FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Instance segmentation and object tracking accuracy comparison on real and virtual scenarios
Student:	Bc. Adam Simek
Supervisor:	Ing. David Hurych, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2020/21

Instructions

1. Study state of the art machine learning methods for visual instance segmentation and tracking.
2. Pre-process the real dataset, create training, validation and testing splits. Pre-annotate the real data by some publicly available instance segmentation method. In VOSSTREX system model the scenes that correspond to testing set scenarios and render testing set images with various amount of details. Render the fisheye distorted images as well as the corrected ones.
3. Discuss the instance segmentation and object tracking models with respect to the model complexity. Select a suitable subset of methods and implement their training algorithm and inference, or use freely available codes, if possible.
4. Train the models (or just fine-tune) for methods selected in step 3) for instance segmentation and tracking on the training set (real captures). Evaluate segmentation and tracking accuracy (potentially also other metrics) on real and artificial testing sets and compare the results.

References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague January 29, 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Instance segmentation and object tracking accuracy comparison on real and virtual scenarios

Bc. Adam Simek

Department of Knowledge Engineering
Supervisor: Ing. David Hurych, Ph.D.

February 12, 2021

Acknowledgements

Special thanks to supervisor David Hurych for extensive support,
Ondřej Zeman from Valeo for help with the Vosstrex and CTU RCI cluster
admins for providing valuable resources for efficient training of CNN models.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on February 12, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Simek, Adam. *Instance segmentation and object tracking accuracy comparison on real and virtual scenarios.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Tématem této práce je porovnání algoritmu detekce a trackingu několika tříd objektů v konfiguraci 4 fisheye kamer a v ekvivalentní konfiguraci virtuální reality, v rámci projektu od firmy Valeo. S tímto srovnáním bude možné ohodnotit použitelnost virtuálního systému Vosstrex pro testování dalších systemu detekce a trackingu.

Klíčová slova Detekce, Tracking, Fisheye, Virtuální Realita

Abstract

This thesis focuses on comparison of detection and tracking various types of object classes on configuration of four fisheye car-mounted cameras in reality and equivalent car system in virtual reality in project Vosstrex by Valeo company. The goal is to compare performance of various detection and tracking systems on both real and virtual data. With this comparison it will be possible to evaluate usability of VR system Vosstrex for validation of other detection and tracking systems.

Keywords Detection, Tracking, Transfer learning, CNN, Kalman Filter, Reidentification, Fisheye, Virtual Reality, Multi-camera

Contents

List of Figures

List of Tables

Introduction

Computer vision is scientific discipline simulating the visual system of living species as well as further processing of collected information and analysis. Artificial intelligence as one of leading research fields is deeply intertwined with computer vision since visual data usually provide crucial information to navigate AI.

There is an incredible demand for automated image and video processing in industry for increasing the quality of analysis tasks, speed of the processing tasks and reduction of the production cost. Furthermore autonomous driving is extremely popular topic, where computer vision grants interaction of AI with reality, providing whole task of extraction and processing of digital data.

The virtual data are much easier to generate than manually collecting data for each developed system for validation of experiments, furthemore the accessibility of digital data for computer vision tasks have been always problematic form ethical perspective and more strict GDPR [1] protection rules, therefore many researchers started to use Virtual Reality to gain easier access to datasets for research on projects [2].

This thesis was developed in cooperation with the company Valeo. "World-leader in producing sensors that enable vehicles to understand their environment." [3] Valeo company provided data from car-mounted system with four fisheye cameras and VR simulator Vosstrex with matching visual setup.

Problem statement

The major goal is to compare the performance of detection/tracking models on real and virtual scenarios with registered sequences of tracking instances, split into training and testing dataset and creating approximate detailed VR scenes reflecting testing scenes 1:1 for comparison.

The comparison is to be performed with model trained on training part of real scenes with automatically generated annotations and tested on (a) testing part of the real dataset and (b) on VR scenes semantically matching their real counterparts. The difference in results will show if and when VR scenes may be used to validate a particular system instead of real data, which are expensive to capture and annotate.

Additional goal is to tackle detection on fisheye distorted scenes and create unified tracking system across all cameras, since there is not a lot of research material on these topics.

Tasks of visual information retrieval

This section describes computer vision tasks, which are used to obtain visual information from scenes. The instance will be referring to an actual object in image, described with location, category (class) and shape approximation (mask). The instances are categorized by their classes and only classes associated with traffic are being focused. Most important classes are pedestrians, vehicles (car, truck, train, bus), bicycles, motorcycles and traffic lights, everything else is labeled as background in this experiment. The characteristics of the instances are described in following four popular computer vision tasks:

Classification is categorization process, where each instance is being labeled with information of human realm representation with possibility of score describing confidence of the prediction.

Localization is used to discover locations of objects in image to improve effect of classification.

Object detection combines classification and localization to find multiple objects. The most popular method of defining the object is bounding box (bbox) - rectangular area around instance of the object.

Semantic segmentation makes prediction of class for each pixel of the image, in result finds all object classes and their exact location and shape, however this discipline does not differentiate between individual instances, therefore it is not possible to recover one instance from clustered class detection.

Instance segmentation consist of all previously mentioned tasks with modification for semantic segmentation to apply locally for each object found by object detection resulting in instance labeled pixels creating a mask.

Tracking links instances in dimension of spacetime in video sequence of images. Tracks provide information about positions of detected objects and provides insight into object state defined by tracked parameters of interest.

Intersection over Union (IoU) is method that determines the overlap between a pair of bounding boxes or masks. Usually ground truth bounding box and a detected bounding box. With IoU metric it is possible to determine matching detection to ground, depending on overlap and chosen threshold. IoU is a confidence type metric, in range $IoU \in < 0,1 >$. This method is computed by obtaining area of both bounding boxes and dividing intersection of both areas by union between them. This method is good for finding overlap of two equally sized areas, however it ignores most cases where one bounding box is significantly larger than the other since the smaller (e.g. car door vs car).

VOSSTREX Virtual Reality

"Purpose of Vosstrex is mainly to be used as real-time simulation system for hardware in the loop benches [3]. It should simulate the environment, the ego vehicle and all of its sensors." [3]

In this thesis Vosstrex is being used as editor tool for creating VR scenes as well as simulating them. Editor provides building blocks of terrains and roads, which means roads are straight, 90° turns or crossroads. Furthermore there is a moderate choice of decorations, buildings and vegetation. However the most important elements are pedestrians, cyclists and vehicles Fig. 4.36.

Most of the pedestrians have predefined functions for running, walking and changing directions with point to point navigation system, while cars have more simulation-like approach, providing acceleration, deceleration and steering wheels. Additionally Vosstrex provides physics engine, for physics of car motion simulation. Unfortunately there are few bugs with diminishing effects for purpose of VR scene creation, since it is still work in progress.

For simulation of the created scenes, Vosstrex offers fully realistic virtual copies of cars with camera systems. In this work Mercedes-Benz [4] W213 Daimler model is used with four fisheye cameras Fig. 0.6 setup providing full visual cover of surrounding objects.

Valeo Drive4U® prototype

Valeo Drive4U is a prototype of autonomous car which is famous for navigating trough streets of Paris while using only production-available sensors.

Mentioned autonomous car is used as reference for camera configuration on car in this work. The cameras used to generate data are four fisheye camera cocoon models placed at front, back and both sides of the car as seen in Fig. 0.2. These cameras provide fisheye distorted vision of area around the car with effective distance of 25m (pink area on image).

Figure 0.1: Valeo Drive4U® prototype thumbnail.

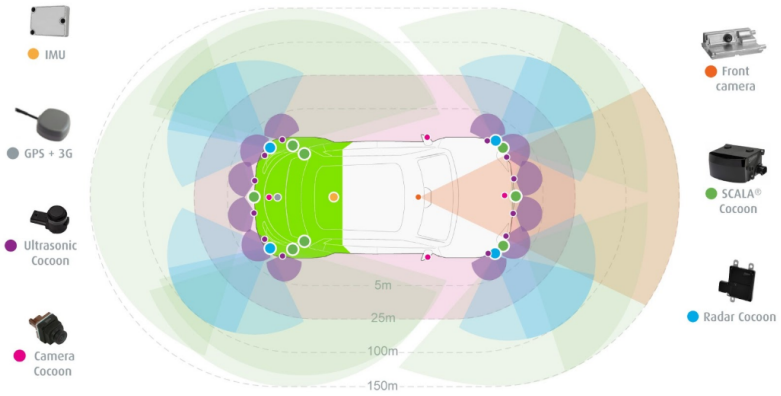


Figure 0.2: Valeo Drive4U® prototype sensors layout, the fisheye cameras used in this thesis are labeled as camera cocoon.

Fisheye camera

Fisheye is a special camera setup providing very wide field of view (FOV), usually above 180°, Valeo cocoon cameras Fig. 0.2 have FOV 192°. The main benefit of fisheye camera setup is large cover of surrounding area, with four cameras it is possible to obtain full cover (360°), especially when cameras are strategically placed on outer positions of car for maximizing vertical and horizontal coverage. The main disadvantage is heavy barrel distortion, all objects increase their size significantly but not in uniform scaling, when approaching center of camera and suffer with rotation, while reaching the edges of camera. Effects of fisheye distortion on state of the art detection frameworks will be discussed at experiments 4.8 part. Because of the extreme distortion a fisheye lens produces, the pinhole model cannot model a fisheye camera.

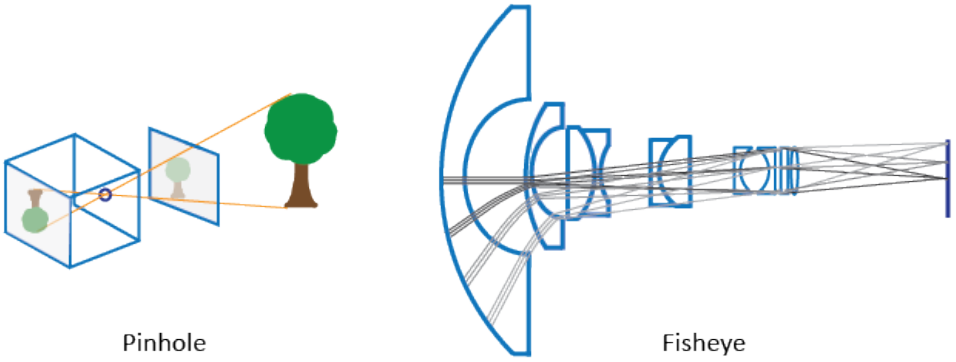
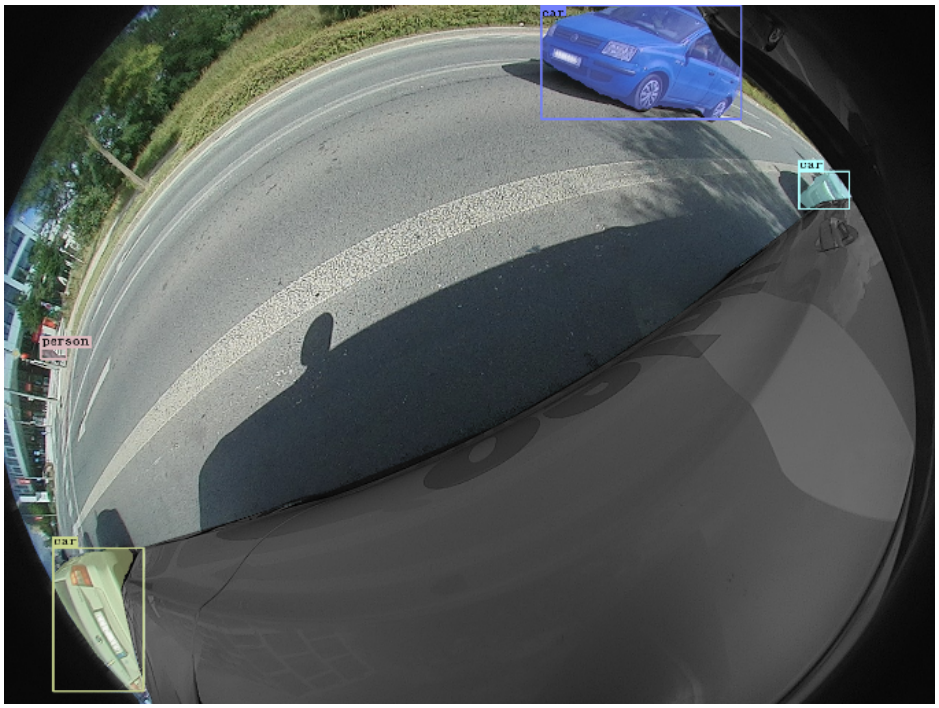


Figure 0.3: Projection model difference between pinhole and fisheye, from Matlab manual for fisheye calibration. [5]

In terms of optical aberrations [6], cameras slightly suffer from vignetting in combination with reflection from metalic nest of camera on car, which are dealt with by masking out critical areas, also the sides and other parts of car itself are masked out for simplification of visual information retrieval tasks. Camera is well fine-tuned in terms of chromatic aberration and diffraction and obviously suffer from heavy barrel distortion since it has fisheye lens.

(a) Vignetting on the left side and reflection causing errors for detection on the blue car



(b) Masked area displayed with monochrome filter suppressing aberrations

Figure 0.4: Masking solution displayed on right-side fisheye camera. The Valeo car was masked out as well to avoid possibility of overfitting in deep learning algorithms.

Fisheye undistort

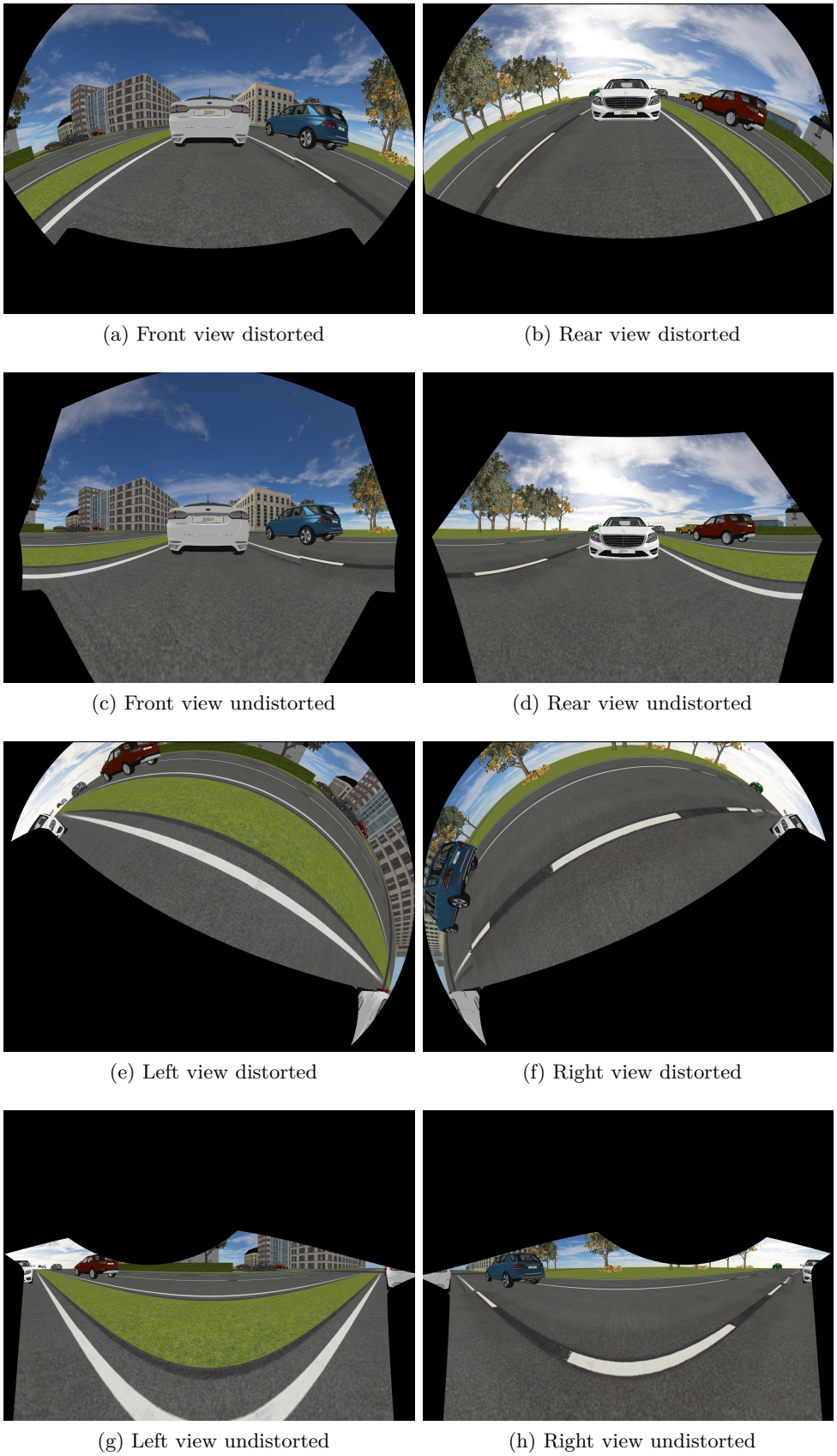
Affected by distortion defects and rotations, preprocessing methods for elimination of distortion are common step in most fisheye camera systems. Undistortion is process where algorithm tries to straighten lines which have curved shape in distorted images. Usually everything is focused into suppressing at least the vertical distortion.

There are many algorithmic solutions using nonlinear least-squares [5] and deep learning methods [7] to find distortion parameters from distorted lines in image, but these methods might be challenging for such heavy distortion as fisheye. Another popular method is Scaramuzza's [6] chessboard calibration, which is implemented in many computer vision libraries, however this method needs possibility of live interaction with cameras and cannot be done "offline".

Fortunately Valeo company provided undistort parameters for building mapping functions between distorted and undistorted image space. Parameters are divided into extrinsic and intrinsic, where extrinsic is used to describe camera in 3-D world system and intrinsic describe radial and tangential distortion as well as center of camera on image.

(a) Left view distorted (b) Left view undistorted

Figure 0.5: Example of transformation from distorted to undistorted state.



(a) Front view distorted

(b) Rear view distorted

(c) Front view undistorted

(d) Rear view undistorted

(e) Left view distorted

(f) Right view distorted

(g) Left view undistorted

(h) Right view undistorted

Figure 0.6: Distorted and undistorted Vosstrex fisheye cameras with masked vehicle.

Related work

Besides the literature describing specifically problems of object detection, instance segmentation and multi-object tracking alone, there are not many works describing problems for tracking with multiple (mostly single camera) cameras mounted on car (moving object). Most of the available literature is dealing with problem of tracking people with multiple cameras with fixed positions.

As for detection with fisheye cameras most of works focus on surveillance top-view cameras [8], there are some works similar, to this with fisheye configuration for detection [9, 10, 11] and tracking [12, 13], most of these articles are focused in subsections of this work, therefore they are used as helpful reference for some experiments, but not covering full scale of this work.

Most of literature sources for this work are related to recent state of the art techniques in fields of real-time object tracking and theirs origins [14] either without re-identification [15, 16, 17], or with re-identification of objects with help of CNN embedded features [18, 19, 20, 21, 22].

As for virtual reality the Vosstrex VR is work in progress and not publicly available, making this task more unique. However using VR data from computer games with highly realistic graphic such as GTA [23] for computer vision tasks is not a new idea, benefits of using the VR are well discribed in article [2].

The detection task is achieved with mask R-CNN [24] architecture and popular YOLOv4 one-shot detector [25], the final choices will be described in following chapter Sec. 1.

Thesis contribution

This work adds interesting ideas how to automatically generate dataset from data, when the state of the art trained models are not very familiar with its configuration (unique fisheye distortion).

The additional perk of this work is successful experiment of fine-tuned CNN model 4.7 which works even with distorted images without the need of undistortion 4.8 which is computationally heavy operation.

And finally this work is valuable for Valeo and Vosstrex developers to evaluate state of art detection and tracking on theirs VR system.

Chapter 1

State-of-the-art

Convolutional Neural Network (CNN) is deep learning (DL) [26] algorithm used in computer vision designed with inspiration of the connectivity pattern of Neurons in the human brain [27]. CNN is capable of processing multidimensional data such as images, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other [26].

The computer vision sector made major progress over the last couple of years, especially in the computation time performance. Object detection, segmentation and re-identification [21] have been very popular for task such as face recognition, autonomous driving, surveillance and many more fields in computer vision.

1.1 Object detection

This part of thesis will describe the building blocks of the architecture of detection framework and their purpose, rather than reaching depths of implementation of the innermost parts of the CNN.

Recently, the CNN [26] models have obtained more generalized structure especially the Faster R-CNN [28] to provide better building foundation in the computer vision tasks, this architecture is usually referred as "Generalized R-CNN" [29].

There are two popular meta-architectures of CNN: two-stage detectors and one-stage detectors, the differences will be mentioned through this chapter. The Generalized R-CNN (Region based CNN, since first stage propose regions) is relied to two-stage detectors which will be described first.

1.1.1 Backbone

The core part of every deep learning image processing method is feature extraction. Feature extraction is special kind of image preprocessing, which is fully orchestrated by R-CNN. The preprocessing might be in simplified version compared to preprocessing image with filters such as first or second derivative kernel used for sharpening or edge detection, however the rules of these filters are learned and defined automatically by R-CNN and all filters can have much more possibilites than mentioned filters.

Feature extraction mostly breaks concept of three color channels (Red, Green, Blue) and creates new channels describing features, which R-CNN considers to be useful, trough backbone the original image content is constantly reduced in dimension of width and height, while the channel dimension rapidly increases, nowadays R-CNN backbones usually contribute 256-1024 channels.

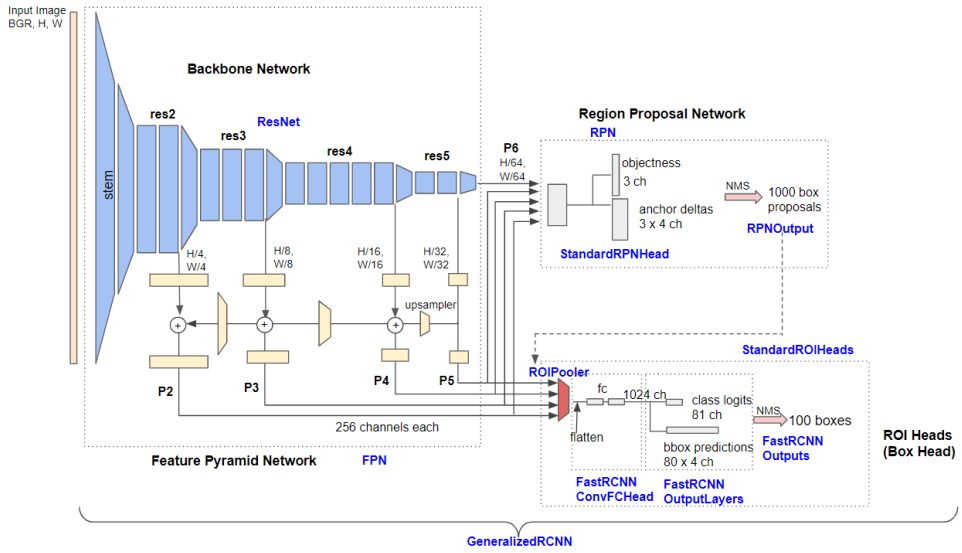


Figure 1.1: Generalized R-CNN architecture used in Detectron 2 [29] with ResNet 50 backbone architecture from article [30].

The backbones can be different than ResNet 50 (e.g. VGG [31]) or having more depth (more parameters which can be learned) resulting in better performance, there is whole article about experiments with various ResNet convolution block depths [32], however with increasing depth of blocks, the computation time increases as well. Fortunately the proposed ResNets Fig. 1.2 are balanced to get best combination of performance and time efficiency.

50-layer ResNet is the middle way of ResNet architectures, the table Fig. 1.2 can be associated with Fig. 1.1 conv1 is input block and res2 equals conv2_x etc., this backbone is used in experiment 4.9.2.

101-layer ResNet was used for vast majority of detection tasks in this work, this configuration have 7.6 billion FLOPs (FLOPs in this case are operations needed to do one iteration of algorithm, FLOPS are used for time performance), which is twice the amount of ResNet 50, this is achieved by deeper conv4_x (res4) block of ResNet backbone, which have more layers.

152-layer ResNet was used for automated annotation generation, to maximize performance with 11.3 billion FLOPs on behalf of computation time, which is less important for annotation generation.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 1.2: ResNet backbone configurations from article [32].

Feature Pyramid Network

Detecting objects in different scales is challenging in particular for smaller objects. However rescaling and cloning image to create mutiple inference tasks with different sizes would be time inefficient. Another solution would be to acquire features not only from the output of the backbone but also from other blocks (e.g. res4, res3, res2 from Fig. 1.1), the problem is that the features might not be fully processed in these stages since they are closer to actual image. To improve this solution in efficient way the Feature Pyramid Network (FPN) is used.

The FPN [33] is pyramid structure interacting with pyramid structure of backbone, the previously mentioned solution had problem with features not being completely processed, to help with this the FPN receives output of backbone and use it to upsample ”half-done” features from blocks res5 up

to res2 in backward-way, where each block with more processed information enriches his parent block. From Fig. 1.1 the FPN description shows, that on top of original output of backbone the R-CNN now have 4 additional outputs with functionality of representing different sizes, helping with smaller objects.

1.1.2 Region Proposal Network

The RPN [28] are convolution layers which provide time efficient search for possible objects on feature map. Given feature map as input from backbone (and FPN) the R-CNN has to learn whether there is an object present on every point of feature map. This task is achieved by placing "anchors" on every point, each anchor can represent object with box of finite options for size and aspect ratios (usually 3 sizes and 3 ratios, resulting in 9 possibilities), each anchor also provides confidence of object being present (in Fig. 1.1 called objectness).

The possibilities are finite to ensure process of RPN is time efficient, the RPN does not find perfect locations of objects this is achieved by bbox regressor 1.1.4, the RPN only estimates objects and backrounds on feature map so following parts of R-CNN have less work to do and can focus more on precise predictions.

The RPN usually have fixed amount of proposals which can be proposed, to ensure there are not many duplicities since anchors next to each other might propose the same object, the Non-maximum Suppression (NMS) [34] which is simple algorithm removing area duplicities with intersection over union over chosen threshold.

1.1.3 Region of Interest Pooling

The RPN proposals will be input for following part of R-CNN, however these proposals have only information of location and objectness, therefore they must be reunited with features from backbone once again, this is achieved by ROI Pooling, furthemore ROI Pooling also reshapes all proposals (now features from location) to have same fixed size, making their processing more time efficient for GPU systems.

1.1.4 ROI Heads

The "heads" are final branches of R-CNN, these branches provides every desired task which is requested on each object, usually there is head for classification and head called bbox regressor providing more accurate bounding box than RPN for task of object detection. The task of mask segmentation of mask R-CNN [24] have also Segmentation head which uses the same inputs as other heads.

1.1.5 One-stage detector

This part will focus particularly on YoloV4 [25] as the newest addition of You Look Only Once family of detectors (at the time of writing project YoloV5 still have not proved its superiority and originality).

The one-shot detectors were very poplar choice of detectors especially if speed of inference (prediction) is more valuable than accuracy of predictions, the Yolo projects have been always on top of speed performance especially thanks to theirs efficient implementation and lightweight model.

The major difference between one-stage detectors and two-stage detectors is the lack of RPN, Yolo does not use RPN to predict region proposals and solves inference as regression problem straight from backbone to estimation of bounding box and class. This method saves lot of computation time, making Yolo good candidate for real-time detection (speed can over 60 FPS, depending on configuration).

The YoloV4 version also implements most of the good ideas from Faster R-CNN backbone such as its own version of FPN and many other state of the art methods.

1.1.6 The choice of detectors

The detector for generation of automatically annotated data should be focusing more on accuracy of prediction, rather than on speed, therefore the best choice was two-stage detector, which was trained to provide good performance.

Detectron 2 [29] have many perks making it good choice for experiments in this project, first reason is the quality of pretrained models, which made good baseline for automated data annotation, second reason is quality of implementation and options of training, to train networks easily on cluster GPUs and being on of the fastest among two-stage detectors. And last reason is good documentation of code, to easily make modifications to code needed to enable this project.

The YoloV4 was picked for experiments as well, since it provides valuable insight how would the the experiment work if there was necessity for real time detection.

1.2 Tracking

The final tracking algorithm is based on DeepSORT [18] tracking algorithm which is a modification of the Simple Online Realtime Tracking (SORT) [16] with deep-learning method for obtaining feature embeddings for re-identification [21]. This tracking algorithm has two important elements the first is localization prediction and the second is re-identification, which support each other in order to fortify their weaknesses. The tracking algorithm uses detections from detectors to define tracking instances

1.2.1 Kalman filter

It is an iterative mathematical process that uses a set of equations and consecutive data inputs, which allows to quickly estimate position and velocity when the predicted values contain random error, uncertainty or variation [35].

With difference to machine learning models which need many inputs from data to interact with to be able to predict properly, Kalman filter can narrow down to possible solution, which is usually close to optimal solution with requirement of very few inputs by understanding theirs uncertainty and variation. This ability is extremely valuable for tracking, since Kalman filter can predict behaviour of the new object very fast and in not so many iterations (frames passed).

1.2.2 Localization prediction

The previously mentioned Kalman filter is the core of localization prediction, this algorithm is used to predict positions of centers of the bounding boxes and with these predictions it improves predictions of next bounding box location for each frame if the track was not lost in the process.

This part of tracking algorithm is performing very well on slow moving or static objects or any objects with stable behaviour, also on scenes with high framerate detections, which makes object movement to appear slower. However any sudden changes in object trajectory, speed or object hiding behind another object can be hard cases for this algorithm to deal with.

1.2.3 Re-identification

The re-identification is used for comparing image information of two objects and making decision, whether the objects are similar to each other (possibly the same) or if they are completely different. Re-identification has possibility to reinstate track which has been lost by Kalman filter, but in some cases the matching potential still is not that powerful to be used alone.

Instead of methods like pixel-wise comparison of two images, this method transforms image cropped to focus the object (crop is provided by detector) into feature embeddings, which are vectors of information, which can be compared to test similarity. The embeddings can be created by encoder [26] working with cropped images directly, which was used to automatically generate tracking data or getting features from ResNet backbone, which is better approach since it uses the features computed by detection and saves time. The modification for Detectron 2 was inspired by feature extractor GLAMOR [36] which is designed for ResNet backbones.

1.3 Multiple object tracking

To make localization predicition and re-identification to work together, there is need for Tracker which will control the life cycle of tracks. The Tracker is modified implementation of DeepSORT [18], with custom re-identification layer.

The life cycle of Tracker:

1. Initialize Tracker
2. Obtain feature embeddings from detection and gather previous tracks
3. Compute the distances d_1 between new positions of objects from detection and Kalman Filter predictions
4. Compute the distances d_2 between new feature embeddings from detection and last feature embeddings of active tracks
5. Compute weighted distance c combined from Kalman prediction distance and feature embeddings:

$$c_{i,j} = \lambda d_1(i,j) + (1-\lambda) d_2(i,j) \tag{1.1}$$

6. Use Hungarian Maximum Matching algorithm 4.4.1 to get perfect matching from weighted distance matrix c
7. Update tracks and Kalman Filter with matches
8. Initialize new tracks and eliminate inactive tracks and goto 2)

There are important initial variables to control lifecycle of Tracks, first is maximum age for track elimination (25 frames is value being used) threshold, which eliminates inactive tracks, when theirs inactivity reaches this threshold, second is λ balancing weighting between Kalman filter and re-identification (set to 0.7 favoring re-identification).

The last is distance function, author of DeepSORT [18] uses Mahalanobis distance, however it is mentioned that, this metric is more favourable for short-term predictions and cosine metric is considered better for recovering in long term predictions. Since Kalman filter is used to deal with short-term predictions, cosine metric is being used as distance function d_2 for computing difference between feature embeddings.

The following equation describes cosine distance function for two feature embedding vectors $\mathbf{A}$ and $\mathbf{B}$.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}, \tag{1.2}$$

Chapter 2

Dataset

The data provided by Valeo [3] company had form of ".dat" binary files with ADTF (Automotive Data and Time-Triggered Framework) [37] format, due to large size of original dataset (6.8 TB), binary images were transformed to ".jpeg" format with 90% compression.

Images of constant size 1280×960 group into video sets, further referred as scenes. The total count of original scenes is 900, each scene has 4 cameras: FV (front), RV (rear), MVL (left), MVR (right) and each camera has up to 1000 frames, with framerate 30 FPS resulting in videos long around 30 seconds depending on number of frames.

2.1 Automatically generated annotations

The data were issued as only images, there were no annotations, therefore it was necessary to generate the dataset with an existing trained model, to make task easier the images were first undistorted Fig. 0.5, then the annotation was executed and finally the mask was transformed into polygon format and the polygon points were transformed to distorted version. For the distorted polygons, new bounding box had to be created by enclosing object mask, afterwards boxes were described with top-left, bottom-right corner points format.

The annotations were saved in Detectron 2 [29] and YoloV4 [25] format to ensure fast data loading for training. The experiments with automated annotation generation are described in section 4.2.

For tracking, the tracks were obtained with original DeepSORT [18] Tracker, the data were saved as unique track identifiers in annotation each object instance.

2.2 Dataset format

The dataset annotations were saved in both ".json" for easier access and later in ".pickle" binary format for speed performance. The annotation file is always on the same directory level as the folder containing all images.

```
Dataset
└─ Scene
   ├─ [FV|RV|MVL|MVR] Camera ..... Folder with images for each camera
   │  └─ *.jpeg ..... Images
   └─ [FV|RV|MVL|MVR] Annotations.json .... Annotations file for each
      camera
```

Figure 2.1: Tree structure of dataset folders and files.

To make dataset usable on memory limited computing cluster enviroment, the data had to be reduced by using only 6 FPS scenes, this was achieved by saving only images with scene id value modulo 5 (final size 350 GB). This data modification, might increase the difficulty of tracking task, where longer intervals between images have higher chance of spontaneously changing direction, weakening Kalman filter [35] predictions 1.2.2. Despite the task becoming slightly harder for inference, this modification does not influence heavily any training task of selected deep-learning models, not even re-identification, since positives in triplet loss are getting harder with greater distance 4.3.3.

2.3 Dataset split

The very important data preprocessing step before application of any machine learning method of machine learning is dataset spliting. In most cases data are divided into three parts: training, validation and test dataset.

The training part of dataset is being used for the training phase 4.3, usually called model fitting. This part is in most cases the largest part of all, since for training it is important to have great amount of data to train complex models to fit the data. With low amount of data trained models might suffer from underfitting, which means the data are too simple or not in enough quantity. Underfitting might also occur when trained model is too complex fro the data.

To eliminate the underfitting, the most straightforward approach is to repeat epoch multiple times, however this approach usually can suffer from another problem, which is overfitting, describing state in which model predicts extremely well on training cases and similar cases, yet the test cases have poor detection performance, because the model was trained on similar data so much that its weights are overtuned towards this particular case, or the data splits are not well designed [26].

Canonical machine learning tasks often prefer random splitting methods and more balanced train-test split, usually 50-75% for training part, since tasks other than image processing have generally much more samples it is relatively forgiving to use random dataset sampling methods.

This is definitely not the case for deep learning in image processing, for image detection the smallest sample unit is image, which can have multiple instances of objects (actual samples which are grouped), unfortunately for tracking tasks, the smallest sample unit is actually one video, resulting in very small number of samples, where each scene usually has its unique characteristics e.g. brightness, dominance of the object class or dynamics of the scene.

Aforementioned grouped sampling requires more supervised care for dataset splitting for obtaining better results as described further below in this chapter. Another trend for deep learning in image processing is creating larger training parts, mostly because the data are very costly in terms of computer memory and their capturing, therefore researchers try to acquire as much possible information from training [38].

The testing part is the part of dataset with very strict rule, that model can never interact with testing data in training phase, only in final evaluation. This part describes the unseen cases when the trained model is being applied in practical use, therefore the rule of interaction with training is very important. From previously mentioned reasons, this part is very small in comparison to other parts, the main reason is the cost of high quality annotations and starving for quantity in testing phase. While being the smallest part, the test part must be balanced with most delicate care to reflect as many real scenarios as possible.

For dataset provided by Valeo with 900 scenes, data were split into 750 training, 100 validation and 50 test scenes. After removing scenes which were corrupted or not usable (there were some scenes with open car doors, which were not compatible with camera masks) the final state of Valeo dataset is 743 training samples, 98 validation scenes, 45 test scenes. The reasons of this split will be described in following section 2.3.1.

	training	validation	test
frames	493524	56224	31728

Table 2.1: Dataset split frames count

2.3.1 Dataset balancing

For deep learning tasks it is very important to balance dataset parts to provide roughly equivalent distribution of object classes, colors or other specific characteristics of images and objects. The approach for this work was to create categories of interest, which would act as the deciding factor of dataset split.

All color based categories were accomplished with hue saturation value (HSV) [6] color spectrum, which is very popular in image processing for color analysis. The frequently used red green blue (RGB) or blue green red (BGR, used in detectors) have three color channels each working with one color as the name suggests. HSV has three channels as well, however only hue describes color with 360° spectrum divided into six overlapping major colors: red, yellow, green, cyan, blue and magenta, while saturation resembles brightness of the color, fully saturated meaning most colorful and lowly saturated meaning pale color and finally value resembling interaction with white and black color.

The HSV spectrum provides much more intuitive splitting of color regions, to analyze color characteristics. The hue color can be simply split into six parts on 60°, however starting from $-30°$ shift, resulting in more accurate color split e.g. there is more red color detections mapping to $330-360°$ sector than $30-60°$ sector which is actually orange.

By reason of saturation and value being free of color element, these channels are perfect for analysis of brightness of scenes, usually brightness corresponds to value, however the saturation value reacts specially to sunlight and sunrays, helping to detect very bright images on sunny days [39]. The metrics used are mean and standard deviation, mean describes average of channel values, while standard deviation indicates spread of values. The strategy is to apply binning on mean of the value channel, while being thresholded by mean of the saturation channel and finally sorted by standard deviation of value channel.

Bright scenes usually occupy high value channel mean and lowest bin of saturation, final results are selected according to standard deviation, the lower the better (low means that most of values are similar to the bright tone).

Dark scenes have low value channel, saturation is not that decisive in this case the strategy with mean and standard deviation is similar.

Saturated scenes represent color contrast images with high saturation, providing another extreme case.

Class dominant scenes are separated by object class, all available classes are related to traffic in some way due to trimming of classes in detection models.

	person	car	bicycle	motorcycle	bus	train	truck	traffic light
FV	387858	802353	11545	4246	14141	4218	55527	59719
RV	379710	822273	9514	4624	10586	3250	50456	19658
MVL	318417	707573	6458	3438	7986	2235	44737	2729
MVR	340625	515945	10600	3159	6620	738	24347	5739

Table 2.2: Train object instance occurrences across all cameras.

The different counts in cameras are caused by different camera angles and position as well as some traffic rules e.g. Tram (train) almost always ride in the middle of street, therefore MVL (left) camera is going to see the tram more times than MVR (right), if cars drive on the right side.

	person	car	bicycle	motorcycle	bus	train	truck	traffic light
FV	81802	136649	3303	1298	5265	1803	14143	7611
RV	82636	143457	2802	1112	3014	1219	9304	3851
MVL	61470	131894	1849	873	1639	680	9628	226
MVR	84192	96787	3296	1252	2630	188	5622	196

Table 2.3: Validation object instance occurrences across all cameras.

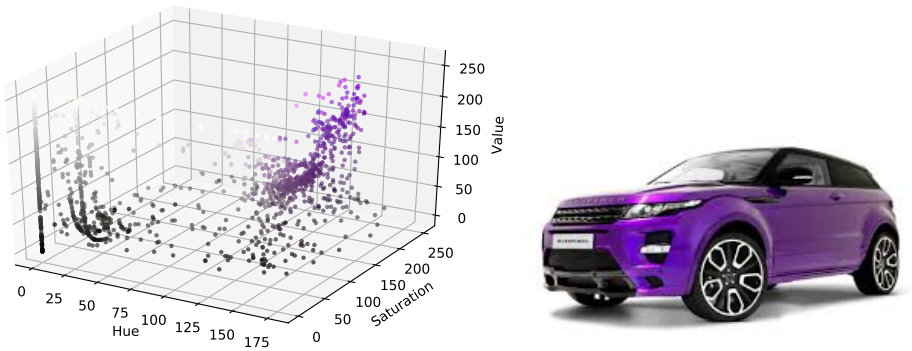
	person	car	bicycle	motorcycle	bus	train	truck	traffic light
FV	39831	74382	2737	935	1884	376	5521	4321
RV	39927	70877	2760	840	863	431	3458	604
MVL	36086	59965	1394	209	791	354	3956	478
MVR	35122	48401	3805	1019	1091	41	3092	180

Table 2.4: Test object instance occurrences across all cameras.

From previously displayed tables it is apparent that dataset split is balanced across classes, actually despite the smaller size of test data, it holds significant representation in each class.

Color balance is the last category of interest, it deals only with car, person and truck instances and balances their color split. Other classes are way too similar or starving on sample size (train, bus, bikes).

The colors are detected as previously mentioned six colors in the hue spectrum and theirs full, light, dark variant accompanied by black, grey and white.

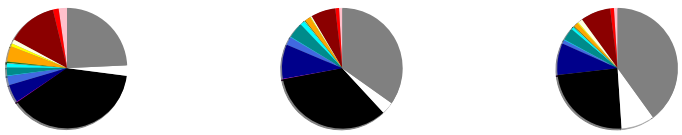


(a) HSV representation of pixels

(b) Sample purple car [40]

Figure 2.2: Distribution of HSV channels on masked car object.

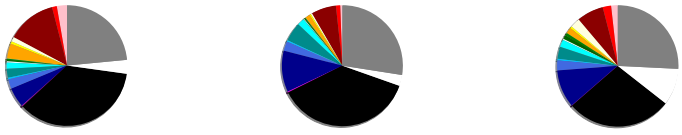
For detection of color, the HSV color space Fig. 2.2 (a) is separated into small color subspaces with four equidistant cuts across value as well as saturation and twelve cuts across hue, colors are detected by thresholding precentage on merged cubical bins color spaces created by channel cuts, the strategies are similar to previously mentioned methods.



(a) Train data person

(b) Train data car

(c) Train data truck



(d) Test data person

(e) Test data car

(f) Test data truck

Figure 2.3: Balanced color split distribution of training and testing dataset.

The Graph Fig. 2.3 shows percentage distribution of colors in dataset, collected from the front camera, each detected color means, there was at least 20% of object visible surface reflecting this particular color. For balancing colors in the test dataset, simple iterative algorithm was used, which selects scene, which fix the most colors at time of each iteration, resulting in fairly balanced split.

The rest of dataset was filled with scenes which were considered interesting for VR scenes, mostly dynamic scenes and scenes with exclusive environment e.g. cobble stone road.

2.4 Vosstrex VR scene generation

The Vosstrex VR simulator scene editor allows to build scenes and save them as objects into ".xml" document, because Vosstrex project is still work in progress, it allows building only straight roads, crossroads and 90° turns, because of this fact the most of scenes were generated from similar seed ground scene, and are quite similar in terms of surroundings, however the most important part of generated scenes is traffic, specifically cars and pedestrians.

Vosstrex provides considerable selection of car and pedestrian models, unfortunately though vehicles like train and bus or truck, do not have yet implemented kinematics, therefore are usable only as static objects. Even despite these problem it is still achievable to create approximate copy of reality into virtual reality.

The final scenes are approximate copies of their real counterpart, meaning the behavior of all moving objects in real scene was replicated to VR scene. All cars have approximately the same trajectories, most of pedestrian as well unless the pedestrians are clustered, then the exact count might not be precise. The same is applied for clustered cars, like parking lots and car lines in side alleys, only the most visible ones are replicated to VR scene.

The types and colors of cars are not matching, since the choices of color and car type were very limited. The final count of Vosstrex scenes used in experiments is 25, the results compared to Vosstrex used these 25 scenes as well.

Chapter 3

Implementation

It is always necessary to carefully pick the right tools for the job, for deep learning tasks there are many interesting choices, however it is not good idea to mix too many deep learning frameworks together, because most of these frameworks provide backpropagation systems which are not compatible between different libraries. Because of the large data it is crucial to select an efficient solution to all data movements as well as calculation. In deep learning the last decade is heavily dominated by graphic card (GPU) computations, since they offer much more parallel processing power than nowadays CPU units.

3.1 Programming language and software

For most of the machine learning and deep learning projects, python [41] language is obvious choice, the main reasons are:

1. Simplicity of code - the python with numpy [42] library provides very useful array and tensor operators, which allow to create codes for modification of large multidimensional data with ease.
2. The numpy library and OpenCV [6] library are C, C++ language based libraries, providing functions with very good performance for most mathematical and image processing tasks.
3. Deep learning support - considering previous statements, it is not suprising fact that most of the researchers use python for data processing tasks, also python has the most machine learning and deep learning libraries available.
4. Platform independency is another useful quality of python, since Vosstrex is not multiplatform.
5. CUDA [43] support, allowing efficient parallel computation on GPU.

3.2 Deep learning libraries

The two most popular deep-learning frameworks are Tensorflow [44] and PyTorch [45]. Tensorflow is project provided with great history in the world of deep learning, however being very popular and developed by many AI researchers, the first version of Tensorflow slowly changed into slightly chaotic direction. Not even mentioning the static computation graph approach meaning, that the computation is build for specific data before computations and is used repeatedly, which might seem like a good idea until the input changes or requires some complex data structures. The final disadvantage was debugging, since Tensorflow does not support many python functionalities, because its own compiling system.

The other alternative is PyTorch which is designed to fully interact with python and use dynamic computation graph, allowing to change parts of deep learning system during execution. Not to mention that PyTorch framework is much more organised and very popular in most image processing tasks. Another quality of PyTorch is performance, there are many projects rewritten from Tensorflow to Pytorch with aim to improve performance.

In response to PyTorch rising popularity, Tensorflow developed second version TF2, which acomplishes most of things that PyTorch already provided. However there are still some very good functions, for example parts of training module in this projects such as Kalman Filter are realized with TF2, however the learning parts of PyTorch and TF2 never interact with each other as it should be.

3.3 The dataset editing tool

For exploration of generated scenes or other examinations of dataset, it was necessary to create simple editor capable of showing four frames at once and walk through scene, the dataset editing tool is capable of listing frames with keyboard arrow buttons.

Another functionality is selection with mouse, which finds all objects at small area around mouse click position, the selected item can be then deleted, the editor also provides tracking with possibility of modifying tracks however this functionality is highly experimental, it was designed to help with track annotation. The editor was created with pygame library [46].

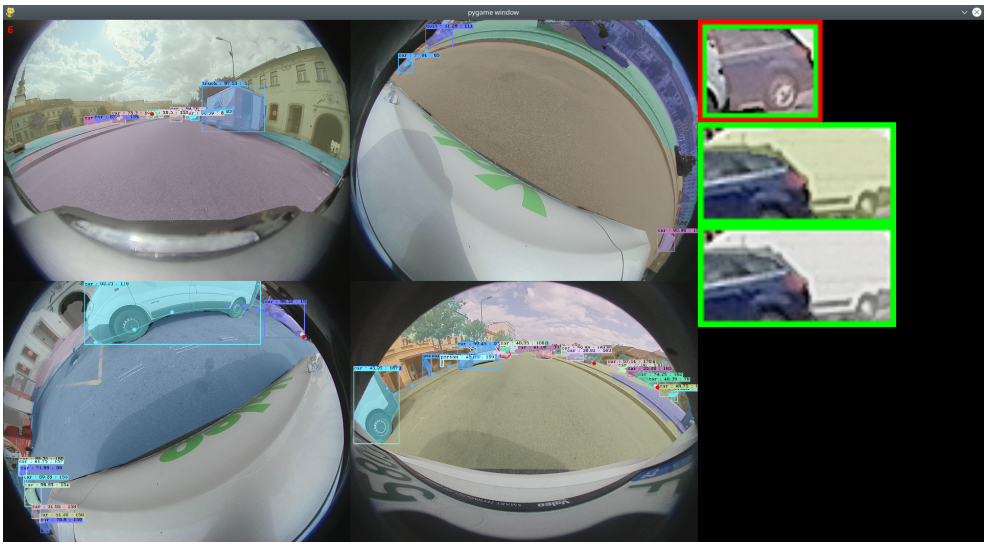


Figure 3.1: Annotation Editor Tool

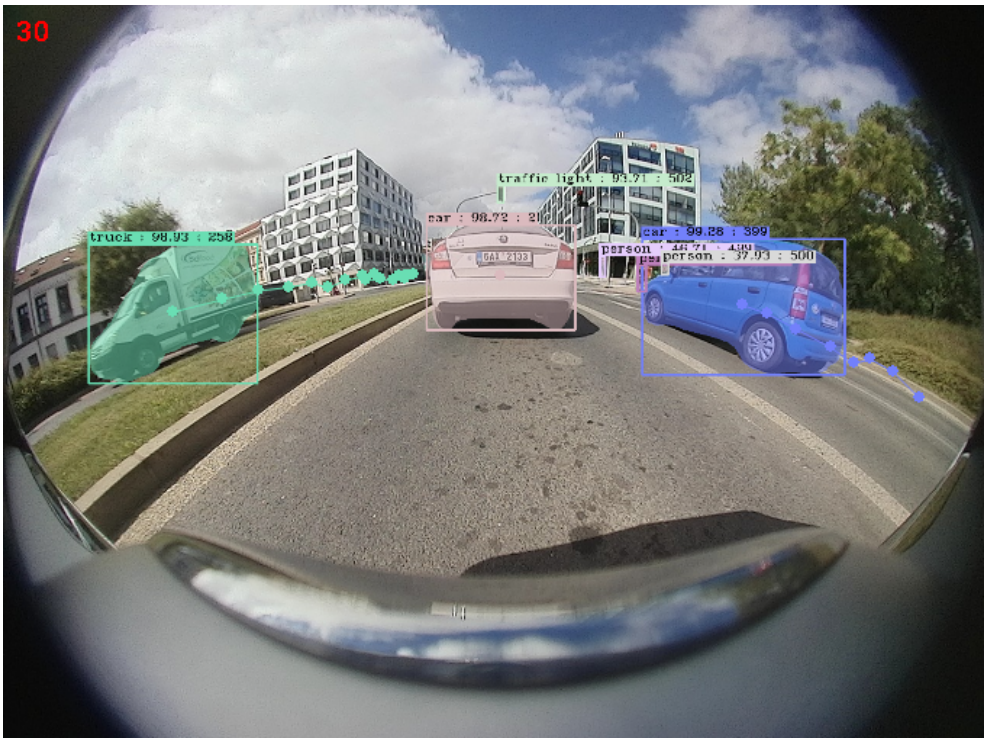


Figure 3.2: Tracking Task

Chapter 4

Experiments

In this part of thesis, all conducted experiments are described. Most of the experiments describe challenging task of training SOTA detectors on fisheye camera, automated annotation generation experiment, tracking experiments and methods to evaluate the level of performance similarity on real and virtual data. The score in experiment tables evaluates experienced efficiency of tasks.

4.1 Fisheye and state of the art detection

The fisheye camera as previously mentioned 0.3 provides greater field of view with the cost of massive barrel distortion. The most feasible solution would be to train state of the art detection model to work on distorted images despite barrel distortion or with undistorted images. The rotation of objects is still extremely challenging feature of object detection, even tough size difference problems were mostly dealt with by FPN 1.1.1. There are many projects such as CapsNet [47], which try to deal with this problem, but stable solution is still out of reach.

The lack of rotation robust model does not mean that this problem is insolvable, most popular solution is data augmentation [26], it is speculative if this approach is really the best solution. The model with some rotation reduction might deal with rotation problem and let CNN deal with the rest, on the other hand the augmentation might just try to fit model on multiple rotation cases while lowering the quality of original weights, therefore the rotation upsampling must be executed with utmost care. For fisheye problem simple rotation augmentation is much more complex than it seems. The radial distortion with combination of extrinsic parameters create very unique distortion especially nearing center point and rotating and moving this object to for example edge of scene might seriously damage learning process of model's understanding of the fisheye.

Another great problem with object detection on fisheye are bounding boxes, the rectangular form is useful for covering convex polygon objects like cars, however complex objects like people usually do not have that much density of theirs objectness in resulting bounding box, still in the end model usually learns how to deal with people objects, expecting them to cover much less area than object like car.

The very interesting cases on fisheye cameras are very long objects such as train, while seen from afar, their shape usually does not differ very much from car, when seen from front side. Still when such long objects gets close for example on side camera Fig. 0.5, the bounding box get from situation of tightly enclosing train to enclosing large object, while having more than 50% of inner area empty. Trains are probably most problematic objects, they are usually detected on most trained models, but have lot of problems with estimating length of train and approximating its mask.

4.1.1 The unsucessful experiments

Table 4.1: Image rotation experiment

model	Mask Scoring R-CNN [48] GT
method	angle rotation of objects
score	★☆☆☆☆☆☆☆☆☆

The idea was to use one of the highest performance ranking detectors at the time with trained model and help the detector by augmentation of images with rotation, even if rotation was helping slightly the models was primarly trained for competition in COCO [49] dataset, which reflected to real performance on random dataset.

The detections were stable only with certain angles and using more angles was too time consuming on top of the already slow model even for mask RCNN. Also the algorithm to merge masks, find the best ones (usually the ones at the angle of object center) and suppress others was very hard to fine-tune and the mask quality was very low from distorted positions.

Table 4.2: Crop classes experiment

model	Detectron2 X152 FPN [29] GT
method	undistort, filter classes
score	★★☆☆☆☆☆☆☆☆

This model used in future successful generation experiments with undistort had small modification of removing unnecessary classes, however by removing classes most of remaining classes started to push into detections of deprecated classes, therefore this idea was banished.

4.2 Annotation generation experiments

The most impactful method in automated annotation generation was using undistort on images before feeding them to the CNN, reduction of distortion helped extremely for rotated objects and deformed long objects like trains and busses.

The final choice of detector model is Detectron2 X152 FPN [29] (also called ResNeXt) trained on ImageNet-5k [50]. This model also has cascade R-CNN functionality, which makes more detections sequentially, the model was obviously much slower then most others mask R-CNN models, however the quality of detections even on undistorted fisheye datasets was much better than previous choice.

Table 4.3: Experiment basic

model	Detectron2 X152 FPN [29] GT
method	undistort, low score threshold 0.3
score	★★★★★☆☆☆☆☆

The cascade feature has small disadvantage and that is generating multiple detections on the same object with highly tolerant score threshold 0.3, the low score threshold is used in future experiments, since it is better to start with more objects and filter them, than have small amount which might possibly be wrong, also the models usually get small scores on datasets which have never been seen by them before.

There are two popular metrics to find overlapping objects:

1. The bounding box intersection over union is very fast and simple method, unfortunately it is not very precise and because of fisheye, which causes very large bounding boxes, creating larger chance of overlap and elimination of actual valid object instance.

2. The mask intersection over union is slow but much more precise, since the processed are is actual area of the object and not surroundings like in case with bounding box. There is possibility to improve speed of mask IoU performance using RTree [6] indexing, creating sorted x, y interval values to decide possibility of intersection of the polygons.

Table 4.4: Experiment NMS

model	Detectron2 X152 FPN [29] GT
method	undistort, bbox nms
score	★★★★★★★★★★

The first chosen method was bounding box intersection, the total process is called non maximum suppression for bounding boxes (bbox nms), picking highest scoring object among others within chosen limit of IoU, this method reduces duplicities, however there is problem that the duplicities are mostly of different classes, and if the wrong class gets disabled, the detection becomes confusing in training.

Cleaning masks and tiny detections

To improve data cleanliness and avoid future errors form masks with shape of point or line, it is important to clean data first. For every mask it is necessary to delete small shards, which usually only make problems in all IoU tasks, also deleting very small objects with low score helps a lot in all tasks. This method was used in all following experiments.

4.2.1 Automated tracking annotations

Tracking has one big advantage over detection in automated annotation generation and that is Track mining. The track mining finds tracks of reasonable size (at least 5 frames) and uses them for training process. Since the training algorithm is mostly interested into these mined tracks and not into overall correctness of all tracks in annotations, this method is much simpler then creating annotations for detection.

The only problem would be if Batch hard Triplet Loss 4.3.3 would receive tracks having Positive and Negative of the same object in reality. This means that one real track would be divided into more tracks and these tracks would have been learned as different even if they are the same track. This problem is mostly avoided by choosing the threshold on length of Track according to framerate.

The tracks are saved as unique identifiers which are globally generated.

Table 4.5: Experiment Tracker

model	DeepSORT [18] Track GT
method	Kalman Filter + Re-Id 1.2.2
score	★★★★★★★★★★

The Track annotations in this work were automatically generated with DeepSORT algorithm [18], the re-id part of DeepSORT [18] was also used to find and save some tracks, which are unified over cameras (same object in different cameras).

4.2.2 Panoptic Merge

Table 4.6: Experiment Panoptic Merge

model	Detectron2 X152 FPN + R101 FPN (panoptic) [29] GT
method	undistort, panoptic merge
score	★★★★★★★★★★

The bbox nms was very fast in execution, but the quality was not that outstanding, even though panoptic merge focuses mostly on removing random false detections in background, it deals with overlapping objects much better than bbox nms 4.9.2.

The panoptic segmentation uses semantic segmentation to detect backgrounds and instance segmentation to detect object instances as it can be seen on image 4.1. The panoptic merge idea came from using merge with semantic segmentation, however none of trained semantic segmentation models could decently detect background objects in undistorted image. This is because the undistorted side camera Fig. (0.5) has the road still heavily distorted and semantic segmentation has problems to deal with this. Fortunately most of the upper backgrounds of image are recognizable by semantic segmentation, since the distortion is lower in these areas.

The panoptic segmentation trained by Detectron 2 [29] uses semantic segmentation combined with secondary instance segmentation, the instance segmentation provided by this model is only used to balance mask borders between instance and background and provide valuable backgrounds such as buildings, trees, grass, wall which can block most of objects detected by primary instance segmentation detector. The detections from panoptic segmentation will be referred as semantics (instances and backgrounds).

The primary instance segmentation (X101 cascade mask R-CNN) is much superior [29] in terms of predictions, however there is no version of panoptic segmentation with this model, also multiple detectors bring more information to work with, even though primary detector is prioritized, some masks might be used from secondary detector under certain conditions.

The panoptic merge works as instance competition to find best combination of instances to be used as ground truth, this method also deals with classes which are not desired, some of them can be kept as ”dontcares” described in training, but only if their presence influences some desired instances.

The panoptic merge instance tournament:

1. Match all detections by mask IoU from primary detector with secondary detector semantics
2. For all semantics if semantic is secondary instance:
   a) Sort all instance matches to semantic by IoU (descending order)
   b) Put previously sorted instances into queue as candidates
   c) While there are candidates:

      The process is to use best candidate and then try to fill the remaining area of semantic with best solution

      i. Pop first candidate from queue and assign it to solution
      ii. For each other candidate and their neighbour instances fill solution, but ignore instances, which have large intersection over minimum area with instances already in solution (currently used threshold 0.7):

      $$intersection_{area}/min(instance_{area},\ semantic_{area}) > threshold$$

      his can be combined with condition to avoid if class is person (used in this work), if it is desirable to keep person detections inside vehicles and not substract from semantic area, otherwise it is great tool to remove person detections form vehicles
      iii. Record solution, compute solution score:

      $$solution_{score} = \sum_{s \in solution} s_{score}\ (1\ if\ wanted\ class\ else\ 0)$$

   d) Get solution with highest score as final and mark instance to keep it, notice that objects with class, which is not desired have no score, therefore they lose against solution with more desired classes
   e) If solution is only one instance it means that it can compete with secondary instance, this part is mostly customizable by experiences with dataset, in this case if primary instance had at least 20% of its mask detected as surrounding objects, then use secondary instance, this helped to fix many masks e.g. trains and busses can usually be detected as buildings (because of windows and shape), and if there was building behind train, the mask of primary detection had tendency to assume the building is part of train, these were tho only cases where secondary instances outclassed primary ones, because the building background already suppresed the train mask misdetections

3. For all semantics if semantic is background:

    a) For all instance matches, if the match has high score (used > 0.5) and is desired class mark to keep it, otherwise mark as deprecated it if it has not been marked to keep it

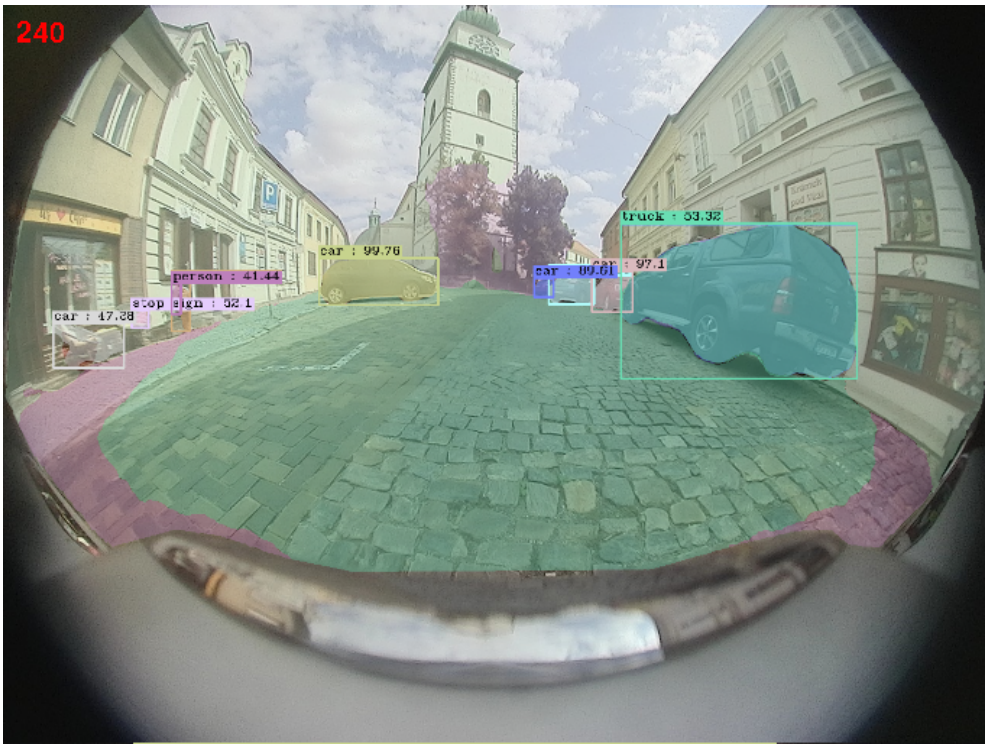


Figure 4.1: Example of panoptic merge, the most left car (not actually car) with score 47% which has building overlapping as background will be deprecated, the person, if semantic is person too will be kept despite low score

4.2.3 Random Custom Bugs

There are some random bugs whose patterns have been learned and the matching masks with same class are systematically removed from dataset in all experiments, for example this light-person bug, which occurs on side of cameras under certain lightning conditions as shown on image to the right.

4.3 Training

The important part of every deep-learning project is the training phase (also called learning), the process to create dataset was described in previous sections and this section will descirbe the training process itself.

In section (2.3) the dataset sections were defined. Each pipeline (process) of training consist of training, validation and testing phase. The training phase describe process of learning model on annotated data with ground truths, with chosen hyper-parameters for training. The hyper-parameters are variables which influence the speed of learning, magnitude of learning and other special settings to improve learning.

There are two popular methods of training NN.

- Learning from random initialization is the standard method, the NN is learning from scratch, this process takes longer time to learn, however it may sometimes bring better results.
- Fine-tuning from pre-trained model is another approach, the benefits are shorter time to learn and more consistent results, however the model might inherit errors of its basic configuration. This approach was chosen for training on automatically generated data, due to possible lower quality of dataset, to extend standard working models to fisheye distorted images.

4.3.1 Training process

Deep-learning neural networks are trained using the Gradient Descent algorithm [26]. The training of neural network consist of two phases:

1. **Forward phase** where input is processed trough network, this phase is almost the same as inference, however all the outputs of all layers are saved for second phase
2. **Backward phase** where the output is compared to ground truth and the error is calculated by loss function and backpropagated through the layers of network to improve the internal model parameters (weights and bias), each layer will receive gradient of loss with respect to its outputs and and return the gradient of loss with respect to its inputs

Loss function is a function used in optimization problems. The loss function given predictions and ground truths computes error value of predictions and in the training task the aim is to minimize this error.

Learning rate is a hyper-parameter influencing the magnitude of learning potential from each input, learning rate is from interval $[0, 1]$, usually closer to 0, value too large might cause overlearning, state of model, where it starts to focus its performance towards particular case and loses its robustness towards other cases.

Batch size is a hyper-parameter that defines the number of samples to work through before updating the internal model parameters. There are three batch methods used in Gradient Descent.

- **Batch Gradient Descent** Batch size = size of training set
- **Stochastic Gradient Descent** Batch size = 1
- **Mini-Batch Gradient Descent** 1 < Batch size < size of training set

The Mini-Batch is the most popular nowadays, since it provides best balance between training time and performance. The sizes of Mini-Batch are usually determined by hardware architecture of machine which computes the training process, this work used Batch size = 8 per GPU (2× GPU = 16), the Batch size was mostly optimized for the time speed of training to be able to try more experiments.

Epoch describes iteration through whole dataset. The number of epochs hyper-parameter is deeply linked with the learning rate, while lowering number of epochs it might be necessary to increase learning rate to get similar results.

One epoch in COCO [49] 2017 dataset has approximately 118k training images the Detectron 2 [29] models are trained with ~37 COCO epochs (4366k iterations of images) or ~12 COCO epochs (1416k iterations) for quick research iteration, one epoch of training dataset from this work has 493k iterations of images (~4 COCO epochs).

4.3.2 Validation process

In the validation process, the validation part of dataset is used to tune the hyper-parameters, the validation is similar to evaluation (4.4), but does not use the testing dataset. Validation was performed after every 100k iterations, to decide if the training should continue, it was used only to find hyper-parameters, most experiments were then used with same parameters to have the equality of opportunity.

The validation process:

1. Divide validation set (100 scenes) into 10 groups
2. Training of 100k iterations of images
3. Evaluate 1 random group if results are better record results for this group continue with training
4. Start evaluating all the groups evaluated before, if any one of them gets better result continue training otherwise stop

This method was used to estimate 4 epochs of training from Detectron 2, the algorithm stopped on 2100k iterations, therefore 4 epochs (1972k iterations) to round the epochs for training balance.

Since YoloV4 was the only experiment with this model, it was spared the cropping after reaching 1600k iterations.

Table 4.7: Experiment ResNet 101

model	Detectron 2 R101 FPN [29] experiments training
learning rate	0.05
batch size	8 per GPU
epochs	4 (1972k iterations)

Table 4.8: Experiment ResNet 50

model	Detectron 2 R50 FPN [29] experiments training
learning rate	0.05
batch size	8 per GPU
epochs	4 (1972k iterations)

To find the right learning rate for fine-tuning the experiments started on half of the originally used learning rate (0.02 to 0.01), however this configuration ended too quickly overfitting, the learning rate 0.001 was too unimpactful, the good balance was achieved on quarter of originally used learning rate.

Similar to the Detectron 2 experiment, YoloV4 was trained with the fine-tuning approach, however the pre-trained models for Yolo were too bad at performance, the learning rate was set just slightly lower than standard training (0.0013 to 0.001) the model took over 3 epochs before starting to overfit.

Table 4.9: Experiment YoloV4

model	YoloV4 [25] training
learning rate	0.001
batch size	16
epochs	3+ (1600k iterations)

4.3.3 Training Tracker

With tracking annotations which were mined from dataset 4.2.1, the Re-Id 1.2.3 part of tracker is trained to differentiate between two feature embeddings.

The difference is calculated with loss function [26], which compares feature embeddings. For this work, the Triplet Loss was selected [51].

For Triplet Loss function, the first step is to create triplets Anchor, Positive and Negative. The Anchor is the object which is actually being processed in learning phase. The Positive is the same object as Anchor, however in different time frame of the track. The Negative is any object, which is not the same object as Anchor (and Positive). There are important relations between these three elements. Anchor is close in terms of similarity to Positive, meaning there will be small distance between A and P, the distances used are usually Cosine similarity, Euclidean distance [51] or Mahalanobis [18]. On the other hand Negative is distant from Anchor.

$$\mathcal{L}\left(A, P, N\right) = \max\left(\| \mathrm{f}\left(A\right) - \mathrm{f}\left(P\right) \|^2 - \| \mathrm{f}\left(A\right) - \mathrm{f}\left(N\right) \|^2 + \alpha,\ 0\right) \tag{4.1}$$

Equation (4.1) describes Triplet Loss function, where f is feature embedding and α is margin between P and N. The goal for learning is to optimize the loss function for Re-Id feature generation in way, such as A and P are close, A and N are distant.

The strategy to create triplets from mined Tracks is Batch hard also from article [51]. In mining the triplets are measured by their loss and categorized as easy - loss is 0, hard loss is greater than margin and semi-hard - loss is inside margin. The Batch hard strategy selects for each A the hardest P and the hardest N to create triplets for learning.

4.3.4 Data augmentations

Data augmentations are very popular method to achieve better results with smaller datasets, the method uses images from original dataset and tries to augment the image by resizing, rotations, crops and other creative ideas.

Despite augmentation being recommended in most of the CNN training tasks, it was after **abandoned after few unsuccessful attempts**. The theory for bad success on this dataset is the fisheye distortion, after trying out some manual attempts for fisheye examples the augmentation does not make that much sense, since each position of fisheye distorted image has its own unique way of modifying object, when resizing the image breaks the rules of distortion, the same with rotations.

The undistorted version did not have attempts that much unsuccessful, but not particularly improving, the measure of improvement was decided from training loss.

4.3.5 Uncertain instance blocking

The experiment panoptic merge Sec. (4.2.2) was extended into this experiment of instance blocking using mentioned deprecated instances. The uncertain instance blocking means that the deprecated instances, have potential to influence the training in negative way and the training decides to ignore these instances. These deprecated instances reintroduced as ”dontcare” instances are instances with low score from automated annotation generation or undesired class to be deleted. There are two ways of dealing with these instances, either delete them or keep them as instances which both exist and do not exist.

When prediction does not match dontcare instance or place where it was deleted, then there is no effect, their benefit applies in case when the prediction actually match dontcare instance or place where it was deleted, because if it was deleted (and it is assumed that there might be something other than background possibly even desired class), it is learned incorrectly as background an possibly harming trained model, however if there is dontcare instance, gradient learning completely ignore fact that this prediction ever happened, in summary dontcare instance exists yet it does not.

Obviously this way reduces data the same way as deletion, even so the benefit to not propagate possible desired classes as background is very powerful as seen in results 4.2.

4.3.6 Class balancing

When training it is desirable to have balanced classes distribution, this way the model does not start "forgetting" any class during learning process. Unfortunately this dataset have some classes like train, bus, motorcycle and bicycle, which are very rare and augmentation is not really working out to upsample these classes. The remaining solution is to boost the loss for these particular classes to improve their learning contribution. To try to avoid boosting uncorrect generated instances, only loss of ground truths with score > 0.7 is $3\times$ boosted, for score > 0.8 is $5\times$ boosted and for score > 0.9 is $10\times$ boosted. The truck class have also low count of samples, however this class usually gets confused with cars and boosting trucks rate could overthrow balance between these two classes.

Table 4.10: Experiment Dontcare

model	Detectron2 X152 FPN + R101 FPN (panoptic) [29] GT
method	panoptic merge, class balancing, uncertain instance deleted
score	★★★★★★★★★☆

Table 4.11: Experiment Purge

model	Detectron2 X152 FPN + R101 FPN (panoptic) [29] GT
method	panoptic merge, class balancing, uncertain instance blocking
score	★★★★★★★★★☆

4.4 Evaluation

Important part of any machine-learning or deep-learning project is properly chosen evaluation method for results. Most of the nowadays object detection and instance segmentation projects use well established evaluators assigned to the datasets being evaluated, such as COCO [49] or Pascal VOC [52].

Likewise automatically generated Valeo dataset 2 requires evaluation methods focusing on problems which come with automated annotation generation. The evaluation methods are based on established baseline calculating confidence threshold precision-recall metric with average and mean features which is used by COCO [49], Pascal VOC [52] and many other evaluators, however two major features are introduced: class mapping and Gaussian localization error evaluation 4.4.10.

Classification differentiates the classes of objects, rather than multi-class classification, multiple binary classification instances for each class are used, since most of the evaluation methods (e.g. precision, recall) are designed for binary classification and the class group sizes are mostly imbalanced, which makes it harder to balance these evaluation methods [26].

For utilizing binary classifier each class is evaluated separately for binary parameters: is $class_i$ and is not $class_i$, for $class_i \in classes$. The upcoming detection and ground truth terms will describe each class as separate problem unless it is specified otherwise.

Localization is measured by the overlapping areas of the objects, the fastest and the most popular approach is to use bounding boxes, however mask overlapping might provide much better results on clustered objects, especially on fisheye distorted images, where bounding boxes might cover less than half of area e.g. on Fig. 0.5, the disadvantage of masks are longer computation speeds.

True Positive (TP) defines correct detection matching ground truth by localization metric e.g. IoU.

False Positive (FP) represents wrong detection, either not matching any ground truth labeled as misdetected FP or scored with localization metric below threshold labeled as supressed FP (also case if outmatched by other detection).

False Negative (FN) specify ground truth annotated sample not matched with any detection.

True Negative (TN) represent every part of image which was correctly not detected, which is usually hard to interpret, it can be defined as pixels, which were correctly not detected, or some another unit representing image space not detected, for two stage detector, the anchors Sec. 1.1.2 might be good candidate.

4.4.1 Perfect matching

While matching detections and ground truths, two or more detections might be considered as TP candidates, furthermore some of these detections might have even better matching with another ground truths, therefore it is necessary to match results as good as possible.

To solve this assignment problem of maximum (for IoU) bipartite graph matching the Hungarian Maximum Matching Algorithm [53] is used. This algorithm finds perfect maximum matching between detections and ground truths and requires adjacency matrix as input, each adjacency means matching between detection and ground truth instance in form of localization metric, which is IoU (no matching: $IoU = 0$) and therefore the matching is being maximized for the best result. The final result of the algorithm are TP candidates (might be denied by low IoU), while the rest of instances are considered FP for detections and FN for ground truths.

4.4.2 Class mapping

In the manually annotated dataset it is assumed that all ground truths have correct class, therefore it is meaningful to evaluate matching on each class separately. However automatically annotated dataset might sometimes see only lower part of truck in image and classify it as car, this results in these classes to more likely influence each other in training and then cause some class

mismatches in testing. This is unavoidable negative part of automated annotations, which might lead into poor evaluation results, with low background knowledge on reasons for such performance.

Naive improved solution is to create all class matching at once, which is on the other hand too misleading, because it is different when car is detected as truck or van than car being detected as person, which is much worse case.

The solution is class mapping which allows each class ground truths to accept not only the detections of the same class but also detections of another classes specified in class mapping.

The mapping is usually decided by class functionality in real world, for example it is reasonable to map vehicles, but leave person as separate.

The current mapping used in Valeo dataset:

Ground truth	Detection
Person	Person
Bicycle	Bicycle, Motorcycle
Motorcycle	Motorcycle, Bicycle
Car	Car, Truck
Truck	Truck, Car, Train, Bus
Train	Train, Truck, Bus
Bus	Bus, Truck, Train
Traffic light	Traffic light

The reason for mapping the bicycle and motorcycle is because generating detector has problems with differentiating between these classes, since they might be sometimes challenging even for human from longer distance or low knowledge of exact bike model.

The car and truck overlap because most of the detectors classify van (might be seen as small truck) as a car, however truck when seen as big truck might overlap with train and bus, although there might be case where car is a train, these cases are not that desirable to tolerate their misclassification, therefore the mapping does not include them.

The person and traffic light are more unique cases thus there is no mapping to them.

The last problem with class mapping is modified **prefect matching** to ensure no instance is repeatedly considered as any TP, FP or FN. To address this problem, the matching is considered for each cluster of classes instead of each class separately.

To obtain cluster of classes, simple depth first search (DFS) [41] is used to get graph components which describe all classes which have some relation to any class in this group for aforementioned mapping one such cluster would be "Car, Truck, Bus, Train".

For each cluster all matches are recorded in adjacency IoU table, however to disable mapping of unmapped adjacencies, for each class c in cluster all unmapped classes ($cluster - map[c]$) adjacencies are filled with zeros in adjacency IoU table. With this modification perfect matching will result in only mapped prefect matching for all classes in cluster.

It is worth mentioning that if cluster consisted of all classes, result would be same as previously mentioned naive improved solution, clustering allows problem to be divided into smallest possible subproblems to be computed more efficiently due lowered matching between cross class instances.

Python pseudocode 4.2 shows general implementation of matching algorithm, which takes detections and ground truths as input, as well as class clusters (if there is no class mapping each cluster equals each separate class) and returns instances of true positives, false positives and false negatives, which are used in evaluation methods precision and recall.

```
def match_frame(gt_frame, # gt instances in frame
                det_frame, # det instances in frame
                clusters): # list of class clusters

  for cluster in clusters:

      gt = []
      det = []
      # data interval separator of classes in cluster
      gt_sep = [0]
      det_sep = [0]

      for c_i in cluster:

          gt  += [x for x in gt_frame.instances if x.cls is c_i]
          det += [x for x in det_frame.instances if x.cls is c_i]
          gt_sep.append(len(gt))
          det_sep.append(len(det))

      # get IoU adjacency matrix
      iou = match(gt, det)

      # hungarian algorithm need square matrix
      iou = square_matrix(iou)

      for i, c in enumerate(cluster):

          if gt_sep[i] < gt_sep[i+1]:
              # get set of classes to be unmapped
              to_zeros = [cluster.index(x) \
              for x in (cluster - class_map[c])]

              for x in to_zeros: # fill unmapped with zeros
                  if det_sep[x] < det_sep[x+1]:
                      zeros(iou[gt_sep[i]:gt_sep[i+1], \
                            det_sep[x]:det_sep[x+1]])

      # perfect matching
      TP = maximum_hungarian_algorithm(iou)
      # detections which are not in perfect matching
      FP = det - TP
      # ground truths which are not in perfect matching
      FN = gt - TP
```

Figure 4.2: Simplified frame matching algorithm

4.4.3 Evaluation metrics

Following metrics are meant to be used for each class separately.

Accuracy is the percentage of correctly predicted examples out of all predictions (class detections but also background).

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.2}$$

Because Accuracy uses TN which was described as problematic, this metric which is popular in other machine-learning tasks **is not reliable metric** for object detection and instance segmentation, due to inconsistency of TN value which might easily outshadow other values.

Precision is being used to provide rate of sucessful detections.

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{all\ detections\ of\ class} \tag{4.3}$$

Precision scores range from 0 to 1, a high precision implies that most detected objects match ground truth objects. For example if precision is 0.8, then 80% of the time the detector is correct, when an object is detected.

Recall express the ability of a model to find all the relevant cases (all ground truth bounding boxes).

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{all\ ground\ truths\ of\ class} \tag{4.4}$$

Recall ranges from 0 to 1 where a high recall score means that most ground truth objects were detected. For example recall is 0.6, then the model detects 60% of the objects correctly.

4.4.4 Precision-Recall curve

Usually with precision and recall there can be scenarios where precision is perfect and recall is miserable, that means detections were correct but there were not enough detections, likewise if recall is prefect and precision is miserable, then all ground truths have been matched by detections, however there were too many extra detections ruining precision.

To better understand trade-off between precision and recall the Precision-Recall (PR) curve - yellow line on Fig. 4.3 is very popular way to evaluate object detector performance. Models that involve confidence score can trade-off precision for recall by adjusting the level of confidence needed to make a prediction. If the model is in a situation where avoiding false positives is more important than avoiding false negatives, it can set its confidence threshold higher to encourage the model to only produce high precision predictions at the expense of lowering its amount of coverage (recall).

The PR metric is calculated by iterative process over sorted detection instances by confidence score, which allows detector to emphasise his confidence in detected object throughout evaluation of PR metric.

On each iteration precision and recall are calculated with cumulative sum of TP, FP and FN.

Throughout the process of the PR curve, it is desirable to keep high precision score, on high confidence samples it means that detector was well trained to recognize this class, the prefect scenario keep precision 1 while reaching recall 1, which is highly improbable. For decently trained model usually precision decreases with small jagged jumps over iterations of the lowering confidence score with increasing recall.

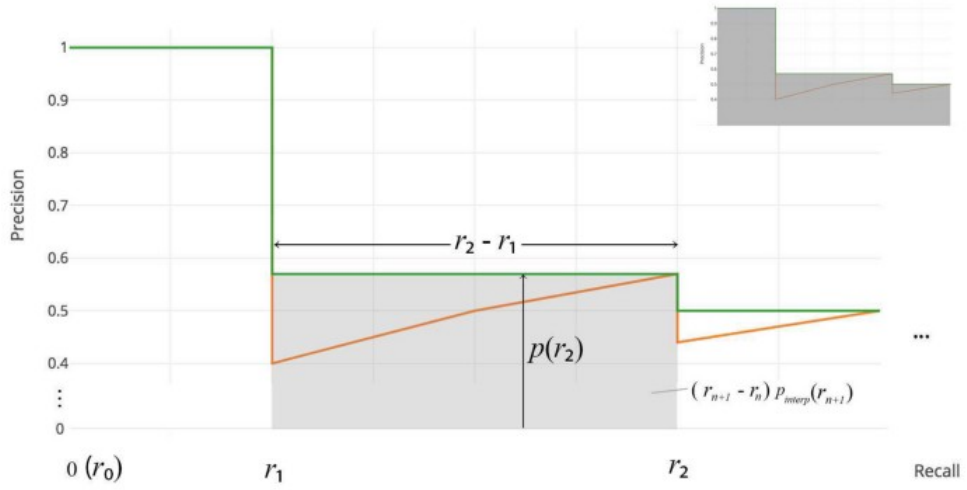


Figure 4.3: Example of computation of smoothened AP from PR curve. [54]

4.4.5 Average Precision

PR curve is very good metric to analyze performance on classes, however for faster interaction with evaluated observations, it is very convenient to group PR curve into some scalar value for smoother interpretation. This method is called Average Precision (AP), the general definition is that AP is the area under the PR curve $p(r)$ where p is precision and r is recall.

$$AP = \int_0^1 p(r)dr \tag{4.5}$$

Since the PR curve might have jagged pattern caused by random misdetection streaks, most SOTA evaluators **smoothen** PR curve to be less suspectable to small variations in the confidence ranking. Smoothening is a form of interpolation in which the precision value for recall $\widetilde{r}$ is replaced with the maximum precision for any recall $\geq \widetilde{r}$, resulting in monotonically decreasing smoothened PR curve.

$$p_{interp}(r) = \max_{\widetilde{r} \geq r} p(\widetilde{r}) \tag{4.6}$$

To calculate smoothened AP it is necessary to find all occurences of maximum precision value droping beforehand and write them down as interval edges $\{r_1, r_2, ...\}$ Then the aforementioned interpolation equation is redefined as:

$$p_{interp}(r_{n+1}) = \max_{\widetilde{r} \geq r_{n+1}} p(\widetilde{r}) \tag{4.7}$$

$$AP = \sum (r_{n+1} - r_n)\ p_{interp}(r_{n+1}) \tag{4.8}$$

Efficiently calculating sum of square area of each interval as shown in Fig. 4.3 where yellow line is PR curve and green is smoothened PR curve.

4.4.6 Mean Average Precision

To further generalize model performance it is possible to obtain mean average precision (mAP) from all AP across all N classes:

$$mAP = \frac{1}{N} \sum_{i=0}^{N-1} AP_i \tag{4.9}$$

4.4.7 Pixel TN

True negatives in image processing are much more complex than in most of the other machine learning disciplines. It is hard to define one unit of background compared to one instance of object, the solution in this case was to use pixel as the units defining size of all the objects in image and also the backround (TN).

Through process of evaluation TP, FP and FN are obtained with prefect matching 4.4.1, in each image frame the rest of pixels which are not TP, FP or FN are considered TN. To ensure that the computation of TN is precise by subtracting TP, FP and FN, it is necessary to ensure that all masks of detections (TP + FP) do not overlap with other detections, which would cause some pixels to be counted twice. The ground truths use the same rule, however ground truths should never be overlapping in the first place.

To fix possible detections overlaps, the detection masks are being added to image in one iteration and each detection cannot claim pixels which have already been allocated by another detection mask. The order of detections in the iteration of adding masks is very important, since it is desirable to keep masks of the best detections with good IoU from matching. To achieve this, all detections are sorted by IoU, where the detections with high IoU are added first. The TP, FP and FN keep the sizes of cropped masks for following evaluations with TN.

4.4.8 False Positive Rate

False Positive Rate (FPR) also known as false alarm rate is proportion of wrong detections FP to all negative instances in binary classification, which in this case are FP and TN (background).

$$FPR = \frac{FP}{FP + TN} \tag{4.10}$$

The pixel TN value is usually much larger than pixel FP since, there is lot of background pixels on image, therefore results of the pixel FPR are very small numbers. The following mentions of FPR will be meant as pixel FPR.

4.4.9 Receiver operating characteristic

Receiver operating characteristic (ROC) is trade-off between True Positive Rate (TPR) which is another term for previously defined Recall and False Positive Rate (FPR). The ROC curve for pixel TPR and FPR is computed similarly to PR curve, it is being calculated by iterating over sorted detections computing FPR and TPR for each step. The examples of ROC curve can be seen in 4.5 and in experiments. The implementation will be shown with Gaussian error included 4.4.12.

4.4.10 Gaussian localization error

With automated generation of bounding boxes and masks it is expected for boxes to have small localization (affecting IoU) error being slightly shifted or having different size than a bounding box manually placed by an annotator. The mask also suffers the same errors also with possibility of malformation of shape. To propagate these errors in final evaluation this work presents Gaussian localization error to describe the uncertainty of automatically generated bounding boxes and mask instances.

Gaussian distribution from probability theory is a type of continuous probability distribution for a real-valued random variable. This distribution is usually used for random values whose distribution is not known, therefore its application is also very popular in many problems outside computer science fields, where is commonly known as "normal distribution".

Because annotation localization error is caused by detections of deep neural network model, and therefore the distribution of error in detection is undefined, this distribution is the best choice for modeling localization error into evaluation.

The Gaussian distribution has important parameters mean μ and standard deviation σ, which influence distribution shape.

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{4.11}$$

Annotation error Let λ_i be the annotation detection of object i created by model, which was used to automatically generate annotations (in this case 4.2).
Let α_i be annotation of the object i which is 100% accurate (perfect mask, bbox). Then the annotation error δ is calculated as normalized (by 4.15) euclidean distance between centers of mass [6] of these objects (4.16). The Gaussian localization error can be estimated from accumulated annotation errors.

By definition of **Central Limit Theorem** [5] the Gaussian localization error would be more precise with increasing amount of objects i and their respective annotation errors δ. This work uses 50 manually annotated objects for each class to collect δ errors from which the Gaussian error is estimated for each class separately.

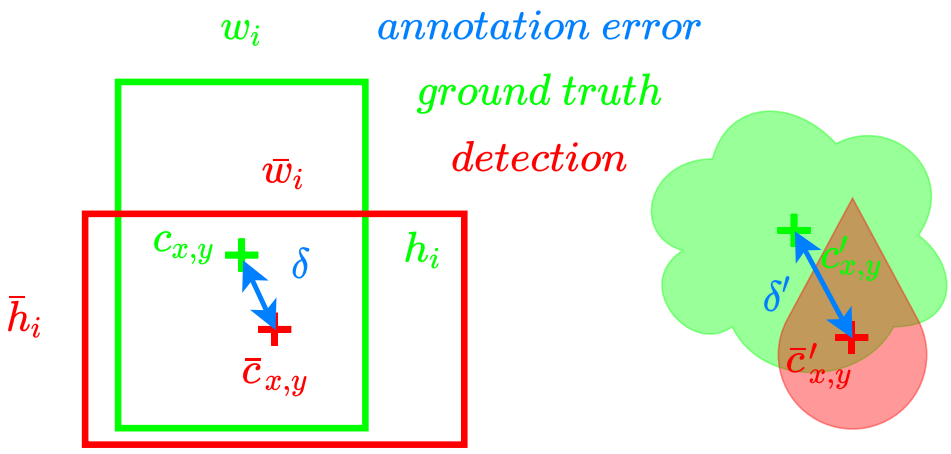


Figure 4.4: The annotation error

The center of mass (4.12, 4.13, 4.14) can be used for both bounding boxes and masks.

$$M_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - \bar{x})^p (y - \bar{y})^q f(x, y)\, dx\, dy \tag{4.12}$$

For $p, q = 0, 1, 2, ...$ adapting to digital image $f(x, y)$ with pixel intensities the previous equation becomes:

$$M_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q f(x, y) \tag{4.13}$$

The center of mass can be obtained from moments with following formula:

$$c_x,\ c_y = \left\{ \frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}} \right\} \tag{4.14}$$

$$c_{max} = \max_{w_i, h_i, \bar{w}_i, \bar{h}_i \ \in \ bounding\ boxes\ of\ objects} \tag{4.15}$$

$$\delta = \sqrt{\left(\frac{|c_x - \bar{c}_x|}{c_{max}} \right)^2 + \left(\frac{|c_y - \bar{c}_y|}{c_{max}} \right)^2} \tag{4.16}$$

After accumulating normalized annotation errors δ_i for all objects i the data are used to generate Gaussian distributions by fitting these data with Gaussian.

For computations with this distribution it is necessary to cut the tails which have too low information value to justify their memory consumption in implementation, for this cut 3-sigma (σ) rule is used, this rule states that Gaussian distribution has approximately 99.73% information within area of 3 lenghts of standard deviation to each direction from mean value (center of distribution).

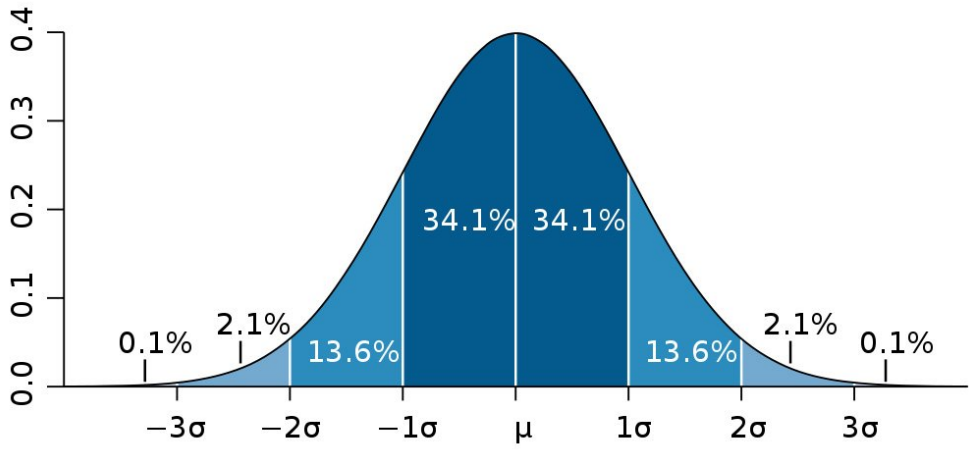


Figure 4.5: The 3-sigma rule from [55]

The obtained distributions are now represented with μ and σ, to obtain error data usable with evaluation it is necessary to get array which represents areas of distribution in bins, the bins are mapping unit to IoU, optimal choice is to use width of each bin corresponding to 1% recall for balance between precision and computational performance.

For faster understanding it is recommended to explore graphs of results with IoU distributions 4.5, where it can be seen how each IoU is applied with Gaussian error 4.6.

To obtain error area for each bin, the Cumulative Distribution Function (CDF) [5] is applied on 3-sigma area of distribution and binned to 1% bins, this method is representing actual computation using python [41] functions. CDF also rescales the total area to $[0, 1]$ which is important to simulate contribution of instance, prefectly annotated instance would have only one bin with value 1 (100%), as in example on result data 4.5.

For example class **car** with Gaussian error distribution $\mu = 0.031$ and $\sigma = 0.019$ will have 3-sigma binned CDF:

$$[0.007, 0.027, 0.083, 0.198, 0.379, 0.591, 0.779, 0.904, 0.967, 0.991, 0.998]$$

To get areas from CDF each bin except first one subtract his left neighbour bin.

$$[0.007, 0.021, 0.056, 0.115, 0.181, 0.212, 0.188, 0.126, 0.063, 0.024, 0.007]$$

The result array is Gaussian localization error which will be used in Gaussian PR, the sum of this array is approximately 3-sigma rule value, the error should be symmetric from center of array, in this example machine precision makes this slightly inaccurate, however it is symmetrical with certain tolerance as it can be seen on graphs Fig. 4.6, 4.7.

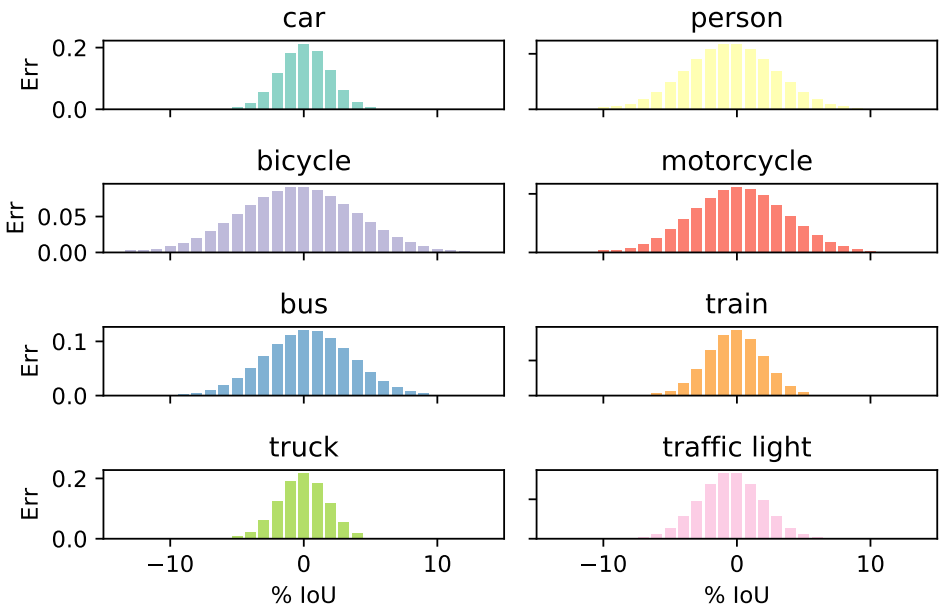


Figure 4.6: Bounding box Gaussian error

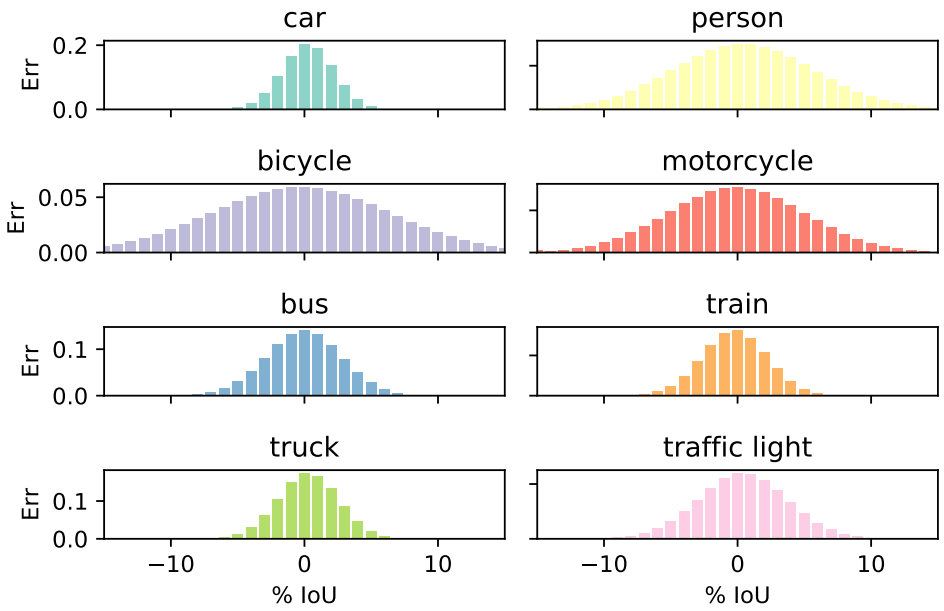


Figure 4.7: Mask Gaussian error

From collected errors on Fig. 4.6, 4.7, there is evidence that cars have the lowest Gauss Localization Error and classes such as person, bicycle and motorcycle have the highest error, this information is not unforseen, since the person masks are always very complex and bikes usually are very thin and sensitive to segmentation, also sometimes person rides bike, which can make segmentation even more chaotic.

4.4.11 Gaussian Precision-Recall curve

Previous section explained process of obtaining Gaussian localization error, this section will describe how to use it in evaluation of PR curve. The PR curve decides if the match with non-zero IoU of detection and ground truth should become TP or FP. The quality of annotation (localization error) influences this part heavily, since one percent of IoU can make great difference if value falls below threshold and becomes FP.

However with Gaussian localization error as the collected distributions Fig. 4.6, 4.7, the cumulative sum of TP and FP can be represented array with length 101 working as histogram for collected IoUs obtained during iterative PR curve process, which are added to histogram on the index that is IoU of match (with center of Gaussian to index).

The following code Fig. 4.8 shows implementation of Gaussian PR, theres small modification for time performance, to avoid heavy code branching for cases of adding Gaussian on sides of histogram (e.g. match with IoU 100% could overflow with right half of Gaussian), therefore the length 101 is increased by length of Gaussian to represent overflow areas below 0 Iou (misdetected FP) and over 100% (TP), this modification needs to remember offset to IoU point 0 which is represented as iou_0 in the code.

The result of code is standard PR curve with calculated Gaussian localization error.

4.4.12 Gaussian Receiver operating characteristic

The Gaussian ROC uses the Gaussian localization error in the same way as Gaussian Precision-Recall, the Recall (TPR) is calculatedin the same way, however there is new input variable all pixels, which is sum of all pixels across all images in dataset, from this value the TN is calculated by subtracting TP, FP and FN.

The final values are always reduced by sigma3rule which is the maximum amount which Gaussian error distribution can contribute 4.5.

The pseudocode for Gaussian ROC is on Fig. 4.9.

4.4.13 Gaussian Average Precision

Gaussian Average Precision (gAP) has the same definition as AP 4.4.5, with the difference that the gAP source data are obtained from Gaussian Precision-Recall implemented in Fig. 4.8 instead of standard Precision-Recall.

4.4.14 Mean Gaussian Average Precision

Mean Gaussian Average Precision (mgAP) has the same definition as mAP 4.4.6. The class data used are gAP 4.4.13 instead of AP.

```python
def gaussian_precision_recall(matches, # data table with TP, FP, FN
                              gauss_loc_err, # Gauss localization errors
                              iou_threshold):

  precision           = {cls:[] for cls in classes}
  recall              = {cls:[] for cls in classes}
  # cumulative sum + tails
  false_true_positive = {cls:zeros(101+len(gauss_loc_err[cls])) \
                         for cls in classes}
  # FP which didnt hit any GT
  false_positive_miss = {cls:0 for cls in classes}
  ground_truth        = {cls:len(gts from matches) \
                         for cls in classes}
  sigma3rule          = {cls:sum(gauss_loc_err[cls]) \
                         for cls in classes}
  # tail offset for cumulative sum array
  iou_0               = {cls:int(len(gauss_loc_err[cls]) / 2) \
                         for cls in classes}

  for (gt_class, det_class, score) \
      in sorted(detections from matches, key=score):

      cls = det_class

      if iou == 0:
          # false positive missed all gt,
          # reduced by 3 sigma rule sum ~ 99.7%
          false_positive_miss[cls] += sigma3rule[cls]
      else:
          if cls != gt_class:
              cls = gt_class # mapped detection, added to gt class
          # add gausssian into detection cumulative sum (TP + FP)
          # works as array addition with IoU offset
          false_true_positive[cls] \
              [int(iou*100):int(iou*100) + len(gauss_loc_err[cls])] \
                  += gauss_loc_err[cls]

      # TP is everything above IoU threshold + tail offset
      TP = sum(false_true_positive[cls] \
          [int(iou_threshold*100) + iou_0[cls]:])

       # TP / (TP + FP)
      precision[cls].append( \
          TP / (sum(false_true_positive[cls]) + false_positive_miss[cls]))

      # TP / (TP + FN), ground_truth is reduced by 3 sigma rule
      recall[cls].append( \
          TP / (sigma3rule[cls] * ground_truth[cls]))
```

Figure 4.8: Simplified Gauss Precision Recall

```
def gaussian_roc(matches, # data table with TP, FP, FN
                          gauss_loc_err, # Gauss localization errors
                          iou_threshold,
                          all_pixels):

  false_positive_rate = {cls:[] for cls in classes}
  true_positive_rate  = {cls:[] for cls in classes}
  false_true_positive = {cls:zeros(101+len(gauss_loc_err[cls])) \
                         for cls in classes}
  false_positive_miss = {cls:0 for cls in classes}
  ground_truth        = {cls:sum(fn_area of gts from matches) \
                         for cls in classes}
  sigma3rule          = {cls:sum(gauss_loc_err[cls]) \
                         for cls in classes}
  iou_0               = {cls:int(len(gauss_loc_err[cls]) / 2) \
                         for cls in classes}

  TN = all_pixels - sum(tp_area, fp_area, fn_area from all in matches)

  for (gt_class, det_class, score, tp_area, fp_area, fn_area) \
      in sorted(detections from matches, key=score):

      cls = det_class

      # same as PR but multiplied by area
      if iou == 0:
          false_positive_miss[cls] += sigma3rule[cls] * fp_area
      else:
          if cls != gt_class:
              cls = gt_class # mapped detection, added to gt class
          false_true_positive[cls] \
              [int(iou*100):int(iou*100) + len(gauss_loc_err[cls])] \
                  += gauss_loc_err[cls] * tp_area

      # TP is everything above IoU threshold + tail offset
      TP = sum(false_true_positive[cls] \
          [int(iou_threshold*100) + iou_0[cls]:])

      FP = false_positive_miss[cls] + sum(false_true_positive[cls] \
          [:int(iou_threshold*100) + iou_0[cls]])

       # FP / (FP + TN)
      false_positive_rate[cls].append( \
          FP / (FP + sigma3rule[cls] * TN))

      # TP / (TP + FN), ground_truth is reduced by 3 sigma rule
      true_positive_rate[cls].append( \
          TP / (sigma3rule[cls] * ground_truth[cls]))
```

Figure 4.9: Simplified Gauss Receiver Operating Characteristic

4.4.15 Tracking Evaluation

For evaluation of tracking the Multi-Object Tracking (MOT) and Segmentation [56] project was used. The MOTS project defines very good baseline for tracking evaluation in this work.

The main metrics in MOTS are MOTS Accuracy (MOTSA) and MOTS Precision (MOTSP).

$$MOTSA = 1 - \left(\frac{FP + FN + IDS}{TP + FN} \right) \tag{4.17}$$

In equation 4.17 The TP, FP and FN are known from detection evaluation 4.3 and the IDS are ground truths, whose first predecessor was tracked with different track id, meaning that the Tracker lost the old track and started new Track (unless it has returned to the correct track).

$$MOTSP = \frac{\sum_{t \in TP} IoU_t}{TP} \tag{4.18}$$

The MOTSP is quite straightforward, this metric measure how precise was the prediction in regards to all TP values with their respective IoUs.

MT (Mostly Tracked), these are all Tracks which were tracked at least 80% of their lifetime.

ML (Mostly Lost), these are all Tracks which were tracked at most 20% of their lifetime.

4.4.16 Multi-Camera Multi-Object Tracking

The multi-camera version is evaluated by modified MOTS [56] for multiple cameras.

From all track detections which are affiliated with objects of ground truth multi-track, group of detection tracks with matching track id must be chosen as alpha track, which is the correct matching of detection multi-track to ground truth and all the tracks with id not matching alpha track are considered mismatches. In the best scenario all the track detections are in group matching id with the alpha track. Since the choice of alpha track is important for final evaluation, the alpha track is a group of detection tracks with matching id across all cameras, which has the most track id instances matched to multi-track ground truth across all time frames - images in video sequences (concept similar to perfect matching of IoU 4.4.1).

The single camera MOTS uses the first predecessor track as alpha track and all identity switches are resolved by comparing track id to this alpha track. Therefore the multi-camera MOTS will keep all aspects of the original MOTS metric, the multi-camera MOTS can achieve at most the same score as MOTS across all cameras evaluated without multi-track. That would be scenario when all tracks are perfectly matched across all the cameras, including all identity switches occurring simultaneously across all the cameras. In the most scenarios the multi-camera MOTS will have worse results since increasing the complexity of matching across all the cameras and also possibility of track on camera, which does not match the alpha track along its whole lifespan.

The MOTSA (4.17) is influenced by the change of identity switches affecting variable IDS. The difference to single camera MOTSA is, that each track may not start as alpha track, therefore it can be counted towards IDS from beginning of the track. The conversion to correctly matched alpha track is valid, since even in single camera version, the track can switch identity and return back to the alpha track identity later.

MOTSP (4.18) is not influenced by multiple cameras.

The multi-camera MOTS was resolved with minimal changes to original algorithm with introduction of alpha track and is still viable for comparison with single camera MOTS. The only implementation changes are finding the alpha track and setting zero predecessor (track with no time frames) instead of first predecessor (4.17) of track as alpha track.

4.4.17 Gaussian localization error in tracking

The Gaussian localization error in MOTSA (4.17) is defined by lower and upper error bound of accuracy with TP and FP uncertainty σ Sec. 4.5 which is standard deviation of Gaussian of accumulated Gaussian errors for PR 4.5. From the 4 combinations of all σ uncertainties, the min and max are chosen as the error interval.

$$MOTSGA_{Error} = 1 - \left(\frac{FP \ \pm \sigma_{FP} + FN + IDS}{TP \ \pm \sigma_{TP} + FN} \right) \tag{4.19}$$

Since MOTSP (4.18) uses only TP, the Gaussian localization error is simply defined by previously mentioned standard deviation σ applied on MOTSP to get the error lower and upper bound interval.

$$MOTSGP_{Error} = MOTSP \pm \sigma_{TP} \tag{4.20}$$

4.5 Introduction to graph system

TP uncertainty is variable reliable on Gaussian localization error, it is calculated from TP IoU. The uncertainty is defined as standard deviation of Gaussian fitting all TP IoUs, which can be seen as purple Gaussian fit in IoU graph Fig. 4.11 calculated from all TP in dataset. The TP uncertainty shows possible margin of error of TP detection.

FP uncertainty is similar to TP uncertainty, it works with all FP, the FP displayed in IoU graph are only those with positive IoU. The FP with IoU 0 usually outshadow FP with non-zero IoU. This creates Gaussian with mean very close to 0 and standard deviation (FP uncertainty) being also small value, therefore the FP uncertainty are not plotted on IoU graphs.

Predicted labeled curves with green color in graph are computed with standard metrics without Gaussian error, these are being used in all Vosstrex evaluations 4.10, since Vosstrex does not use Gaussian localization error, because the annotations are 100% accurate. In the PR graphs Fig. 4.12 the predicted version is also displayed to show difference between Gaussian and predicted curve.

Gaussian labeled dashed curves with blue color in graph are computed with Gaussian versions of metrics 4.8, 4.9, including Gaussian localization error.

There first graph explained Fig. 4.10 is the ROC curve graph with Gaussian localization error 4.4.17. This graph plots the ROC curve with True Positive Rate (TPR = Recall) on y axis with $(0, 1)$ scale and False Positive Rate (FPR) on x axis proportional to all pixels in all images which are scaled to value 1 (100% pixels). The TP uncertainty is calculated in each step of ROC computation, representing error margin of ROC curve.

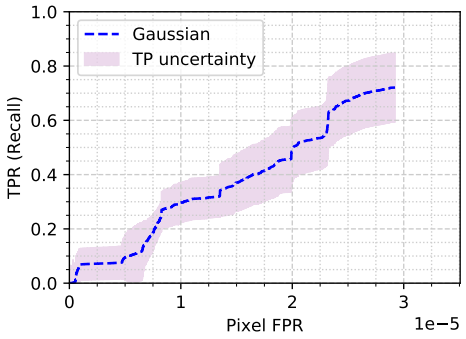


Figure 4.10: ROC curve graph

The second graph Fig. 4.11 is graph describing histogram of TP and non-zero FP values, with discretized IoU with bins of size 1% an x axis and cumulative sum on y axis.

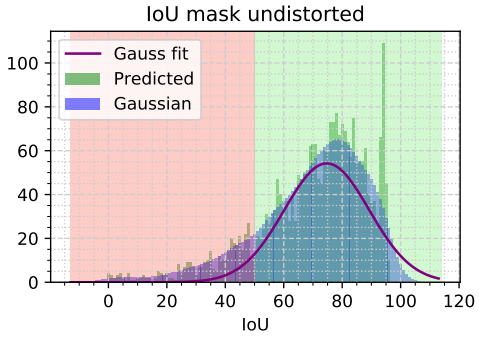


Figure 4.11: IoU distribution graph

The histogram was achieved with code 4.8, the blue bins represent accumulated Gaussians with the help of Gaussian localization error 4.4.10 and the error distribution arrays Fig. 4.6. The green bins describe IoU without error, which have been accumulated with the same method, but with array of size with value 1 (zero error spread). In most cases the bins without error are just for comparison and explanation of what Gaussian version does, however Vosstrex uses only this version because of the perfect annotations.

The light red area below chosen IoU 50% are FP with non-zero IoU. The light green area are TP and the aforementioned Gauss Fit describes the TP uncertainty across only the TP area. The mean value (center) of fitted Gaussian is good indicator how to set strict IoU threshold for inference.

The last graph Fig. 4.12 the PR curve shows trade-off between Precision on y axis and Recall on x axis on scale $(0, 1)$ for both standard metric previously mentioned as Predicted (green) and Gaussian (blue) metric 4.5.

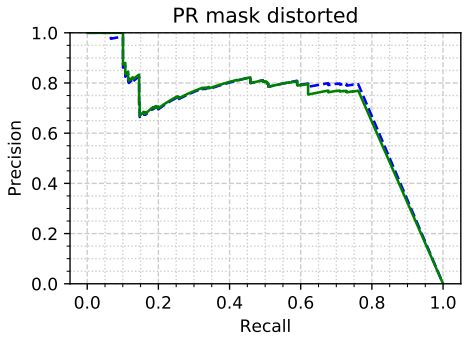


Figure 4.12: PR curve graph

The error margin is not displayed in PR graph since it is already included in IoU graph 4.11, which displays the last step of PR computation with all TP and FP values and the final error margin is standard deviation of Gauss fit.

Another reason is because these graphs were made small so all graphs can be showed at one page for comfortable comparison, therefore the error margins would make the results too overcrowded and confusing.

4.6 The legend to detection models

This small subsection serves as crossroad for all model configurations. The tables use gAP 4.4.13 as metric for separate classes and mgAP 4.4.14 for summary.

The models consist of word or two word tags of the major experiments used to create them. The most common tag is **un** which means that model was trained on undistorted data opposing to distorted which is represented by missing the tag **un**, the difference between these two configurations can be seen in Fig. 0.5. Valid for all experiments.

The tag **seg clean** stands for segmentation clean, it describes that the experiment of 4.2.2 was used. This version has two modifications **dontcare** 4.10 and **purge** 4.11. From experiments [4.8, 4.9.1, 4.9.2, 4.10].

The tag **standard** stands for very low data modification 4.4. From experiments [4.9.2].

The tag **YoloV4** is self explanatory 4.9. From experiment [4.9.2].

There are also tags for backbones 1.1.1 **r50**, **r101** for ResNet 50 respective 101. All experiments.

The rare tag **vanilla** for model which was not modified, in this case Detectron 2 [29] model. From experiment [4.7].

And finally **vosstrex** tag for the VR scenes 2.4. From experiment [4.10].

4.7 Unmodified detector experiment

The first stage of tracking algorithm is detection. The detector must be well trained to successfully recognize objects from image, if there is a case of weak detector, it will most likely influence tracking in negative way, either not providing enough detections or detecting random objects.

To properly start detection, it is necessary to define starting point, and goal. The starting point will be Detectron 2 R101 mask R-CNN publicly available trained model, the goal was to train it to perform on undistorted images and additional task was to try if it is possible to train it to perform on distorted images. The following table with Average Precision per class and mean AP show the performance of ResNet 101 model with configuration 4.7, but not trained on this dataset.

The model configurations and addtional information for following AP table can be found in legend Sec. 4.6.

	person	**bicycle**	**car**	**motorcycle**	**bus**	**train**	**truck**	**traffic light**	**mgAP**
vanilla r101 bbox	0.292	0.243	0.348	0.326	0.237	0.308	0.452	0.436	0.320
vanilla r101 mask	0.273	0.228	0.348	0.304	0.240	0.301	0.457	0.430	0.312
vanilla r101 un bbox	**0.425**	**0.357**	**0.566**	**0.386**	**0.399**	**0.565**	**0.634**	**0.446**	**0.461**
vanilla r101 un mask	**0.410**	**0.346**	**0.565**	**0.367**	**0.401**	**0.563**	**0.634**	**0.438**	**0.454**

Figure 4.13: Results of unmodified detector model

The results show that even the model trained by Detectron 2 [29] on completely different dataset can perform decently on undistorted images, however on distorted data the performance is much lower, this explains why undistorted images were so important for automated annotation generation, to compare staring point model with other models, visit results overview Sec. 4.11.

4.8 Undistorted and distorted experiment

The undistorted detection model works with images from fisheye cameras, which have been converted to undistorted version reducing fisheye distortion Fig. 0.3. The undistorted detection model should be stronger in evaluation on generated data, since the data were generated by this method.

The model and training configurations and addtional information form following AP table can be found in legend Sec. 4.6.

	person	bicycle	car	motorcycle	bus	train	truck	traffic light	mgAP
seg clean dontcare r101 bbox	0.494	0.316	0.601	0.415	0.649	0.639	0.752	0.527	0.549
seg clean dontcare r101 mask	0.470	0.309	0.602	0.390	0.652	0.632	0.758	0.514	0.541
seg clean dontcare r101 un bbox	**0.529**	**0.405**	**0.634**	**0.461**	**0.681**	**0.742**	0.756	0.493	**0.588**
seg clean dontcare r101 un mask	**0.480**	**0.394**	**0.618**	**0.441**	**0.682**	**0.739**	0.758	0.451	**0.570**

Figure 4.14: Results of undistorted and distorted comparison experiment

The distorted detection model completely skips the step of removing distortion, resulting in having little bit of disadvantage, however the distorted detection would be success even if only this model could hold up to the undistorted model, the other configurations of the distorted detection model are the same as the ones the undistorted model use.

The following table describes TP, FP and FN statistics in this experiment. The FP miss is FP which completely missed all GT, the FP hit is used for all FP which have non-zero IoU but are FP.

True Postive			False Positive hit	False Positive miss	False Negative
person	person	53647	4110	21457	15715
bicycle	motorcycle	29			
	bicycle	2284	345	1479	1108
car	truck	878			
	car	161100	11051	42421	29873
motorcycle	bicycle	36			
	motorcycle	853	121	280	414
bus	train	91			
	truck	184			
	bus	2173	46	436	595
train	bus	98			
	truck	41			
	train	389	4	106	89
truck	car	1176			
	bus	182			
	train	4			
	truck	6414	289	1374	825
traffic light	traffic light	6290	817	3845	2199
235869			**16783**	**71398**	**50818**

True Postive			False Positive hit	False Positive miss	False Negative
person	person	50585	4438	21573	18777
bicycle	motorcycle	47			
	bicycle	1827	459	1041	1547
car	truck	1022			
	car	159292	12566	49597	31537
motorcycle	bicycle	54			
	motorcycle	748	113	260	501
bus	train	90			
	truck	537			
	bus	1716	71	423	700
train	bus	136			
	truck	31			
	train	300	20	110	150
truck	car	1302			
	bus	118			
	train	4			
	truck	6205	249	1187	972
traffic light	traffic light	6310	727	3263	2179
230324			**18643**	**77454**	**56363**

Figure 4.15: Instance evaluation tables, the first one is for undistorted data, the second one represents distorted data

The instance evaluation tables between models actually look very similar, the upper table have obviously better results, however when all fields are compared the distorted detection model does not show any major difference. This result is very good, although before drawing final conclusion it is better compare other data.

Another very interesting part in this evaluation is always the class mapping from Sec. 4.4.2. This model achieved to keep most detections within theirs classes and misclassification even among vehicles is rare, only trucks got little bit mixed with cars but this is very common.

The following 8 (1 for each class) pages show the PR graphs, the legend to these graphs is in Sec. 4.5. In following PR curve graphs on top will start with undistorted detection model and interleave with distorted detection model. The PR curves are displayed together in comparison with distorted model. There are always 4 per page, showing results for undistorted detection model for bbox, then distorted detection for bbox and then mask for both in the same order.

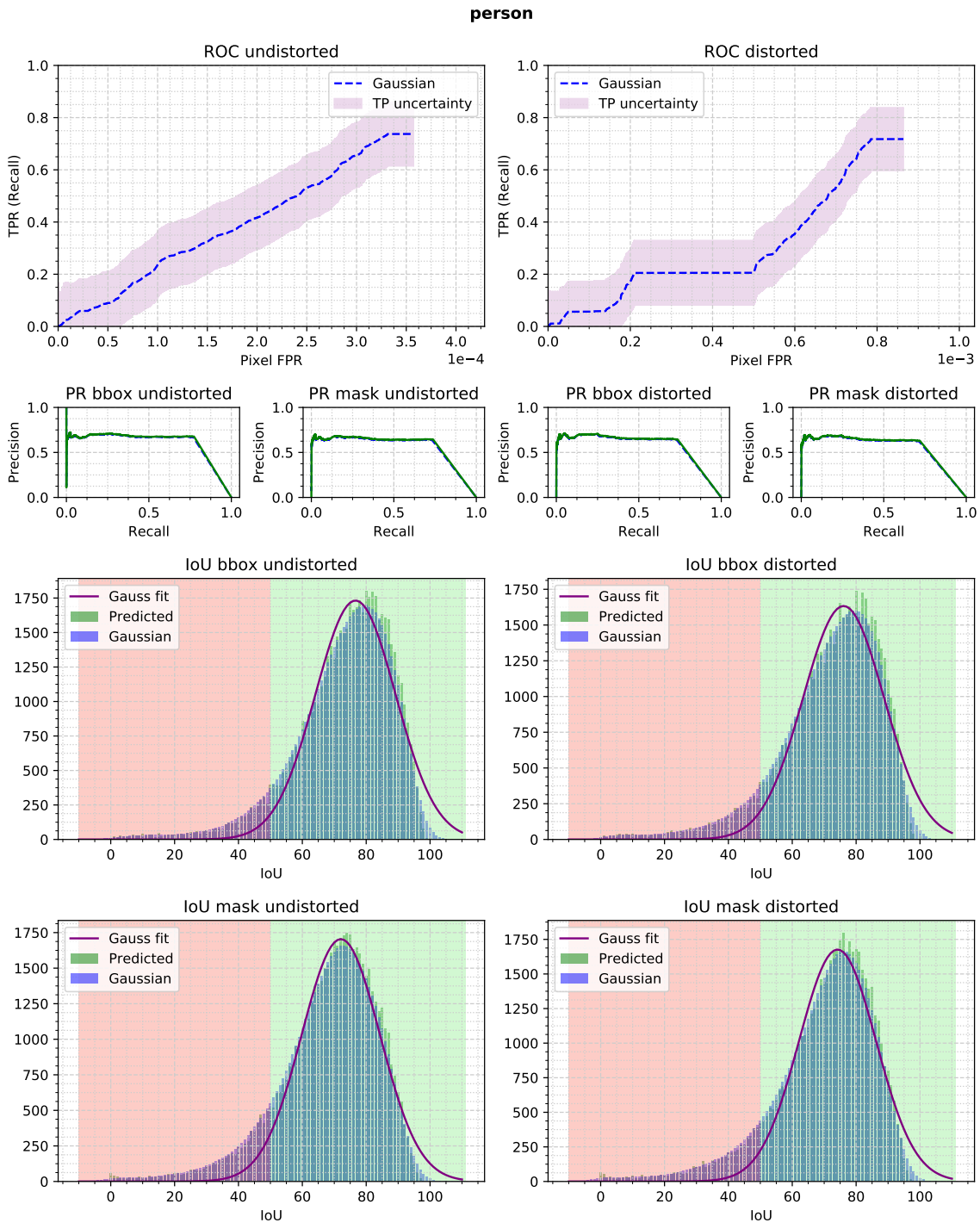


Figure 4.16: The ROC curve has good trade-off between TPR and FPR for undistorted images, however for distorted there is area with stable increase of FPR. This means, there is lot of FP with high score, since samples were sorted by score and this phenomenon is in the left side of ROC curve. This is partly due to detecting objects which have been removed in data generation 2.1 and were evaluated as FP. This theory is supported by fact that distribution of matched IoUs and recall is very similar, therefore most of the FP in the distorted version mostly miss all ground truths. This experiment is great success, the detection of distorted persons without undistortion of image is achievable.

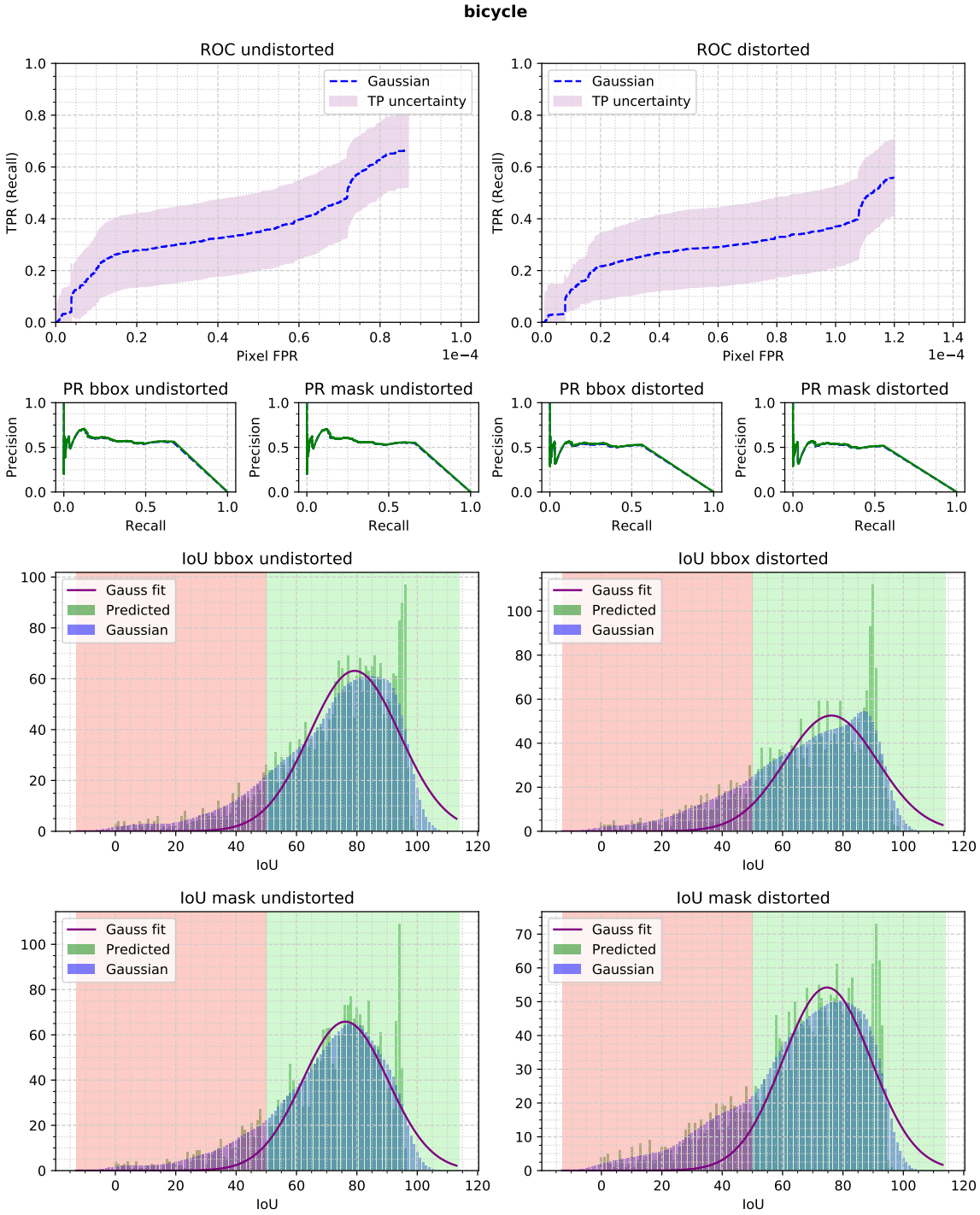


Figure 4.17: For bicycle there is not much difference on graphs, only on PR curve for bboxes, there is small difference in IoU distribution, this might be caused by larger empty spaces in bbox of distorted bike.

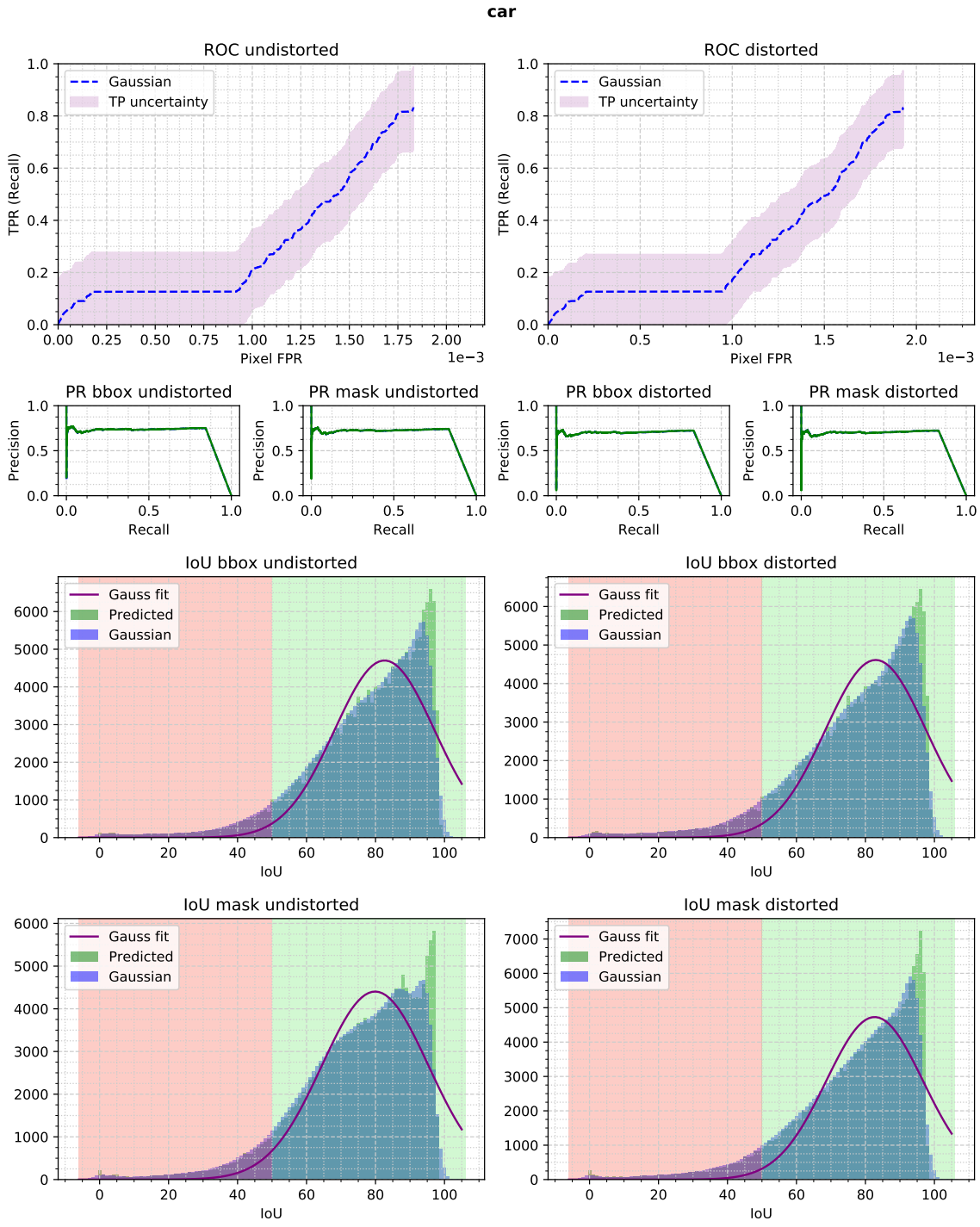


Figure 4.18: The car results show the same phenomenon as persons on distortion, most of the background cars are now detected as FP with high score, causing stable increase of FPR. This time even model trained on undistorted data is influenced. Otherwise all graphs are almost identical, resulting in another success.

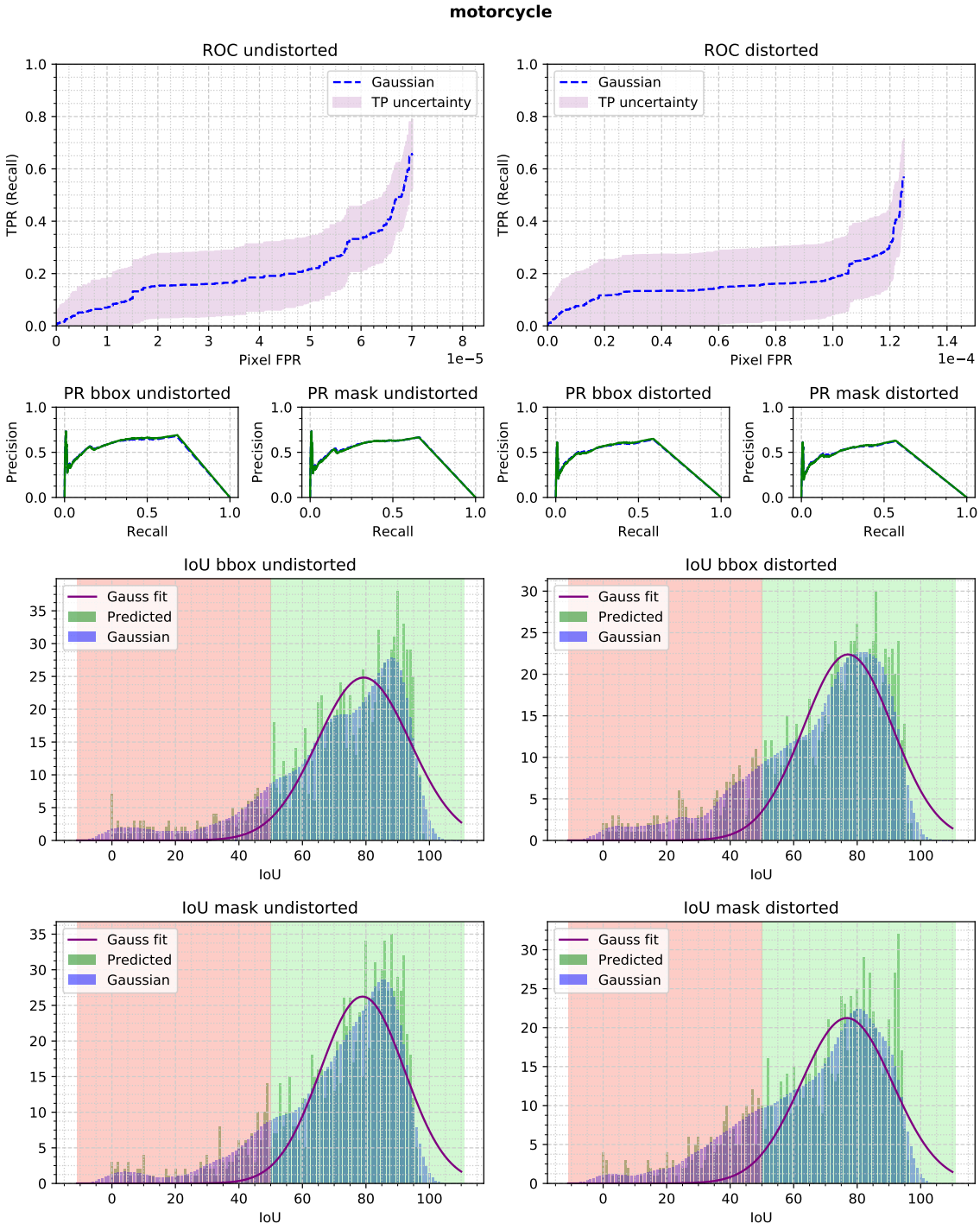


Figure 4.19: The motorcycles have behaviour similar to bicycles, which is not surprising, since both object classes have many similarities.

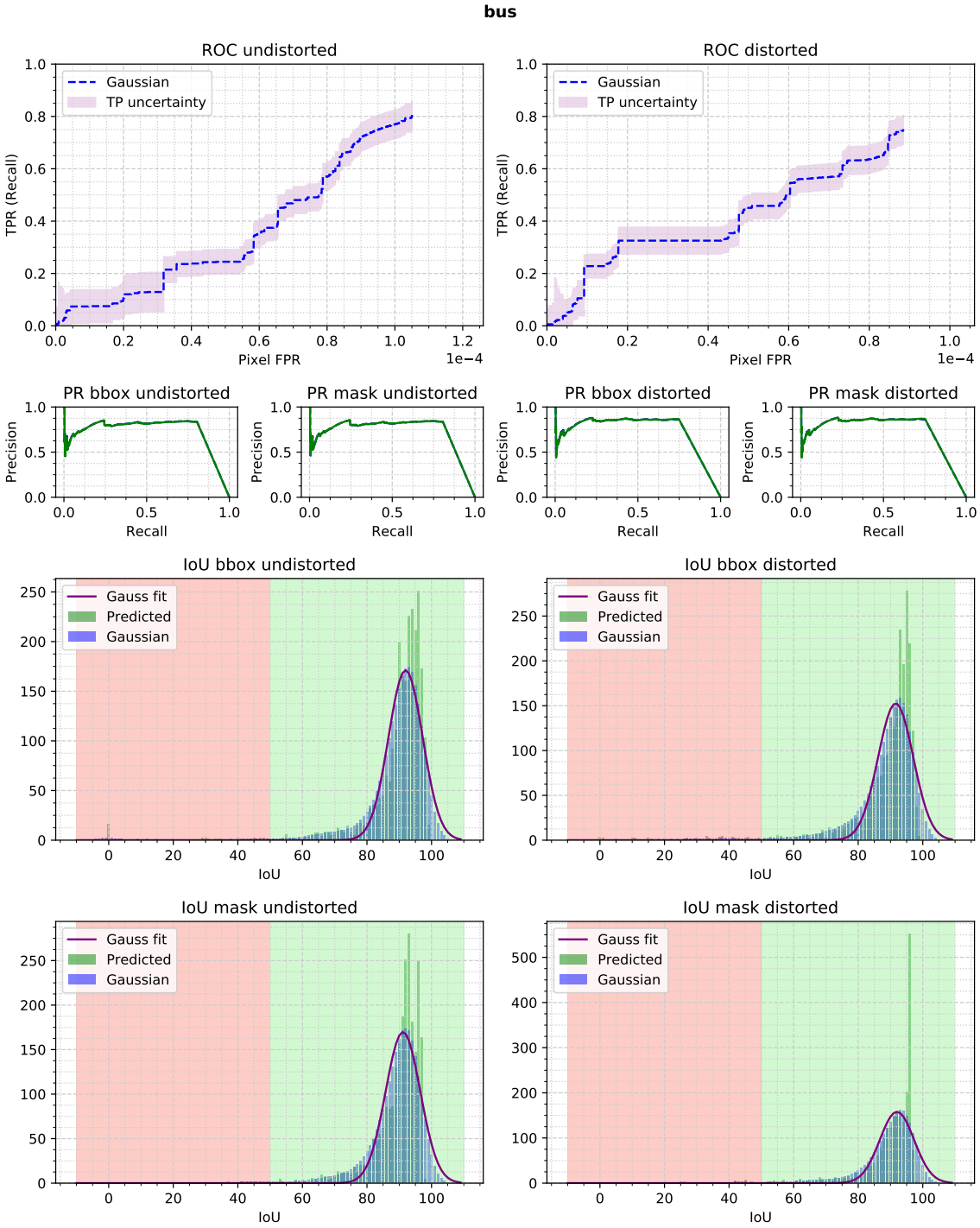


Figure 4.20: The buses are very similar, the IoU for mask might seem different on graph, but that is only because lot of buses luckily hit the same IoU and rescaled graph. The distributions are otherwise similar.

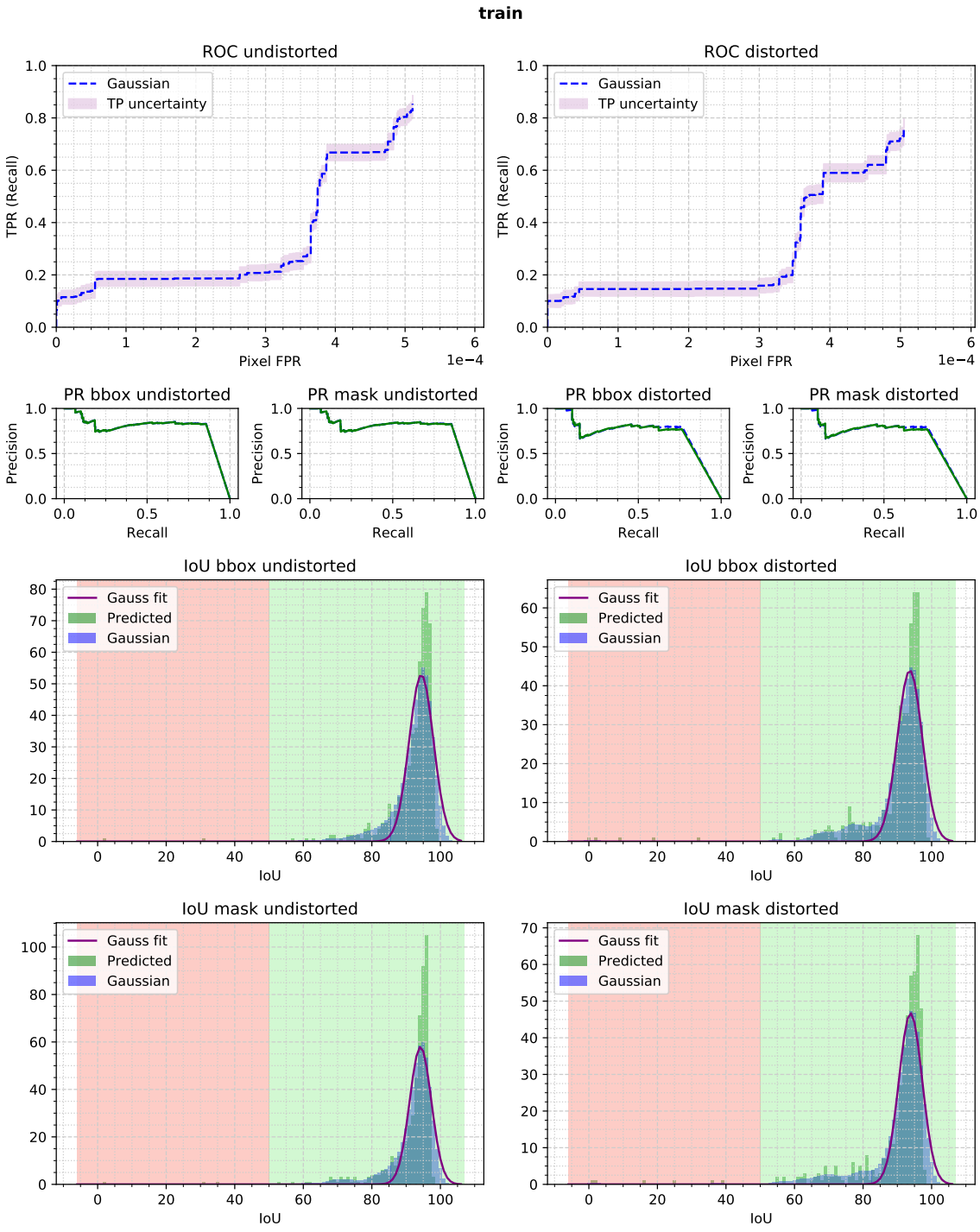


Figure 4.21: The trains also show stable increase in FPR, because the FP are counted as all FP pixels and trains are large especially when close to cameras. Also the closer the trains get, the harder is the detection due to distortion. Since this is the case of most of the FP, the ROC curve does not seem that good in comparison with PR curve which counts only objects and not pixels. In terms of similarity there are no problems.

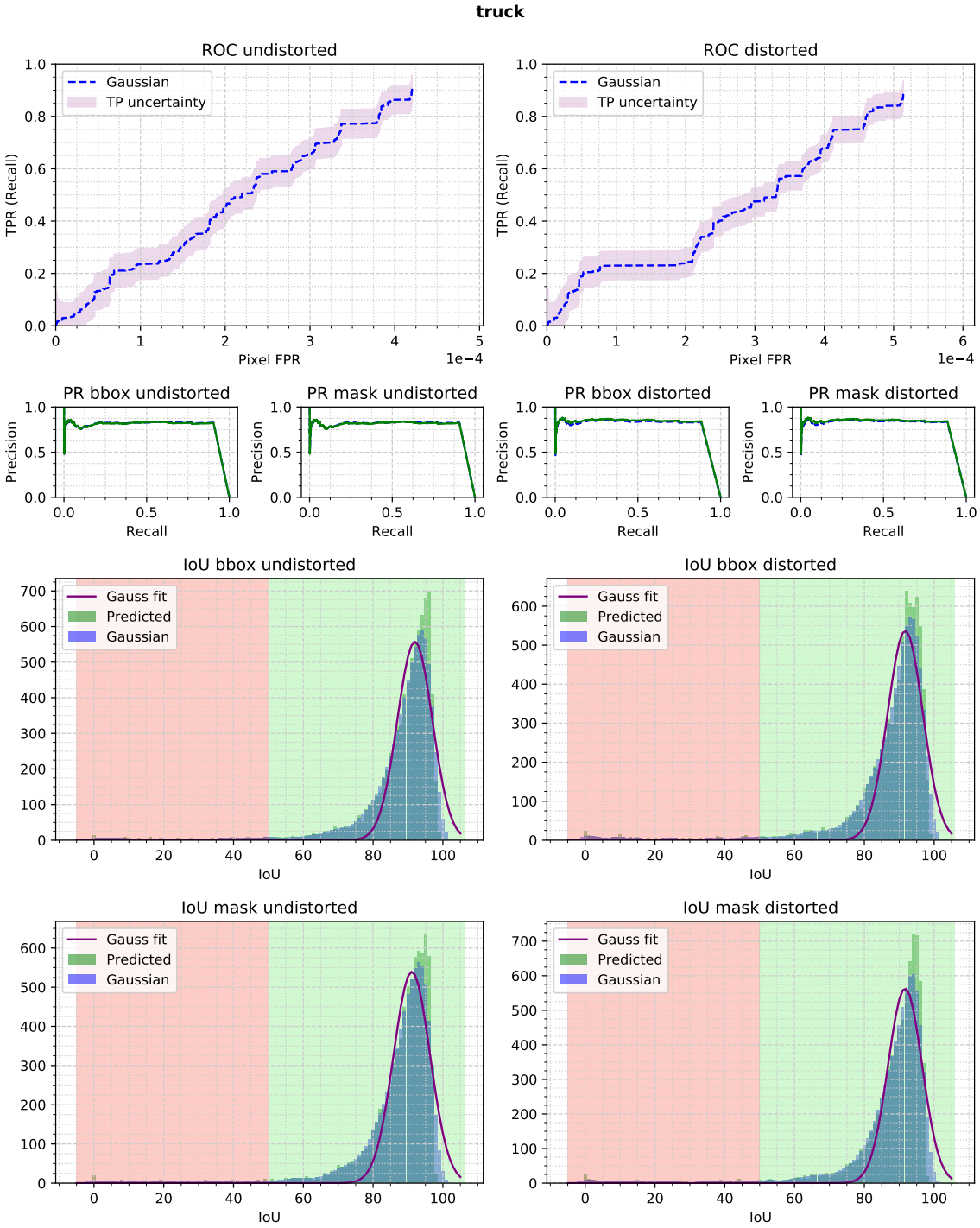


Figure 4.22: The trucks have large overlaps with cars since most of vans are considered a truck by detector. The conclusion for trucks is the same as for cars, since most of the graphs are almost identical.

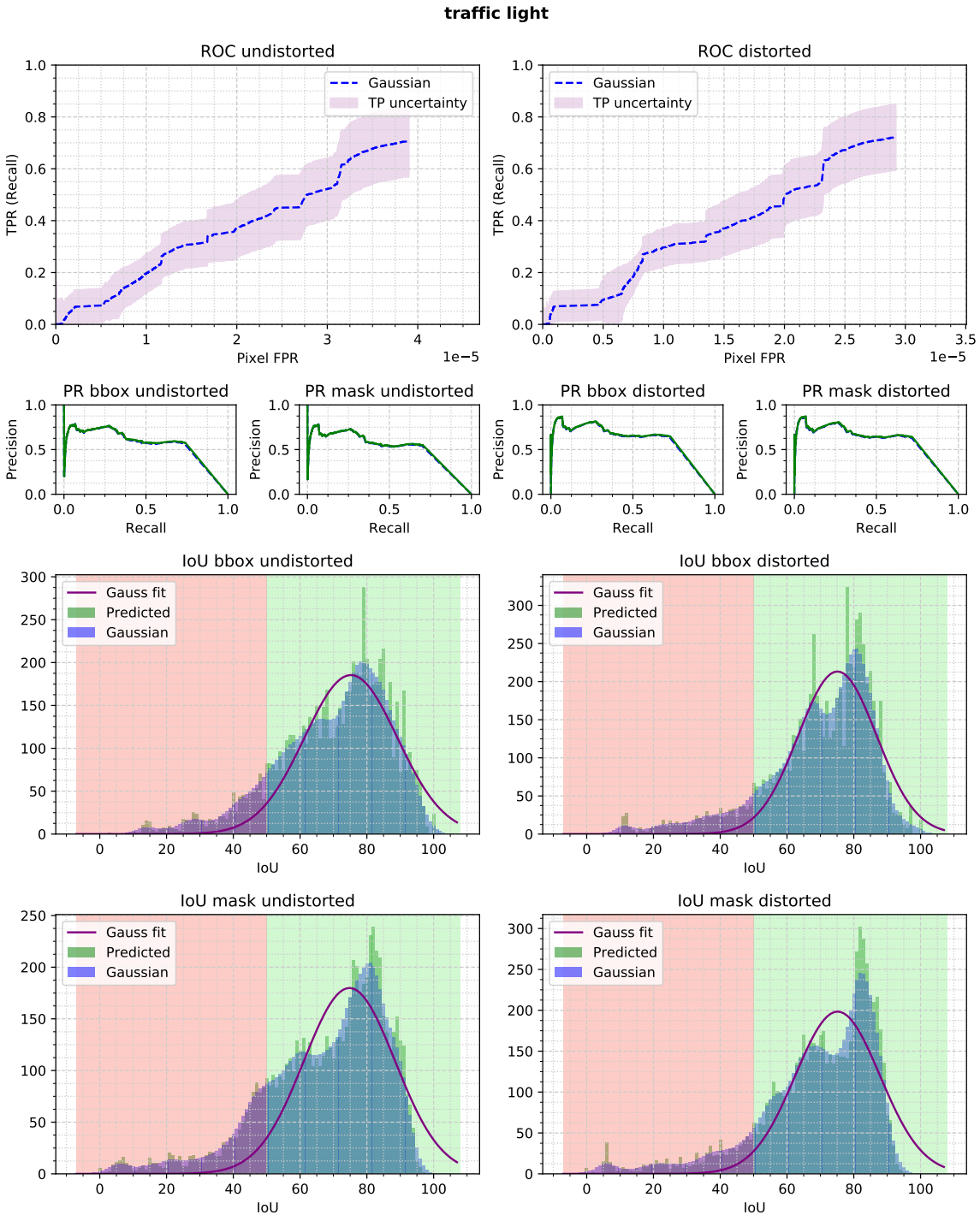


Figure 4.23: The traffic lights have another set of almost identical graphs, also showing good results on trade-off with ROC and PR.

In conclusion the the training of distorted model was a success, enabling encouragement for more similar experiments in the future.

4.9 Experiments on distorted images

4.9.1 Experiment on separate cameras

The experiment in this section make detection model focus on only certain cameras.

In this experiment there will be defined new tags for cameras:

- FV stand for forward view and decribes front camera
- RV stand for back view and decribes rear camera
- MVR is used for right view and decribes right camera
- MVL is used for left view and decribes left camera
- MVX is used for side view and decribes left and right camera

Each AP table on page 80 have experimented camera in leftmost top corner of table.

The tags in table will match previously explained car views model, configurations and addtional information can be found in legend Sec. 4.6.

All the cameras had the best performance when trained focusing only on theirs respective view. Only left camera showed different behavior and gained better training from both sides and suprisingly for masks preferred model trained on distorted images, being left view in country where cars drive on the right side, that means there were probably some trams in the middle preferring this way.

FV	person	bicycle	car	motorcycle	bus	train	truck	traffic light	mgAP
seg clean dontcare r101 bbox	0.500	0.390	0.611	0.435	0.784	0.789	0.779	0.692	0.623
seg clean dontcare r101 mask	0.488	0.394	0.603	0.408	0.783	0.789	0.785	0.690	0.618
seg clean dontcare r101 un bbox	0.526	0.482	0.657	0.424	0.805	**0.859**	0.783	0.647	0.648
seg clean dontcare r101 un mask	0.497	0.485	0.641	0.394	0.804	**0.859**	0.785	0.632	0.637
seg clean dontcare r101 fv bbox	0.506	0.475	0.637	**0.499**	0.788	0.823	0.795	0.701	0.653
seg clean dontcare r101 fv mask	0.492	0.477	0.628	**0.481**	0.788	0.823	0.799	**0.695**	0.648
seg clean dontcare r101 un fv bbox	**0.535**	**0.532**	**0.659**	0.420	**0.826**	0.852	**0.800**	**0.703**	**0.666**
seg clean dontcare r101 un fv mask	**0.506**	**0.534**	**0.643**	0.393	**0.828**	0.852	**0.801**	0.686	**0.655**

RV	person	bicycle	car	motorcycle	bus	train	truck	traffic light	mgAP
seg clean dontcare r101 bbox	0.481	0.289	0.621	0.399	0.545	0.757	0.792	0.223	0.513
seg clean dontcare r101 mask	0.457	0.280	0.619	0.380	0.551	0.760	0.798	0.196	0.505
seg clean dontcare r101 un bbox	0.507	0.366	0.648	0.413	0.575	0.849	0.800	0.217	0.547
seg clean dontcare r101 un mask	0.470	0.352	0.633	0.407	0.578	0.849	0.800	0.173	0.533
seg clean dontcare r101 rv bbox	0.503	0.321	0.637	**0.472**	**0.605**	0.830	0.823	**0.284**	0.559
seg clean dontcare r101 rv mask	**0.480**	0.312	0.635	0.428	**0.607**	0.832	0.823	**0.242**	0.545
seg clean dontcare r101 un rv bbox	**0.513**	**0.378**	**0.661**	0.470	0.552	**0.870**	**0.825**	0.232	**0.563**
seg clean dontcare r101 un rv mask	0.472	**0.372**	**0.644**	**0.440**	0.565	**0.870**	**0.827**	0.188	**0.547**

MVL	person	bicycle	car	motorcycle	bus	train	truck	traffic light	mgAP
seg clean dontcare r101 bbox	0.465	0.373	0.615	0.781	0.608	0.485	0.736	0.343	0.551
seg clean dontcare r101 mask	0.425	0.374	0.620	0.786	0.614	0.460	0.741	0.308	0.541
seg clean dontcare r101 un bbox	0.518	0.491	0.641	0.812	**0.649**	0.563	0.747	0.219	0.580
seg clean dontcare r101 un mask	0.436	0.478	0.624	0.804	**0.647**	0.557	0.744	0.070	0.545
seg clean dontcare r101 mvl bbox	0.520	0.494	0.648	**0.823**	0.620	0.511	0.758	0.199	0.572
seg clean dontcare r101 mvl mask	**0.481**	0.488	**0.650**	**0.839**	0.624	0.510	**0.761**	0.187	**0.568**
seg clean dontcare r101 mvx bbox	0.500	0.381	0.636	0.764	0.575	0.546	0.743	**0.428**	0.572
seg clean dontcare r101 mvx mask	0.466	0.388	0.639	0.790	0.576	0.546	0.750	**0.359**	0.564
seg clean dontcare r101 un mvl bbox	**0.523**	0.509	**0.657**	0.800	0.637	0.568	0.743	0.240	0.585
seg clean dontcare r101 un mvl mask	0.436	0.503	0.636	0.819	0.638	0.566	0.747	0.151	0.562
seg clean dontcare r101 un mvx bbox	0.518	**0.529**	0.648	0.794	0.609	**0.578**	**0.762**	0.305	**0.593**
seg clean dontcare r101 un mvx mask	0.436	**0.518**	0.629	0.774	0.610	**0.567**	0.760	0.178	0.559

MVR	person	bicycle	car	motorcycle	bus	train	truck	traffic light	mgAP
seg clean dontcare r101 bbox	0.532	0.297	0.553	0.378	0.576	0.240	0.724	0.218	0.440
seg clean dontcare r101 mask	0.507	0.277	0.559	0.342	0.588	0.240	0.730	0.226	0.434
seg clean dontcare r101 un bbox	0.567	0.368	0.589	0.498	**0.708**	0.520	0.727	0.280	0.532
seg clean dontcare r101 un mask	0.511	0.349	0.572	0.470	**0.722**	0.520	0.734	0.222	0.512
seg clean dontcare r101 mvr bbox	0.572	0.356	0.596	0.503	0.573	0.496	0.753	0.324	0.522
seg clean dontcare r101 mvr mask	**0.550**	0.339	**0.602**	0.464	0.592	0.496	0.760	**0.335**	0.517
seg clean dontcare r101 mvx bbox	0.561	0.352	0.580	0.486	0.588	0.493	0.743	0.305	0.513
seg clean dontcare r101 mvx mask	0.538	0.338	0.586	0.442	0.581	0.493	0.752	0.309	0.505
seg clean dontcare r101 un mvr bbox	**0.572**	0.393	**0.602**	**0.566**	0.703	**0.538**	**0.758**	**0.329**	**0.558**
seg clean dontcare r101 un mvr mask	0.515	**0.375**	0.583	**0.555**	0.690	**0.538**	**0.762**	0.249	**0.534**
seg clean dontcare r101 un mvx bbox	0.564	**0.394**	0.598	0.541	0.662	0.439	0.756	0.258	0.526
seg clean dontcare r101 un mvx mask	0.509	0.372	0.579	0.533	0.673	0.444	0.756	0.169	0.505

Figure 4.24: The experiments on separate cameras

4.9.2 Experiment on different models and annotation cleaning methods

This experiment is bit of unorganized collection of all secondary detection models which have been trained in this work, with presence of smaller speed performance models such as the YoloV4 1.1.5, which will be interesting comparison to larger models, especially since this experiment is trained exclusively on distorted data only.

The model configurations and addtional information for following AP table can be found in legend Sec. 4.6.

	person	bicycle	car	motorcycle	bus	train	truck	traffic light	mgAP
seg clean dontcare r101 bbox	0.487	0.270	**0.616**	**0.518**	0.645	**0.741**	0.726	**0.537**	**0.567**
seg clean dontcare r101 mask	**0.469**	0.270	**0.615**	**0.522**	0.648	**0.741**	0.729	**0.534**	**0.566**
standard r101 bbox	0.485	**0.346**	0.610	0.378	0.590	0.678	**0.760**	0.502	0.544
standard r101 mask	0.461	**0.336**	0.609	0.355	0.591	0.670	**0.767**	0.490	0.535
standard r50 bbox	0.470	0.334	0.608	0.414	0.560	0.666	0.753	0.485	0.536
standard r50 mask	0.446	0.326	0.608	0.381	0.562	0.667	0.756	0.484	0.529
seg clean purge r101 bbox	**0.490**	0.305	0.602	0.379	**0.653**	0.640	0.751	0.531	0.544
seg clean purge r101 mask	0.467	0.298	0.602	0.356	**0.656**	0.638	0.755	0.520	0.537
YoloV4 bbox	0.372	0.153	0.416	0.133	0.517	0.443	0.641	0.334	0.376

Figure 4.25: The experiments on different models and annotation generation

The YoloV4 detailed comparison to Resnet 101 distorted data model is included in the appendix A.

Unfortunately the YoloV4 did not perform too well in this experiment it might be because it works only with bounding boxes, which might not be optimal for some distortion cases, the smaller ResNet 50 architecture did actually fine despite having much smaller backbone and the panoptic merge 4.2.2 and dontcare 4.10 were the best from data cleaning techniques.

4.10 Experiment on Virtual Reality scenes

The greatest difference between automatically generated scenes and VR scenes is that VR scenes have absolutely correct ground truth. Therefore the Gauss localization error will not be used for evaluation on VR scenes. However the perfect ground truths have small inconvenience which is being too precise over limitations of human eye and detector with small sizes of ground truths.

The models evaluated on reality do not use objects with area lower than 96, since these objects are too small to notice theirs presence (meaning that such small person or car would be too far away).

From table below it is noticeable what is the main problem of these sometimes even one pixel size objects. These tables describe statistics for bboxes on undistorted VR images.

True Postive	False Positive hit	False Positive miss	False Negative
77078	**22799**	**7242**	**103814**

The balance between TP and all FP is reasonable, however the number of FN - missed ground truths is devastating. Obviously after calculating AP for all objects, the recall will be very bad and is going to influence the results, these will be shown only in Average Precision for now, the PR curves will be show on better experiment configuration.

	person	bicycle	car	bus	train	truck	mgAP
seg clean dontcare r101 vosstrex bbox	**0.202**	0.042	**0.370**	0.317	**0.249**	0.233	0.235
seg clean dontcare r101 vosstrex mask	**0.172**	**0.178**	**0.361**	**0.353**	**0.251**	0.239	**0.259**
seg clean dontcare r101 un vosstrex bbox	0.184	**0.084**	0.362	**0.336**	0.238	**0.287**	**0.249**
seg clean dontcare r101 un vosstrex mask	0.142	0.155	0.332	0.344	0.240	**0.288**	0.250

Figure 4.26: Vosstrex evaluation without size filters

Both undistorted and distorted detection task have very low AP scores, this experiment shows problems that might happen while not thresholding small objects. For second experiment the objects with area of bbox lower than 32 were disabled to see which of the TP, FP or FN lose the most objects. The area of 32 pixel is extremely tiny, the final threshold will be 96, this threshold just demonstrates how many tiny object can be present.

True Postive	False Positive hit	False Positive miss	False Negative
74582	**16591**	**6007**	**46282**

The table shows that after droping tiny objects the FN lose more then half objects which is extremely large difference. The TP and FP were not influenced that much.

area > 32	person	bicycle	car	bus	train	truck	mgAP
seg clean dontcare r101 vosstrex bbox	**0.407**	0.054	**0.534**	0.355	**0.286**	0.253	0.315
seg clean dontcare r101 vosstrex mask	**0.353**	**0.231**	**0.526**	**0.396**	**0.289**	0.260	**0.343**
seg clean dontcare r101 un vosstrex bbox	0.372	**0.109**	0.525	**0.371**	0.274	**0.311**	**0.327**
seg clean dontcare r101 un vosstrex mask	0.293	0.201	0.484	0.381	0.276	**0.312**	0.324

Figure 4.27: Vosstrex evaluation with tiny size filter

With confusing tiny objects away the AP of all classes rapidly increased, for example person doubled, which is not unforeseen since there are usually many people in background, far away from road which makes their area very tiny. The final reduction of data will be achieved with threshold 96 of area size, after this modification the VR evaluation will have same conditions as the other evaluations.

True Postive			False Positive hit	False Positive miss	False Negative
person	person	9056	1531	668	4457
bicycle	bicycle	235	91	24	434
car	truck	4180			
	car	47319	7046	2468	15350
bus	train	2			
	truck	890			
	bus	385	26	122	1144
train	bus	189			
	truck	14			
	train	102	1	2	541
truck	car	43			
	bus	6			
	train	0			
	truck	502	328	172	959
62923			**9023**	**3455**	**22885**

Figure 4.28: Vosstrex instance evaluation table with small size filter

The final drop of small objects halved FN again, also great amount of FP and TP was reduced, this table also descirbes the class mapping to see which classes are confusing in VR. Because VR simulator Vosstrex does not have motorcycles implemented, they will not influence bicycles. The bicycles have also one major problem in Vosstrex and that is their masks being shared with the person riding the bike (bicycle is the bike and person counted as one in Vosstrex).

To deal with this very unfortunate problem the ground truths of bicycles were cloned and labeled as person, however this still leaves them in disadvantage, because they need almost perfect IoU to barely pass even the 50% threshold. To give chance to bicycles theirs IoU will be exclusively thresholded at 25%.

Returning to class mapping cars are mostly cars, but truck is also popular choice, which is usual behaviour, also trucks sometimes identify as car however in VR trucks rarely confuse themselves with train and bus, which is interesting. The train and bus however gets mixed way too much, this might be caused by very low quality models of these objects Fig. (4.36).

The model configurations and addtional information for following AP table can be found in legend Sec. 4.6.

area > 96	person	bicycle	car	bus	train	truck	mgAP
seg clean dontcare r101 vosstrex bbox	**0.625**	0.084	**0.686**	0.408	**0.355**	0.298	0.409
seg clean dontcare r101 vosstrex mask	**0.530**	**0.263**	**0.662**	**0.457**	**0.358**	0.304	**0.429**
seg clean dontcare r101 un vosstrex bbox	0.602	**0.170**	0.675	**0.427**	0.338	**0.346**	**0.426**
seg clean dontcare r101 un vosstrex mask	0.486	0.242	0.631	0.439	0.340	**0.346**	0.414

area > 96 & bicycle IoU thr = 0.25	person	bicycle	car	bus	train	truck	mgAP
seg clean dontcare r101 vosstrex bbox	**0.625**	**0.466**	**0.686**	0.408	0.355	0.298	0.464
seg clean dontcare r101 vosstrex mask	**0.530**	**0.471**	**0.662**	0.457	0.358	0.304	0.459
seg clean dontcare r101 un vosstrex bbox	0.602	0.459	0.675	0.427	0.338	0.346	0.468
seg clean dontcare r101 un vosstrex mask	0.486	0.456	0.631	0.439	0.340	0.346	0.445
seg clean dontcare r101 bbox	0.487	0.270	0.616	0.645	0.741	0.726	0.567
seg clean dontcare r101 mask	0.469	0.270	0.615	0.648	0.741	**0.729**	0.566
seg clean dontcare r101 un bbox	0.529	0.359	0.643	**0.689**	**0.820**	**0.729**	**0.598**
seg clean dontcare r101 un mask	0.486	0.351	0.629	**0.689**	**0.820**	0.729	**0.586**

Figure 4.29: Vosstrex evaluation with small size filter and bicycle IoU condition

The two tables with calculated AP show performance after thresholding area size below 96, with the second table addressing the bicycle problem mentioned before. The IoU benevolence for bicycle shows that the matching were there waiting just below the border and now the AP of bicycle is improved.

The second table also includes models evaluated on 1:1 real data scenarios, for the AP and mAP the reality scene still win, due to Vosstrex bad performance on busses, trains and trucks, however these objects are very low quality and that might be the decisive factor for that low performance, the examples of these objects might be seen in Fig. 4.36. On the other hand the cars in VR have very good AP score, same with persons and bicycles, which got little bit of handicap as mentioned before.

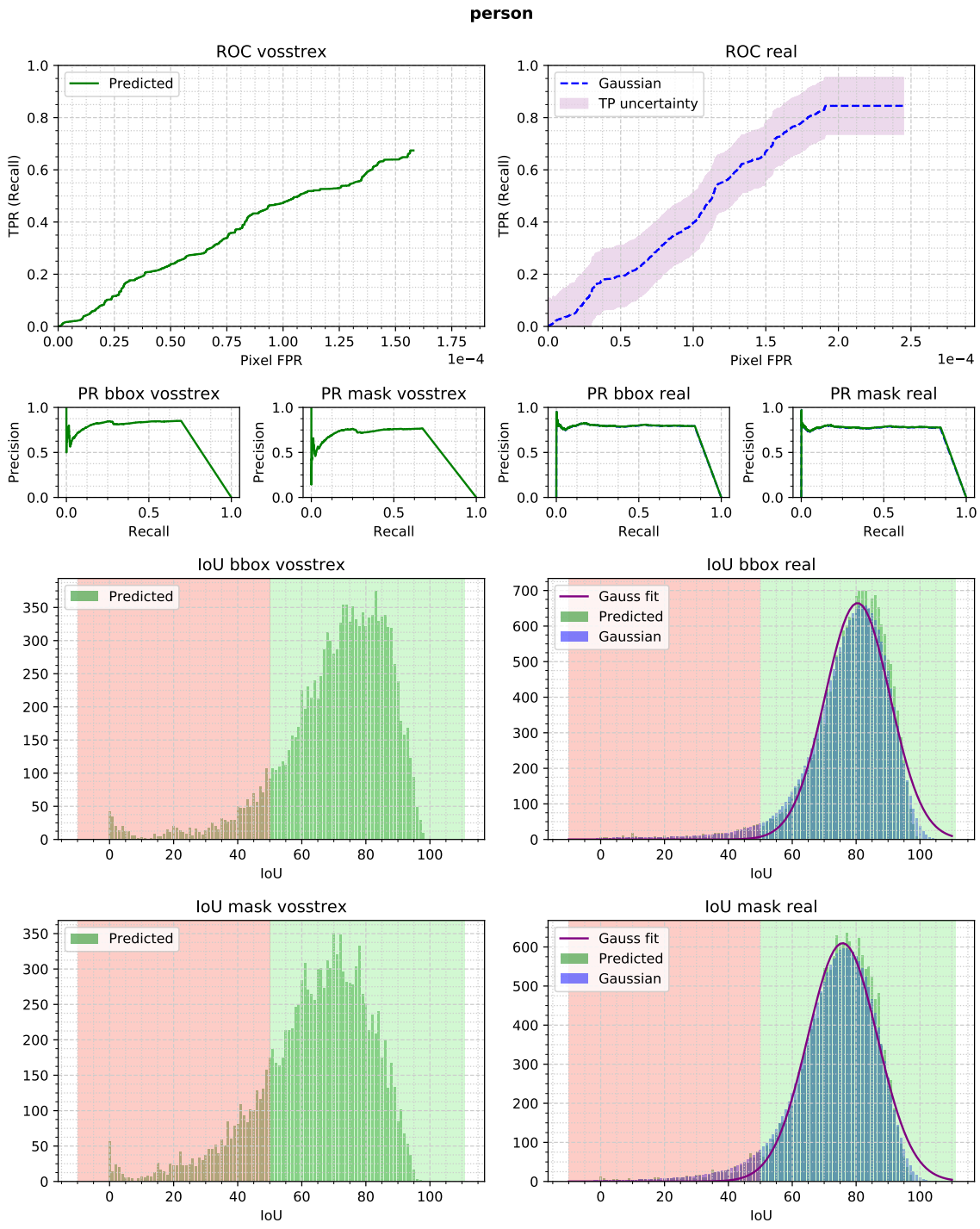


Figure 4.30: The person results are interesting, the VR scene has much less people, mostly because of simplification of scene and making clusters of people smaller, but despite the number difference, both IoU distributions are almost the same and both are closer to mean of 80% IoU which is very nice result. The equivalence in this case is better than VR outperforming real life detection, because for purpose of training models offline in VR to work in real world, it is necessary to know if the models will perform relatively the same in both cases.

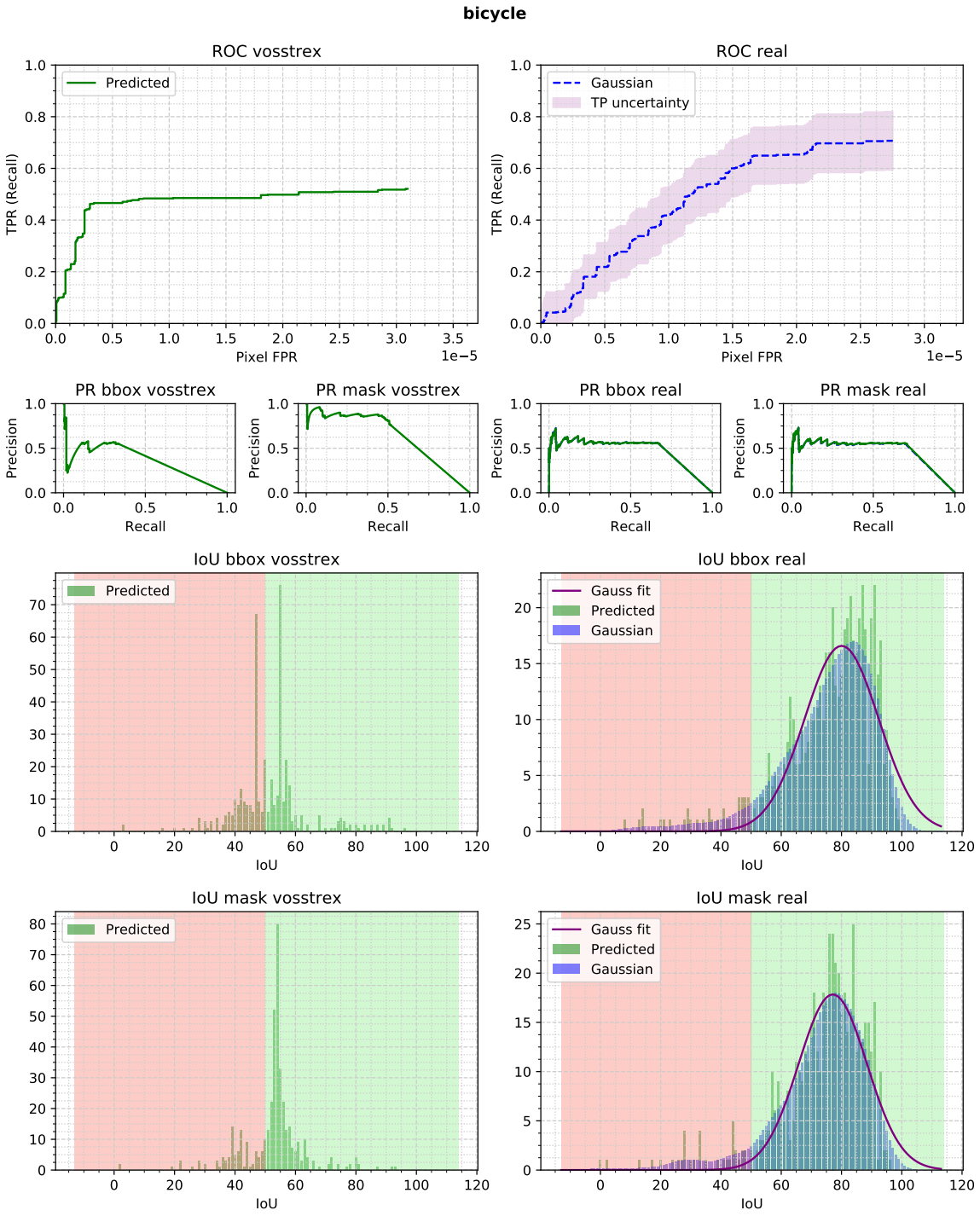


Figure 4.31: The bicycles preform decently, if it is accounted for the problems of shared bicycle with person ground truths of Vosstrex.

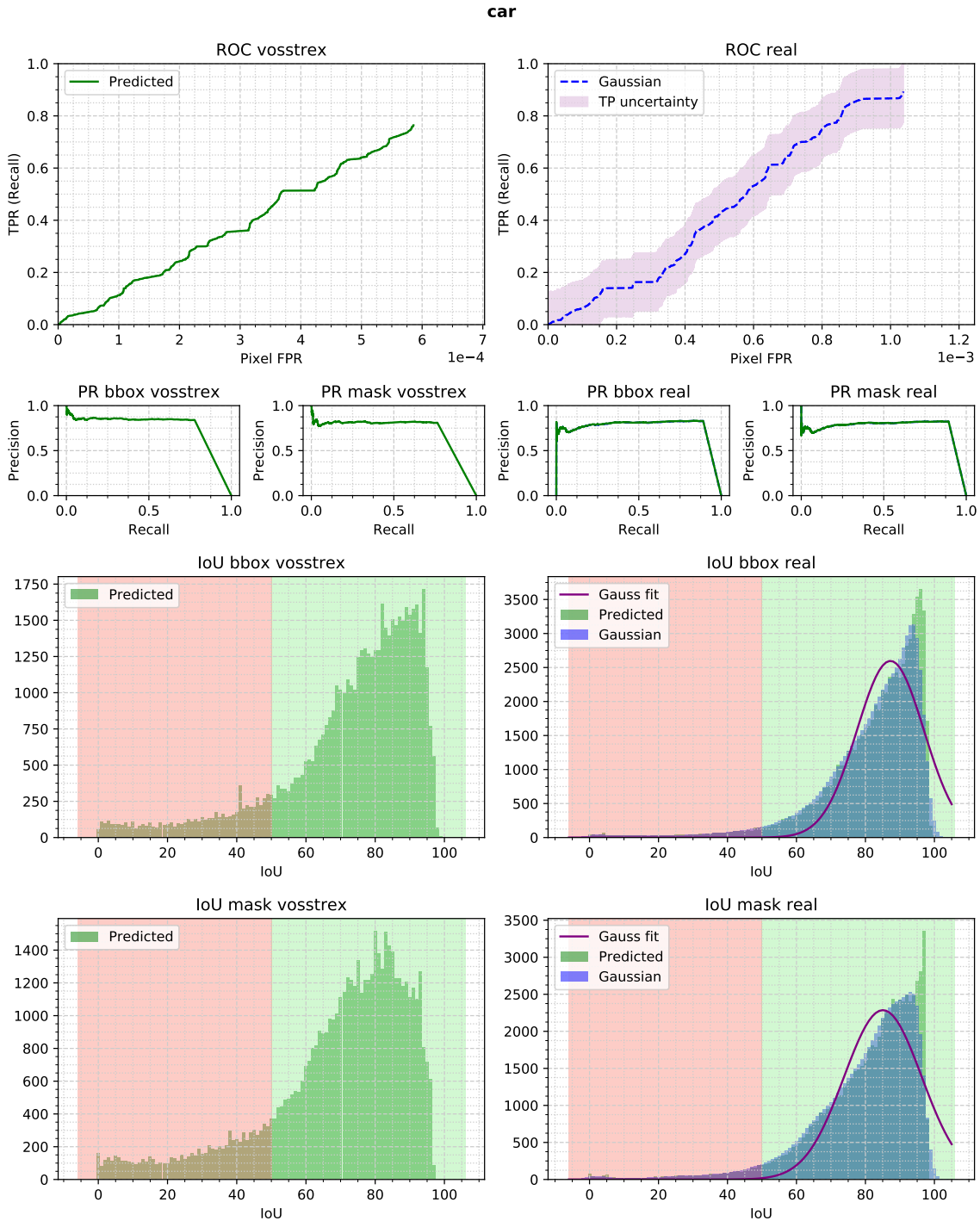


Figure 4.32: The cars have very good performance and similarity of results, this is mostly because cars have the most detailed models to match the real world. The FPR is approximately 70% larger for real data than for VR data, this is because the real data can have more background car objects than VR data as it is mentioned in 2.4 (the 70% is still viable size difference for this hypothesis), therefore the trade-off curve and its shape and slope are better for comparison than the FPR total size, the shape and slope of ROC curve shows similarity in this case.

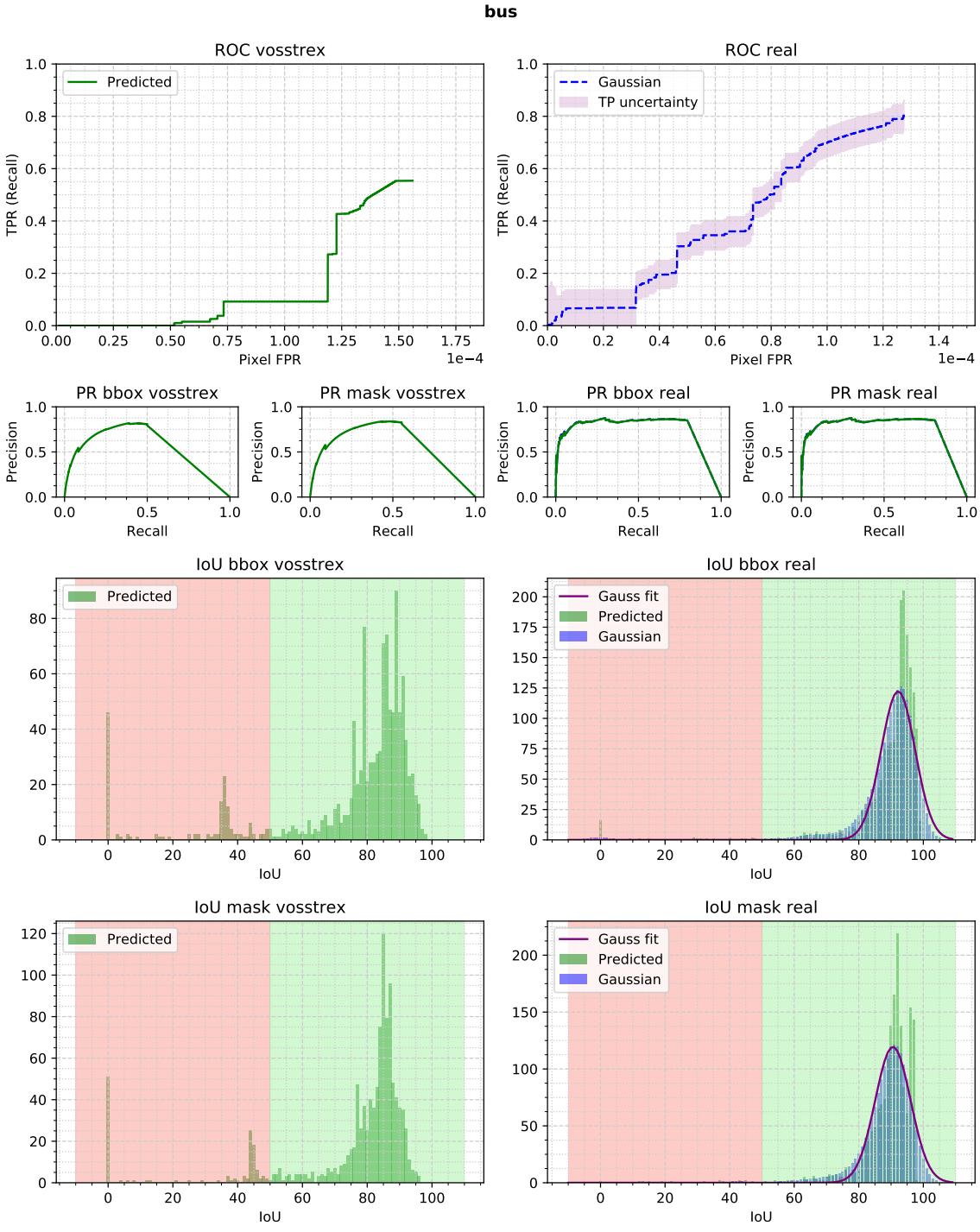


Figure 4.33: For the bus detection matches (FP + TP) even Vosstrex has good good IoU distribution, however the recall is very low meaning lot of GT were not detected, these are probably the cases when the VR car gets too close to object when the low resolution starts to hurt detection Fig. 4.36.

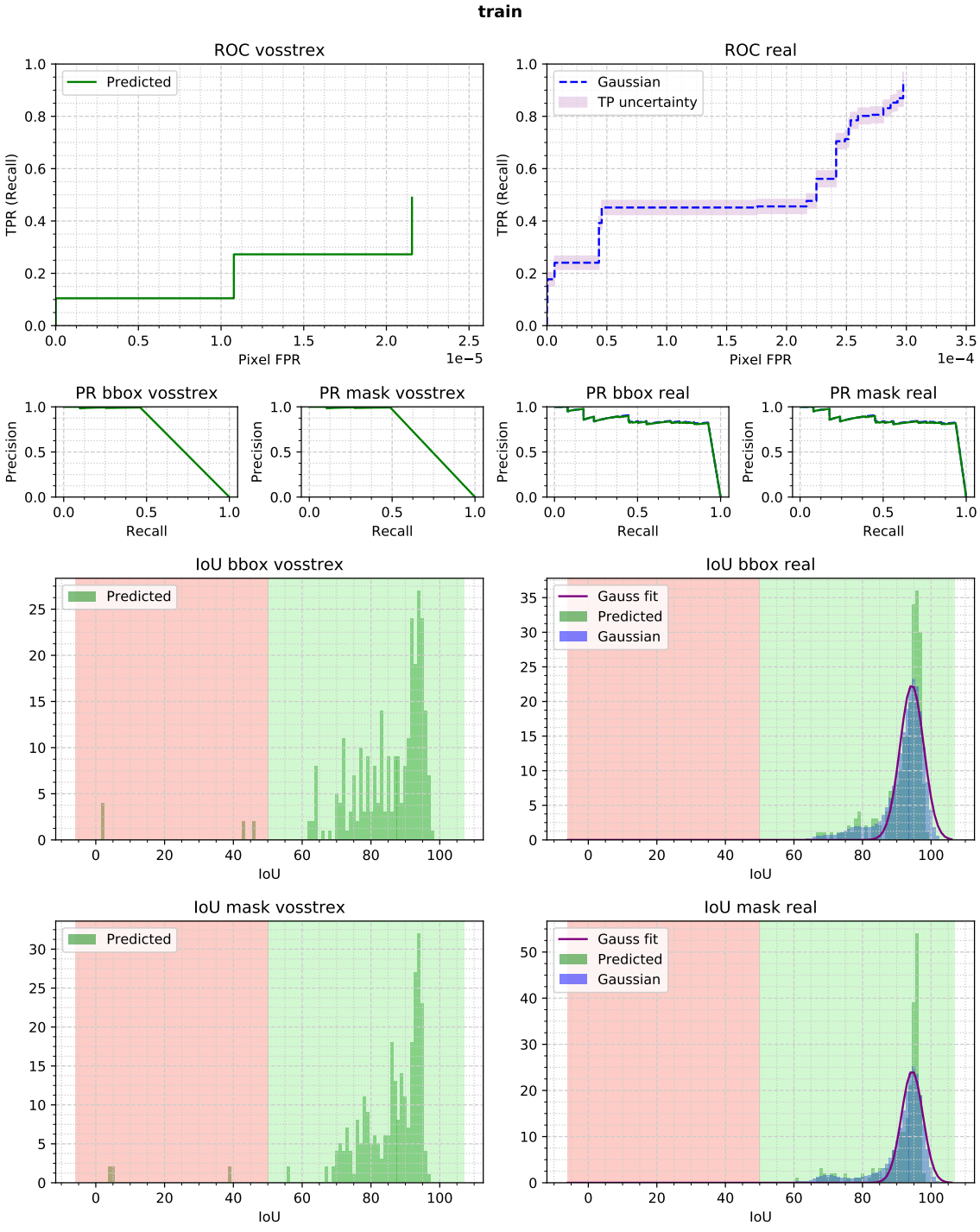


Figure 4.34: The trains also suffer from very low recall, but also low occurrences of FP resulting in not very informative ROC curve. The trains are already hard to detect in reality and low quality VR models makes this task even harder 4.36.

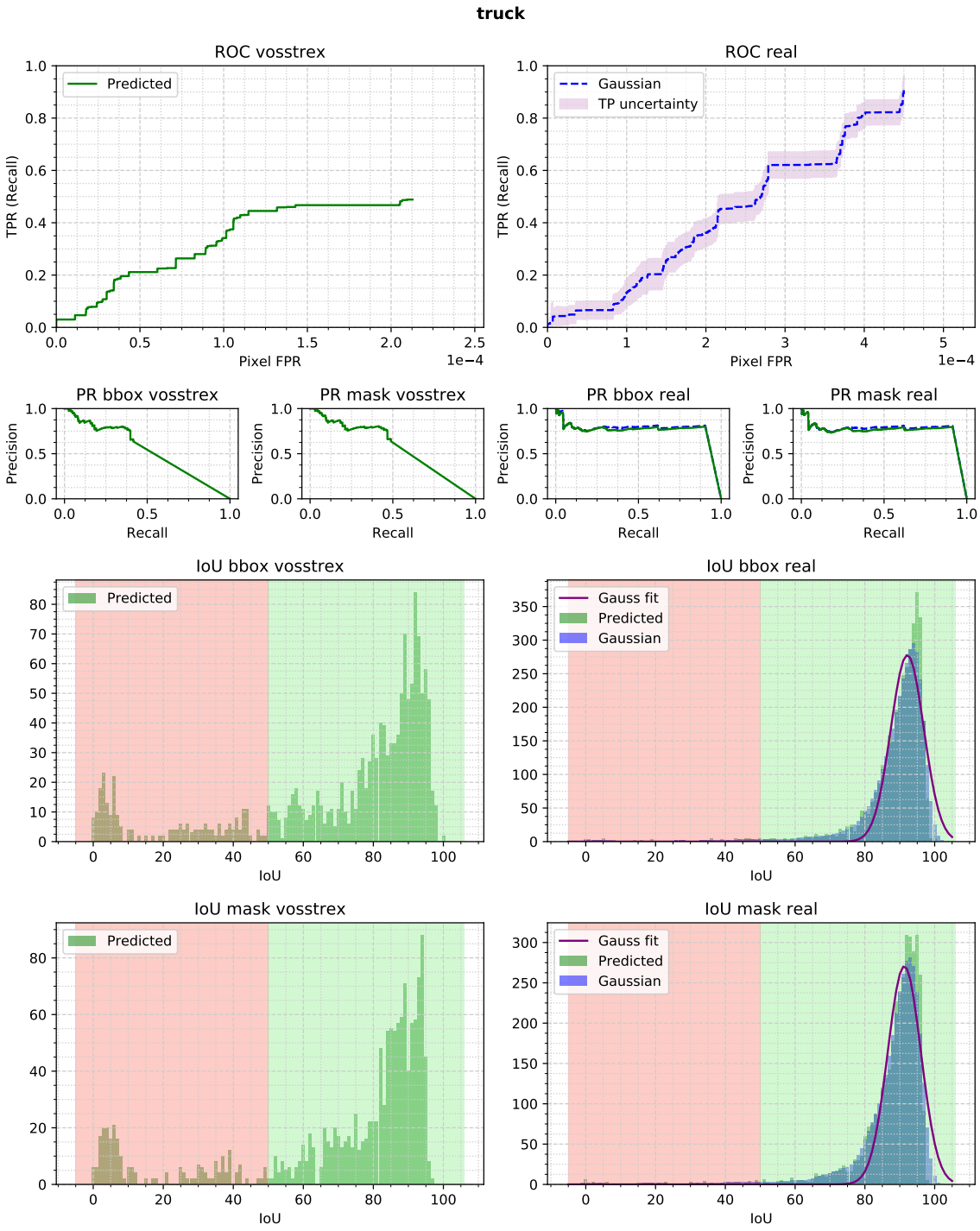


Figure 4.35: The trucks have problems mostly identical to buses and trains, except the ROC curve, which has better trade-off in this case, but this might be caused by overlapping with the car class.

In conclusion for Real and Virtual data comparison, The results were overall very similar and that is very good for successful completion of goal of this work. In statistics Vosstrex was performing slightly better on some classes and a bit worse on the others, what was even more confusing, was that the model trained on distorted data running on VR was slightly outperforming model trained on undistorted data running on real data, bringing nice twist to results in this experiment.

Figure 4.36: Samples of Vosstrex classes, most of the cars, people and bike have high quality models, unfortunately the buses, trains and trucks, are mostly to fill the class place and there are problems with their detection.

4.11 Detection results overview

	person	bicycle	car	motorcycle	bus	train	truck	traffic light	mgAP
vanilla bbox	0.292	0.243	0.348	0.326	0.237	0.308	0.452	0.436	0.320
vanilla mask	0.273	0.228	0.348	0.304	0.240	0.301	0.457	0.430	0.312
vanilla un bbox	0.425	0.357	0.566	0.386	0.399	0.565	0.634	0.446	0.461
vanilla un mask	0.410	0.346	0.565	0.367	0.401	0.563	0.634	0.438	0.454
seg clean dontcare r101 bbox	0.494	0.316	0.601	0.415	0.649	0.639	0.752	0.527	0.549
seg clean dontcare r101 mask	0.470	0.309	0.602	0.390	0.652	0.632	0.758	0.514	0.541
seg clean dontcare r101 un bbox	**0.529**	**0.405**	**0.634**	**0.461**	**0.681**	**0.742**	0.756	0.493	**0.588**
seg clean dontcare r101 un mask	**0.480**	**0.394**	**0.618**	**0.441**	**0.682**	**0.739**	0.758	0.451	**0.570**
standard r101 bbox	0.485	0.346	0.610	0.378	0.590	0.678	**0.760**	0.502	0.544
standard r101 mask	0.461	0.336	0.609	0.355	0.591	0.670	**0.767**	0.490	0.535
standard r50 bbox	0.470	0.334	0.608	0.414	0.560	0.666	0.753	0.485	0.536
standard r50 mask	0.446	0.326	0.608	0.381	0.562	0.667	0.756	0.484	0.529
seg clean purge r101 bbox	0.490	0.305	0.602	0.379	0.653	0.640	0.751	**0.531**	0.544
seg clean purge r101 mask	0.467	0.298	0.602	0.356	0.656	0.638	0.755	**0.520**	0.537
YoloV4 bbox	0.372	0.153	0.416	0.133	0.517	0.443	0.641	0.334	0.376
YoloV4 mask	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Figure 4.37: Results on testing dataset

	person	bicycle	car	bus	train	truck	mgAP
seg clean dontcare r101 vosstrex bbox	**0.625**	**0.466**	**0.686**	0.408	0.355	0.298	0.464
seg clean dontcare r101 vosstrex mask	**0.530**	**0.471**	**0.662**	0.457	0.358	0.304	0.459
seg clean dontcare r101 un vosstrex bbox	0.602	0.459	0.675	0.427	0.338	0.346	0.468
seg clean dontcare r101 un vosstrex mask	0.486	0.456	0.631	0.439	0.340	0.346	0.445
seg clean dontcare r101 bbox	0.487	0.270	0.616	0.645	0.741	0.726	0.567
seg clean dontcare r101 mask	0.469	0.270	0.615	0.648	0.741	**0.729**	0.566
seg clean dontcare r101 un bbox	0.529	0.359	0.643	**0.689**	**0.820**	**0.729**	**0.598**
seg clean dontcare r101 un mask	0.486	0.351	0.629	**0.689**	**0.820**	0.729	**0.586**

Figure 4.38: Results on Vosstrex testing dataset with 1:1 matching scenes

4.12 Tracking results

The tracking was evaluated with MOTS [56] metrics described in Sec. 4.4.15, only the tracking on real data was evaluated with Gaussian localization error 4.4.17, since the VR annotations have no uncertainty.

As for tracking results, there were small complications with the evaluation on real data, because the trackers such as DeepSORT [18] are still not performing too well for automatically generated track annotations, even if this tracker is probably the most universal towards unseen datasets.

%	person	bicycle	car	motorcycle	bus	train	truck	traffic light
MOTSA	53.7	48.3	59.1	49.4	53.4	51.8	52.3	49.7
MOTSA $+Error$	+6.1	+6.5	+7.6	+6.7	+2.7	+1.8	+1.7	+6.5
MOTSA $-Error$	−6.8	−7.8	−8.1	−7.2	−3.0	−2.1	−1.9	−7.1
MOTSP	64.2	48.5	58.9	43.8	67.1	58.4	65.3	53.6
MOTSP $\pm Error$	±12.2	±14.1	±15.4	±13.2	±5.3	±3.1	±5.2	±13.6
Mostly Tracked	64.6	41.6	64.8	49.3	57.8	59.6	53.7	47.3
Mostly Lost	15.3	18.5	2.3	14.3	12.7	13.5	12.1	18.7

Figure 4.39: Tracking on real scenes

The results for the tracking on real scenes Fig. 4.39 look very good, however this is because of the annotations were far from perfect and it was checked in tracking video as well. If the Tracker creating the annotations make the same mistakes as inference tracker, then it can have these misleading results. The automatically generated track annotations had good enough quality to train the re-identification, however the inconsistency of tracks makes them not viable for testing, especially compared to VR, which has 100% accurate tracks. Since it was not possible to obtain automatically generated testing data to 1:1 match VR scenes, the 1:1 comparison of tracking was deprecated in this work. For planned tracking experiments only Vosstrex was used, since the Vosstrex generated testing track annotations are very good benchmark for trained tracker.

%	person	bicycle	car	bus	train	truck
MOTSA	32.5	24.7	35.1	29.8	24.8	27.3
MOTSP	55.2	35.5	60.3	36.3	38.9	37.6
Mostly Tracked	8.5	11.6	17.8	11.3	10.3	13.6
Mostly Lost	36.2	32.3	29.3	32.3	38.7	37.1

Figure 4.40: Tracking on VR scenes

Tracking on Vosstrex ended up worse in score, but at least it was much better evaluation of tracking then the real testing data. Most likely the only metric to uncover the bad performance of tracker which have good accuracy is that the Mostly Tracked were suspiciously good in the real data case. However The VR has also the problem with great amount of GT which are badly detectable and which might be reason for low Mostly Tracked metric on VR.

%	person	bicycle	car	bus	train	truck
MOTSA	26.8	18.3	26.0	23.2	19.7	22.4
MOTSP	55.2	35.5	60.3	36.3	38.9	37.6
Mostly Tracked	4.8	8.3	9.9	4.8	6.7	8.3
Mostly Lost	38.1	34.5	28.9	33.7	39.0	39.3

Figure 4.41: Tracking on VR scenes with only re-identification

In comparison between VR and reality the detection is the main task which needs to be properly compared. The proposed tracking method 1.2 interacts with VR only with the re-identification, the other parts are not interacting directly with image data where the reality and VR differs. This experiment was about setting the weighting between Kalman Filter and re-identification to 100% 1.3 for re-identification to see if the re-identification trained on real data works on VR data.

With bit worse results for discarding Kalman Filter, the tracker is still capable to find tracks, however the Mostly Tracked rating is much worse, that could be because lot of identity switches on some re-id matches, most likely when the object rotates and changes its appearance, these cases usually are guessed correctly with Kalman Filter, since it is interested in position and not appearance.

%	person	bicycle	car	bus	train	truck
MOTSA	39.7	28.3	50.1	31.2	38.7	39.4
MOTSP	57.3	36.7	62.5	38.4	41.2	39.7
Mostly Tracked	12.8	13.1	25.8	20.3	24.7	25.0
Mostly Lost	28.4	27.6	15.2	22.7	18.9	19.3

Figure 4.42: Tracking on VR scenes 5× frame rate

The experiment of tracking with 30 FPS instead of 6 FPS (the tracked objects are closer to each other which is easier) is showing, that the theories about trying to track fast to increase performance [15] have some truth in them. Especially cars started to get good tracking results, but these are probably the easiest to track by Kalman Filter.

%	person	bicycle	car	bus	train	truck
MC MOTSA	20.3	18.4	27.9	22.3	20.4	19.8
MOTSP	55.2	35.5	60.3	36.3	38.9	37.6
Mostly Tracked	7.8	9.3	13.9	10.0	8.6	11.4
Mostly Lost	37.3	35.8	31.2	44.8	39.0	38.5

Figure 4.43: Multi-camera Tracking

The last experiment was multi-camera tracking which has the worst results, this task would have been much easier if this dataset had some distance measure from surroundings, this way the tracker relied only on re-identification, therefore the results are not that amazing.

Overall the tracking results were lesser success because the comparison between real and virtual data was not achievable in conditions when it is not possible to obtain good real testing data. In conclusion the 1:1 comparison of tracking was not successful, however there were experiments on VR data which proved that re-identification which is the part of tracker most sensitive to changing from reality to VR still achieved acceptable results.

Conclusion

To conclude finished work in this thesis, the dataset annotations were automatically generated and new data cleaning techniques were tested, the detectors ended up trained very well, the secondary project of detection on distorted images was success as well.

Convenient tools for tasks similar to this one were created and new evaluator capable of including Gaussian localization error of annotation misplacement to evaluation was explained and implemented.

The trained detection models on both virtual and real testing data performed very well and the patterns in their results were very similar, which is good for claiming that the detection comparison was a success.

The tracking was tested as well in many experiments, including multi-camera tracking. Unfortunately the automatically generated real testing data track annotations did not have very good quality for testing dataset. Only the tracking learning algorithm was usable on real data, due to mining triplets from training dataset instead of being forced to require full tracks. This resulted in deprecating the full 1:1 tracking comparison. Instead the tracking was at least tested on VR and the re-identification part of tracker was functional even on VR images, which is also considered success.

The final conclusion is that VR project Vosstrex is viable for validation of detection systems. For tracking the re-identification, which is the only part of tracker vulnerable to difference between real and VR data proved to be working on VR as well, however to completely certify viability of tracking it would require better real tracking testing data.

Bibliography

[1] Council of European Union. Council regulation (EU) no 269/2014. 2014, http://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1416170084502&uri=CELEX:32014R0269.

[2] Richter, S. R.; Vineet, V.; et al. Playing for Data: Ground Truth from Computer Games. 2016, 1608.02192.

[3] VALEO. https://www.valeo.com/.

[4] Daimler: Mercedes Benz. 2020. Available from: https://www.daimler.com/company/business-units/mercedes-benz-cars/

[5] The Mathworks, Inc., Natick, Massachusetts. *MATLAB version 9.3.0.713579 (R2017b)*. 2017.

[6] OpenCV. https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/.

[7] Roxas, M.; Oishi, T. Real-Time Variational Fisheye Stereo without Rectification and Undistortion. 2019, 1909.07545.

[8] Zhu, J.; Zhu, J.; et al. Object detection and localization in 3D environment by fusing raw fisheye image and attitude data. *Journal of Visual Communication and Image Representation*, volume 59, 2019: pp. 128 – 139, ISSN 1047-3203, doi:https://doi.org/10.1016/j.jvcir.2019.01.005. Available from: http://www.sciencedirect.com/science/article/pii/S1047320319300069

[9] Deng, L.; Yang, M.; et al. CNN based semantic segmentation for urban traffic scenes using fisheye camera. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 231–236, doi:10.1109/IVS.2017.7995725.

[10] Álvaro, S.; M., B. L.; et al. Real-Time Semantic Segmentation for Fisheye Urban Driving Images Based on ERFNet. 2017, 1705.04608.

[11] Goodarzi, P.; Stellmacher, M.; et al. Optimization of a CNN-based Object Detector for Fisheye Cameras. In *2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, 2019, pp. 1–7, doi:10.1109/ICVES.2019.8906325.

[12] Baek, I.; Davies, A.; et al. Real-time Detection, Tracking, and Classification of Moving and Stationary Objects using Multiple Fisheye Images. 2018, 1803.06077.

[13] Bertozzi, M.; Castangia, L.; et al. 360° Detection and tracking algorithm of both pedestrian and vehicle using fisheye images. *2015 IEEE Intelligent Vehicles Symposium (IV)*, 2015: pp. 132–137.

[14] Wang, Z.; Zheng, L.; et al. Towards Real-Time Multi-Object Tracking. 2020, 1909.12605.

[15] Bergmann, P.; Meinhardt, T.; et al. Tracking without bells and whistles. 2019, 1903.05625.

[16] Bewley, A.; Ge, Z.; et al. Simple online and realtime tracking. *2016 IEEE International Conference on Image Processing (ICIP)*, Sep 2016, doi:10.1109/icip.2016.7533003. Available from: http://dx.doi.org/10.1109/ICIP.2016.7533003

[17] Beyer, L.; Breuers, S.; et al. Towards a Principled Integration of Multi-Camera Re-Identification and Tracking through Optimal Bayes Filters. 2017, 1705.04608.

[18] Wojke, N.; Bewley, A.; et al. Simple Online and Realtime Tracking with a Deep Association Metric. 2017, 1703.07402.

[19] Chen, L.; Ai, H.; et al. Real-Time Multiple People Tracking with Deeply Learned Candidate Selection and Person Re-Identification. *2018 IEEE International Conference on Multimedia and Expo (ICME)*, Jul 2018, doi:10.1109/icme.2018.8486597. Available from: http://dx.doi.org/10.1109/ICME.2018.8486597

[20] Chen, L.; Ai, H.; et al. Real-Time Multiple People Tracking with Deeply Learned Candidate Selection and Person Re-Identification. *2018 IEEE International Conference on Multimedia and Expo (ICME)*, Jul 2018, doi:10.1109/icme.2018.8486597. Available from: http://dx.doi.org/10.1109/ICME.2018.8486597

[21] Zheng, Z.; Zheng, L.; et al. A Discriminatively Learned CNN Embedding for Person Reidentification. *ACM Transactions on Multimedia Computing, Communications, and Applications*, volume 14, no. 1, Jan 2018: p. 1–20, ISSN 1551-6865, doi:10.1145/3159171. Available from: http://dx.doi.org/10.1145/3159171

[22] Liu, W.; Liao, S.; et al. High-Level Semantic Feature Detection: A New Perspective for Pedestrian Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[23] *Grand Theft Auto V*. New York, NY: Rockstar Games, 2014.

[24] He, K.; Gkioxari, G.; et al. Mask R-CNN. 2018, 1703.06870.

[25] Bochkovskiy, A.; Wang, C.-Y.; et al. YOLOv4: Optimal Speed and Accuracy of Object Detection. 2020, 2004.10934.

[26] Goodfellow, I.; Bengio, Y.; et al. *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[27] Quian, Q. R.; L., R.; et al. Invariant visual representation by single neurons in the human brain. volume 435, 2005: pp. 1102 – 1107, doi: https://doi.org/10.1038/nature03687.

[28] Ren, S.; He, K.; et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 2016, 1506.01497.

[29] Wu, Y.; Kirillov, A.; et al. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[30] Honda, H. Digging into Detectron2 [online]. https://medium.com/@hirotoschwert/digging-into-detectron-2-47b2e794fabd, 2020.

[31] Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2015, 1409.1556.

[32] He, K.; Zhang, X.; et al. Deep Residual Learning for Image Recognition. 2015, 1512.03385.

[33] Lin, T.-Y.; Dollár, P.; et al. Feature Pyramid Networks for Object Detection. 2017, 1612.03144.

[34] Hosang, J.; Benenson, R.; et al. Learning Non-Maximum Suppression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[35] Welch, G.; Bishop, G. An Introduction to the Kalman Filter. Technical report, USA, 1995.

[36] Suprem, A.; Pu, C. Looking GLAMORous: Vehicle Re-Id in Heterogeneous Cameras Networks with Global and Local Attention. 2020, 2002.02256.

[37] Data Definition Language Library - the ADTF data description language (Automotive Data and Time-Triggered Framework). https://github.com/audi/ddl.

[38] Jacob Solawetz. https://blog.roboflow.com/train-test-split/.

[39] Bora, D.; Gupta, A.; et al. Comparing the Performance of L*A*B* and HSV Color Spaces with Respect to Color Image Segmentation. 06 2015.

[40] Purple Range Rover. 2020. Available from: http://www.iluxurystyle.com/luxury/cars/purple-people-eater.html/

[41] Van Rossum, G.; Drake Jr, F. L. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[42] Harris, C. R.; Millman, K. J.; et al. Array programming with NumPy. *Nature*, volume 585, no. 7825, Sept. 2020: pp. 357–362, doi:10.1038/s41586-020-2649-2. Available from: https://doi.org/10.1038/s41586-020-2649-2

[43] NVIDIA; Vingelmann, P.; et al. CUDA, release: 10.2.89. 2020. Available from: https://developer.nvidia.com/cuda-toolkit

[44] Abadi, M.; Barham, P.; et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[45] Paszke, A.; Gross, S.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. Available from: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[46] Pygame contributors. Pygame Front Page — Pygame v1.9.2 documentation. Available online: http://www.pygame.org/docs/ [Accessed 14/01/2018], 2018.

[47] Mukhometzianov, R.; Carrillo, J. CapsNet comparative performance evaluation for image classification. 2018, 1805.11195.

[48] Huang, Z.; Huang, L.; et al. Mask Scoring R-CNN. In *CVPR*, 2019.

[49] Lin, T.-Y.; Maire, M.; et al. Microsoft COCO: Common Objects in Context. 2015, 1405.0312.

[50] Deng, J.; Dong, W.; et al. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[51] Hermans, A.; Beyer, L.; et al. In Defense of the Triplet Loss for Person Re-Identification. 2017, 1703.07737.

[52] Everingham, M.; Van Gool, L.; et al. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, 2012.

[53] Kuhn, H. W.; Yaw, B. The Hungarian method for the assignment problem. *Naval Res. Logist. Quart*, 1955: pp. 83–97.

[54] Hui, J. Mean average Precision for object detection. 2020. Available from: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

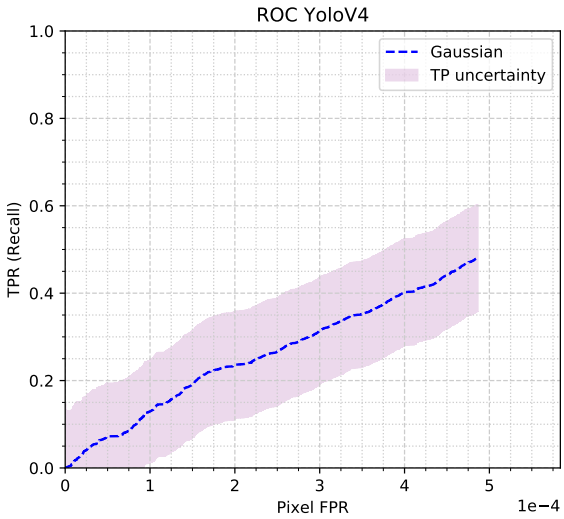
[55] MIT: The 3-sigma rule. 2020. Available from: https://news.mit.edu/2012/explained-sigma-0209

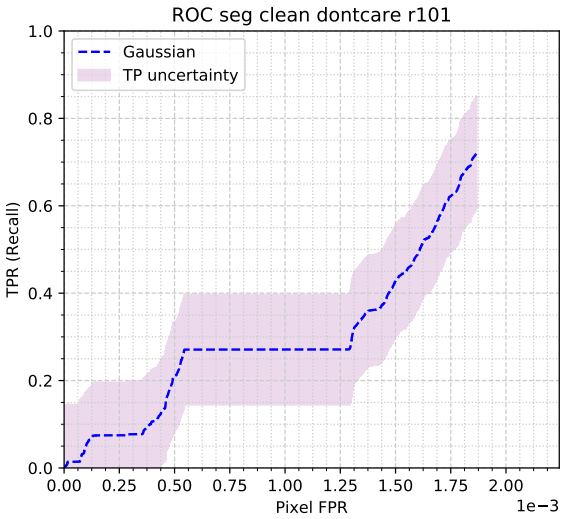
[56] Voigtlaender, P.; Krause, M.; et al. MOTS: Multi-Object Tracking and Segmentation. 2019, 1902.03604.

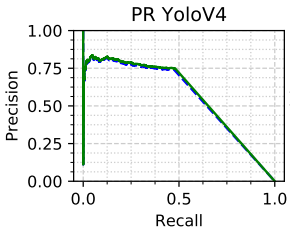
Appendix A

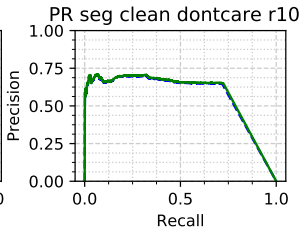
YoloV4 distorted images comparison

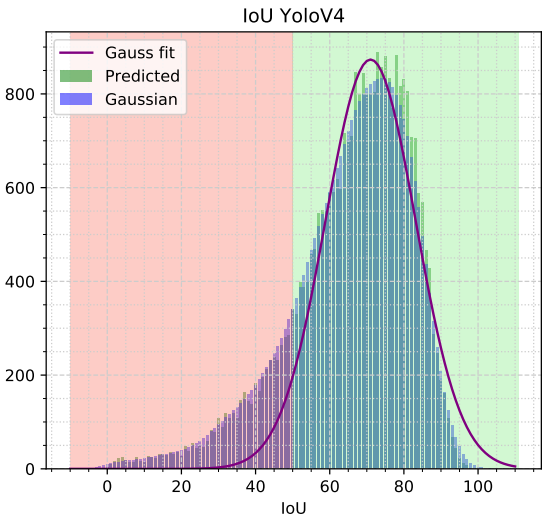
person

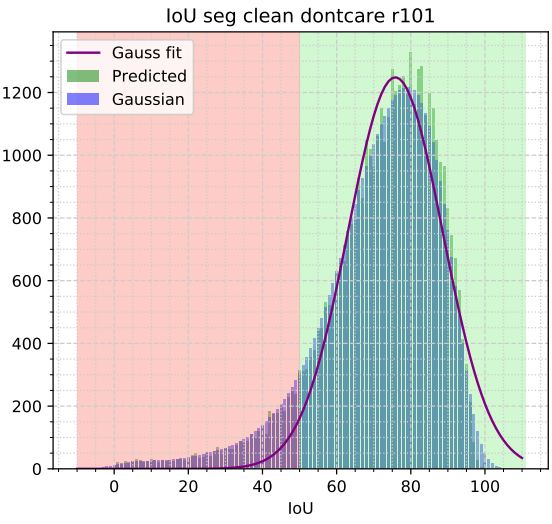

ROC YoloV4
Gaussian
TP uncertainty
TPR (Recall)
Pixel FPR
1e−4


ROC seg clean dontcare r101
Gaussian
TP uncertainty
TPR (Recall)
Pixel FPR
1e−3

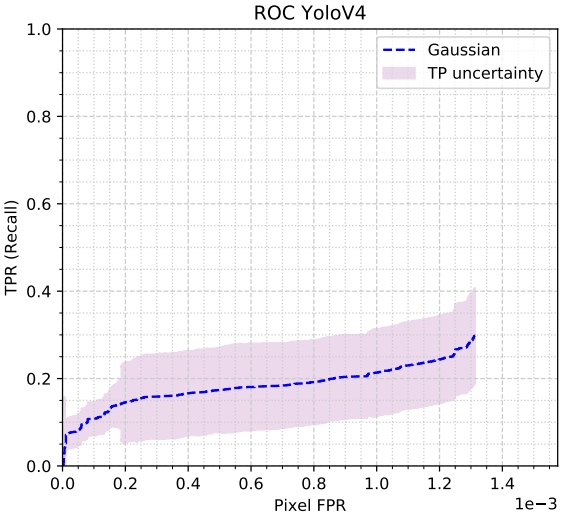

PR YoloV4
Precision
Recall

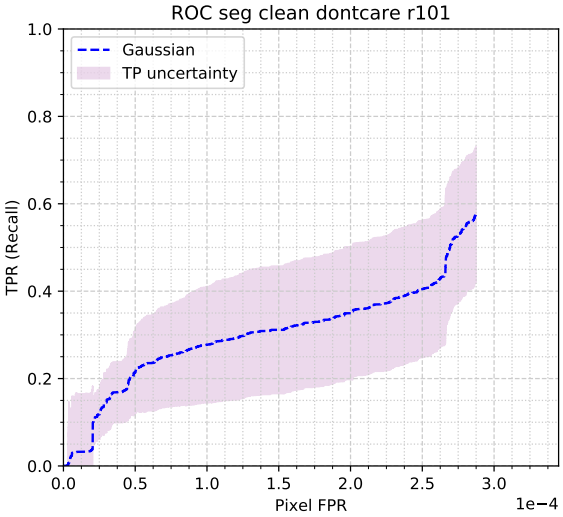

PR seg clean dontcare r101
Precision
Recall

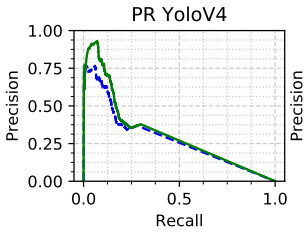

IoU YoloV4
Gauss fit
Predicted
Gaussian
IoU

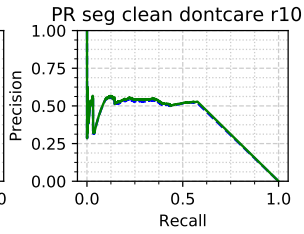

IoU seg clean dontcare r101
Gauss fit
Predicted
Gaussian
IoU

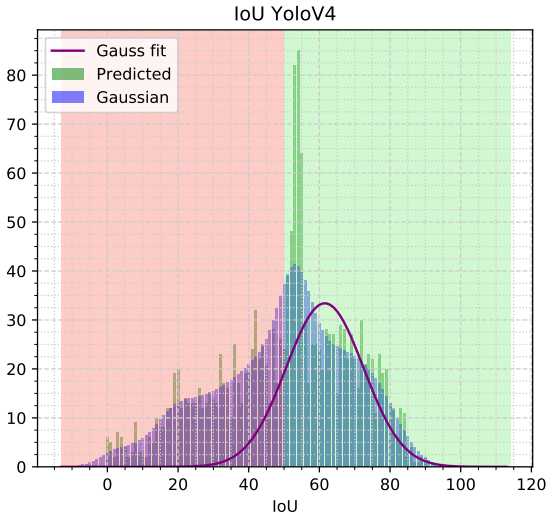
bicycle

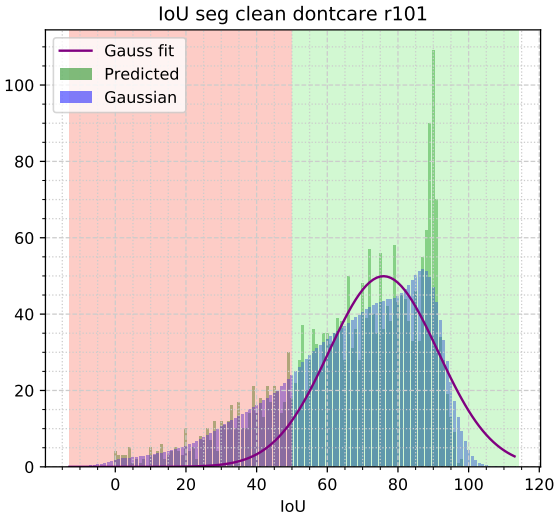

ROC YoloV4
Gaussian
TP uncertainty
TPR (Recall)
Pixel FPR
1e−3


ROC seg clean dontcare r101
Gaussian
TP uncertainty
TPR (Recall)
Pixel FPR
1e−4

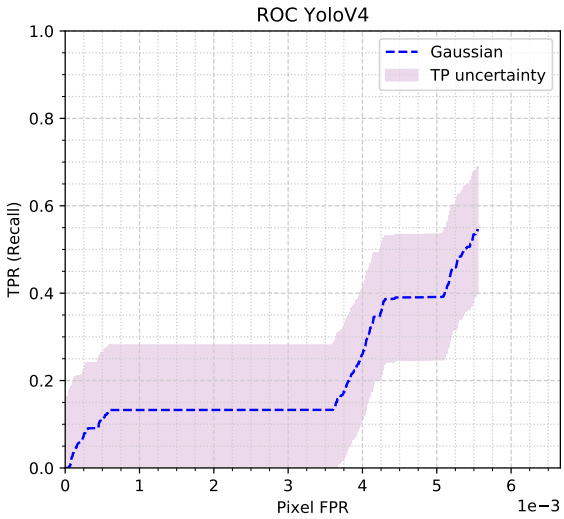

PR YoloV4
Precision
Recall

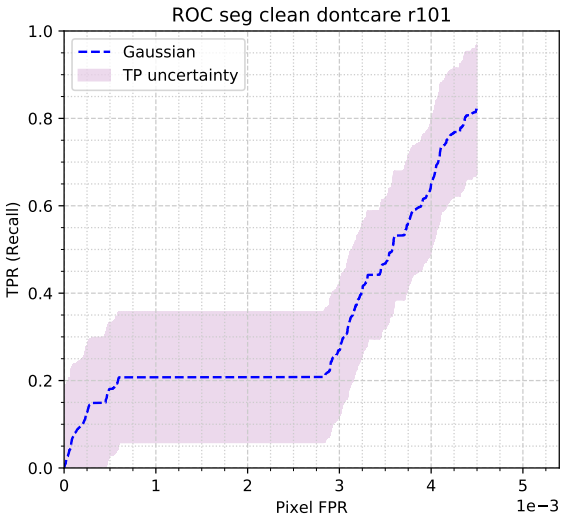

PR seg clean dontcare r101
Precision
Recall

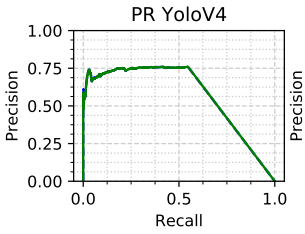

IoU YoloV4
Gauss fit
Predicted
Gaussian
IoU

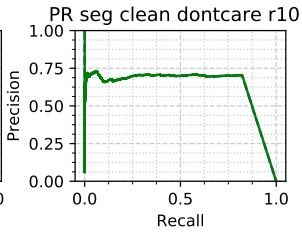

IoU seg clean dontcare r101
Gauss fit
Predicted
Gaussian
IoU

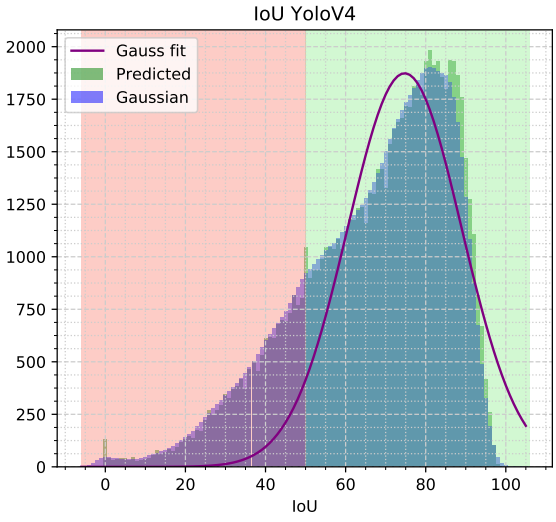
car

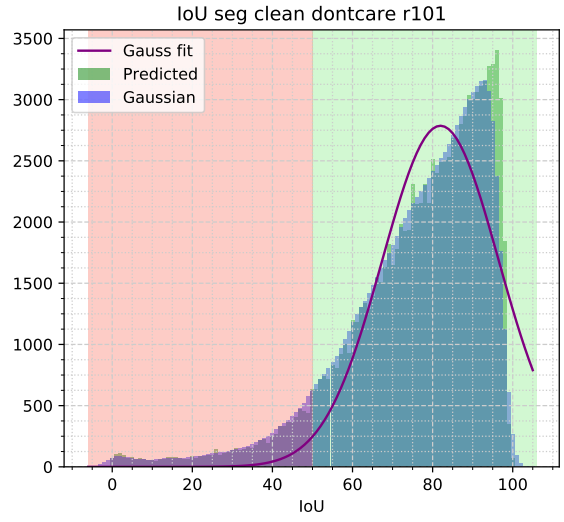

ROC YoloV4
Gaussian
TP uncertainty
TPR (Recall)
Pixel FPR
1e−3


ROC seg clean dontcare r101
Gaussian
TP uncertainty
TPR (Recall)
Pixel FPR
1e−3

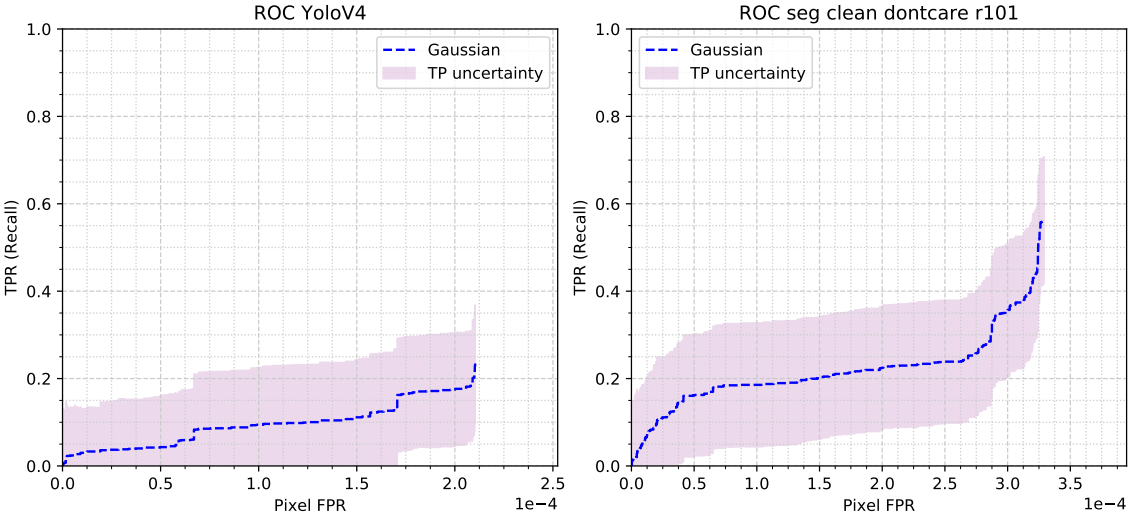

PR YoloV4
Precision
Recall


PR seg clean dontcare r101
Precision
Recall

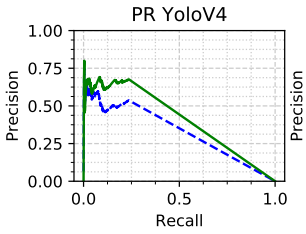

IoU YoloV4
Gauss fit
Predicted
Gaussian
IoU

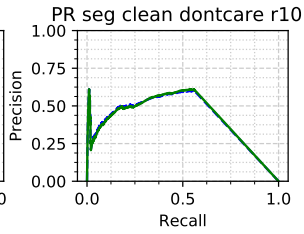

IoU seg clean dontcare r101
Gauss fit
Predicted
Gaussian
IoU

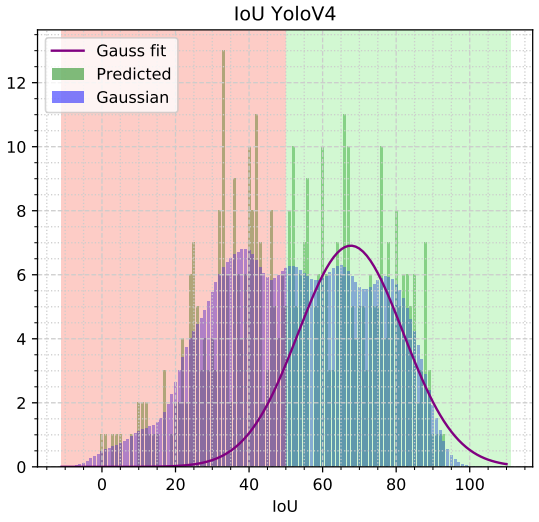
motorcycle

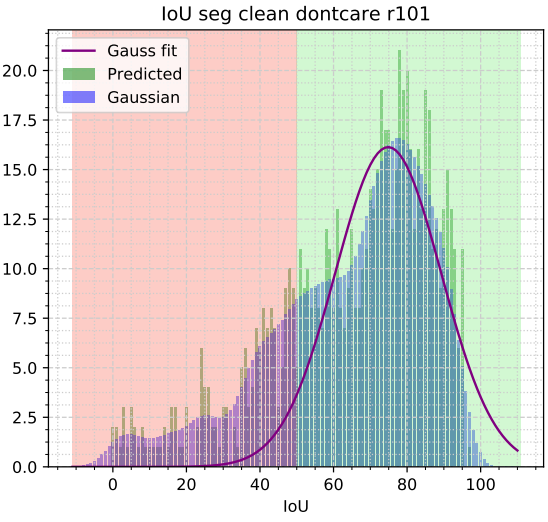

ROC YoloV4
Gaussian
TP uncertainty
TPR (Recall)
Pixel FPR
1e−4

ROC seg clean dontcare r101
Gaussian
TP uncertainty
TPR (Recall)
Pixel FPR
1e−4

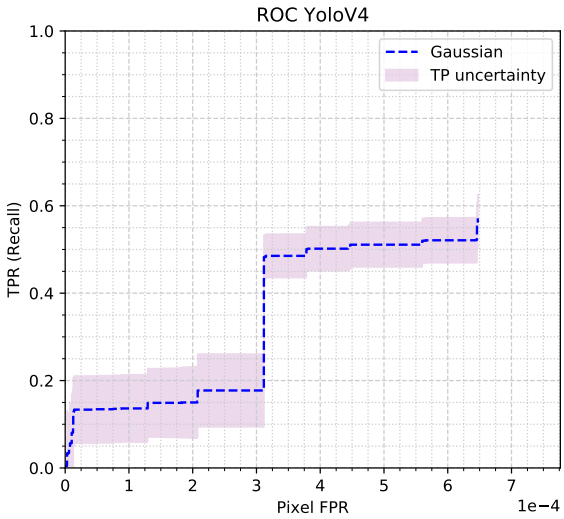

PR YoloV4
Precision
Recall

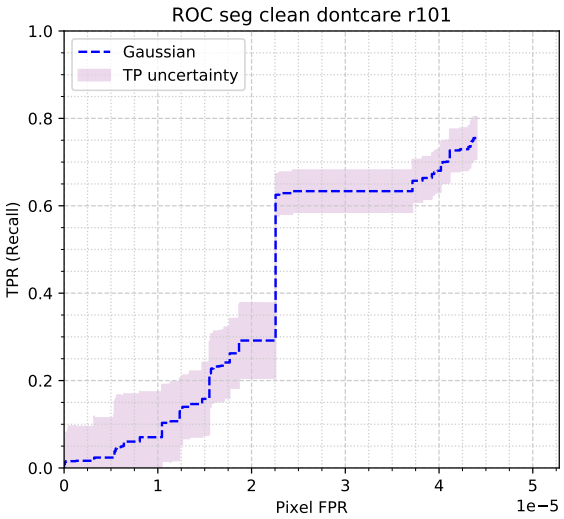

PR seg clean dontcare r101
Precision
Recall

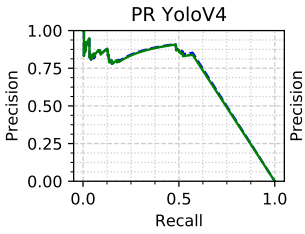

IoU YoloV4
Gauss fit
Predicted
Gaussian
IoU

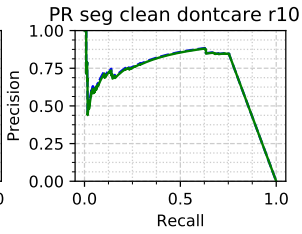

IoU seg clean dontcare r101
Gauss fit
Predicted
Gaussian
IoU

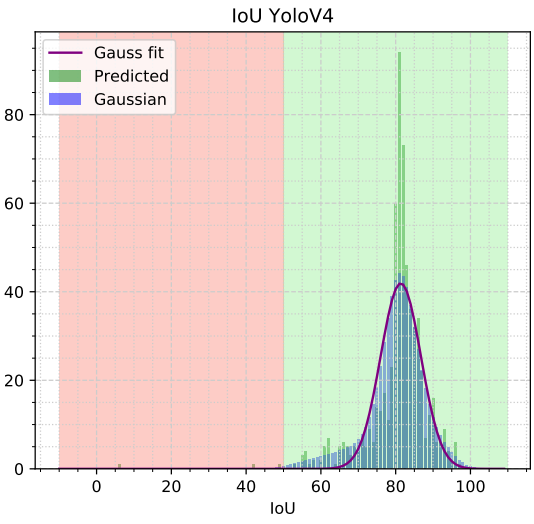
bus

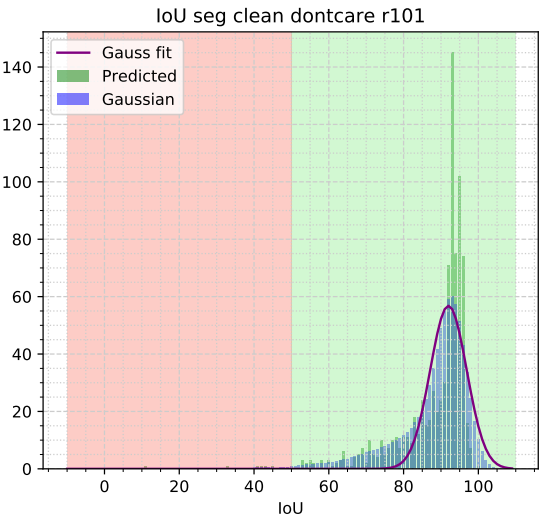

ROC YoloV4
Gaussian
TP uncertainty
TPR (Recall)
Pixel FPR
1e−4


ROC seg clean dontcare r101
Gaussian
TP uncertainty
TPR (Recall)
Pixel FPR
1e−5

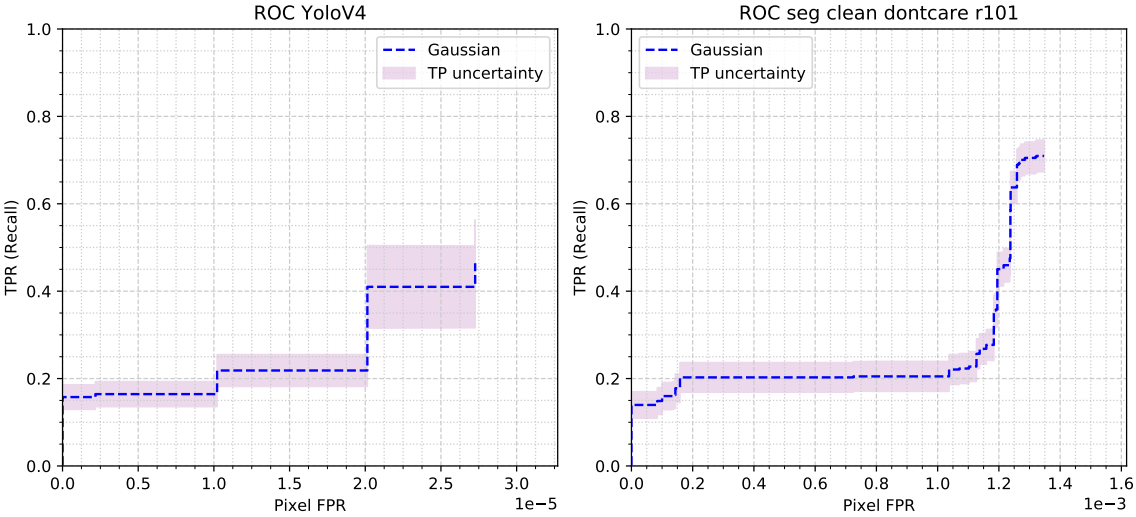

PR YoloV4
Precision
Recall


PR seg clean dontcare r101
Precision
Recall

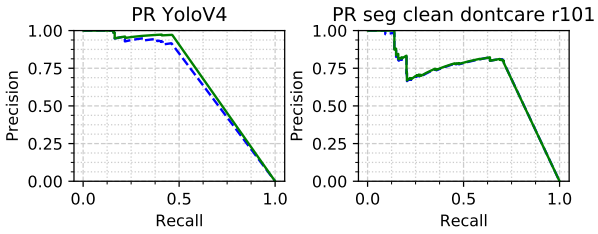

IoU YoloV4
Gauss fit
Predicted
Gaussian
IoU

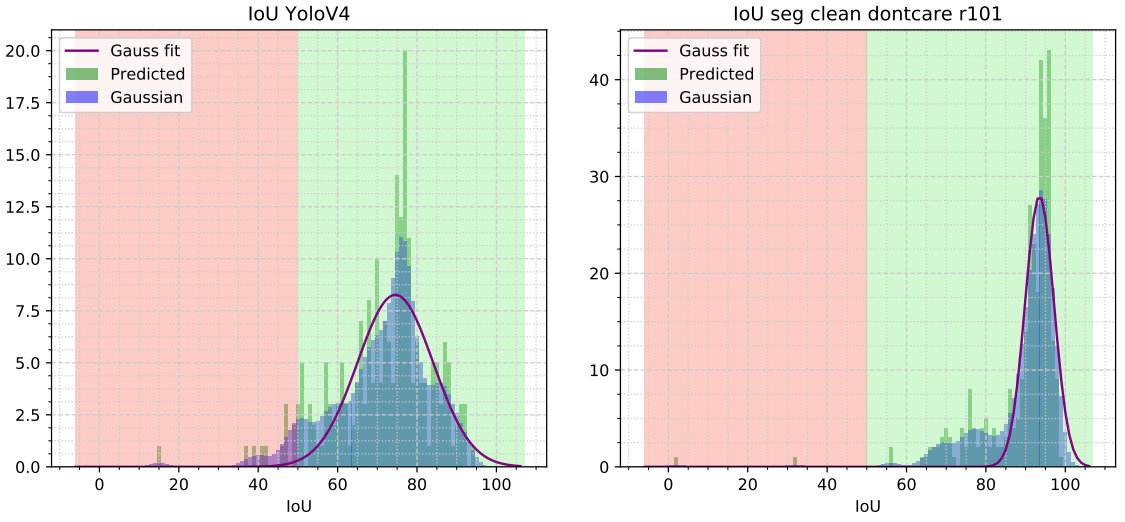

IoU seg clean dontcare r101
Gauss fit
Predicted
Gaussian
IoU

train


ROC YoloV4
Gaussian
TP uncertainty
TPR (Recall)
Pixel FPR
1e−5

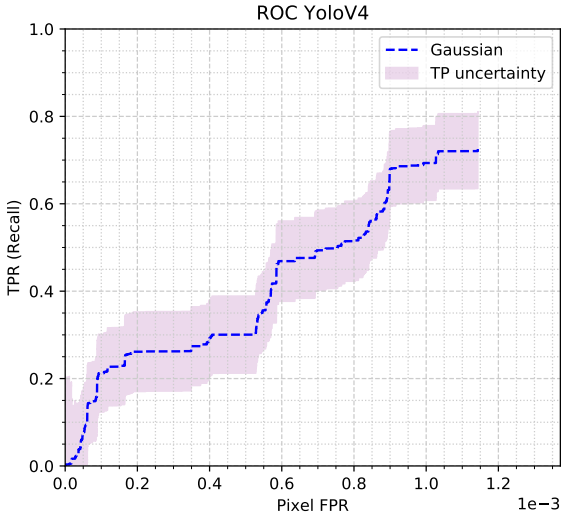
ROC seg clean dontcare r101
Gaussian
TP uncertainty
TPR (Recall)
Pixel FPR
1e−3

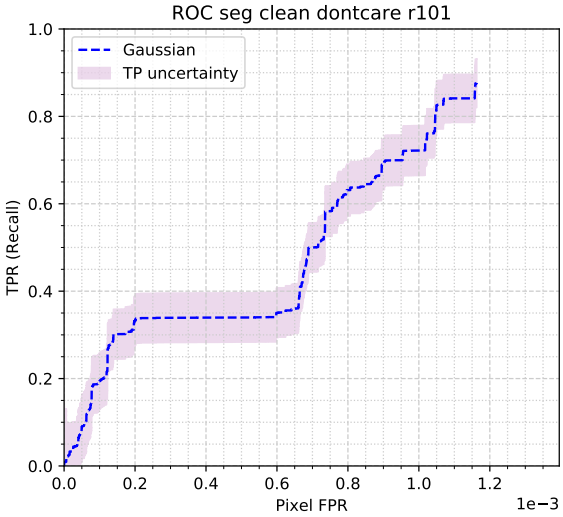

PR YoloV4
PR seg clean dontcare r101
Precision
Recall

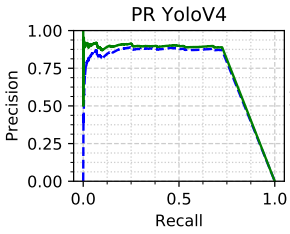

IoU YoloV4
Gauss fit
Predicted
Gaussian
IoU

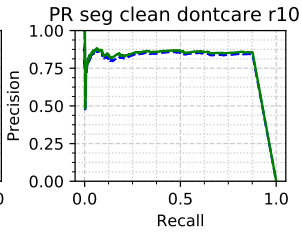
IoU seg clean dontcare r101
Gauss fit
Predicted
Gaussian
IoU

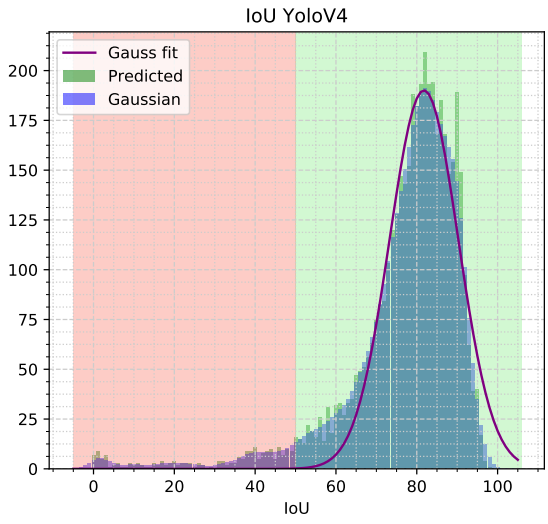
truck

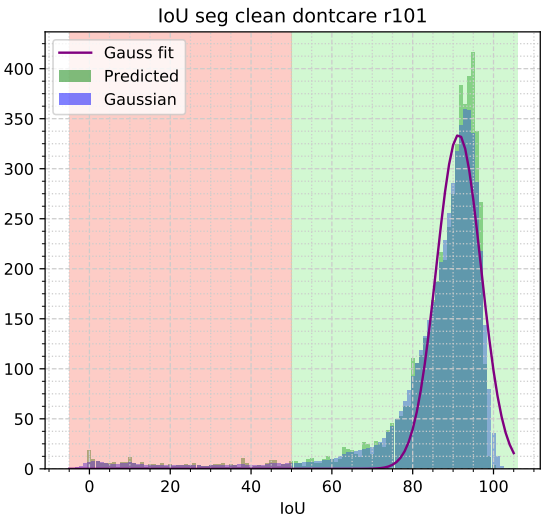

ROC YoloV4
Gaussian
TP uncertainty
TPR (Recall)
Pixel FPR
1e−3


ROC seg clean dontcare r101
Gaussian
TP uncertainty
TPR (Recall)
Pixel FPR
1e−3

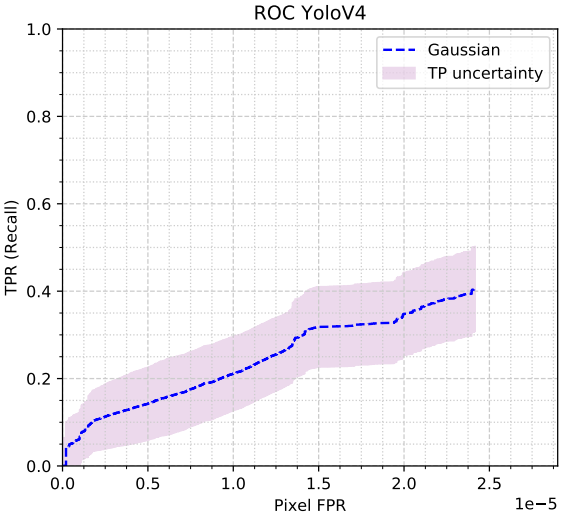

PR YoloV4
Precision
Recall

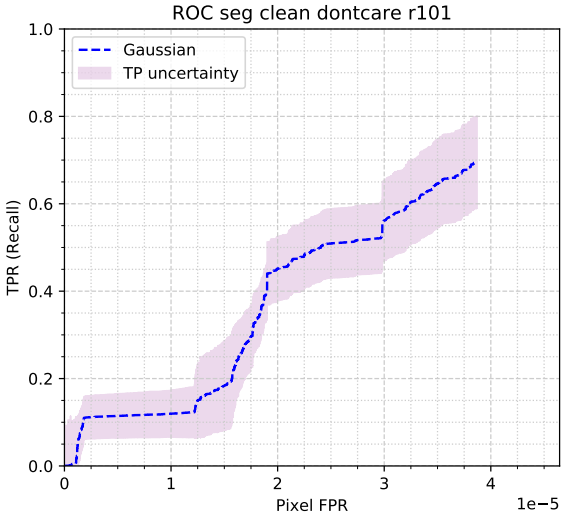

PR seg clean dontcare r101
Precision
Recall

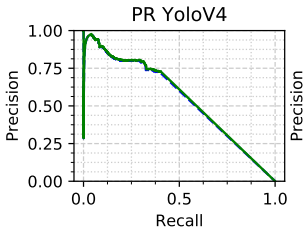

IoU YoloV4
Gauss fit
Predicted
Gaussian
IoU

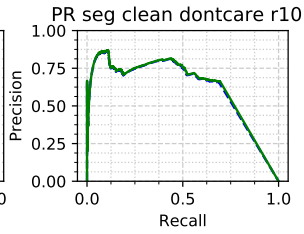

IoU seg clean dontcare r101
Gauss fit
Predicted
Gaussian
IoU

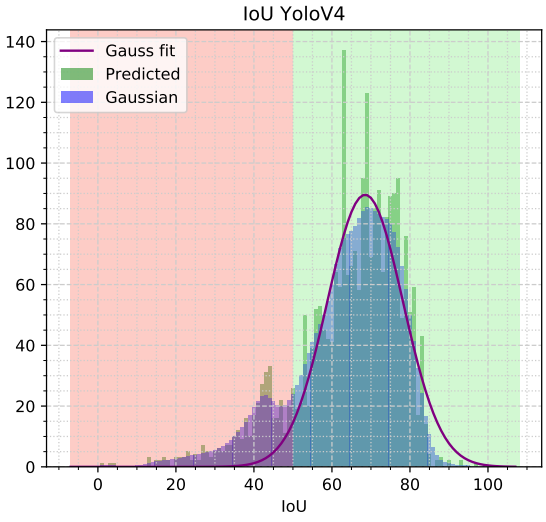
traffic light

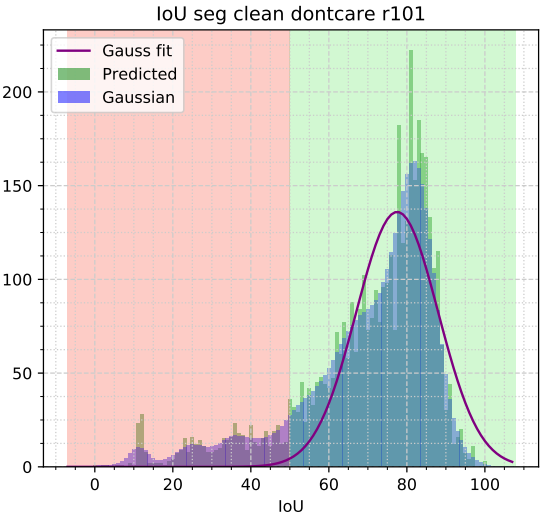












Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `thesis` ..... the directory of LaTeX source codes of the thesis
  - `trackframework` . the directory of detection and tracking source codes
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format