used in a particular experiment.

Parallel categorical plots are typically used to visualize multi-dimensional categorical datasets and this makes them ideal for the above-mentioned use-case. This section tries to provide a brief primer of how to read these parallel categorical plots from the context of this work. For this, the plot from the mAP Analysis of experiment 3 is chosen, as it appears visually complex.

Figure B.1: Parallel categorical plot explained.
In the above plot, the \(mAP\) column represents the \(mAP\) value achieved by a particular network on a particular \(mAP\) calculation scenario. The columns \(predictionType\) and \(groundTruthType\) jointly determine what \(mAP\) calculation scenario is being used. If both \(groundTruthType\) and \(predictionType\) are \textbf{bounding box} then it is the \(mAP\)-(Box)x(Box) calculation scenario. If the \(groundTruthType\) is \textbf{mask} and the \(predictionType\) is \textbf{bounding box} then it is the \(mAP\)-(Mask)x(Box) calculation scenario. Finally, if the \(groundTruthType\) is \textbf{mask} and the \(predictionType\) is \textbf{octagon} then it is the \(mAP\)-(Mask)x(Octagon) calculation scenario. \(paramSize\) determines if the network under question is a bounding box (value = 4) predicting network (SqueezeDet) or an octagonal (value = 8) mask predicting network (SqueezeDetOcta). \(fineTuneMode\) determines what transfer-learning setting is used to train the network. \textbf{fine_all} value indicates that all the layers of the network are fine-tuned, \textbf{fine_last} indicates that only the last layer of the network is fine-tuned and \textbf{disable} means the network is trained from scratch. In this experiment, all the candidate networks use logarithmically extracted anchors and the center-offset encoding/decoding scheme, and hence the \(anchorType\) and \(gtEncode\) placeholders are not shown in the plot. There is one flow-line moving from left to right for each candidate network whose \(mAP\) is being measured on a particular \(mAP\) calculation scenario. For this explanation, consider the most complex portion of the plot where there are multiple flow-lines overlapping each other (marked by the red ellipse). The steps to decode the values in the plot are as mentioned below,

1. Start from the right-most column i.e., the \(mAP\), and select the flow-line for a particular value of \(mAP\) which needs to be decoded.

2. Follow the flow-line to the previous column and record the value for that column. If there is excessive overlap between the flow-lines at a particular junction, the shaded borders of the flow-line can be used as a guide to get through the junction.

3. Repeat this till the left-most column is reached.

4. The recorded values at each column describe the candidate network and the \(mAP\) calculation scenario used to get the selected \(mAP\) value.

For instance, let us consider, we want to find what combination of candidate network and the \(mAP\) calculation scenario resulted in an \(mAP\) value of 13.77 ± 0.18. At the junction, there are several flow-lines overlapping each other. The shaded borders on the flow-line for 13.77 ± 0.18 indicates that the \(groundTruthType\) is \textbf{mask}. Moving towards the left, it can be seen that at \(predictionType\) column, the value is \textbf{bounding box}. Moving further towards the left, the value at \(fineTuneMode\) column is \textbf{fine_all}, and finally, the value at the column \(paramSize\) is 8. The white dashed line illustrates the traced flow-line. Decoding all these values provides the following information,

1. The candidate network was an octagonal mask predicting network (SqueezeDetOcta) that was trained by fine-tuning all of its layers.

2. The \(mAP\) calculation scenario was \(mAP\)-(Mask)x(Box).
Appendix B. Interpreting the plots

Let us consider another example where we want to find what combination of candidate network and the mAP calculation scenario resulted in an mAP value of 13.20 ± 0.10. By concentrating on shaded borders on the flow-line for 13.20 ± 0.10, it can be seen that it passes through the value **mask** on the column groundTruthType. Moving on to predictionType column, it can be easily seen that the value is **bounding_box**. Moving further left to the column fineTuneMode indicates that the value the flow-line passes through is **disable**. Moving towards the column paramSize a heavily congested junction is encountered. However, following the flow-line by using its shaded borders as a guide indicates that it passes through the value of 4 at the paramSize. The white dashed line illustrates the traced flow-line. Finally, decoding all these values provides the following information,

1. The candidate network was a bounding box predicting network (**SqueezeDet**) that was trained from scratch.

2. The mAP calculation scenario was mAP-(Mask)x(Box).

Similarly, the other flow-lines can be easily decoded.

### B.2 Sun-burst plots

![Sun-burst plot explained](image)

**Figure B.2:** Sun-burst plot explained.

Interpreting sun-burst plots is relatively straight forward.
1. The entire plot is primarily partitioned into two circular regions.

2. The inner most circular region is further partitioned into three sectors. Each sector corresponds to one experimental candidate network and the value specified in the sector is the total inference time of that experimental candidate.

3. Every sector in the inner circular region is associated with 4 sectors in the outer circular region. These sectors represent the time shares of the different stages in the inference pipeline (FP: Forward-pass time, FT: Filtration time, IT: Interpretation time, NMS: Non Maximum Suppression time). The summation of these 4 time shares should be equal to the total time mentioned in the inner sector.

4. The numerical values in each sector represent the time in the format $\mu_t \pm \sigma_t$ where $\mu_t$ is the mean time in milliseconds and $\sigma_t$ is the standard deviation in milliseconds.

Now let us extract the run-time information from the Figure B.2.

- The sector labeled bbox_log_nor_no represents one candidate network. The numerical values below this label represent the total inference time. The mean inference time is 124.75 ms and the standard deviation is 87.62 ms. The time shares of the different stages are,
  - **Forward-pass time**: mean time = 18.62 ms and standard deviation = 1.76 ms
  - **Interpretation time**: mean time = 10.84 ms and standard deviation = 2.00 ms
  - **Filtration time**: mean time = 0.34 ms and standard deviation = 0.47 ms
  - **Non Maximum Suppression time**: mean time = 94.95 ms and standard deviation = 87.40 ms

- The sector labeled bbox_lin_nor_no represents one candidate network. The numerical values below this label represent the total inference time. The mean inference time is 107.16 ms and the standard deviation is 72.66 ms. The time shares of the different stages are,
  - **Forward-pass time**: mean time = 18.40 ms and standard deviation = 0.59 ms
  - **Interpretation time**: mean time = 16.77 ms and standard deviation = 38.10 ms
  - **Filtration time**: mean time = 0.34 ms and standard deviation = 0.48 ms
  - **Non Maximum Suppression time**: mean time = 71.65 ms and standard deviation = 60.83 ms

- The sector labeled octa_log_nor_no represents one candidate network. The numerical values below this label represent the total inference time. The mean inference time is 181.56 ms and the standard deviation is 128.74 ms. The time shares of the different stages are,
  - **Forward-pass time**: mean time = 18.73 ms and standard deviation = 3.72 ms
  - **Interpretation time**: mean time = 13.61 ms and standard deviation = 14.14 ms
  - **Filtration time**: mean time = 0.45 ms and standard deviation = 0.51 ms
  - **Non Maximum Suppression time**: mean time = 148.77 ms and standard deviation = 126.93 ms
Qualitative results

Following are some of the qualitative results for some of the samples from Stuttgart in the Cityscape dataset. They represent images collected using the same camera setup and under similar lighting conditions as the dataset used for training.

Following are some of the qualitative results for some of the samples from New York. These samples are collected using a different camera setup and under vastly different lighting conditions. The trained
network make a lot of overly optimistic predictions but it seems to generalize quite well to the classes.
References


References


[38] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan Yuille. The role of context for object detection and semantic segmentation in the wild. 06 2013. doi: 10.13140/2.1.2577.6000.


References


