A Staged Approach to Training Mask R-CNN on Cityscapes with Limited Resources

Ben Sebastian Attias
bensebastian.attias@studenti.unipd.it

Timur Oner
timur.oner@studenti.unipd.it

Abstract

Instance segmentation is a popular computer vision task that involves detecting objects and creating masks outlining each detected object. This is a challenging and computationally demanding task as creating correct masks for objects involves very precise localization. Applications range from medical imaging to autonomous vehicles. In this work we demonstrate our approach to training Mask R-CNN with limited computing resources using a staged approach on a particularly challenging dataset that has high relevance for self-driving cars: the Cityscapes dataset. Cityscapes includes various urban sceneries with high quality instance level annotations across streets of German cities. We show a respectable performance of our Mask R-CNN implementation, discuss challenging aspects of working with Cityscapes dataset and describe approaches we took to manage working with huge size of Cityscapes data with limited compute and time resources.

1. Introduction and Background

Instance segmentation has become one of the most relevant and challenging tasks in computer vision research. It has notable applications ranging from robotics and self-driving cars to medical devices [7]. Microsoft COCO and Cityscapes are among the most popular general-purpose datasets used as a benchmark for instance segmentation tasks. A wide range of architectures were proposed to tackle the instance segmentation task ranging from proposal based networks (R-CNN, Mask R-CNN), proposal-free approaches (SOLOV2, PolarMask) to reinforcement learning based methods [16]

1.1. The Instance Segmentation Task

Instance segmentation is one of the most widely used tasks in computer vision. In a sense, it combines the semantic segmentation and object detection task. In semantic segmentation tasks, we classify each pixel according to the semantic category it belongs (e.g. car or pedestrian) but we do not worry about differentiating different instances of the same semantic class. On the other hand, in object detection task we do no care about the classes of individual pixels, we only try to separate each instance of each class and localize each instance using a bounding box. In the instance segmentation task, we try to both detach each instance of a class and create a pixel-wise mask to precisely highlight the detected object. Because of this, in a way instance segmentation task is harder than object detection and semantic segmentation tasks. In addition to this, panoptic segmentation combines instance segmentation and semantic segmentation but is outside of the scope of this work [9].

1.2. Performance Measures and Loss Functions

1.2.1 Mean Average Precision (mAP)

Mean Average Precision (mAP) measures instance segmentation performance by combining precision and recall. A prediction is considered correct only if it matches the ground truth class, sufficiently overlaps with the ground truth mask (IoU above a threshold), and has the highest confidence among overlapping predictions.

$$\mathrm{mAP} = \frac{1}{C}\sum_{c=1}^{C}\mathrm{AP}_c, \quad \mathrm{AP}_c = \int_0^1 P_c(R)dR \qquad (1)$$

Predictions are sorted by confidence to compute precision and recall at each level, forming a precision-recall curve. The area under this curve gives Average Precision (AP), which is averaged over multiple IoU thresholds and then across all classes to obtain the final mean Average Precision (mAP). The forumla to calculate mAP is shown in (1).

1.2.2 IoU and GIoU

IoU (Intersection over Union) is the go-to measure to quantify how well the predicted and ground truth masks (or bounding boxes) overlap.The formula to compute IoU is written in 2. IoU can take values between 0 (no overlap)

and 1 (perfect overlap).

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} = \frac{A_{\text{intersection}}}{A_{\text{union}}} \quad (2)$$

[15] proposed an alternative way to measure intersection between areas: Generalized IOU or GIOU. GIOU solves a shortcoming of IOU: IOU becomes 0 and keeps being 0 once the 2 areas aren't overlapping. This introduces a plateau in the loss function when we try to optimize the model weights w.r.t. bounding box and mask locations. GIOU solves this by taking into the consideration the distance between the areas. According to the equation 3, C is the smallest convex object that encloses both A and B: the areas for which GIOU is calculated. GIOU can take values from -1 to +1, +1 meaning a perfect overlap and values below 0 meaning no overlap. More negative values signify that areas that are further apart.

$$GIoU = IoU - \frac{|C \backslash (A \cup B)|}{|C|} \quad (3)$$

2. Mask R-CNN and Related Architectures

2.1. Residual networks and the Resnet50 backbone

Residual networks are frequently used as backbone networks for regional proposal based instance and object detection architectures. The backbone network is used to extract semantically rich features from the processed image. Without the backbone network the Region Proposal Network would lack semantically rich representations and it would be much harder to make good proposals. ResNet-50 and ResNet-101 are among the most popular choices for the backbone network of Mask R-CNN.

2.2. Feature Pyramid Networks (FPN)

FPNs [11] are a key component of Mask R-CNN, operating as a backbone feature extractor [8]. The framework allows for the generation of feature pyramids easily from existing ConvNet architectures. FPNs enable the simultaneous generation of multi-scale feature maps, significantly improving detection performance for images containing objects of widely varying sizes.

FPN's leverage the pyramidal structure of ConvNets to produce feature maps at different scales that all contain high-level semantic information. FPNs perform a backwards pass upsampling the representation at each layer starting from the lowest resolution layer and up to the highest resolution level. Upsampled maps are merged with lateral connections from the backbone, yielding a pyramid of semantically rich feature maps across multiple resolutions.

2.3. Region-based CNNs and Faster R-CNN

Region Based CNN [6] was the first architecture to introduce the idea of combining region proposals with CNN features. The initial implementation required a CNN pass for each of the thousands of proposals. This was very computationally demanding and as an improvement Fast R-CNN [5] was introduced with RoI pooling and end-to-end training. Faster R-CNN [14] increased the speed further by eliminating the need for external region proposals and introducing the novel Regional Proposal Network (RPN). This shares convolutional features and directly predicts and classifies region proposals by refining predetermined anchor boxes of various scales and aspect ratios. RPNs improved both speed and accuracy and laid the groundwork for modern instance segmentation models.

2.3.1 The Region Proposal Network (RPN)

The RPN takes as input an image of any dimensions and outputs a set of rectangular proposals for object locations with an "objectness" score representing whether the content of the patch belongs to any object (termed as the foreground), or the background. The proposals are generated by sliding a small network (3×3 in the original paper) across all levels of the hierarchical FPN map. The results are fed into 2 sibling fully connected layers: one of them performs box regression to refine the coordinates of each proposal, and the other performs box classification to predict the "objectness" score. These proposals are generated from predefined anchors and filtered using thresholding operations, including Non-Maximum Suppression and confidence score thresholds, to remove unlikely or redundant boxes.

2.4. Mask R-CNN

Mask R-CNN [8] is built on Faster R-CNN by retaining its two-stage design, first generating candidate regions via the RPN, then applying classification and fine box regression heads. It extended this architecture with a third branch: a small fully convolutional network (FCN) that predicts an $m \times m$ segmentation mask for each RoI.

The key change enabling pixel-accurate masks was RoIAlign, which replaces RoIPool to avoid quantization misalignments by using bilinear interpolation. Without this change extracted features would not precisely align with RoIs.

2.4.1 RoI Align instead of RoIPool

RoIPool [5] was the standard approach for generating small feature maps with a fixed spatial dimension of $H \times W$ (typically 7×7). This method involves first quantizing the region of interest (RoI) by dividing the coordinate space by the feature map stride and then rounding to the nearest grid point. Subsequently, each RoI is subdivided into spatial bins, and features within each bin are aggregated, commonly via max pooling. Although this quantization process has little impact on classification accuracy due to its robustness to minor

translations, it can adversely affect the precise prediction of pixel-level masks.

RoIAlign [8] improves upon this approach by sampling four fixed points within each spatial bin of the RoI. Instead of rounding coordinates, exact floating-point coordinate values are retained, and bilinear interpolation is employed to estimate the feature value at each sampling point. The features sampled from these points are then combined, either through max or average pooling, to produce a representative feature vector for each bin.

2.4.2 Classification and Mask Prediction Heads

The heads themselves function as follows. Taking the feature maps produced by RoIAlign that correspond to each RoI from the RPN, the classification head will perform a standard classification task to identify which class of object is present inside the RoI (argmax among the classes), box regression is performed to adjust the exact position and size of the bounding box and the mask head produces a pixel-level segmentation mask for identifying which pixels in the RoI correspond to the object.

Both box regression and mask generation are class specific, i.e. the model produces a separate output for each possible class. During training only the loss for the ground truth box or mask are propagated back and during inference once a class is decided by the classifier head only the corresponding box and mask are retained. An important thing to consider is that initially the mask heads compute the masks for each of the possible class, but after the classification is being made only the mask of the predicted class is kept and the rest of the mask predictions are discarded.

2.4.3 Classification, Box and Mask Loss

Mask R-CNN [8] uses a multi-task loss function 4 defined on each RoI.

$$L = L_{cls} + L_{box} + L_{mask} \tag{4}$$

Where, L_{cls} is the classification loss, L_{box} is the box loss as defined in [5] i.e. log loss and L_1 loss respectively. L_{mask} is the loss defined by the average binary cross entropy loss between the predicted mask and the ground truth.

3. Dataset

We use the Cityscapes dataset [4], a large-scale dataset that contains diverse urban scenes from various German cities. It consists of more than 3000 high-quality images (2048px by 1024px) with instances of objects that are commonly encountered during driving (pedestrians, cars etc.) and their respective instance annotations and pixel-level instance segmentation masks. There are also more than 20000 semantically annotated images that can be used for semantic annotation and pre-training tasks.

For our experiments, 2,975 finely annotated images are used for training and 500 for validation. Of the dataset, 1,525 images are reserved for benchmarking and lack public labels, so they were excluded from training. The coarse annotations from the `trainextra` split were utilized to provide additional semantically annotated samples that we used to pretrain resnet50 backbone that we used in our Mask R-CNN implementation. Due to storage and memory constraints, we used only 4120 coarsely annotated images.

3.1. Statistics of the Cityscapes Dataset

To estimate distribution of ground truth objects in the dataset we counted and statistically analyzed all objects in 1 partition of the training dataset (372 images). The object size and class distributions are shown in table 1 and table 2 respectively. While the size distribution is relatively balanced, there is a significant imbalance in class distributions, cars and persons constituting a large majority of all annotated objects.

Table 1. Size distribution of objects

Category	**Count (%)**
Small objects ($\leq 32\times32$ px)	1302 (23.9%)
Medium objects (32–96 px)	2333 (42.8%)
Large objects ($> 96\times96$ px)	1813 (33.3%)

Table 2. Class distribution of objects

Class	**Count (%)**
car	3177 (58.3%)
person	1630 (29.9%)
bicycle	215 (3.9%)
truck	138 (2.5%)
rider	126 (2.3%)
motorcycle	109 (2.0%)
bus	37 (0.7%)
train	16 (0.3%)

3.2. Challenges of the Dataset

The cityscapes dataset is a particularly challenging dataset to work with. One of the most obvious challenges is that each image is 1920x1080px and high-resolution samples increase training time, making hyperparameter experimentation more time-consuming. Working with large-sized samples constraints batch size as the GPU we used (Nvidia L4 GPU) could handle only up to 4 images per batch during training.

Another unique challenge introduced by cityscapes dataset is large number of small and occluded objects. As

shown in table 1, small objects constitute a significant portion of all annotated objects. Presence of small and occluded objects makes reaching high detection performance significantly more challenging.

Finally, as discussed in the previous section, the dataset exhibits a significant class imbalance, which makes achieving high average precision (AP) for underrepresented classes particularly challenging.

4. Methods

4.1. Dataset Preparation and Preprocessing

The training and validation set consisted total of 3475 RGB images and the corresponding annotations. Each annotation mask contained pixel values encoded according to the following expression:

$$\text{encoded\_pixel} = (\text{class\_id} \times 1000) + \text{instance\_id}$$

Each image was accompanied by an instance mask with same dimensions as the image that contained all of the instance annotations in that image encoded according to the equation presented above. To prepare the dataset for training and validation we converted each instance segmentation mask into a dictionary of class IDs, bounding boxes and masks for each object. Bounding boxes were obtained by simply taking the extreme coordinates of the instance mask of each object in the image.

To make the dataset loading faster to Google Colab Python runtime, we converted all of the images and the corresponding annotation files into safetensor format. This conversion reduced dataset loading times each time we started a new Google Colab session from 10 minutes to about half a minute.

To enable dynamic data loading and deletion during training, we have split the training and validation dataset into 8 and 4 partitions, respectively. For example, a train partition with 372 samples was saved as a safetensor file with dimensions of [372,3,2048,1024] for the images and [372,2048,1024] for the annotations. The first 2 partitions of the validation set were used for hyperparameter selection and validation loss tracking during the training procedure. The last 2 validation partitions were reserved for obtaining the final test scores.

4.2. Training Loop Structure and Optimization

We used a slightly modified loss function that combined the standard multitask loss 4 with the GIoU [15] between prediction boxes and ground truth object boxes. We defined the composite loss in (5). In addition, one way to improve training times and make more efficient use of GPU memory is mixed precision training [12] . This technique uses half-precision (FP16) floating point numbers for weights, activations, and gradients during training, while maintaining a single-precision (FP32) copy of the weights to accumulate gradients during optimization. Furthermore, another computational optimisation was the usage of dynamic data loading. We divided the large training dataset into 8 partitions which were loaded and deleted from GPU memory throughout the training process to be able to train on the whole dataset without running out of RAM assigned by Google Colab.

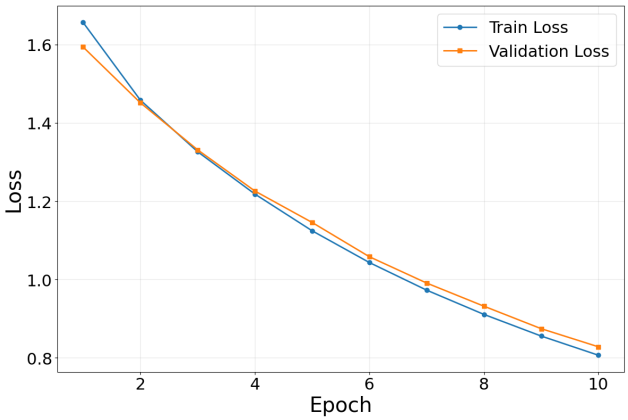

Figure 1. Train/val loss curve for backbone pretraining

$$L_{\text{composite}} = 0.9\, L_{\text{standard}} + 0.1\, L_{\text{GIoU}} \tag{5}$$

5. Experiments

We tested and trained the Mask R-CNN model in 3 different stages, progressively extracting better performances. Before these stages we also pretrained the resnet50 backbone on a subset of the Cityscapes coarsely annotated `trainextra` set, see 5.1. All the training code can be found in [13]. All of the experiments were run on Google Colab with Nvidia L4 GPU.

5.1. Stage 0: Pre-training on Cityscapes

We initialized our backbone with DeepLabv3 weights [3], which provided richer feature representations from prior semantic segmentation training compared to standard ImageNet-pretrained backbones. In out pretraining we used 4120 semantically annotated images from train-extra set provided by Cityscaèes. Pre-training was run for 12 epochs with cross entropy loss and Adam optimizer, with a learning rate of 1e-6. ReduceLROnPlateau was used as a learning rate scheduler. Figure 1 shows the training curve for the backbone pre-training. It can be inferred from the figure that convergence was not reached, however due to computational constraints we stopped the training at a reasonable point. Use of a validation set showed no signs of overfitting.

Parameter Name	Search Space
`fg_iou_thresh`	$[0.3, 0.7]$
`rpn_positive_iou_thresh`	$[0.6, 0.9]$
`rpn_negative_iou_thresh`	$[0.1, 0.4]$
`fpn_grad_flow`	{True, False}

Table 3. Hyperparameters were tuned with Optuna: IoU thresholds sampled continuously, FPN update as a categorical choice.

Best Hyperparameters	
Parameter	Value
FastRCNN IoU threshold	0.415
RPN positive IoU threshold	0.789
RPN negative IoU threshold	0.220
FPN gradient flow	True

Table 4. The optimal hyperparameters found from Optuna tuning trials.

5.2. Stage 1

In the first stage of the training procedure, we have frozen the parameters of the resnet50 backbone and unfroze the parameters of feature pyramid network (FPN), region proposal network (RPN), RoI Align, and the heads (classification, box and mask). We did not perform any detailed fine-tuning during this stage other than trying with a couple different learning rate and weight decay configurations to achieve a stable convergence. We trained Mask R-CNN on the full training dataset (all of 8 training partitions, 2975 samples) for 4 epochs. The main aim of this stage was to get the FPN to adapt the resnet50 backbone features, and the task. We also tried a couple of experiments where we unfreezed the resnet50 backbone too, but doing so caused noticeable over-fitting after only a couple of epochs. Our hypothesis is that the huge parameter count (around 25 million) of the resnet50 backbone is one of the main culprits of this over-fitting phenomenon.

5.3. Stage 2: Hyperparameter Tuning and Training With The New Hyperparameters

We focused on tuning these four hyperparameters with potential to significantly influence region proposal quality and training stability: the foreground IoU threshold, RPN positive/negative IoU thresholds, determine whether a region proposal is classified as "positive" or "negative" in objectivity based on overlap with ground truth (IoU). The importance of IoU thresholds in defining true positives and preventing noisy detections is well documented: too low, and proposals include many false positives; too high, and training suffers from a scarcity of positive examples and inference mismatches [2].

We considered whether to apply gradients to the FPN as part of our stage-wise training regime. Hyperparameters were efficiently explored using Optuna [1] on a reduced subset of the training data, with validation on a separate partition to avoid bias. After 42 trials of 5 epochs each, the optimal hyperparameter configuration was identified and is shown in Table 4.

Another metric provided by the Optuna framework is the importance score for each tuned parameter, which quantifies how much changes to that parameter influence the objective score. For `fpn_flow_grad_flow` this importance score was negligible (see figure 3) so we decided to remain with FPN weights frozen to avoid the computation necessity to train those additional weights, despite the tuning suggesting "True" was the optimal choice.

We adapted anchor sizes for Cityscapes by performing k-means clustering on ground-truth bounding boxes and using the five most representative scales to better detect small and varied objects. With all hyperparameters selected, the model was trained for 6 epochs with the backbone frozen, and the training curves for Stages 1 and 2 are shown in Figure 2.

5.4. Stage 3: The Final Stage of the Training for Refinement

As a final refinement, we unfroze the entire network including the resnet50 backbone and FPN and trained the network on validation set partitions that were used to select hyperparameters in stage 2 and track the validation loss in both stage 1 and stage 2. We lowered the learning rate by a factor of 10 to prevent overfitting. The aim of this stage was to let the components of the model that were trained separately in previous stages to adapt to each other. Since the validation set partitions we have used to get final test scores were not used in any hyperparameter selection or training procedure, we do not expect any noticeable data leakage.

6. Results and Discussion

Table 5 summarizes the final performance of our model after each training stage. Although the improvements in stage 2 were modest, this can be attributed to our limited ability to perform comprehensive parameter tuning. Nevertheless, this stage was essential for the overall training process. As seen from the table, our model does not reach state-of-the-art levels, which is expected given that Mask R-CNN is an older architecture and the scope of this work was not to achieve SOTA performance but to demonstrate good results under computational constraints.

6.1. Challenges and Future Work

A recurrent challenge encountered during this work was limited computational resources. The Cityscapes dataset comprises thousands of high-resolution images (1024×2048 pixels), necessitating extensive work to develop an adequate data loader prior to model training. Additionally,

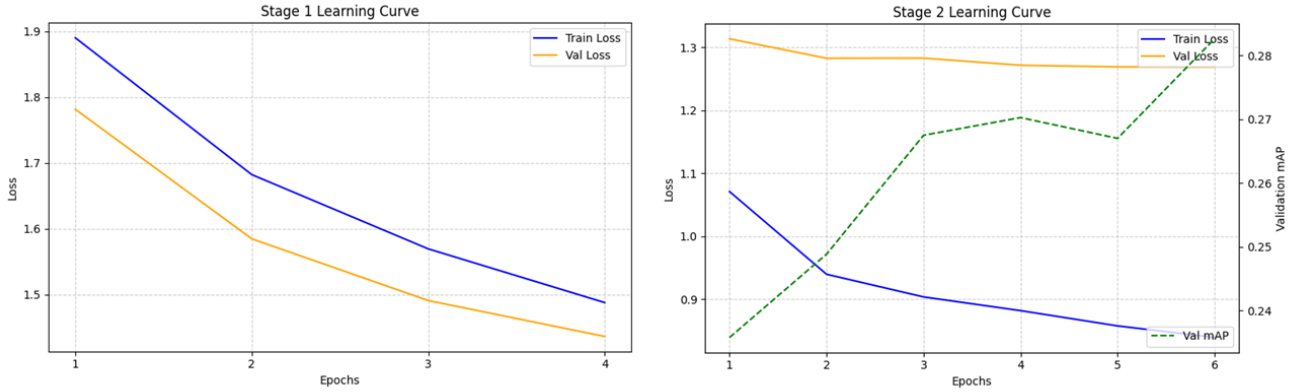

Figure 2. Training and Validation Loss curves for Stage 1 and 2.

Table 5. Evaluation Metrics for Model Comparisons

Model	mAP	mAP@50	mAP [small]	mAP [medium]	mAP [large]	mAR100
Stage 1: DeepLabv3 Feats + Pre-training	0.2706	0.5081	0.0497	0.2063	0.4690	0.376
Stage 2: Hyperparameter tuning	0.2751	0.5037	0.0360	0.2183	0.4665	0.383
Stage 3: The Final Stage	0.2978	0.5290	0.0950	0.2352	0.5033	0.402
HRI-TRANS	0.445	0.714	–	–	–	–
PolyTransform + SegFix + BPR	0.427	0.665	–	–	–	–

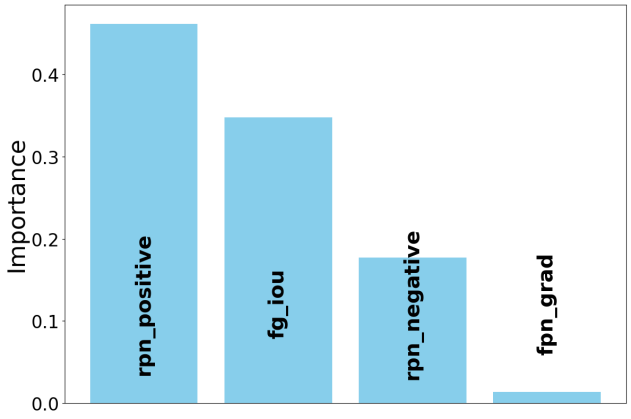

Figure 3. Importance of the tuned parameters to mAP during Optuna trials

instance segmentation is inherently computationally intensive; Mask R-CNN involves numerous complex operations, and the ResNet-50 backbone contains approximately 25 million parameters. Consequently, large-batch training protocols and data augmentation strategies were impractical due to prohibitive computational costs, despite their potential to enhance performance. The primary objective was not to achieve state-of-the-art results but to demonstrate the viability of certain methodological approaches in principle.

Mask R-CNN performing poorly on small objects was also noted in [10]. The authors propose oversampling data augmentation techniques to improve model performance in this area, other works have proposed different architectures that can be beneficial.

7. Conclusion

In this work, we implemented and evaluated Mask R-CNN on the Cityscapes dataset. Despite limited computational resources, our staged training achieved a mean Average Precision of 0.2978, even though below SOTA still acceptable for many applications. The study highlights both the feasibility of training competitive models under constraints.

Working with high-resolution, densely annotated data imposed heavy computational demands, limiting large-batch training and advanced augmentations. In future work, there is a possibility to explore more recent instance segmentation architectures, explore how in addition to training also inference can be performed with limited amount of GPU and storage (e.g. in edge devices) and to explore resource-friendly data augmentation methods to push our results closer to SOTA benchmarks.

Our results show how resource limitations influence model performance and provide insights for developing efficient, practical instance segmentation methods.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 2623–2631, New York, NY, USA, July 2019. Association for Computing Machinery.

[2] Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: Delving into High Quality Object Detection, 2017. Version Number: 1.

[3] Liang-Chieh Chen, G. Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. *ArXiv*, June 2017.

[4] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[5] Ross Girshick. Fast r-CNN.

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

[7] Abdul Mueed Hafiz and Ghulam Mohiuddin Bhat. A survey on instance segmentation: state of the art. *International journal of multimedia information retrieval*, 9(3):171–189, 2020.

[8] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-CNN.

[9] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. Panoptic segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9404–9413, 2019.

[10] Mate Kisantal, Zbigniew Wojna, Jakub Murawski, Jacek Naruniec, and Kyunghyun Cho. Augmentation for small object detection, 2019.

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

[12] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed Precision Training, Feb. 2018. arXiv:1710.03740 [cs].

[13] Timur Oner and Ben Sebastian Attias. Cityscapes instance segmentation with mask r-cnn. `https://github.com/TimurOner/cityscapes_instance_segmentation_maskedrcnn/tree/main`, 2025. Accessed: 2025-08-31.

[14] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-CNN: Towards real-time object detection with region proposal networks.

[15] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 658–666, Long Beach, CA, USA, June 2019. IEEE.

[16] Rabi Sharma, Muhammad Saqib, Chin-Teng Lin, and Michael Blumenstein. A survey on object instance segmentation. *SN Computer Science*, 3(6):499, 2022.