Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Multimediakommunikation und Signalverarbeitung

Prof. Dr.-Ing. André Kaup

Master Thesis

Saliency Coding of Video Data for Machine to Machine Communication

Felix Fleckenstein

March 2020

Supervisors: Kristian Fischer, M.Sc.
Prof. Dr.-Ing. André Kaup

Lehrstuhl für Multimediakommunikation und Signalverarbeitung
Prof. Dr.-Ing. A. Kaup

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Masterarbeit

für

Felix Fleckenstein

Saliency Codierung von Videodaten für maschinelle Kommunikation

Saliency Coding of Video Data for Machine to Machine Communication

Bei der Codierung von Videodaten für den Menschen als Endbenutzer hat sich gezeigt, dass es nützlich ist, psychovisuelle Aspekte in die Codierung mit einfließen zu lassen. So können Bereiche, in denen die Artefakte nicht so stark auffallen, mit einer gröberen Quantisierung codiert werden. Die dabei eingesparten Bits können an anderen, auffälligeren Stellen dazu verwendet werden die subjektive Qualität dort zu erhöhen. Die dafür benötigte Unterscheidung erfolgt dabei zumeist auf Coding Tree Ebene. Das Prinzip, Blöcke innerhalb eines Frames mit unterschiedlicher Qualität zu codieren, wird Saliency Codierung genannt.

Dieser Saliency Ansatz soll nun von der Codierung für den Menschen, auf die Codierung für neuronale Detektionsnetzwerke als finaler Nutzer übertragen werden. Dabei soll vor der Codierung ein Detektionsalgorithmus die Bereiche erkennen, welche später wahrscheinlich detektiert werden. Die anderen Bereiche können dann mit einer höheren Quantisierung codiert werden, was letztendlich zu einer Einsparung der benötigten Bits führt, bei möglichst gleichbleibender Erkennungsrate des daran anschließenden Detektionsnetzwerks.

Für diesen Ansatz soll nun eine Saliency Codierung in die Versatile Video Coding Referenzsoftware eingebaut werden. Dabei kann sich an bestehenden Ansätzen wie dem adaptiven Quantisierungsparameter orientiert werden. Zur Bestimmung der relevanten Regionen sollen geeignete Erkennungsalgorithmen oder –netzwerke gefunden und implementiert werden. Als finales Erkennungsnetzwerk kann zum Beispiel das Faster CNN benutzt werden. Abschließend soll analysiert werden, ob das implementierte Framework Bits bei der Codierung einsparen kann, wenn der Endbenutzer kein Mensch ist, sondern ein maschinelles Erkennungsnetzwerk. Dabei sollen auch unterschiedliche Parametrierungen getestet werden.

Im Rahmen der Arbeit wird auf eine saubere Dokumentation der Arbeit, des Quellcodes sowie eine umfassende Analyse der Ergebnisse besonderer Wert gelegt.

Beginn: 03. September 2019
Abgabe: 03. März 2020

A. Kaup

(Prof. Dr.-Ing. A. Kaup)

Declaration

I confirm that I have written this thesis unaided and without using sources other than those listed and that this thesis has never been submitted to another examination authority and accepted as part of an examination achievement, neither in this form nor in a similar form. All content that was taken from a third party either verbatim or in substance has been acknowledged as such.

Place, Date

Felix Fleckenstein
Obere Schmiedgasse 22
90403 Nürnberg

Contents

Kurzfassung

Saliency-Codierung beschreibt die inhaltsabhängige Kompression von Frames mit unterschiedlicher Qualität und hilft dabei, visuell störende Artefakte zu reduzieren, indem Eigenschaften des menschlichen Sehsystems berücksichtigt werden. Genauer gesagt haben Experimente gezeigt, dass Kompressionsartefakte in Bereichen mit wenig visueller Aktivität als störender wahrgenommen werden, als in stark strukturierten Bereichen. Aus diesem Grund werden Regionen mit niedriger visueller Aktivität mit einer höheren Qualität und Regionen mit hoher visueller Aktivität mit einer niedrigeren Qualität codiert. Aufgrund der jüngsten Fortschritte im Bereich computergestütztes Sehen gibt es immer mehr Videodaten, die von Algorithmen verarbeitet statt von Menschen betrachtet werden. Dies ist z.B. im Bereich Videoüberwachung oder autonomes Fahren der Fall. In diesen Systemen ist es oft notwendig, die aufgenommenen Videodaten zu einem leistungsfähigeren zentralen Server weiterzuleiten, der dann zur weiteren Verarbeitung der Daten genutzt wird, z.B. um Objekte zu erkennen. Da die Bandbreite der Übertragungskanäle begrenzt ist, müssen die Daten auf eine effiziente Weise komprimiert werden, sodass die Qualität der wichtigen Bereiche im Bild erhalten bleibt. Deshalb wird die Idee der Saliency-Codierung für den Menschen in dieser Arbeit auf die Saliency-Codierung für maschinelle Kommunikation übertragen. Regionen, die wahrscheinlich Objekte beinhalten, werden mit einer besseren Qualität codiert und irrelevante Regionen wie der Hintergrund werden mit einer schlechteren Qualität codiert. Um Regionen zu detektieren, die Objekte beinhalten, werden Algorithmen mit niedriger Komplexität angewandt, die auch von Systemen mit niedriger Leistungsfähigkeit verwendet werden können. Anschließend können die Bilder auf zwei Arten komprimiert werden. Im ersten Ansatz wird die an die Wahrnehmung angepasste Bitallokation, die im Versatile Video Coding Test Model (VTM) 6.1 verwendet wird, an die maschinelle Kommunikation angepasst. Im zweiten Ansatz wird ein Durchschnittswert-Filter als Vorverarbeitungsschritt hinzugefügt, der Regionen im Hintergrund filtert. Die Ergebnisse dieser Arbeit zeigen, dass die Dateigrößen der Bilder signifikant verringert werden können, ohne die Leistungsfähigkeit von Objektdetektionsnetzwerken wie Faster Region-based Convolutional Neural Network (R-CNN) oder Mask R-CNN zu verschlechtern.

Abstract

Saliency coding describes the content-adaptive compression of frames with different quality and helps to reduce visually annoying compression artifacts by taking properties of the human visual system into account. More specifically, experiments have shown that compression artifacts are perceived as more annoying in smooth regions than in highly textured areas. Because of that, smooth regions are compressed with higher quality and regions with high visual activity are coded with worse quality. Due to the recent advances in computer-vision-based systems, there is a growing amount of video data being processed by algorithms instead of being watched by human viewers. This is the case, for instance, in video surveillance or autonomous driving. In these systems, it is often required to transmit the obtained video data to a computationally more powerful central server that is used for further processing of the data, e.g., for detecting objects. As the bandwidth of the transmission channels is limited, the data has to be compressed efficiently such that the quality of the important parts in the image is preserved. The idea of saliency coding for human viewers is hence transferred to saliency coding for machine to machine communication in this work. Regions that are likely to contain objects are compressed with better quality and irrelevant regions like the background are coded with worse quality. In order to detect regions that contain objects, low-complexity algorithms are used that can be employed by systems with low computational power. Consequently, the images may be compressed in two ways. In the first approach, the perceptually optimized bit allocation used in Versatile Video Coding Test Model (VTM) 6.1 is adapted to machine to machine communication. In the second approach, an averaging filter that is used to filter regions in the background is added as a pre-processing step. The results of this work show that the file sizes of the images can be decreased significantly without a degradation of the performance of object detectors like Faster Regions with CNN features (R-CNN) or Mask R-CNN.

Symbols and Notations

x	Input of neuron (Artificial Neural Networks)/
	Mean position of edge group (Edge boxes)
y	Output of neuron
w	Weight
f	Activation function (Artificial Neural Networks)/Feature vector (FES)
Σ	Summation
b	Bias
σ	Sigmoid function (Artificial Neural Networks)/Scale value (Edge boxes)/
	Standard deviation(FES)
max	Maximum
r	Dilation rate (Artificial Neural Networks)/Radius (FES)
$S(y_i)$	Softmax function
Pr	Probability
$p(r)$	Precision-recall curve
$p_{\text{interp}}(\text{r})$	Interpolated precision-recall curve
$D(i,j)$	Discrete cosine transform
C	Coefficient for discrete cosine transform
N	Width of block
$p(x,y)$	Intensity of pixel
$Qstep$	Quantization step size
R	Rate
D	Distortion
α	Step size (Edge boxes)/Attenuation parameter (FES)
τ	Parameter for adjusting the aspect ratio
s	Edge group
$a(s_i,s_j)$	Affinity of edge groups
θ	Mean orientation
θ_{ij}	Angle between two edge groups
γ	Sensitivity to changes in orientation
b	Bounding box
S	Set of edge groups (Edge boxes)/
	Set of object proposals (Video Compression for Object Detection)
m	Magnitude
p	Edge

T	Ordered path of edge groups
h_b	Score of candidate bounding box
b_w	Width of candidate bounding box
b_h	Height of candidate bounding box
κ	Scale sensitivity
h_b^{in}	Score of bounding box with subtraction of center
b^{in}	Box centered in bounding box
β	NMS threshold for object proposals
i	Size of window
s_l	Score for window
$\mathbf{w}$	Linear model
$\mathbf{g}_l$	Normed gradients feature
o_l	Final objectness score
v_i	Learnt parameter for each quantized size
t_i	Learnt parameter for each quantized size
$\bar{x}$	Position in the image
H_x	Saliency of pixel
K	Center
B	Surround
$\mathcal{G}(\cdot)$	Gaussian kernel
n	Number of samples
$S_r^n(x)$	Saliency
$\mathcal{A}_c$	Circular averaging filter
$*$	Convolution
$S(x)$	Final saliency
M	Number of scales (FES)/
	Number of pixels in the local neighborhood (averaging filter)
$S_{r_i}^{n_i}(x)$	Saliency computed at different scales
$h[i,j]$	Averaging filter
N	Local neighborhood
$D_{\text{pic}}^{\text{wSSE}}$	Weighted Sum of Squared Errors
s	Original image
$\hat{s}$	Coded image
D_{pic}	Overall distortion
$\{\mathbf{p}_k\}$	Block coding parameters
λ	Lagrange multiplier

B_k	Block
M_k^t	Objectness map
t	Frame number
$\Psi_s(p)$	Pixel enclosed by bounding box

Abbreviations and Acronyms

ANN	Artificial Neural Network
AP	Average Precision
AUC	Area under Curve
BD	Bjontegaard Delta
BING	Binarized Normed Gradients
CABAC	Context-Adaptive Binary Arithmetic Coding
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
CTU	Coding Tree Unit
CU	Coding Unit
DCT	Discrete Cosine Transform
DST	Discrete Sine Transform
FCN	Fully Connected Network
FES	Fast and Efficient Saliency
FN	False negative
FP	False positive
FPS	Frames per second
GPU	Graphics Processing Unit
HEVC	High Efficiency Video Coding

IDCT Inverse Discrete Cosine Transform

IOU Intersection over Union

KDE Kernel Density Estimation

LMCS Luma Mapping with Chroma Scaling

mAP Mean Average Precision

MTSE Multi-thresholding Straddling Expansion

NG Normed gradient

NMS Non-Maximal Suppresion

QP Quantization Parameter

QPA Quantization Parameter Adaptation

R-CNN Region-based Convolutional Neural Network

ReLU Rectified Linear Unit

ResNet Residual Network

ROI Region of Interest

RPN Region Proposal Network

SAO Sample-adaptive Offset

SSE Sum of Squared Errors

SVM Support Vector Machine

TP True positive

VGG Visual Geometry Group model

VOC Visual Object Classes

VTM Versatile Video Coding Test Model

VVC Versatile Video Coding

YOLO You Only Look Once

ZF Zeiler and Fergus model

Chapter 1

Introduction

Due to the recent advances in computer-vision-based systems such as video surveillance [1], human-computer interaction [2] and autonomous driving [3], there is a growing amount of videos being observed by algorithms. Typical tasks of these systems include, for instance, identifying anomalous activities or detecting people and objects, as can be seen in Fig. 1.1. One problem that emerges in many of these applications is that the processing power of the cameras used is usually relatively low and they do not have a Graphics Processing Unit (GPU). That is why they are not capable of employing complex object detectors such as Faster Region-based Convolutional Neural Network (R-CNN) [4] or Mask R-CNN [5] in real time. A solution to this issue is transmitting the video streams to a more powerful central server which is used for further processing of the video data. As the bandwidth of the transmission channels is limited and there may also exist bandwidth bottlenecks on the server itself, an efficient compression of the video data is crucial for achieving a high performing computer-vision-based system.

This work presents two approaches that can be used to compress video streams in computer-vision-based systems efficiently. The basic idea behind both approaches is to employ a low-complexity saliency detector such as Binarized Normed Gradients (BING) [6], Edge boxes [7], Fast and Efficient Saliency (FES) [8] or You Only Look Once (YOLO) [9] that finds regions that are likely to contain an object. Consequently, these regions are coded with a good quality and the background is coded with a worse quality using the emerging Versatile Video Coding (VVC) standard [10]. This allows for an efficient compression and hence fast transmission to the central server. In the next step, the objects in the images are detected and classified using a Faster R-CNN or Mask R-CNN on the central server.

The first approach is based on the Quantization Parameter Adaptation (QPA) [11] which is used in VVC. QPA has been developed to achieve visually pleasing results for human viewers. If there is low activity in a region in the image, i.e., in smooth regions, a good quality is chosen for that region because coding artifacts are more annoying

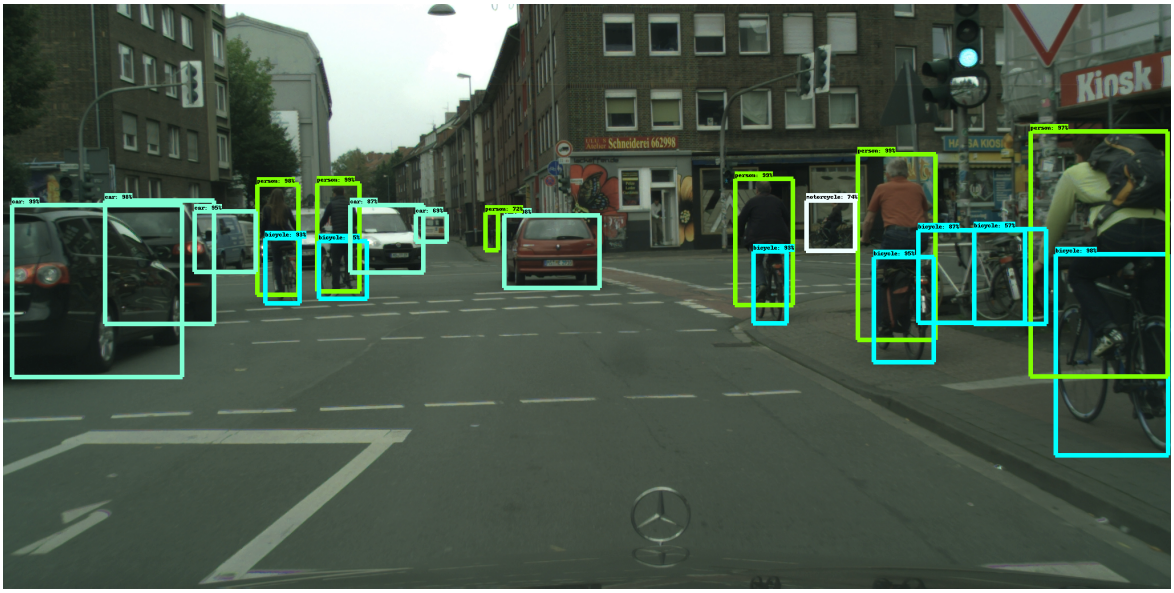


Figure 1.1: Example of a typical computer-vision-based system

in these areas. In contrast, if the current region has high activity, for example at highly textured areas, a worse quality is sufficient as artifacts are less visible then. In this work, this method is modified in such a way that the good quality is employed for regions that are likely to contain an object and the worse quality is used for the background.

The concept of the second approach is employing averaging filters [12] to the background in order to filter out high frequencies in these areas. This may also improve the compression of the video streams.

This Master's thesis is structured as follows. In Chapter 2, the theoretical background of neural networks is explained in detail with a focus on networks for object detection. Additionally, information on video coding and the different saliency detectors as well as averaging filters is given. In Chapter 3, some related work about saliency coding is presented. A description of the developed framework, the dataset used for evaluation and the experiments that were conducted is presented in Chapter 4. Chapter 5 gives an overview of the obtained results, followed by a conclusion and an outlook to possible future work in Chapter 6.

Chapter 2

Theoretical Background

In this chapter, the theoretical background about Artificial Neural Networks (ANNs) and video coding is explained in detail.

2.1 Artificial Neural Networks

An ANN is an interconnected assembly of processing elements that loosely model the behavior of neurons in an animal brain [13]. The most interesting property of ANNs is that they are capable of familiarizing with problems by means of training. After sufficient training, ANNs are able to solve unknown problems of the same class. In the following, the structure of ANNs is explained in detail.

The basic element of an ANN is an artificial neuron which has several inputs x_1, x_2, ..., x_n and an output y that can take continuous values between 0 and 1. Each input is multiplied by a weight w_1, w_2, ..., w_n that represents the importance of the specific input to the output [14]. The output is computed by

$$y = f\left(\sum_{j=0}^{m} w_j x_j + b\right), \tag{2.1}$$

where f is the activation function and b is the bias that makes it possible to move f to the left or the right which is often necessary in order to produce the required output value. An overview of the structure of an artificial neuron is depicted in Fig. 2.1.

The purpose of f is to ensure that small changes in the weights and bias produce only small changes in the outputs. This is an important property, because it will allow a neural network to learn. Another important property is that the activation function has to be nonlinear, because only in this case, neural networks are able to solve complex problems using a small number of artificial neurons. A commonly used activation function is the sigmoid function [14]. It is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.2}$$

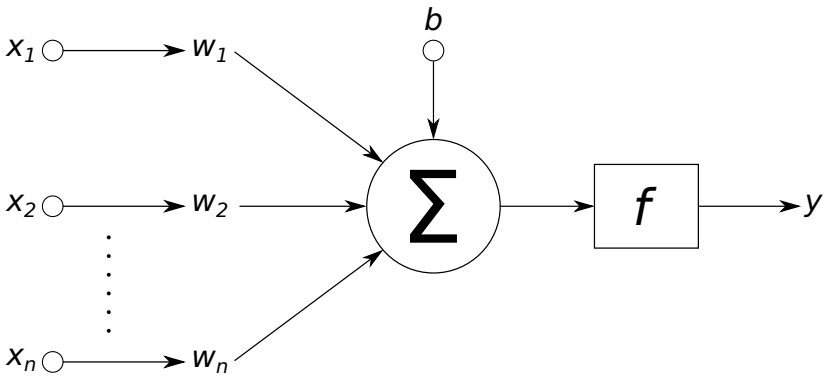
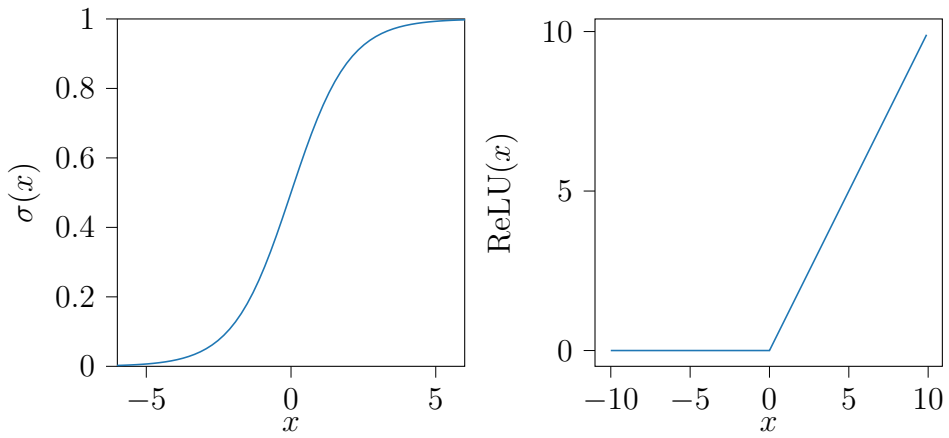


Figure 2.1: Structure of a neuron used in neural networks [15]



(a) Sigmoid function

(b) ReLU

Figure 2.2: Two activation functions commonly used in neural networks

For large positive values, $\sigma(x)$ converges towards 1 and for large negative values towards 0, as can be seen in Fig. 2.2a. The smooth behaviour of $\sigma(x)$ ensures that small changes in the weights and in the bias will generate only a small change in the output.

In the field of computer vision, the rectifier function is commonly used for f. It is defined as

$$f(x) = \max(0, x), \tag{2.3}$$

where x is the input to an artificial neuron. A plot of the rectifier function is illustrated in Fig. 2.2b. Units that employ this function are referred to as Rectified Linear Units (ReLUs) [16].

Multiple artificial neurons can be combined to form an ANN. An example is depicted

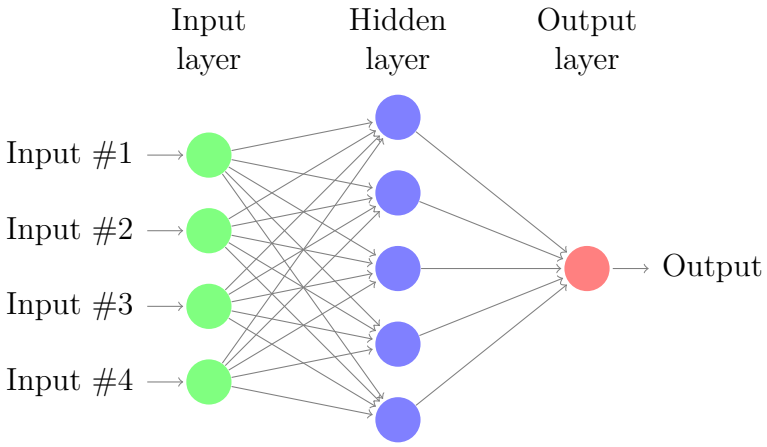


Figure 2.3: Example of an ANN [17]

in Fig. 2.3. The different columns of neurons in the network are referred to as layers. The first layer is called the input layer. The last layer is the output layer and layers that are neither inputs nor outputs are referred to as hidden layers. If there are many hidden layers present in the neural network, it is referred to as deep neural network.

A crucial part of the development of every neural network is learning from a data set. The training set is comprised of input patterns and the expected results in the form of the exact activation of each output neuron. For each training sample that is fed into the network, the output of the network can be compared with the correct solution which is referred to as ground truth. Consequently, the network weights can be adjusted according to the difference between the output of the network, i.e., the prediction, and the ground truth data. In order to determine how to adjust the weights, a particular process called backpropagation is employed. Its basic idea is to go back in the neural network and investigate each connection to check how the predicted output would behave when changing the weight of a connection. The goal of training is to adjust the weights in such a way that the network cannot only associate the input and output patterns from the training set independently, but can provide reasonable results to unseen input patterns, i.e., it generalizes.

2.2 Object Detection

2.2.1 Convolutional Neural Networks

A specialized kind of neural network is the Convolutional Neural Network (CNN). It is used for processing data that exhibits a grid-like structure, e.g. for images and

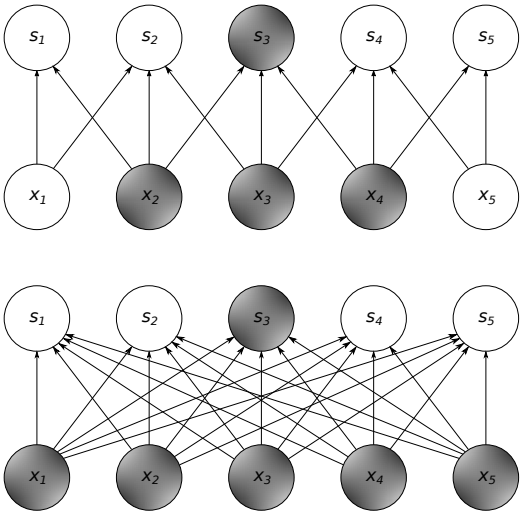


Figure 2.4: Comparison of sparse connectivity (top) and full connectivity (bottom) [19]. It can be observed that the output unit s_3 is affected by fewer input units x_i when using sparse connectivity.

videos. CNNs are defined as networks that employ convolution instead of general matrix multiplication in at least one of their layers [18]. The fundamental ideas of CNNs are sparse interactions, parameter sharing and equivariant representations. In Fully Connected Networks (FCNs), each neuron in layer i is connected to each neuron in layer $i + 1$, as can be seen in Fig. 2.3. However, this is not necessary when processing images, because small and meaningful features like edges can be detected by using a much smaller kernel that occupies only parts of the input image. Thus, fewer parameters are used in CNNs which reduces memory requirements as well as computational complexity and improves statistical efficiency. Another benefit of CNNs is that the input size can be chosen arbitrarily. The idea of sparse connectivity is visualized in Fig. 2.4.

A further advantage of CNNs is parameter sharing which refers to employing the same parameter for more than one function in a model. When calculating the output of a layer, each weight is only used once in traditional networks. In CNNs, however, each kernel weight is employed at each position of the input. This means that the same feature is detected by each neuron in the first hidden layer, just at different locations. This is useful because the ability to detect e.g. a vertical edge in a certain part of the input image might also be helpful in another region of the image [14].

Sharing the parameters in CNNs leads to equivariance to translation [18]. More

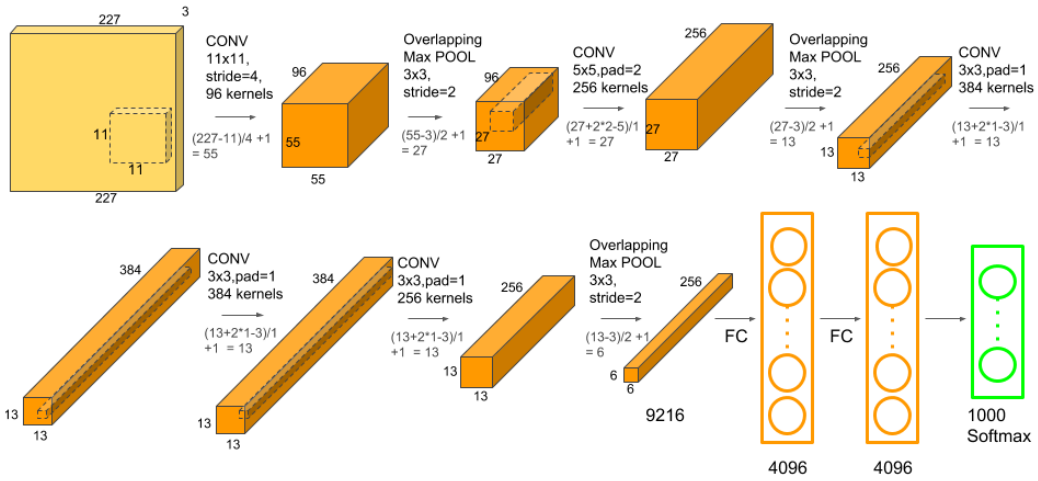


Figure 2.5: Architecture of AlexNet [20]

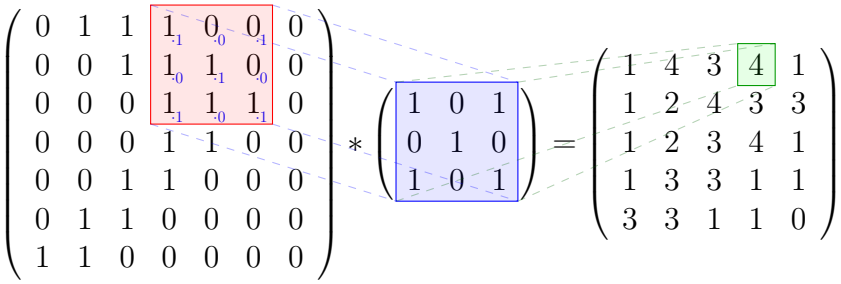


Figure 2.6: Example of a convolution employed in convolutional layers [22]

specifically, the translation of an image results in a corresponding translation of the output features. An exemplary architecture of a deep CNN, the AlexNet [21], is illustrated in Fig. 2.5. The input is an image of size 227×227 with 3 color channels. The hidden layers are a combination of convolutional and pooling layers. In convolutional layers, the activity of each neuron is computed by employing a discrete convolution. Therefore, a learnable filter kernel of relatively small size, e.g. 3×3 pixels, is shifted across the input, as can be seen in Fig. 2.6. The amount by which the filter shifts is referred to as stride. It is possible to adapt this parameter individually for each layer.

In some network architectures, atrous convolution [23] is used instead of the conventional convolution. Its concept is depicted in Fig. 2.7. The idea of atrous convolution

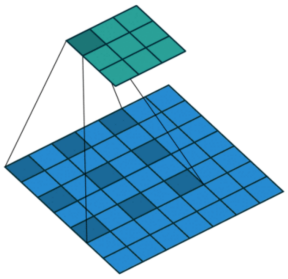


Figure 2.7: Visualization of atrous convolution with $r = 2$ [24]

is to introduce $r - 1$ zeros between consecutive filter values, where r is referred to as the dilation rate. This leads to a wider field of view without increasing the number of parameters or the computational cost. A big advantage of atrous convolution is that it allows for finding the best trade-off between precise localization (small rs) and context assimilation (big rs).

In practice, many different filters are used for each convolutional layer. In AlexNet, for instance, there are 96 filters in the first convolutional layer.

Pooling layers are responsible for reducing the spatial size of the convolved features, i.e., they simplify the information in the output from the convolutional layers. In networks used for object detection, max pooling is usually applied in pooling layers, i.e., each feature map is divided into equal-sized sections and for each section, the maximum value is copied to the output buffer. An example of how max pooling is applied is depicted in Fig. 2.8. Max pooling can be thought of as a way for the network to find out whether a given feature is present in a region of the image. The basic idea is that the exact location of a feature is not as important as its rough location relative to other features. A big advantage of this method is that a lot of computation time can be saved as the dimensionality is reduced [14].

The final layers of a CNN are fully connected layers that are used to classify the features. The number of neurons in the last fully connected layer usually corresponds to the number of classes that the network needs to distinguish. In the last layer, a softmax function

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{2.4}$$

is applied, where y_i is the output of the i^{th} neuron from the last fully connected layer and the denominator represents a sum over all outputs from this layer. The softmax function converts the outputs to a probability distribution of all class candidates [14].

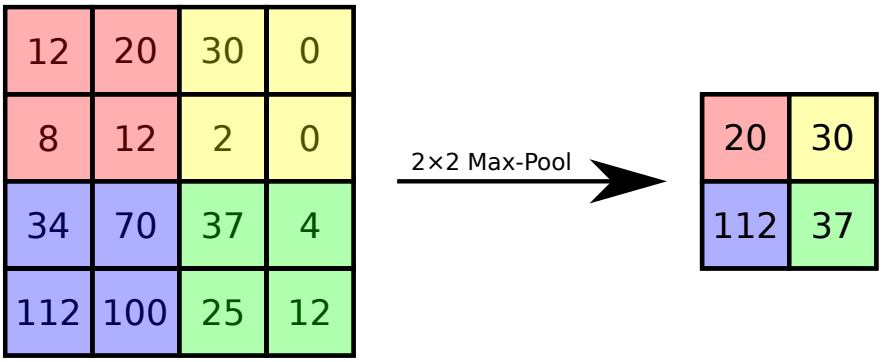


Figure 2.8: Example of max pooling [25]

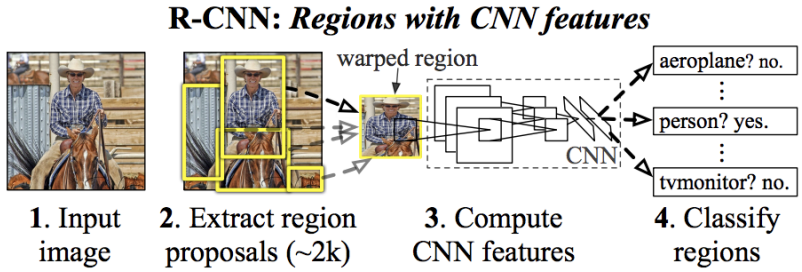


Figure 2.9: Structure of R-CNN [26]

2.2.2 Regions with CNN features

The following sections describe the history of Faster R-CNN which is mainly used in this work. Therefore, the network which Faster R-CNN is based on, namely R-CNN, is explained first. R-CNN [27] is a network used to find and classify objects in an image. The structure of R-CNN is two-staged. An overview is depicted in Fig. 2.9. The first step is to generate about 2000 category-independent region proposals using selective search [28], whose basic idea is to generate an initial sub-segmentation of the input image, followed by a recursive combination of similar regions into larger ones using a greedy algorithm. These regions are then used to generate the final region proposals. The next step of R-CNN is to warp the proposed regions to a fixed size which is required by the CNN. Consequently, each warped region is fed into the CNN that extracts a 4096-dimensional feature vector. Finally, the vector is fed into a Support Vector Machine (SVM) in order to predict the corresponding class and to increase the precision of the bounding box.

A problem with R-CNN is that it is not possible to implement it in real time, as it takes around 47 seconds to detect and classify objects in one image. The reason for

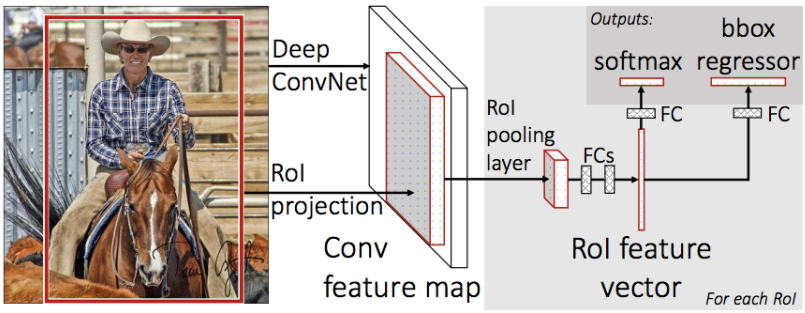


Figure 2.10: Structure of Fast R-CNN [26]

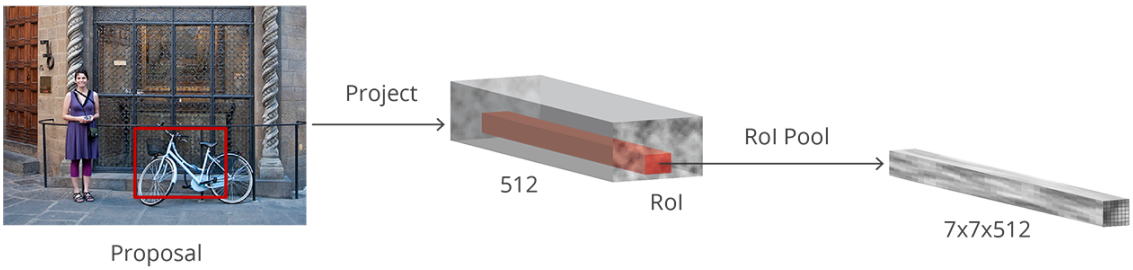


Figure 2.11: ROI pooling [30]

this is that each region proposal is fed independently into the network. Additionally, the selective search algorithm used for proposing the regions is fixed and non-learnable. This may lead to the generation of bad candidate region proposals.

2.2.3 Fast R-CNN

In order to mitigate some drawbacks of R-CNN, the same author proposed Fast R-CNN [29] which is presented in Fig. 2.10. The idea is similar to the one of R-CNN which has been explained in the previous section. The difference is that the CNN feature vectors are no longer extracted separately for each region proposal. Instead, the whole input image is fed into a CNN with several convolutional and max pooling layers in order to produce a feature map. Afterwards, a fixed-size feature vector is extracted from the feature map for each object proposal using a Region of Interest (ROI) pooling layer. This means that for each ROI the corresponding section of the feature map is selected, as can be seen in Fig. 2.11.

Afterwards, this section is scaled to a predefined size, e.g. 7×7. In the Fast R-CNN architecture, this is done by applying max pooling. Then, each feature vector is fed into

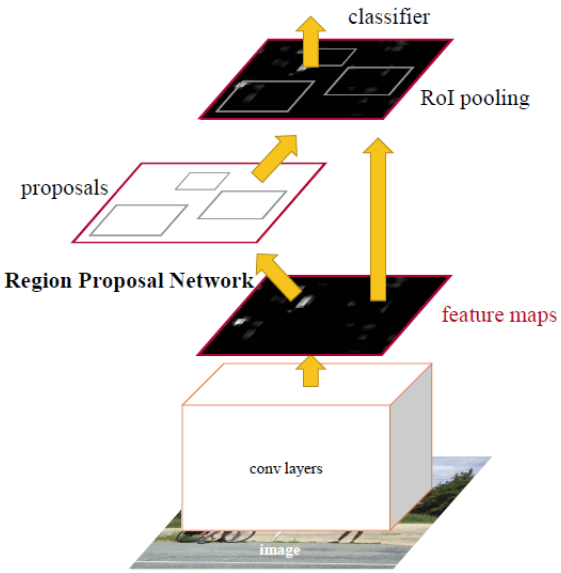


Figure 2.12: Visualization of Faster R-CNN [4]

a set of fully connected layers. Finally, the model branches into a softmax estimator that predicts the class and a bounding-box regression model which is used for the refinement of the ROI coordinates.

Compared to R-CNN, Fast R-CNN is 213 times faster in detecting and classifying objects in one image but the regions proposals are still generated by the selective search algorithm which is time-consuming, as it only runs on a Central Processing Unit (CPU) and not on a GPU, for instance.

2.2.4 Faster R-CNN

A solution to the problem of the slow region proposal algorithm used in R-CNN and Fast R-CNN was proposed in Faster R-CNN [4] which consists of two parts. The first part is a so called Region Proposal Network (RPN), which takes an image of arbitrary size as input and outputs rectangular bounding boxes that have high probability to contain an object. The second part is the Fast R-CNN detector, which is used to process the proposed regions. The structure of Faster R-CNN is visualized in Fig. 2.12. The goal of Faster R-CNN is to share computation between the two parts, which is a big advantage over R-CNN and Fast R-CNN, where the region proposal approaches and the detection networks are decoupled. During the development of Faster R-CNN, the Zeiler and Fergus model (ZF) [31] with 5 shareable convolutional layers and the Visual Geometry Group model (VGG) [32] with 13 shareable convolutional layers,

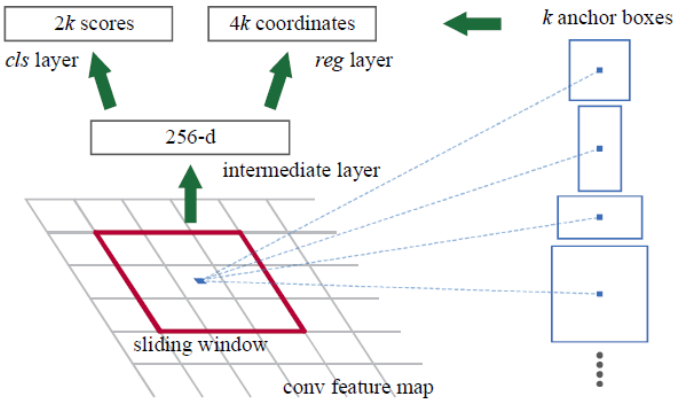


Figure 2.13: Visualization of the output of RPN [4]

followed by ReLUs [16], were investigated.

In the first step, features are extracted using convolutional layers. After that, an $n \times n$ sliding window is applied to the convolutional feature map output by the last shared convolutional layer. Each sliding window is mapped to a lower-dimensional feature, which is either 256-d for ZF or 512-d for VGG. At each location, region proposals are generated by using k so called anchor boxes. These boxes are created by combining different scales and different aspect ratios. By default, 9 anchor boxes with 3 scales (128, 256, 512) and 3 aspect ratios (1:1, 1:2, 2:1) are used. The k anchor boxes are processed by a box-regression layer (*reg*) which outputs $4k$ coordinates of the bounding boxes and a box-classification layer (*cls*) that predicts whether there is an object or not in the box. The *cls* layer is implemented as a two-class softmax layer for simplicity, which means that the number of outputs is $2k$. An overview of the output of RPN is presented in Fig. 2.13.

Subsequently, the ROIs obtained by the RPN and the feature map are fed into the detection network, as can be seen in Fig. 2.12. As this network expects fixed-sized feature maps for each proposal, ROI pooling needs to be applied. The next step is flattening the feature maps and processing them with two fully connected layers of size 4096 with ReLU activation. Finally, a fully-connected layer composed of $N + 1$ units is used, where N is the number of classes and the additional unit is used for classifying proposals as background. Another fully-connected layer with $4N$ units is used for the refinement of the ROI bounding boxes.

The use of an RPN instead of selective search leads to significant improvements in the time needed to detect and classify objects. Faster R-CNN runs at 5 frames per second and may therefore be used for real-time object detection.

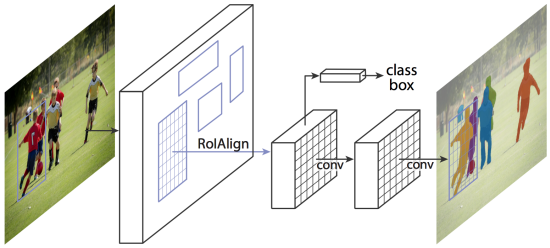


Figure 2.14: Visualization of Mask R-CNN [5]

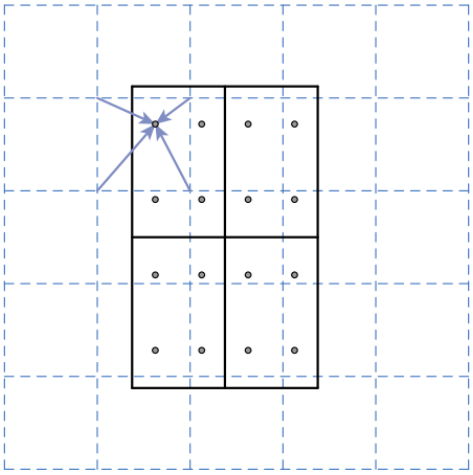


Figure 2.15: Idea of ROIAlign [34]. The ROI feature map is represented by the dashed grid, the ROI bins, in this example of size 2×2, are illustrated by the solid lines and the sampling points are represented by the dots.

2.2.5 Mask R-CNN

Mask R-CNN [33] is an extension to Faster R-CNN and is used for pixel-wise instance segmentation. Mask R-CNN adopts the two-stage procedure of Faster R-CNN, where the first stage, the RPN, is identical. The difference lies in the second stage. Additionally to the prediction of the class and the refinement of the bounding boxes, an input object's spatial layout encoded as a binary mask is predicted for each ROI, as can be seen in Fig. 2.14.

More specifically, K binary masks of size $m \times m$ are predicted for each ROI using an FCN, where K is the number of classes. Due to the pixel-to-pixel behavior of the masks, an appropriate alignment of the ROI features needs to be conducted in order to preserve the per-pixel spatial correspondence. For this purpose, ROIAlign is employed which avoids the rounding errors of ROI pooling. This is done by allowing floating point values for all coordinates used in the ROI, its bins or the sampling points, as

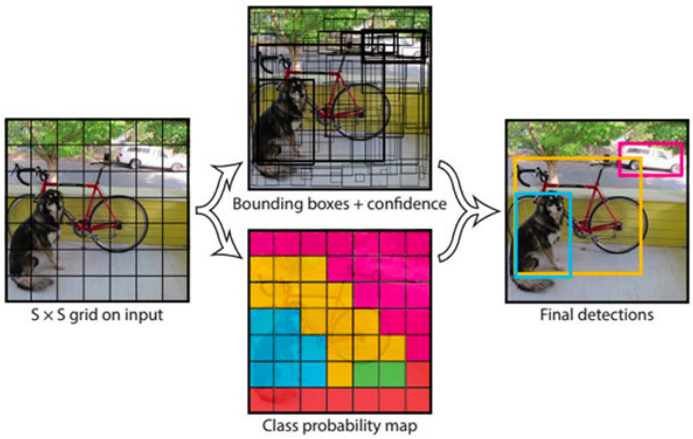


Figure 2.16: YOLO architecture [36]

can be seen in Fig. 2.15.

In order to calculate the exact values of the ROI features at the sampled locations, bilinear interpolation [35] is employed. Then, the result for each ROI bin is obtained by using max or average pooling.

2.2.6 YOLO

Another popular network used for object detection is YOLO [9]. An overview of YOLO is presented in Fig. 2.16. In contrast to the R-CNN based approaches which are two-staged, YOLO runs the image on a CNN model and gets the detections on a single pass. This is where the name You Only Look Once comes from. The input image is divided into an $S \times S$ grid, where each grid cell is responsible for detecting exactly one object. For each grid cell, B bounding boxes and the respective confidence scores are predicted. The confidence score represents the probability of the corresponding bounding box containing an object Pr(Object) and the accuracy of the bounding box measured by the Intersection over Union (IOU) between the predicted box and a ground truth box of any class. The idea of IOU is visualized in Fig. 2.17.

Additionally, for each grid cell, one set of C conditional class probabilities is predicted, denoted as $\Pr(\text{Class}_i|\text{Object})$, where C is the number of classes. Finally, class-specific confidence scores for each box are calculated by

$$\Pr(\text{Class}_i|\text{Object}) \cdot \Pr(\text{Object}) \cdot \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) \cdot \text{IOU}_{\text{pred}}^{\text{truth}}. \tag{2.5}$$

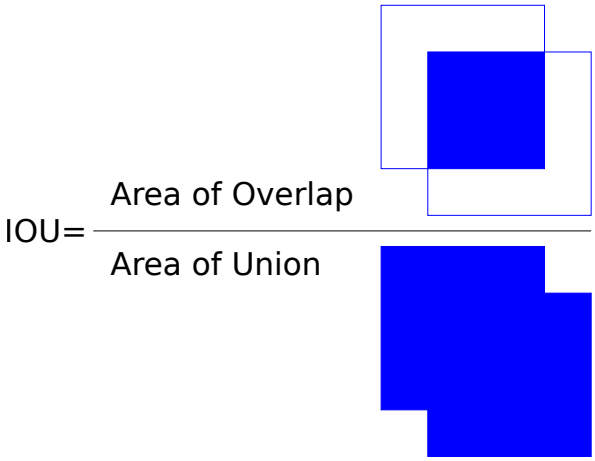


Figure 2.17: Calculation of IOU [37]

2.2.7 Performance Evaluation using Mean Average Precision

A common metric used to measure the performance of object detectors is Average Precision (AP). When computing AP, the first step is to determine the number of true positives (TPs), false positives (FPs) and false negatives (FNs). TPs are defined as predictions that have bounding boxes with an IOU greater than a predefined threshold with the corresponding ground truth box. FPs are bounding boxes whose IOU with the ground truth is below the threshold. A commonly used threshold for the IOU is 0.5. If multiple bounding boxes have an IOU greater than the threshold with one ground truth box, all bounding boxes except the one with the highest IOU are marked as FP [38]. Those objects where the object detector failed to produce a bounding box with an IOU higher than the threshold are referred to as FNs [39].

The next step is to calculate precision and recall. Precision is defined as the ratio of TPs and the total number of predicted positives [40], i.e.

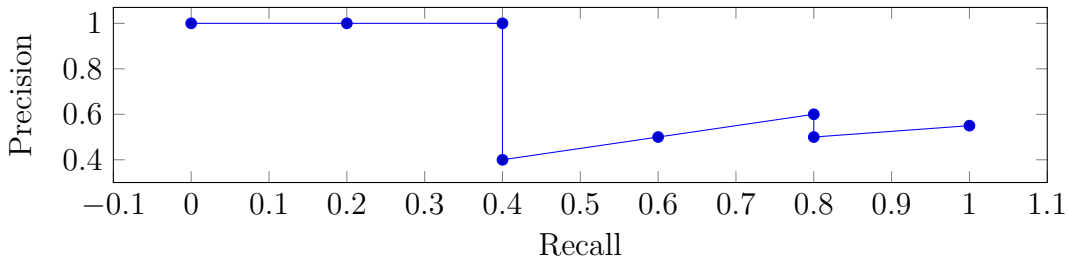
$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{2.6}$$

Recall is the fraction of relevant detections that are retrieved, which can be formulated as

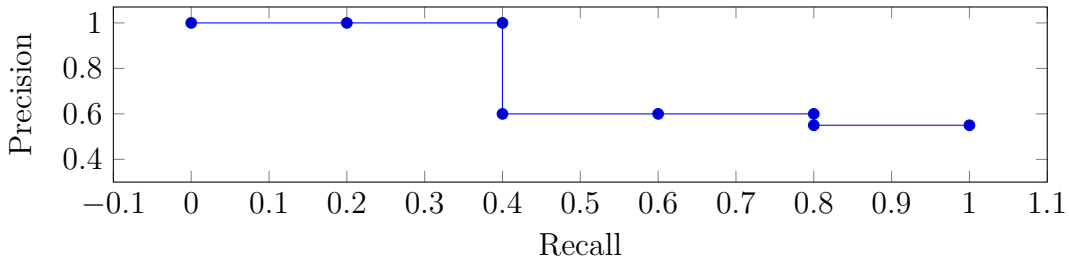
$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{2.7}$$

In general, the AP is defined as the area under the precision-recall curve, i.e.

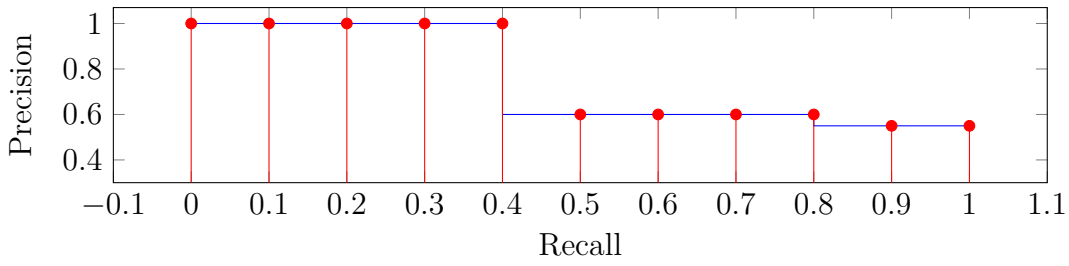
$$\text{AP} = \int_0^1 p(r)dr. \tag{2.8}$$



(a) Initial precision-recall curve



(b) Smoothed out curve



(c) Interpolated curve

Figure 2.18: Example of a precision-recall curve

An example of a plot of recall against precision is depicted in Fig. 2.18a. In practice, the curve is often smoothed out, as can be seen in Fig. 2.18b. This can be formulated as

$$p_{\text{interp}}(r) = \max_{\tilde{r} \geq r} p(\tilde{r}). \tag{2.9}$$

The last step is interpolating the smoothed out curve which is illustrated in Fig. 2.18c. The AP for each class is then calculated as

$$\text{AP} = \frac{1}{11} \sum_{r \in \{0.0,\ldots,1.0\}} p_{\text{interp}}(r). \tag{2.10}$$

The number of interpolating points depends on the data set. In the Pascal Visual Object Classes (VOC) challenge [39], equation (2.10) with 11 points is used, wheras

for the Common Objects in Context (COCO) dataset [41], 101 points are used for the interpolation. Another innovation in the COCO dataset is that the AP is calculated for ten different IOU thresholds independently and averaged. The IOUs range from 0.5 to 0.95 in steps of 0.05.

When averaging the APs for all classes, the result is referred to as Mean Average Precision (mAP).

2.3 Versatile Video Coding

2.3.1 Overview

A video codec is a software used to compress or decompress digital video. The general block diagram of a video coding system is depicted in Fig. 2.19. At the start of the signal processing chain, a digital video signal is captured, for instance, by a camera. During this step, an image of a 3D scene is projected onto the surface of an image sensor that samples the optical signal. The output of the sensor consists of arrays of discrete-amplitude samples. Subsequently, it is possible to apply an optional pre-processing step which includes, for instance, improving the contrast or reducing noise in the image. In the encoding step, the arrays of samples are mapped into a sequence of bits called bitstream which usually has a significantly lower bit rate than the raw data captured by the camera. After encoding, the bitstream is transmitted over a channel like the internet to the decoder. Another option besides transmission would be storing the bitstream on optical discs like DVD or Blu-ray or other storage devices. In the decoding step, the sample arrays are reconstructed from the received bitstream. In order to reduce transmission errors and coding artifacts, an optional post-processing step may be conducted. Finally, the obtained video signal is usually displayed and perceived by human beeings. However, as a result of the recent advances in computer vision systems, there is a growing amount of videos being watched by algorithms [42].

This work focuses on the encoding and decoding steps marked by the red box in Fig. 2.19. Thus, the latest video coding standard VVC [10] is described in detail in the following. VVC is a standard being developed by the Joint Video Experts Team. Compared to its predecessor High Efficiency Video Coding (HEVC) [43], VVC will achieve a reduction of about 50% of the bitrate at the same subjective quality. VVC has a block-based hybrid coding architecture that combines intra-picture and inter-picture prediction, transform coding and entropy coding. An overview of VVC is depicted in Fig. 2.20. First, the uncompressed input video is divided into blocks called

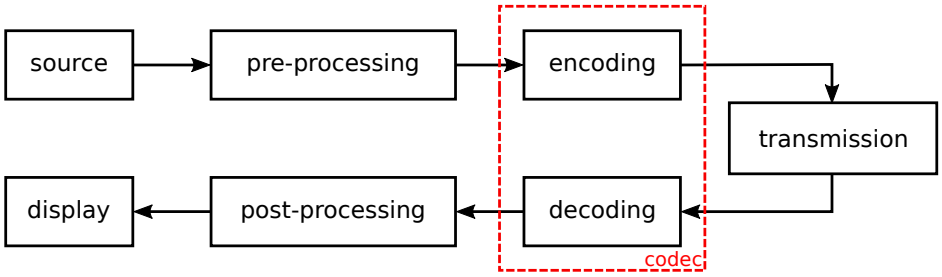


Figure 2.19: Overview of a typical video coding system [42]

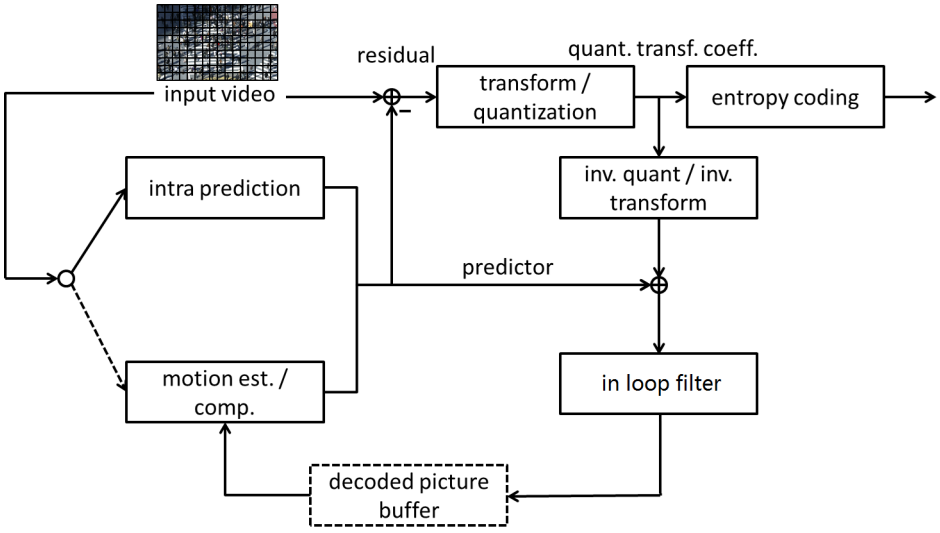


Figure 2.20: Overview of VVC [10]

Coding Tree Units (CTUs). An exemplary CTU is depicted in Fig. 2.21. CTUs can be subdivided into smaller blocks called Coding Units (CUs). For each CU, intra- or inter-picture prediction is applied. The term intra-picture prediction refers to exploiting the spatial redundancy within one frame. More precisely, pixel values are predicted by extrapolation from already coded pixels. Intra-picture prediction is based on the assumption that texture of a picture region has a high correlation with the texture in the local neighborhood and can hence be predicted from neighboring regions. Using this prediction, the data rate can be decreased significantly. Inter-picture prediction, however, tries to take advantage of temporal redundancy between consecutive frames within a video. The idea behind inter-picture prediction is that frames within a video consist of objects that move in the scene. For consecutive frames, the scene has only small differences which mainly occur due to motion. Instead of encoding each frame independently, the encoder will thus search for a block resembling the one it is encoding on a previously encoded reference frame. If a similar block is found, motion estimation

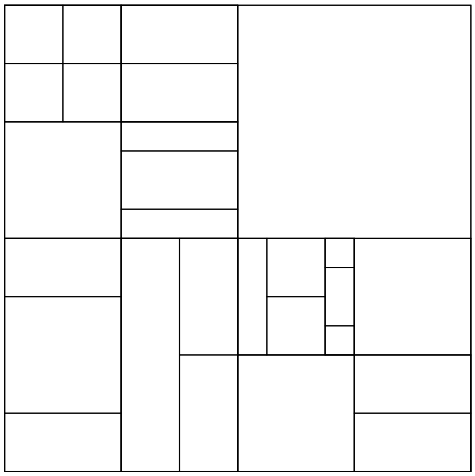


Figure 2.21: Exemplary CTU used in VVC

is employed which describes the determination of the displacement of these two blocks by a motion vector. It is highly unlikely, that the two blocks are identical. Thus, the difference between both blocks is calculated which is referred to as prediction error. For inter-coded blocks, the motion vector and the corresponding prediction error is transmitted to the decoder. The advantage of this method is that in many cases the size of the encoded motion vector and the prediction error is smaller than the size of the encoded original pixel values. Subsequently, the residual is transformed and quantized. In VVC, the techniques used for transformation are Discrete Cosine Transform (DCT) [44] and Discrete Sine Transform (DST), respectively. DCT is the most widely used linear transformation in data compression [45] and is especially used for lossy compression because of its energy compaction property [46]. The DCT of a block of size $N \times N$ is defined as

$$D(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p(x,y) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right], \quad (2.11)$$

where

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0, \\ 1 & \text{if } u > 0. \end{cases} \quad (2.12)$$

For most images, the majority of information lies at low frequencies that appear in the upper left corner of the DCT, as can be seen in Fig. 2.22. Higher frequencies

$$\begin{pmatrix} 26 & -5 & -5 & -5 \\ 64 & 52 & 8 & 26 \\ 126 & 70 & 26 & 26 \\ 111 & 52 & 8 & 52 \end{pmatrix} \rightarrow \text{DCT} \rightarrow \begin{pmatrix} 158.0 & 92.3 & 55.0 & -12.3 \\ -82.5 & -30.7 & -27.0 & 9.3 \\ -41.0 & -21.6 & 12.0 & 7.9 \\ 3.3 & 18.3 & -1.2 & 16.7 \end{pmatrix}$$

Figure 2.22: Example of the calculation of the DCT

with small DCT coefficients can be neglected without distorting the image too much. In VVC, the DCT coefficients are divided by a quantization step size, e.g. 32, and rounded to integers, i.e.,

$$\begin{pmatrix} 158.0 & 92.3 & 55.0 & -12.3 \\ -82.5 & -30.7 & -27.0 & 9.3 \\ -41.0 & -21.6 & 12.0 & 7.9 \\ 3.3 & 18.3 & -1.2 & 16.7 \end{pmatrix} \cdot \frac{1}{32} \approx \begin{pmatrix} 5 & 3 & 2 & 0 \\ -3 & -1 & -1 & 0 \\ -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}. \tag{2.13}$$

The rounding and quantization represents the lossy part of compression. In VVC, the quantization step size may be adapted using the so called Quantization Parameter (QP). As in the previous standards, the relationship between QP and the quantization step size $Qstep$ is [47]

$$Qstep(QP) = (2^{\frac{1}{6}})^{QP-4}. \tag{2.14}$$

As can be seen in (2.13), a lot of DCT coefficients are zero after quantization which simplifies the next step in VVC which is entropy coding of the quantized matrix using Context-Adaptive Binary Arithmetic Coding (CABAC) [48]. CABAC is an efficient method for lossless compression of binary data. Its basic concept is that the encoder may adapt probability models for each symbol beeing 0 or 1 based on the statistics of recently coded data symbols. In order to improve the quality of the reconstructed images, in-loop filters are used which have a significant impact on the overall performance of the video codec. There are three kinds of in-loop filters in VVC. The first one is the deblocking filter. It is used to mitigate the effect of blocking artifacts that occur as a result of block-wise motion compensated prediction as well as block-wise processing of the residual signal. Secondly, there are Sample-adaptive Offset (SAO) filters [49] which can be used, for instance, to reduce ringing artifacts or to correct level values that have been shifted as a result of quantization. The third filter is Luma Mapping with Chroma Scaling (LMCS) which is used to reshape luma components and improve rate distortion.

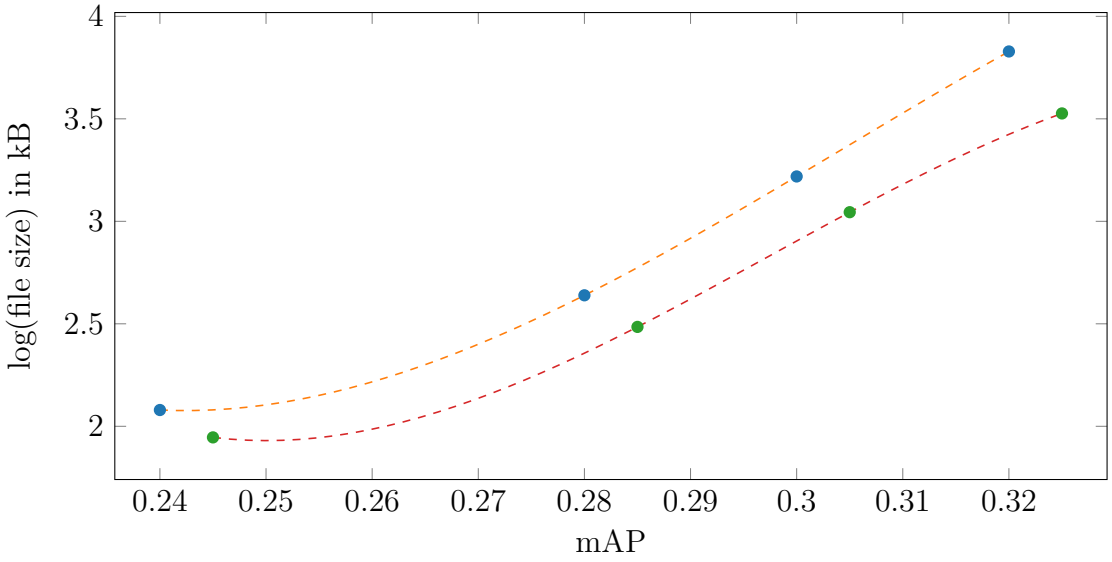


Figure 2.23: Example of two mAP-rate curves

The decoded picture buffer is used to store the reconstructed frames until they are scheduled for display. Additionally, the buffered images may be used as reference pictures for inter-frame prediction. Before transmission, the prediction error is transformed, quantized and entropy coded.

On the decoder side, the matrix is decoded, dequantized and the Inverse Discrete Cosine Transform (IDCT) is employed, i.e.

$$\begin{pmatrix} 160 & 96 & 64 & 0 \\ -96 & -32 & -32 & 0 \\ -32 & -32 & 0 & 0 \\ 0 & 32 & 0 & 32 \end{pmatrix} \rightarrow \text{IDCT} \rightarrow \begin{pmatrix} 21 & -5 & -5 & -9 \\ 64 & 46 & 0 & 30 \\ 148 & 52 & 29 & 15 \\ 116 & 55 & 19 & 63 \end{pmatrix}. \tag{2.15}$$

2.3.2 Performance Evaluation with Bjontegaard Delta

Bjontegaard Delta (BD) [50] is a frequently used metric to compare two rate-distortion curves. Traditionally, the PSNR is used to describe the distortion. In this work, however, the PSNR is replaced by the mAP which is more meaningful in the case of machine to machine communication. An exemplary constellation of two mAP-rate curves is depicted in Fig. 2.23. When calculating Bjontegaard delta, the first step is to fit two third-order polynomials of the form

$$R = a + b \cdot D + c \cdot D^2 + d \cdot D^3 \tag{2.16}$$

through the two curves, where R and D represent rate and distortion, respectively. After that, the integrals of both functions are evaluated using the integration interval

$$[\max(\min(D_1, D_2)), \min(\max(D_1, D_2))]. \tag{2.17}$$

Finally, the average difference between the two functions is calculated by

$$\overline{\Delta R} = \frac{\int_{D_{\min}}^{D_{\max}} R_1(D) - R_2(D) dD}{D_{\max} - D_{\min}} \tag{2.18}$$

and converted to a percentage

$$\overline{\Delta R}_{\%} = (e^{\overline{\Delta R}} - 1) \cdot 100. \tag{2.19}$$

Equation (2.19) can be thought of as the average bandwidth savings for equivalent quality levels between two rate-distortion curves.

The average difference of the distortion may be calculated in a similar manner by changing the axes, i.e.,

$$\overline{\Delta D} = \frac{\int_{R_{\min}}^{R_{\max}} D_1(R) - D_2(R) dR}{R_{\max} - R_{\min}}. \tag{2.20}$$

2.4 Saliency Detection

In this section, the different saliency detectors used in this work are described in detail. They are used for detecting regions in images that are likely to contain an object.

2.4.1 Edge Boxes

The idea of Edge boxes [7] is that edges provide a sparse yet informative representation of an image and the number of contours fully enclosed by a bounding box indicates the likelihood of the box containing an object. In the following, the algorithm is explained in detail.

The first step is to compute an edge map using the Structured Edge detector [51] and generate candidate bounding boxes by employing a sliding window search over position, scale and aspect ratio. The corresponding step sizes are chosen in such a way that one step results in an IOU equal to α for two neighboring boxes, where $\alpha = 0.65$ by default. The scale values are in the range from $\sigma = 1000$ pixels to all pixels in the image and the aspect ratio ranges from $1/\tau$ to τ, where $\tau = 3$ as standard. For each candidate bounding box, the next step is to identify edges that overlap the bounding

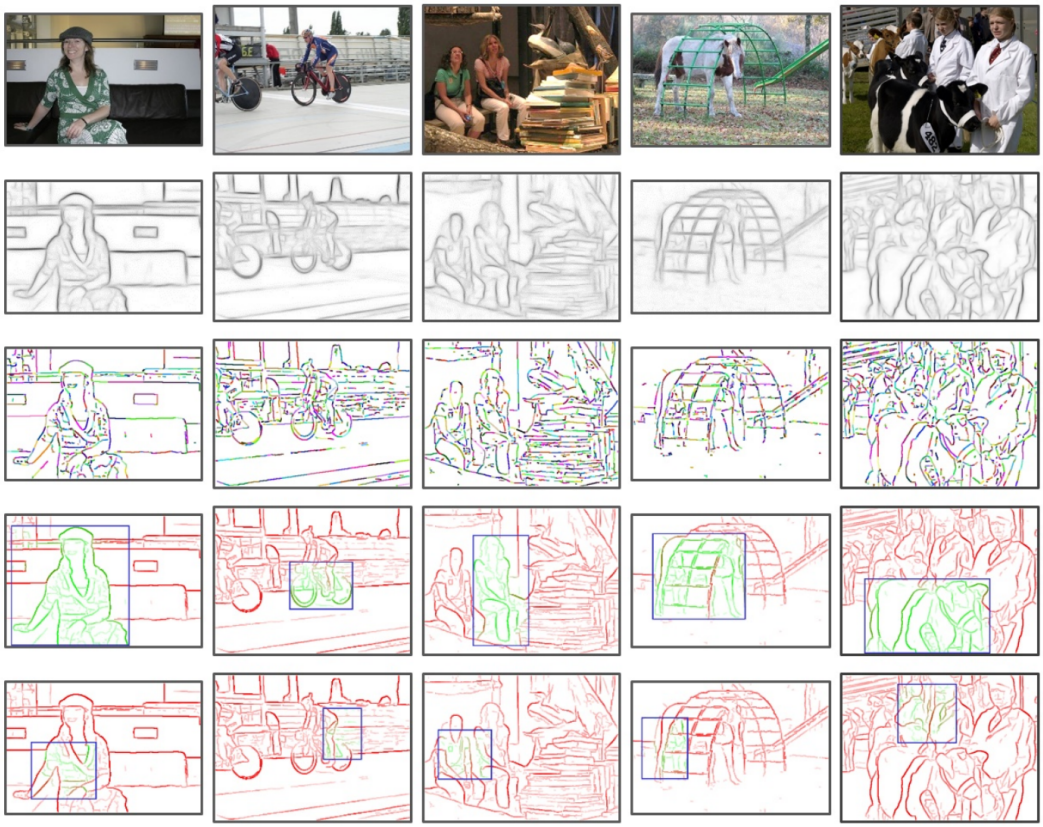


Figure 2.24: Idea of Edge boxes [7]. The original images are depicted in the first row, followed by the Structured Edges [51] in the second row. Edge groups are visualized in the third row. Examples for correctly predicted edge labels and bounding boxes can be seen in the fourth row and examples for incorrect predictions are depicted in the last row.

box boundary, as the likelihood that these edges belong to an object contained by the bounding box is low. For each pixel, its maximum affinity with an edge on the bounding box is calculated. In order to reduce computational complexity, edges with high affinity are grouped together, as can be seen in the third row of Fig. 2.24. Therefore, a simple greedy algorithm is used that combines 8-connected edges until the sum of their orientation differences exceeds $\frac{\pi}{2}$. Additionally, an edge group s_i can be merged with a neighboring group s_j. Therefore, the affinity is calculated depending on their mean positions x_i and x_j and mean orientations θ_i and θ_j by

$$a(s_i, s_j) = |\cos(\theta_i - \theta_{ij})\cos(\theta_j - \theta_{ij})|^{\gamma}, \tag{2.21}$$

where θ_{ij} is the angle between x_i and x_j and γ is used to control the sensitivity to changes in orientation. If there is a gap of more than 2 pixels between two edge groups

Figure 2.25: Exemplary image with Edge boxes applied to it. In this example, β is set to 0.9 and κ is set to 1.9. The remaining parameters are set to the default values.

or $a(s_i, s_j) < 0.05$, the affinity is set to zero.

An objectness score for any bounding box b can be computed from the set of edge groups S and the corresponding affinities. First, the sum m_i of the magnitudes m_p for all edges p in the group s_i are calculated. Then, a continuous weight $w_b(s_i) \in [0, 1]$ is computed for each group s_i. If s_i crosses the boundary of b or $\bar{x}_i \notin b$, then $w_b(s_i)$ is set to 0, else 1. The weights for the remaining edge groups are calculated by

$$w_b(s_i) = 1 - \max_T \prod_j^{|T|-1} a(t_j, t_{j+1}), \tag{2.22}$$

where T is an ordered path of edge groups with a length of $|T|$ that starts at an edge group crossing the bounding box boundary and ends at $t_{|T|} = s_i$. Thus, if s_i has no affinity to an edge group crossing the bounding box boundary, $w_b(s_i)$ is set to 1. After finding the weights for each edge group, the score h_b of the candidate bounding box may be calculated by

$$h_b = \frac{\sum_i w_b(s_i) m_i}{2(b_w + b_h)^\kappa}, \tag{2.23}$$

where b_w and b_h represent the width and height of the candidate bounding box and $\kappa = 1.5$ is used to take the bias of larger windows into account that usually have more edges on average. As edges near the bounding box boundaries are more important

than edges in the center [52], the edge magnitudes in the center may be subtracted by

$$h_b^{in} = h_b - \frac{\sum_{p \in b^{in}} m_p}{2(b_w + b_h)^\kappa}, \tag{2.24}$$

where b^{in} is a box of size $\frac{b_w}{2} \times \frac{b_h}{2}$ centered in b. Each bounding box with a score h_b^{in} above a small threshold is refined by employing a greedy iterative search in order to maximize h_b^{in} over position, scale and aspect ratio. Finally, a Non-Maximal Suppresion (NMS) is employed that discards a box if its IOU is greater than β with a box with higher score.

Edge boxes achieve over 96% object recall at an overlap threshold of 0.5 and 76% for a threshold of 0.7, given only 1000 object proposals. With a run time of only 0.25 seconds per image, Edge boxes are faster than most of the other state-of-the-art methods. An example of how the Edge boxes look like when applied to an image is depicted in Fig. 2.25.

2.4.2 Binarized Normed Gradients for Objectness Estimation

BING [6] is an objectness measurement based on a two stage cascaded SVM approach [53]. The idea of BING is that objects can be detected by investigating the euclidean norm of gradients of the corresponding image windows. In order to find objects, BING scans over a predefined set of quantized window sizes. Therefore, the whole input image is resized to 36 different sizes i, as can be seen in Fig. 2.26b. The widths and heights are a combination of {16, 32, 64, 128, 256, 512}. Then, 8×8 windows are used in the downsized images to extract the normed gradients (NGs). Each window is scored with

$$s_l = \langle \mathbf{w}, \mathbf{g}_l \rangle, \tag{2.25}$$

where $\mathbf{w} \in \mathbb{R}^{64}$ is a linear model learned using a linear SVM and $\mathbf{g}_l$ is the NG feature. Exemplary objects and the corresponding NG features are depicted in Fig. 2.26a and Fig. 2.26c, respectively, and a single learnt model $\mathbf{w}$ is illustrated in Fig. 2.26d. As can be observed from the Figures a and c, the NGs of the objects share strong correlation and can be well distinguished from the background.

From each size i, a small set of candidate bounding boxes is selected using NMS. As the likelihood of a box containing an object also depends on the size of the box, the final objectness score is defined as

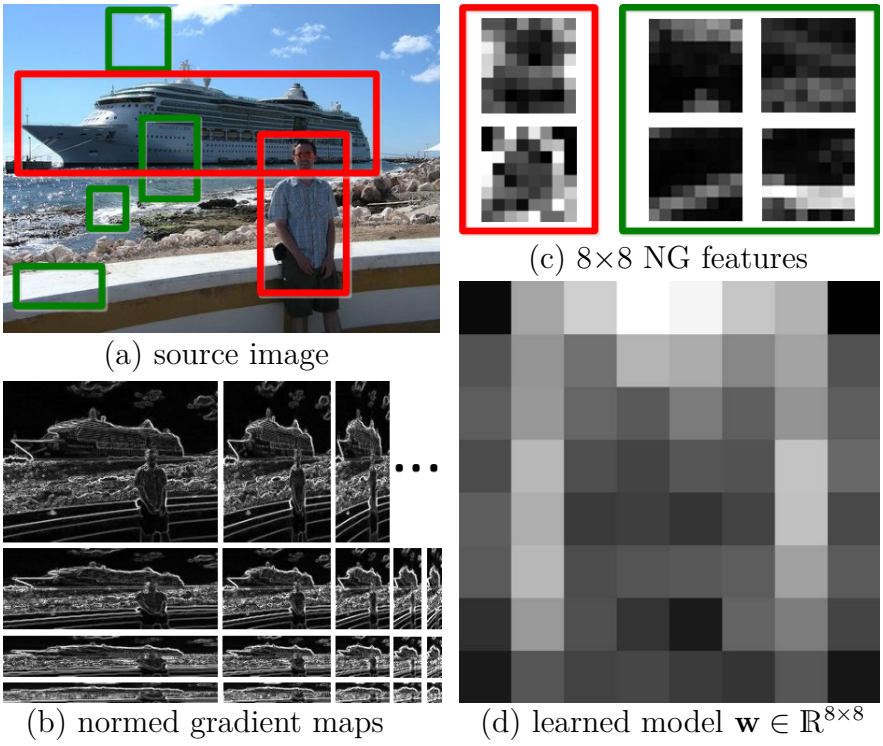


Figure 2.26: Overview of BING [54]

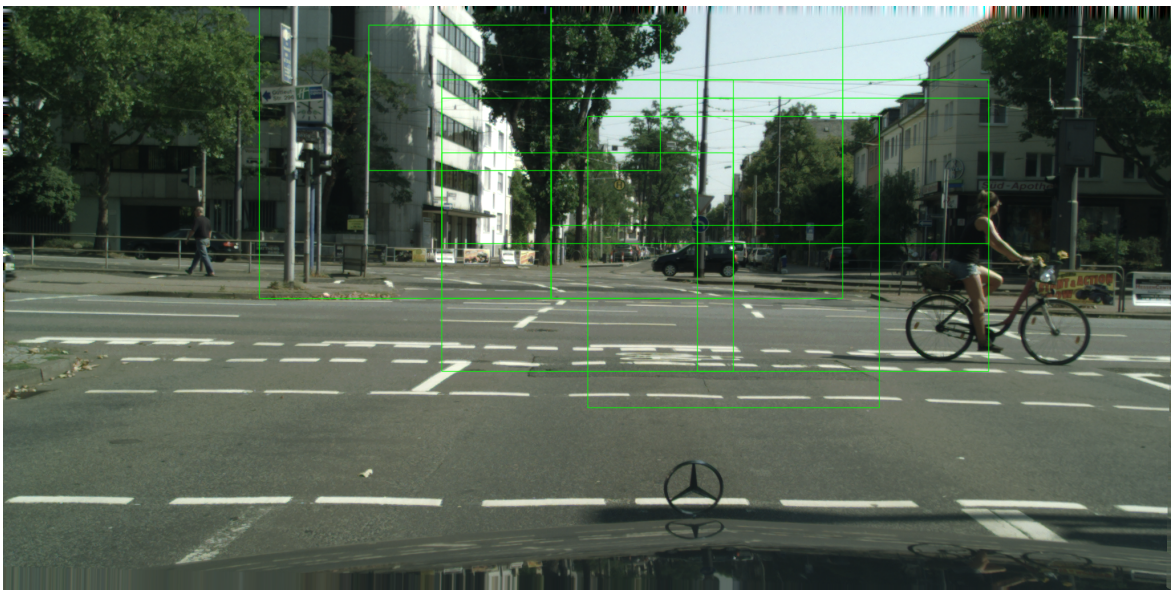


Figure 2.27: Example of an image with BING applied to it. The maximum number of bounding boxes is set to 20 in this example.

$$o_l = v_i \cdot s_l + t_i, \tag{2.26}$$

where v_i and t_i are learnt for each quantized size using a linear SVM. In order to accelerate the feature extraction and testing process, the linear model $\mathbf{w}$ is approximated by a binary model [55, 56]. Its basic concept is to avoid loops computing access to 64 positions required by the 8×8 BING feature. Instead, only a few atomic operations like add, bitwise shift, etc., are used in order to calculate the BING features.

Using an IOU threshold of 0.5 and 1000 object proposals, BING achieves an object recall of 96.2%. With a speed of 300 frames per second (FPS), BING runs faster than most of the other state-of-the-art region proposal algorithms. An example of BING applied to an image is illustrated in Fig. 2.27.

2.4.3 Fast and Efficient Saliency Detection Using Sparse Sampling and Kernel Density Estimation

Another saliency measurement used in this work is FES detection [8] which is based on estimating saliency of local feature contrast in a Bayesian framework. A center-surround technique is used which means that each window is divided into a center containing an object and a surround, as can be seen in Fig. 2.28. Each pixel is defined as $x = (\bar{x}, f)$, where $\bar{x}$ is the position in the image and f is a feature vector. The decision whether a pixel is salient is represented by a binary variable

$$H_x = \begin{cases} 1 & \text{if } x \text{ is salient,} \\ 0 & \textit{otherwise.} \end{cases} \tag{2.27}$$

Consequently, the probability if a pixel is salient is computed using the Bayes rule by

$$P(H_x = 1|f, \bar{x}) = P(1|f, \bar{x}) = \frac{P(f|\bar{x}, 1)P(1|\bar{x})}{P(f|\bar{x})} = \frac{P(f|\bar{x}, 1)P(1|\bar{x})}{P(f|\bar{x}, 1)P(1|\bar{x}) + P(f|\bar{x}, 0)P(0|\bar{x})}. \tag{2.28}$$

In order to estimate $P(f|\bar{x}, 1)$ and $P(f|\bar{x}, 0)$, Kernel Density Estimation (KDE) is used which is one of the most widely-used methods to estimate the underlying probability density function of a dataset [57]. Using KDE, the probabilty that a pixel is salient given a specific feature vector is estimated by

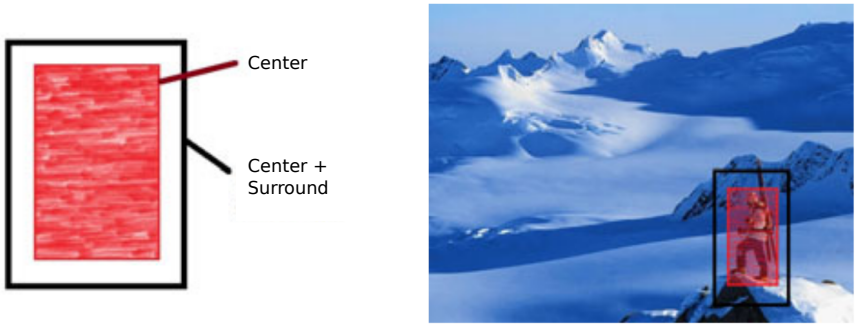


Figure 2.28: Visualization of the center surround concept [8]

$$P(1|f,\bar{x}) = \frac{\frac{1}{m}\sum_{i=1}^{m}\mathcal{G}(f-f_{\bar{x}_{Ki}})P(1|\bar{x})}{\frac{1}{m}\sum_{i=1}^{m}\mathcal{G}(f-f_{\bar{x}_{Ki}})P(1|\bar{x}) + \frac{1}{n}\sum_{i=1}^{n}\mathcal{G}(f-f_{\bar{x}_{Bi}})P(0|\bar{x})}, \tag{2.29}$$

where m and n are the number of samples, $x_{Ki} = (\bar{x}_{Ki}, f_{\bar{x}_{Ki}})$ is the i^{th} sample that belongs to the center K, $x_{Bi} = (\bar{x}_{Bi}, f_{\bar{x}_{Bi}})$ is the i^{th} sample belonging to the surround B and $\mathcal{G}(\cdot)$ is a Gaussian kernel. As the goal is to compute the saliency of only one pixel in the center, the Gaussian kernel used for K can be simplified to

$$\mathcal{G}(f-f_{\bar{x}_{Ki}}) = \frac{1}{\sqrt{2\pi}\sigma_1}\exp(-\frac{\|f-f\|^2}{2\sigma_1^2}) = \frac{1}{\sqrt{2\pi}\sigma_1}, \tag{2.30}$$

where σ_1 is the standard deviation. When assuming that only a few samples scattered uniformly on a circle with radius r around a pixel contribute to $P(f|\bar{x},0)$, (2.29) can be written as

$$P_r^n(1|f,\bar{x}) = \left(1 + \frac{\sigma_1(1-P(1|\bar{x}))}{n\sigma_0 P(1|\bar{x})}\sum_{i=1}^{n}\exp\left(\frac{\|f-f_{\bar{x}_{Bi,r}}\|^2}{2\sigma_0^2}\right)\right)^{-1}, \tag{2.31}$$

where σ_0 and σ_1 are standard deviations, n is the number of samples taken from B and r represents the radius at which samples will be taken. The concept of (2.31) is visualized in Fig. 2.29.

The distribution $P(1|\bar{x})$ is estimated by computing an average fixation map over a training set. Consequently, the saliency is computed by

$$S_r^n(x) = \mathcal{A}_c * [P_r^n(1|f,\bar{x})]^\alpha, \tag{2.32}$$

where $\mathcal{A}_c$ is a circular averaging filter, $*$ represents convolution and $\alpha \geq 1$ attenuates high probability areas.

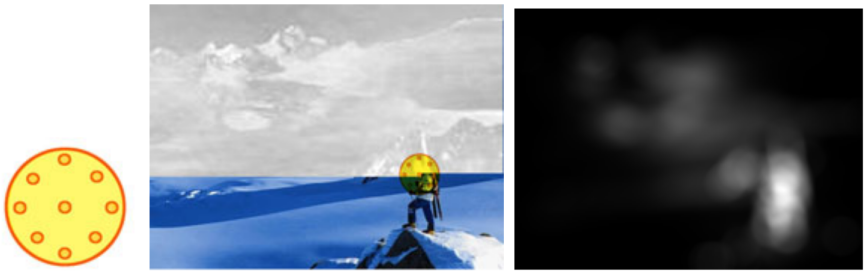


Figure 2.29: Visualization of FES [8]. The left image represents the window used for the selection of a center pixel and its surrounding samples. An example of how it is applied in an image is depicted in the middle. The right image is the corresponding saliency map obtained using FES.

(a) Original image (b) Filtered image

Figure 2.30: Example of the application of an averaging filter using a 15×15 local neighborhood

In order to take account for objects of different sizes, a multi-scale measure is employed in FES which is based on changing the size of the window. It is only necessary to adapt the radius r and the number of samples n. The final saliency is calculated by averaging (2.32) over the different scales, i.e.,

$$S(x) = \frac{1}{M} \sum_{i=1}^{M} S_{r_i}^{n_i}(x), \tag{2.33}$$

where M is the number of scales and $S_{r_i}^{n_i}(x)$ is the i^{th} saliency computed at a different scale using (2.32). FES achieves the highest Area under Curve (AUC) [58] among similar high-performance methods. Furthermore, with a run time of 7.6 μs per pixel, it is the fastest among the high-performance methods. The output of FES is depicted on the right side of Fig. 2.29.

2.5 Averaging filter

An averaging filter is one of the simplest linear filters used in image processing [12]. Each pixel value is replaced by the mean of all values in the local neighborhood. Formally, the averaging filter is defined as

$$h[i,j] = \frac{1}{M} \sum_{(k,l)\in N} f[k,l], \tag{2.34}$$

where M is the number of pixels in the local neighborhood N. An example of how it looks like when applying an averaging filter to an image is depicted in Fig. 2.30.

Chapter 3

Related Work

In this chapter, some related work on saliency coding is presented.

3.1 Perceptually Optimized Bit-Allocation for Block-Based Image or Video Coding

In conventional video coding, one fixed QP is used for the whole image. However, it is known that input-invariant quantization can lead to visually suboptimal results [11]. Thus, a simple perceptually motivated QPA is proposed in [11]. This means that the encoder uses different QPs for different blocks in the image depending on the visual activity in the corresponding block B. Therefore, a weighted Sum of Squared Errors (SSE) $D_{\text{pic}}^{\text{wSSE}}$ between the original image s and the coded image $\hat{s}$ is introduced which is calculated by

$$D_{\text{pic}}^{\text{wSSE}} = \sum_k D_k^{\text{wSSE}} = \sum_k \left(w_k \cdot \sum_{(x,y)\in B_k} (s[x,y] - \hat{s}[x,y])^2 \right), \tag{3.1}$$

where k is the index of the corresponding block and w is the associated weighting factor that is determined by high-pass filtering the input image because this approximates the difference of Gaussians behavior of the human retina [59] and then analyzing the activity in the resulting image. As it has already been mentioned in the introduction of this thesis, if there is low activity in a block, a high weight is assigned to that block because coding artifacts are more annoying in these areas. Conversely, if the current block has high activity, a low weight is assigned to this block as artifacts are less visible then.

The main goal of an encoder control is to minimize the overall distortion D_{pic} while avoiding a transgression of a given rate budget. In order to achieve this, the concept of Lagrangian bit-allocation [60] is employed in modern image and video coding. When using a fixed QP for the whole image, the optimization problem can be formulated as

$$\min_{\{\mathbf{p}_k\}} D_{\text{pic}}(\{\mathbf{p}_k\}) + \lambda \cdot R_{\text{pic}}(\{\mathbf{p}_k\}), \tag{3.2}$$

where $\{\mathbf{p}_k\}$ represents the block coding parameters and λ is the Lagrange multiplier. If any block-additive distortion metric is used, which means that the distortion for a full image is equal to the sum of the distortions for each block, and the dependencies between the blocks are ignored, which is a sufficient approximation, the optimization can be reformulated as

$$\forall k \quad \min_{\mathbf{p}_k} D_k(\mathbf{p}_k) + \lambda \cdot R_k(\mathbf{p}_k), \tag{3.3}$$

where R_k is the rate which is required for transmitting all decisions $\mathbf{p}_k$ for a block B_k and $D_k(\mathbf{p}_k)$ is the corresponding distortion. If (3.1) and (3.3) are combined, the result is

$$\min_{\mathbf{p}_k} w_k \cdot D_k(\mathbf{p}_k) + \lambda \cdot R_k(\mathbf{p}_k) \Leftrightarrow \min_{\mathbf{p}_k} D_k(\mathbf{p}_k) + \lambda_k \cdot R_k(\mathbf{p}_k), \tag{3.4}$$

where $\lambda_k = \frac{\lambda}{w_k}$. This means that for each block, only the Lagrange multiplier λ_k is adjusted. Experiments have shown that the relationship between λ_k and $Qstep$ is proportional [61, 62]. Combining this property with (2.14) yields

$$Qstep_k \propto 2^{\frac{QP_k}{6}} \tag{3.5}$$

and consequently

$$QP_k - \lfloor 3 \cdot \log_2 \lambda_k \rceil = \text{const.}, \tag{3.6}$$

where $\lfloor \cdot \rceil$ indicates rounding to an integer. Let QP' be a predefined QP for a whole image. Then, the QP_k for each block is calculated by

$$QP_k = QP' - \lfloor 3 \cdot \log_2 \frac{\lambda}{\lambda_k} \rceil = QP' - \lfloor 3 \cdot \log_2 w_k \rceil. \tag{3.7}$$

The evaluation results in [11] showed that using QPA instead of one overall QP can lead to significant improvements of visual coding quality.

3.2 Video Compression for Object Detection Algorithms

In [63], an adaptive video coding approach for computer-vision-based systems was developed. The goal of the proposed method is to obtain a saliency map that can

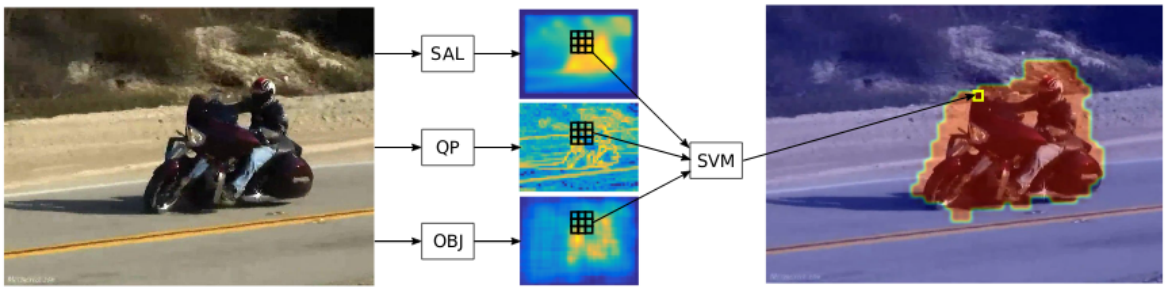


Figure 3.1: Overview of the method proposed in [63]

adapt video compression in a way that is friendly for computer vision algorithms. A binary saliency map is created according to the quadtree partitioning of the CTU. CTUs that are likely to contain an object are coded with the originally estimated QP and the rest is coded with the maximum possible QP which is 51. The binary saliency map is computed by combining 3 different methods, namely objectness, visual saliency and QP. An overview of the proposed method is depicted in Fig. 3.1. First, 3 different saliency maps are calculated which are then fused using an SVM.

The objectness saliency map M_k is obtained by calculating a set of object proposals $S_k \subseteq S$ first. The algorithm used for that is BING which has been explained in Section 2.4.2. For each pixel p of frame t, the number of proposals enclosing it is calculated by

$$M_k^t = \sum_{s \in S_k^t} \Psi_s, \tag{3.8}$$

where

$$\Psi_s(p) = \begin{cases} 1 & \text{if } p \in s, \\ 0 & \text{otherwise.} \end{cases} \tag{3.9}$$

Visual saliency is based on FES which has been described in Section 2.4.3. The last part of the overall saliency map is based on the quadtree representation of the QP values used in the CTU of the HEVC encoder.

The experiments conducted in [63] have shown that using the saliency map improves video compression without damaging the performance of object detectors like Faster R-CNN. In fact, the performance improves, especially for low bitrates. Additionally, the proposed method preserves perceptual quality in the salient regions better than standard codecs and the coding time is more than two times faster than HEVC.

Chapter 4

Saliency Coding

This chapter contains an explanation of the saliency coding approaches used in the experiments. Additionally the Cityscapes dataset used for all evaluations is introduced, followed by an explanation of two newly developed metrics called CTU precision and CTU recall that can be used for evaluating the performance of different saliency detectors. Afterwards, the newly developed framework is presented. Finally, the conducted experiments are described in detail.

4.1 Saliency Coding using Quantization Parameter Adaptation and Average Filtering

In this work, an approach similar to the one proposed in Section 3.2 is presented. Salient regions in the image are detected using BING, Edge Boxes, FES and YOLO. Consequently, a binary mask is computed which indicates whether to use a low QP for the current CTU or a higher QP. The basic concept is illustrated in Fig. 4.1. The camera which is installed at the front of the car captures an image. As the computational power of the camera is relatively low, it is not capable of detecting objects in real-time which is crucial for autonomous driving systems. Thus, some low-complexity algorithms are used which run fast even on a CPU. In the saliency

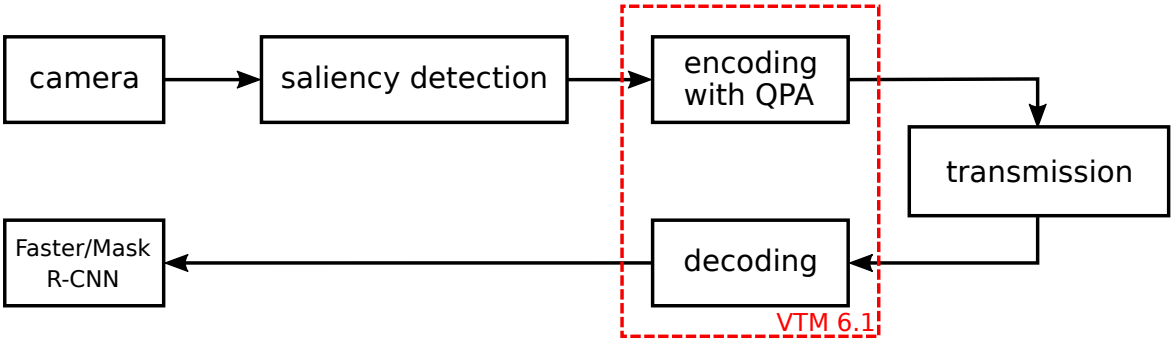


Figure 4.1: Basic concept of the approach developed in this work

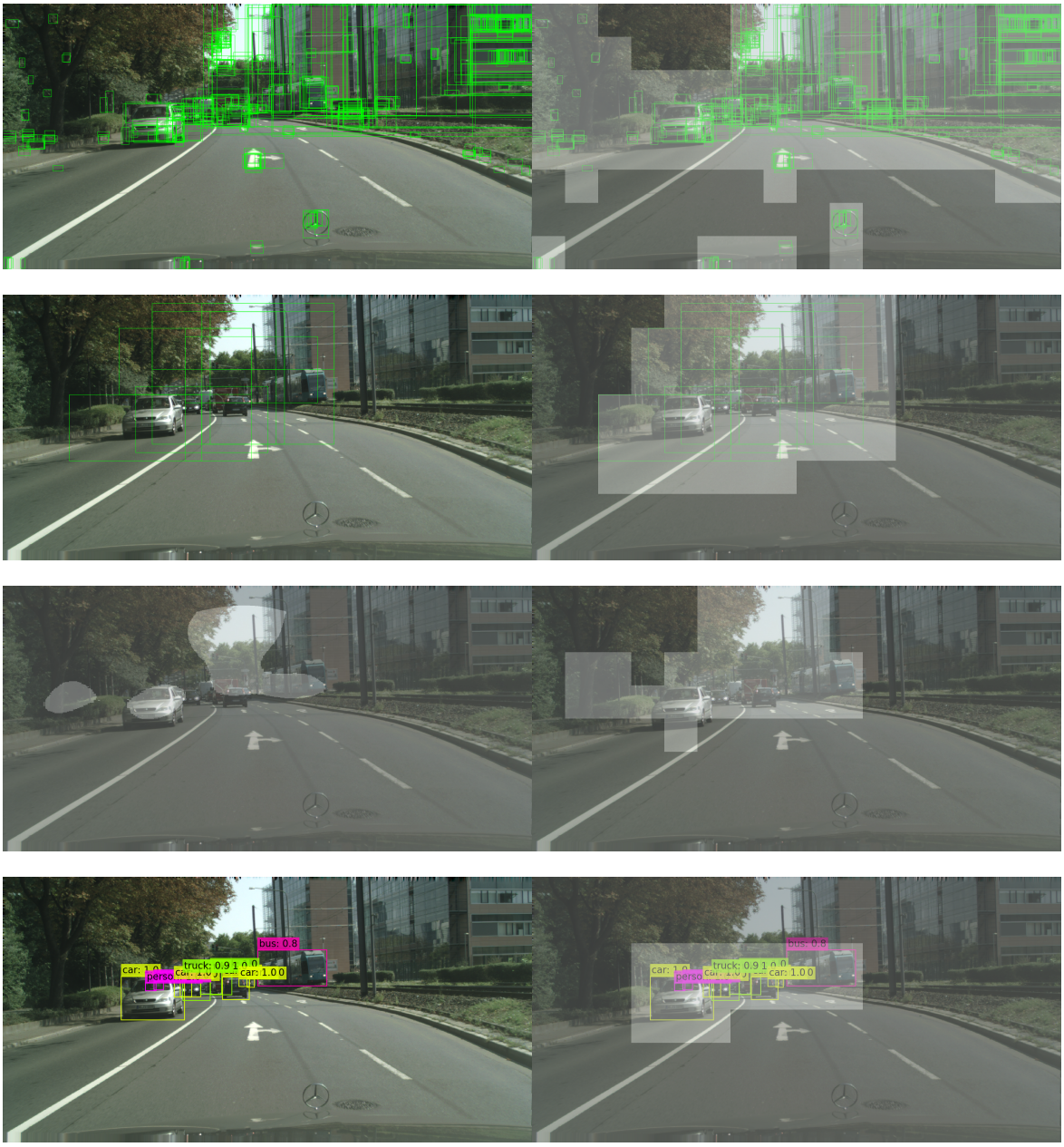


Figure 4.2: Example of the mapping of the salient regions (left column) to a CTUmap which is represented by the brighter regions in the right column. The different rows represent the different saliency detectors. Edge boxes is used in the top row, followed by BING in the second row. The third row represents FES and YOLO is used in the bottom row.

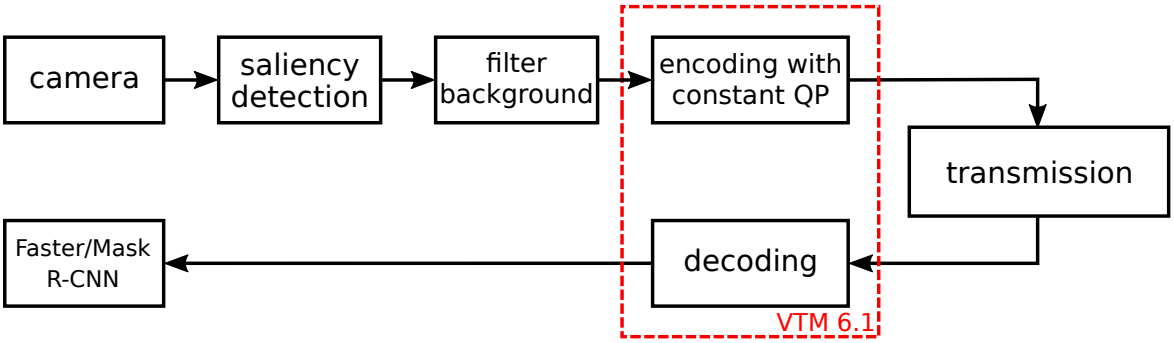


Figure 4.3: Idea of the second approach that is based on filtering the background

Figure 4.4: Exemplary images from the Cityscapes Dataset (left: original image; right: ground truth data)

detection stage, the detected objects are mapped to a CTU map as can be seen in Fig. 4.2. This map is forwarded to the VVC encoder that processes the uncompressed input image using QPA. In the transmission step, the encoded bitstream is passed on from the camera to a more powerful CPU that decodes the bitstream. A GPU may then be used to detect and classify objects in the image using Faster or Mask R-CNN.

The other approach which is conducted in this work is applying an averaging filter to the regions that are not salient in the images. The basic concept is illustrated in Fig. 4.3. The averaging filter is added as a pre-processing step in the processing chain, so VTM is not modified in this approach. Each pre-processed image is coded with a constant overall QP.

4.2 Dataset

The Cityscapes Dataset [38] is used for all evaluations in this work. It focuses on semantic understanding of urban street scenes and consists of a large, diverse set of stereo video sequences recorded in streets from 50 different cities. An exemplary

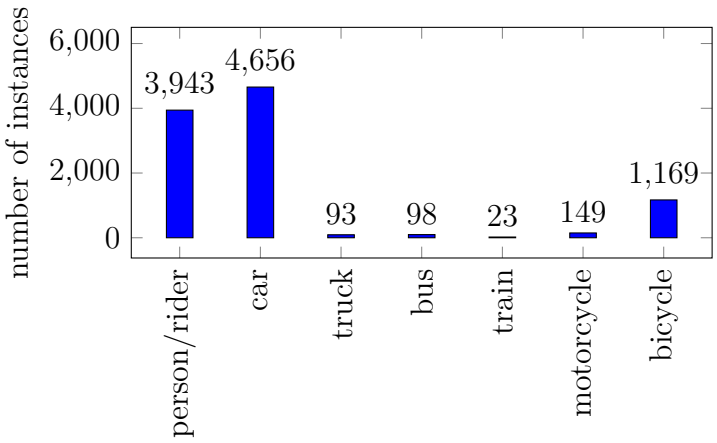


Figure 4.5: Distribution of the different classes in the validation set

image from the data set and the corresponding ground truth annotations are depicted in Fig. 4.4. Originally, 19 classes are used for evaluation in the data set. They include, for instance, road, building, vegetation, car and person. This work focuses on the classes depicted in Fig. 4.5 and the remaining classes are ignored. As can be seen from the histogram, the number of instances are distributed very unevenly among the different classes. Thus, the calculation of the mAP as shown in `evalInstanceLevelSemanticLabeling.py` is adapted in order to take this distribution into account. The AP for each class is weighted according to the corresponding number of instances, i.e.,

$$\text{mAP} = \frac{1}{10131} \sum_{i=1}^{7} w_i \cdot \text{AP}_i, \tag{4.1}$$

where 10131 is the total number of instances for all classes presented in Fig. 4.5 and w_i is the number of instances for one specific class i. Another modification that has been conducted in the calculation of the mAP is combining the classes person and rider. Originally, there is a distinction between these two classes in the ground truth data of the Cityscapes dataset. However, most of the neural networks do not make this distinction and classify riders as person. That is why each ground truth instance that is marked as rider is converted to person in order to consider this property of the neural networks. An additional modification has to be done when evaluating the performance of Faster R-CNN. As can be seen in Fig 4.4, the ground truth data is represented by masks that encode an input object's spatial layout. The outputs of Faster R-CNN are, however, bounding boxes that do not have this pixel-to-pixel behavior. This is why the ground truth polygons are converted to rectangular bounding boxes for the

Figure 4.6: Visualization of the CTU precision proposed in this work. It is defined as the ratio between the relevant CTUs (red area) and the detected number of CTUs (red area and blue area).

evaluation of Faster R-CNN in this work. More specifically, this is done by determining the minimum and maximum x and y coordinates of each ground truth polygon and consequently converting it to a bounding box

$$b = [y_{\min}, x_{\min}, y_{\max}, x_{\max}]. \tag{4.2}$$

4.3 CTU-based evaluation metrics

In order to evaluate the performance of the different saliency detectors, two new evaluation metrics are introduced in this work. These metrics allow for a fast evaluation of the performance of different saliency detectors. The first one is based on calculating the ratio between the relevant number of CTUs and the total number of CTUs detected by the algorithm. This concept is visualized in Fig. 4.6. Formally, the metric is defined as

$$\text{CTU precision} = \frac{\sum_{i=1}^{500} \#\text{relevantCTUs}_i}{\sum_{i=1}^{500} \#\text{detectedCTUs}_i}, \tag{4.3}$$

where i is the index of the corresponding image from the Cityscapes validation set. This metric resembles the precision from (2.6) and the goal is, of course, to achieve a CTU precision of 1, i.e., to minimize the blue area in Fig. 4.6.

The second metric is similar to the recall defined in (2.7). It calculates the ratio between the number of detected relevant CTUs and the total number of relevant CTUs, i.e.,

$$\text{CTU recall} = \frac{\sum_{i=1}^{500} \#\text{detectedRelevantCTUs}_i}{\sum_{i=1}^{500} \#\text{relevantCTUs}_i}. \tag{4.4}$$

Figure 4.7: Visualization of the CTU recall proposed in this work. It is defined as the ratio between the number of detected relevant CTUs (blue area) and the number of relevant CTUs (red area).

Figure 4.8: Example of a file that is used to store the CTU addresses

Again, the goal is to achieve a CTU recall of 1 which can be thought of as minimizing the red area in Fig. 4.7.

4.4 Framework

The framework developed in this work is structured as follows. The saliency detection stage is conducted first. The code for Edge boxes is taken from [64]. For BING, the source code was taken from [65]. The code for FES can be found in [66]. The code for YOLO was taken from [67].

In order to convert the bounding boxes to CTU addresses, a new Python module `compute_CTU_addresses.py` has been developed. Here, a CTU in an image is marked as salient if a bounding box from the saliency detectors intersects with the corresponding CTU. The addresses of the salient CTUs are stored in a file at the end of the saliency detection stage. An example of such a file is depicted in Fig. 4.8. The CTU addresses stored in the files are used in VTM 6.1 to determine whether to

code the corresponding CTU with the base QP or a higher QP. In order to read the information from the files, a new class `CSVReader` has been added to the common library of VTM 6.1. This class depends on the library `boost_1_71_0` [68], so it has to be installed on the system.

The main code modifications in VTM 6.1 have been conducted in the method `applyQPAdaptation()` of the class `EncSlice`. This method is actually used to apply the perceptually optimized bit-allocation that has been described in Section 3.1. For each CTU, an individual QP value is calculated depending on the activity in the block. This approach is adapted in such a way that a base QP is used for the salient CTUs and a higher QP is used for the CTUs in the background.

The adapted encoder can be called by

```
bin/EncoderAppStatic -c cfg/encoder_intra_vtm.cfg
  -i <inputYUV> -b <bitstreamFile>
  -f 1 -fs 0 -fr 30 -q 12
  -hgt 1024 -wdt 2048 --InputBitDepth=8
  --InternalBitDepth=8 --PerceptQPA
  --SliceChromaQPOffsetPeriodicity --HigherQP=32
  --RelevantCTUs=<relevantCTUs.txt>,
```

where the parameters `--PerceptQPA` and `--SliceChromaQPOffsetPeriodicity` are necessary to enable the perceptually optimized bit-allocation. The standard configuration file that is used in this work is `encoder_intra_vtm.cfg`. This file has not been modified in the course of this work. The input of the encoder is an uncompressed png image from the Cityscapes dataset. As the encoder expects yuv files as inputs, the png files from the dataset have to be converted first. In this work, the conversion is always done using

```
ffmpeg -i <inputPNG> -pix_fmt yuv420p <outputYUV>,
```

where the parameter `-pix_fmt` denotes the pixel format. The encoder of VTM expects the format 4:2:0 which is used in most international standards. It represents a reduction of the color resolution by a factor of 2. The big advantage of choosing this format is that the data rate is reduced significantly in comparison to a unmodified resolution of the color components but the loss in visual quality is negligible [69].

After the encoding is finished, the next step is decoding the bitstream with the unmodified decoder of VTM 6.1. This can be done by using

```
bin/DecoderAppStatic -b <bitstreamFile> -o <outputYUV>.
```

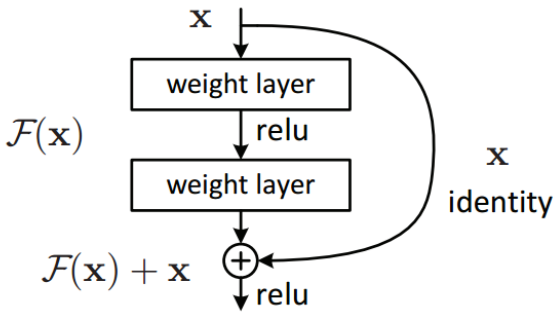


Figure 4.9: Visualization of the skip connection used in ResNets. It adds the output from the previous layer to the layer ahead [74]

The next step in the framework is converting the decoded yuv file back to a png file using

`ffmpeg -s 2048x1024 -i <inputYUV> <outputPNG>,`

where the parameter `-s` defines the size of the image which is always 2048×1024 pixels in the Cityscapes dataset. Consequently, the images are fed into a Faster R-CNN Inception Resnet v2 or a Mask R-CNN Resnet 101 both pretrained on the COCO dataset [41]. For the implementation of the networks, tensorflow-gpu 1.14.0 [70] was used. The corresponding models were taken from the Tensorflow detection model zoo [71]. The architecture that is used for Faster R-CNN in this work is `faster_rcnn_inception_resnet_v2_atrous_coco`. It runs at 620 ms per image and achieves a COCO mAP of 37%. The architecture used for Mask R-CNN is `mask_rcnn_resnet101_atrous_coco_2018_01_28`. This network runs at 771 ms per image and has a COCO mAP of 36%. Both network architectures are based on Residual Networks (ResNets) [72]. The basic idea behind ResNets is to introduce shortcut connections into the network that allow for an alternative path for the gradient to flow through. This technique solved the well-known vanishing gradient problem [73] which may occur during training of ANNs. More specifically, it could happen in deep neural networks with many layers that the gradient of the loss function goes to zero, making it hard to train the network. The concept of a shortcut connection used in ResNets is shown in Fig. 4.9.

The architecture that is used for Faster R-CNN, the Inception ResNet v2 [75], combines the concept of Inception modules [76] with ResNets. Inception modules have been designed to take the large variation of the size of objects in images into account. More precisely, for larger objects, it is beneficial to choose larger filter sizes whereas for small objects smaller filter sizes should be chosen. Because of that, Inception modules

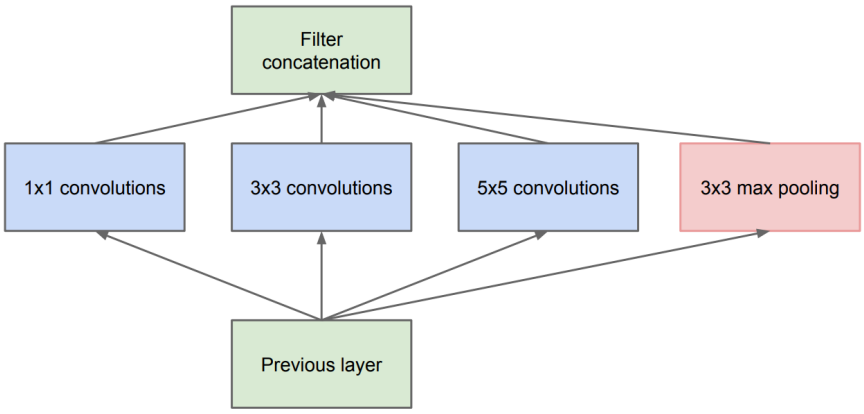


Figure 4.10: Visualization of an Inception module [76]

(a) Faster R-CNN (b) Mask R-CNN

Figure 4.11: Exemplary outputs of the networks

introduce different filter sizes that operate on the same level, as shown in Fig. 4.10.

Exemplary outputs of the networks are depicted in Fig. 4.11. In the developed framework, the information about the bounding boxes and contours is additionally stored in text files and mask files, respectively. The performance of the neural networks is evaluated using the Cityscapes evaluation script [77]. The output of the script is a file that lists the AP and AP50 for each class over all Cityscapes validation images.

4.5 Description of the experiments

The experiments in this work were conducted as follows. In the first approach, the uncompressed images from the Cityscapes validation set were processed using VTM 6.1 with QPA. For the QP values, a combination of {2, 7, 12, 17, 22, 27, 32, 37, 42, 47, 52, 57, 62} was used. The salient regions of the images were coded with the so called base QP and the rest was coded using a higher QP. In the other approach which is conducted in this work an averaging filter is applied to the regions that are not salient

(a) Image coded with an overall QP of 32

(b) Image with QPA applied to it (base QP: 32, higher QP: 62)

(c) Image with averaging filter of size 15×15 applied to it (QP=32)

Figure 4.12: Visualization of the two approaches used in this work

in the images using the function `cv2.blur()` [78]. The filtering operations run at 3 – 5 ms per image and can hence be used for real time applications.

An example of how the images look like when applying the two different approaches is depicted in Fig. 4.12. The image in on the top is coded with the unmodified VTM 6.1 using a QP of 32 for the whole image. The picture in the middle is coded with the adapted QPA approach, where the QP is set to 32 for the salient CTUs and to 62 for the background. The image coded with the average filter-based approach is depicted on the bottom. It can be observed from Fig. 4.12b and Fig. 4.12c that the quality of the images is clearly better in the regions with people and cars in it. Irrelevant parts of the image like the leaves of the trees exhibit a worse quality.

Chapter 5

Results

This chapter contains a documentation of the performance of the different saliency detectors and of the results for Faster R-CNN and Mask R-CNN. Section 5.1 gives an overview of the CTU precision and CTU recall for the different saliency detectors. In Section 5.2, the results of Faster R-CNN for constant QPs, QPA and the application of averaging filters are presented. Analogously, the results for Mask R-CNN are documented in Section 5.3.

5.1 CTU precision and CTU recall

The CTU-precision-recall curves for the different saliency detectors are depicted in Fig. 5.1. The different points on one curve represent a variation of a specific parameter of the corresponding saliency model. For BING, the maximum number of bounding boxes is varied from 10 to 50 in steps of 10. For Edge boxes, the maximum number of bounding boxes was set to 500, 1000, 2000 and 10000. The threshold for object proposals β was set to 0.9 and the scale sensitivity κ was set to 1.9. The remaining parameters were set to the default values for Edge boxes. In the case of FES, the threshold value used for the decision whether a pixel is salient or not was changed from 60 to 130 in steps of 10, where for lower thresholds more pixels are marked as salient. The IOU threshold for YOLO was set to 0.4 which means that each bounding box that has an IOU greater than 0.4 with a box with higher score is removed. All bounding boxes that have a confidence less than the NMS threshold 0.05 are removed as well.

It can be observed from Fig. 5.1 that YOLO achieves a very high CTU precision in comparison to the other detectors. The main reason for this is that YOLO is the only detector that is capable of classifying the detected objects into different classes. CTUs that contain instances that are not relevant can be discarded. In contrast, BING, Edge boxes and FES can only distinguish between whether there is an object of any class present in the current CTU or not. This is why a lot of unnecessary objects are

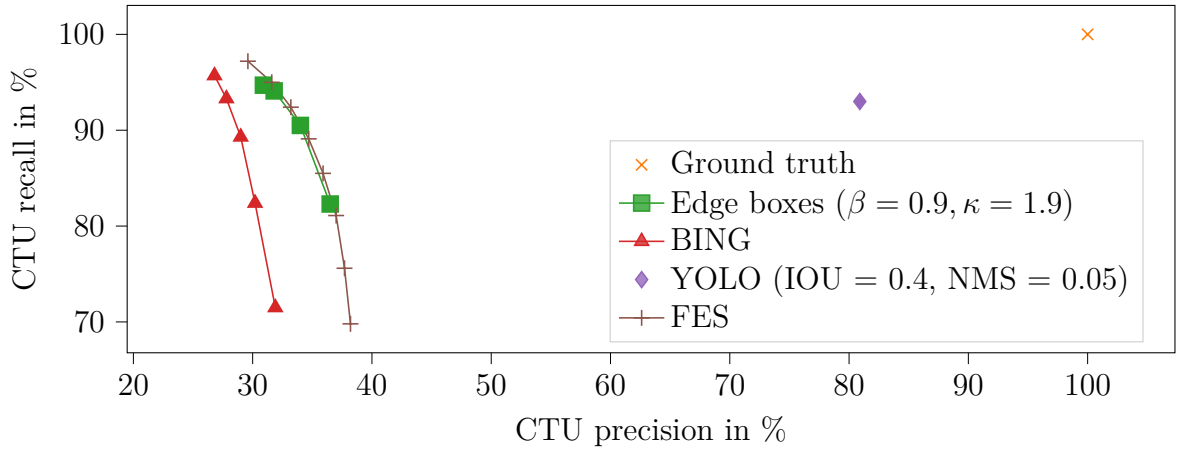
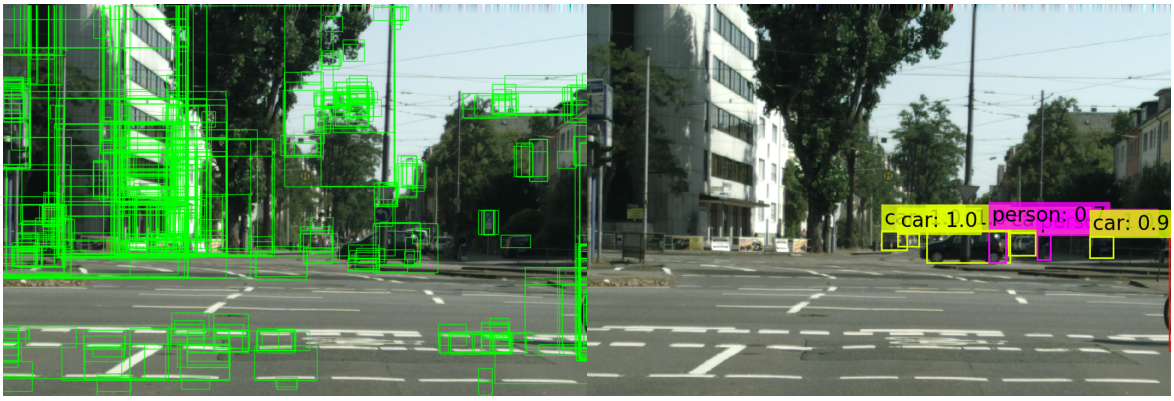


Figure 5.1: CTU-precision-recall plots for different saliency detectors. The number of maximum bounding boxes is varied for BING and Edge boxes. For FES, the threshold is changed.



(a) Edge boxes (b) YOLO

Figure 5.2: Visualization of the differences between Edge boxes and YOLO

detected and the CTU precision is relatively low. An example of where Edge boxes performs badly and a comparison to YOLO is depicted in Fig. 5.2. The signs on the street, for instance, are detected as objects by Edge boxes because their contours are fully enclosed by the corresponding bounding boxes. In contrast, YOLO performs better because of its classifier and only detects the relevant objects.

Another observation from the plot in Fig. 5.1 is that the curves for BING, Edge boxes and FES are monotonically decreasing. The reason for this is that the less boxes are detected the less relevant objects are detected as well.

Among the three detectors running on a CPU only, FES achieves the best results,

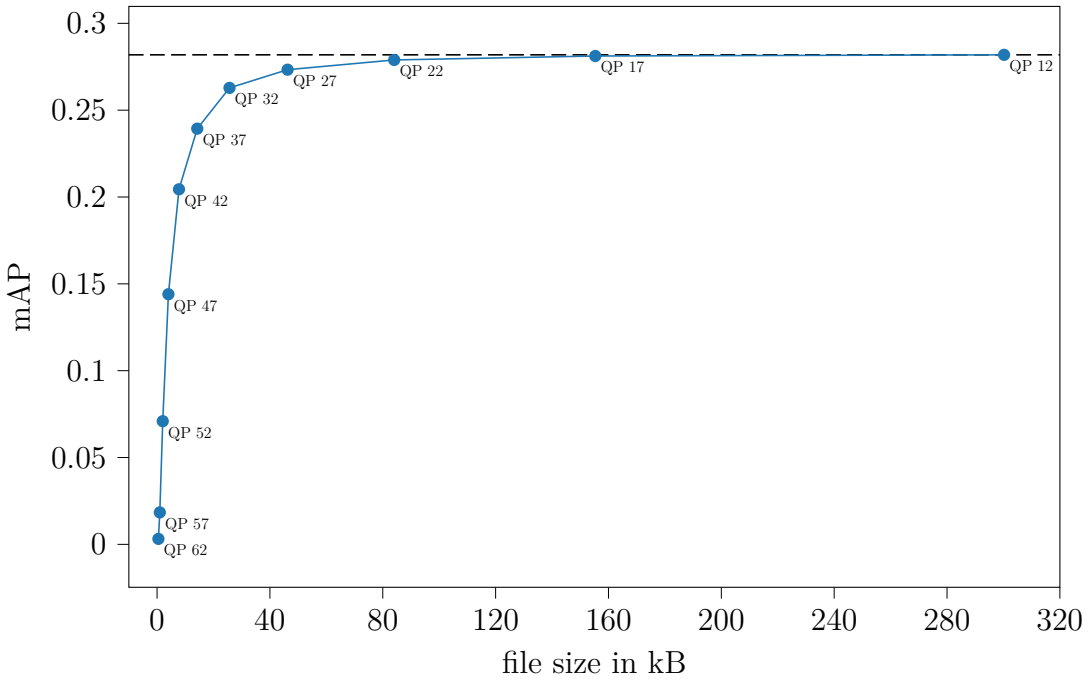


Figure 5.3: Performance of Faster R-CNN for different QPs on the Cityscapes validation set

followed by Edge boxes and BING has the worst results in terms of CTU precision. An ideal detector would have a CTU recall and a CTU precision of 100%, i.e., the points in the plot would be located in the top right corner. This is the case for ground truth, as can be seen in the figure.

5.2 Faster R-CNN

This section gives an overview of the results obtained for Faster R-CNN. Therefore, its performance on the images coded by the unmodified VTM are presented first, followed by the performance analysis of the QPA method. Afterwards, the results for the average filtering approach are described.

5.2.1 Constant Quantization Parameter

The impact of a change of the overall QP on the performance of Faster R-CNN is depicted in Fig. 5.3. The horizontal dashed line represents the mAP obtained for the uncompressed images. In this experiment, one fixed QP was used for all images. The QPs were varied from 12 to 62 in steps of 5. It can be observed that for QPs ≤ 17, there is no noteworthy difference to the performance with the uncompressed images. For

(a) QP=7 (b) QP=32

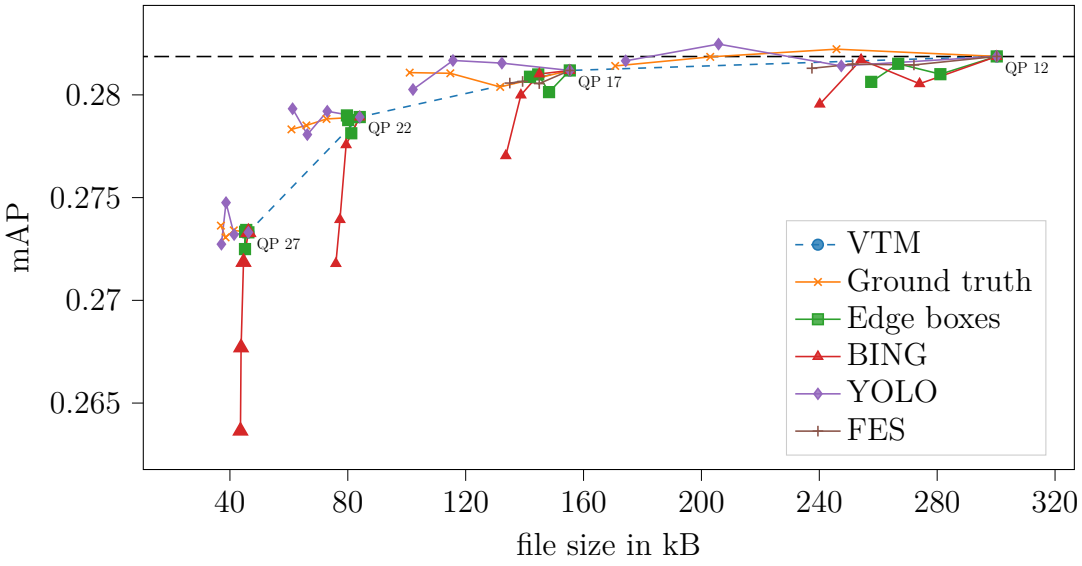
(c) QP=57 (d) QP=62

Figure 5.4: Visualization of the results for Faster R-CNN when using different QPs for the image `frankfurt_000000_000294_leftImg8bit.png`

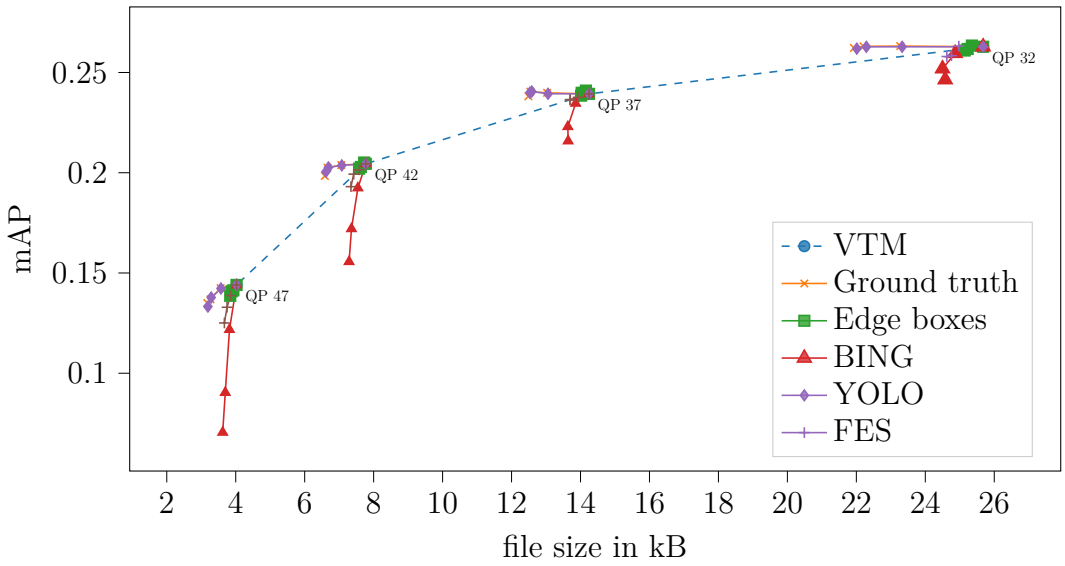
higher QPs, the mAP decreases. The reason for this is that the images are distorted so much that Faster R-CNN cannot detect the objects correctly anymore, as can be seen in Fig. 5.4. In this example, most of the relevant objects are detected for QP = 7. For QP = 32, one person is not detected. When choosing a QP of 57, the car is the only object being detected. In the case of QP = 62, no object is detected at all.

5.2.2 Quantization Parameter Adaptation

The result for using QPA with base QPs of 12 to 47 in steps of 5 is depicted in Fig. 5.5. The points where the lines join represent the mAPs for the unmodified VTM from Fig. 5.3, where the images have been coded with constant QPs. These points are connected by the dashed blue line. The remaining points represent mAPs that have been obtained for the images that have been coded with QPA. The CTU width was set to 128. For each base QP, the corresponding next three higher QPs in steps of 5 are plotted in the figure. For a base QP of 12, for instance, the combinations (12,17), (12,22) and (12,27) were chosen, where the first value represents the base QP and the second value indicates the higher QP. For a base QP of 17, the combinations (17,22), (17,27) and (17,32) were used. The combinations for the remaining base QPs were



(a) QP 12 – 37



(b) QP 32 – 47

Figure 5.5: Result of the application of QPA with varying higher QPs for Faster R-CNN

picked analogously.

For YOLO, an IOU threshold of 0.4 and an NMS threshold of 0.05 was used. The maximum number of detections was set to 10 for BING. For FES, a threshold of 120 was used for the decision whether a pixel is salient or not. For Edge boxes, the NMS threshold for object proposals β was set to 0.9, the scale sensitivity κ was set to 1.9 and the maximum number of bounding boxes per image was set to 1000. The remaining parameters were set to the default values. The parameters used for the

Table 5.1: Higher QPs used for each base QP

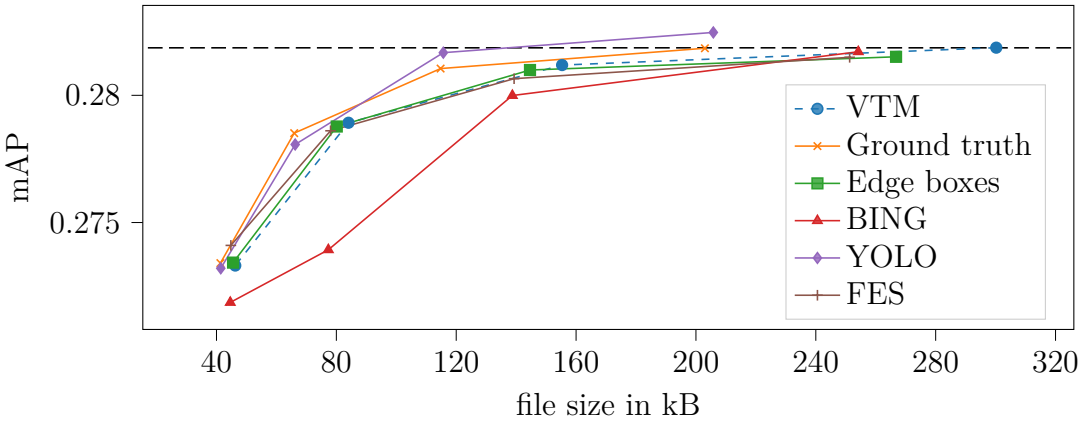
Base QP	12	17	22	27	32	37	42	47
Higher QP	22	27	32	32	37	42	47	52

different saliency detectors are used for all experiments in the remainder of this work in order to allow for a comparison of the different approaches.

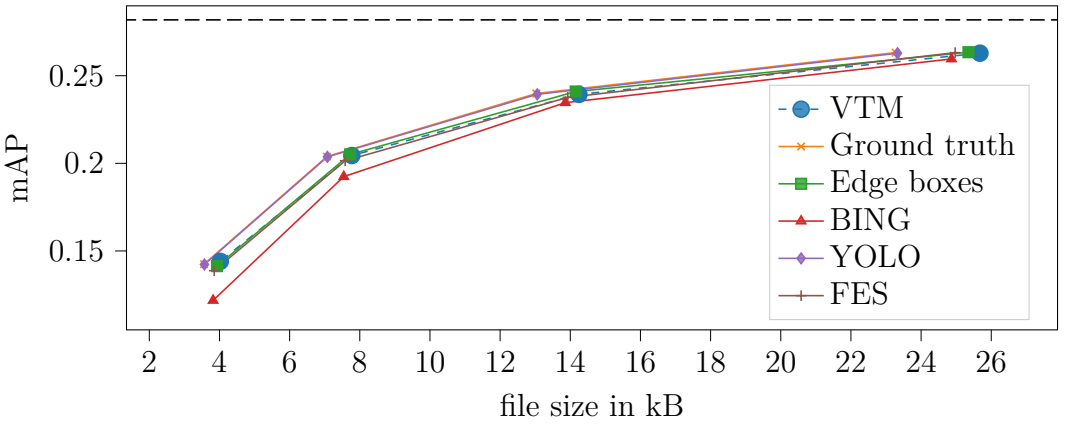
It can be observed from Fig. 5.5 that the application of QPA leads to a significant compression of the file sizes while the impact on the mAP is negligible, at least for ground truth and YOLO. For Edge boxes, BING and FES, the mAP tends to decrease for increasing background QPs. As the goal of this work is to keep the degradation of the mAP as small as possible, the background QPs are chosen accordingly to satisfy that condition, i.e., the difference between base QP and background QP is kept relatively small. The QP combinations used for all the following experiments based on QPA are depicted in Tab. 5.1. For low QPs, a larger difference between base QP and background QP was chosen than for high QPs, because for high QPs the mAP decreases more quickly when increasing the background QP, as Fig. 5.5 shows.

The rate-mAP curves for the different saliency detectors using the QP combinations from Tab. 5.1 are depicted in Fig. 5.6. The corresponding BD rates and BD mAPs are listed in Tab. 5.2. It should be noted that the BD rate represents the relative difference in percent to the unmodified VTM whereas the BD mAP describes the absolute average difference in percentage points. As can be seen, employing the base QP only to CTUs that contain ground truth instances achieves the highest compression and also a small improvement of the mAP. The reason for achieving the highest compression is that most of the irrelevant CTUs are coded with the background QP for the ground truth model in comparison to the other models. The increased mAP can be attributed to the reduction of false positives. YOLO achieves the second best results on average in terms of BD rate but performs even better than ground truth for the base QPs 12 and 17, achieving the maximum mAP of 28.2% for an average file size of 206 kB. As it has been described in Section 5.1, the reason for the good performance of YOLO is the high CTU precision and CTU recall. Among the three saliency detectors running on a CPU only, Edge boxes achieves the best results, followed by FES. BING has the worst results. A reason for this is the low CTU recall of only 71.5% when setting the number of detections to 10.

Another observation from these results is that QPA is more useful for lower base QPs, as higher rate savings are achieved.



(a) QP 12 – 27



(b) QP 32 – 47

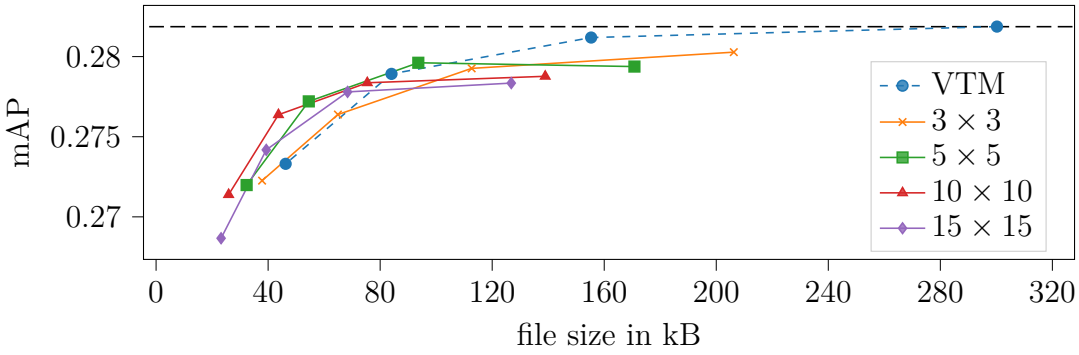
Figure 5.6: Rate-mAP curves for Faster R-CNN using the different saliency detectors with QPA

Table 5.2: BD values for Faster R-CNN using the different saliency detectors with QPA

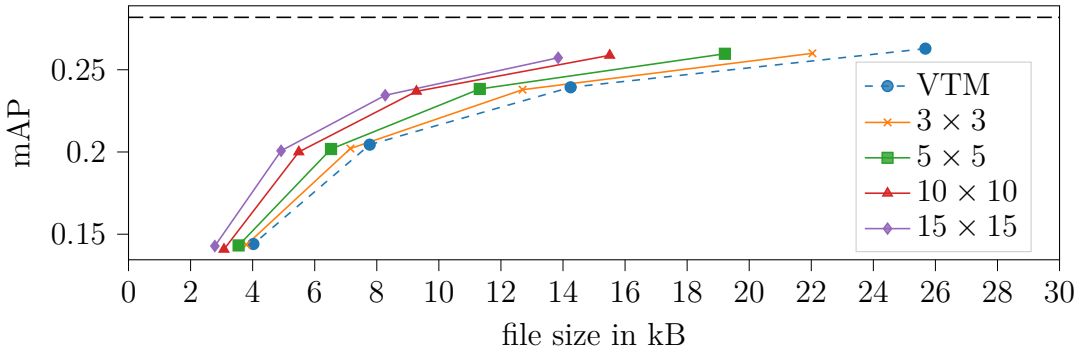
	BD rate (QP 12 – 27)	BD mAP (QP 12 – 27)	BD rate (QP 32 – 47)	BD mAP (QP 32 – 47)
Ground truth	-15.37%	+0.09%	-9.14%	+0.61%
Edge boxes	-1.04%	0.00%	-1.90%	+0.12%
BING	+52.79%	-0.21%	+11.59%	-0.74%
YOLO	-14.98%	+0.11%	-8.54%	+0.57%
FES	-0.16%	0.00%	+0.95%	-0.06%

5.2.3 Filtering

In this section, the results for the experiments based on average filtering are described in detail. The first experiment based on averaging filters was to investigate the impact of the number of pixels in the local neighborhood of the filter on the file sizes and the



(a) QP 12–27



(b) QP 32–47

Figure 5.7: Rate-mAP curves for average filtering of regions that do not contain ground truth instances using varying local neighborhoods N

Table 5.3: BD values for average filtering of regions that do not contain ground truth instances using varying local neighborhoods N

N	BD rate (QP 12 – 27)	BD mAP (QP 12 – 27)	BD rate (QP 32 – 47)	BD mAP (QP 32 – 47)
3×3	+14.73%	-0.08%	-5.74%	+0.41%
5×5	-15.95%	+0.02%	-14.17%	+1.06%
10×10	-23.00%	+0.01%	-26.27%	+2.16%
15×15	-10.75%	-0.02%	-33.49%	+2.82%

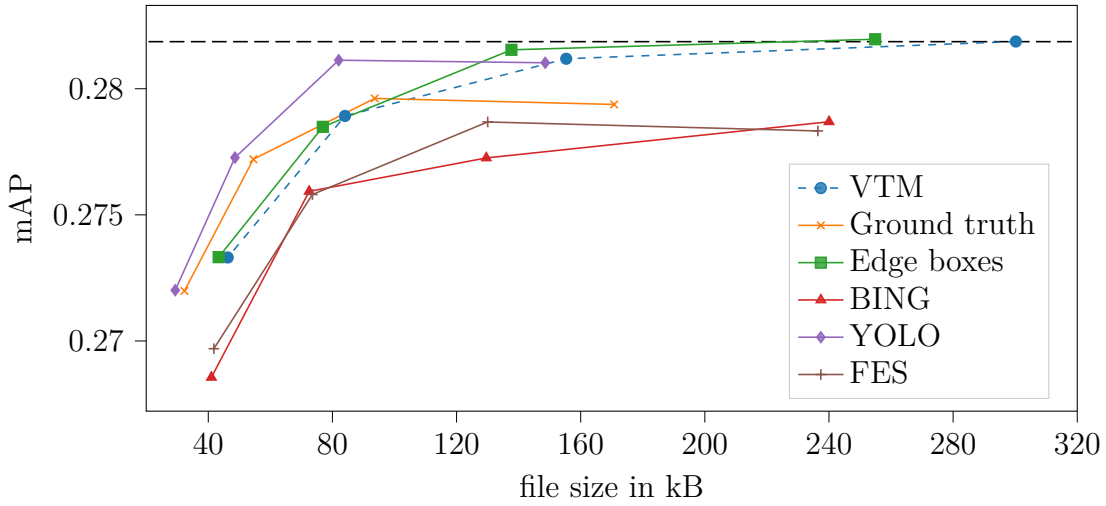
mAP. Therefore, the CTUs that do not contain ground truth instances were filtered with averaging filters of size 3×3, 5×5, 10×10 and 15×15, respectively. The filtered images were then compressed using the unmodified VTM with constant QPs. Finally, Faster R-CNN was used to detect and classify objects in the images.

The resulting rate-mAP curves of the experiment are depicted in Fig. 5.7 and the corresponding BD values are listed in Tab. 5.3. As can be seen, filtering regions that are not important can reduce the file sizes significantly. The maximum negative BD rate

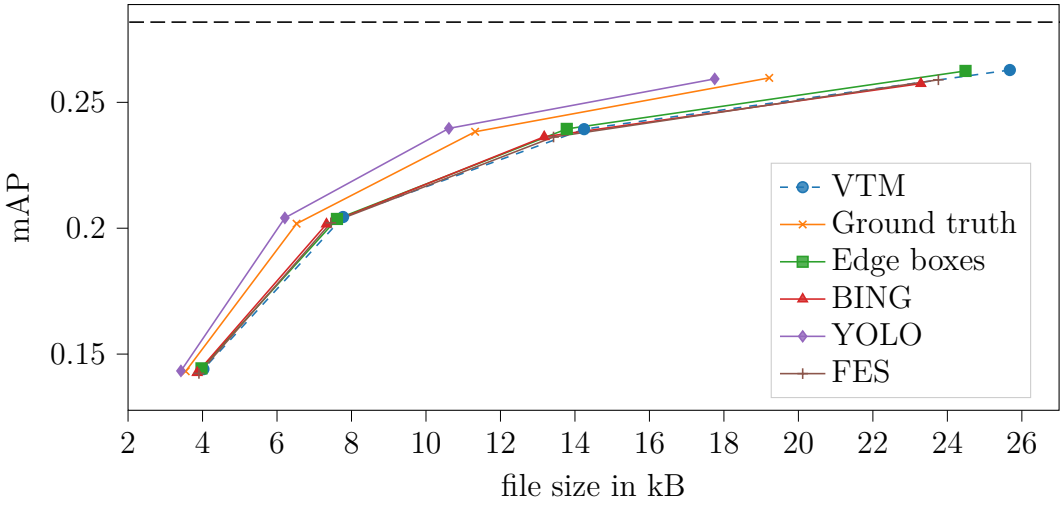
-33.49% is achieved for $N = 15 \times 15$ and QPs of 32 – 47. Additionally, the highest BD mAP +2.82% can be achieved with these parameters. In general, it seems like applying the average filters is more useful for high QPs than for low QPs. For QPs < 22, for instance, all filter curves are mainly located on the right side of the dashed curve that represents the unmodified VTM. This means that the filtered versions perform worse than the unmodified VTM for these QPs. The uncompressed level cannot be reached at all for all filter sizes. In contrast, for higher QPs like 32 – 47, it is beneficial to apply averaging filters for all QPs, as can be seen in Fig. 5.3b. The larger the filter size, the more data can be saved for these QP values. Additionally, the mAP gets better for increasing filter sizes. A reason for the different behavior for low QPs and high QPs might be that the quality differences have a high impact on the performance of the neural networks. If the quality difference between salient regions and background is too large, it seems like the mAP decreases, at least for low QPs. This can be observed from the curve with $N = 15 \times 15$ in Fig. 5.3a. Due to the different results for low QPs and high QPs, the filter sizes 5×5 and 15×15 are tested in detail for the different saliency detectors in the following.

The rate-mAP for the different saliency detectors for a filter size of 5×5 are illustrated in Fig. 5.8 and the corresponding BD values are shown in Tab. 5.4. The filtering operations were conducted as follows. For the method ground truth, the averaging filter was applied to all CTUs of size 128×128 that do not contain a ground truth instance, as it has also been done in Fig. 5.7. A similar approach was conducted for FES. Each CTU of size 128×128 that does not contain at least one salient pixel is filtered. In the case of Edge boxes, BING and YOLO, the averaging filter was applied to each pixel that is not located in a bounding box produced by the corresponding saliency method, so the bounding boxes were not mapped to CTUs for these methods.

As can be seen in Tab. 5.4, YOLO achieves the best results among the saliency detectors for QPs 12 – 27. This is due to the high CTU precision and recall which one can observe from Fig. 5.1. It performs even better than ground truth, because everything but the bounding boxes was filtered in the case of YOLO whereas the filtering for ground truth was conducted on a CTU basis which is less precise. Among the saliency detectors running on a CPU, Edge boxes achieves the best results. For QP = 12, for instance, a reduction of about 15% is achieved in comparison to the unmodified VTM. Additionally, the maximum mAP of 28.2% is achieved by Edge boxes at QP = 12. The methods BING and FES perform worse than the unmodified VTM for low QPs. This is due to the worse CTU recall.



(a) QP 12 – 27

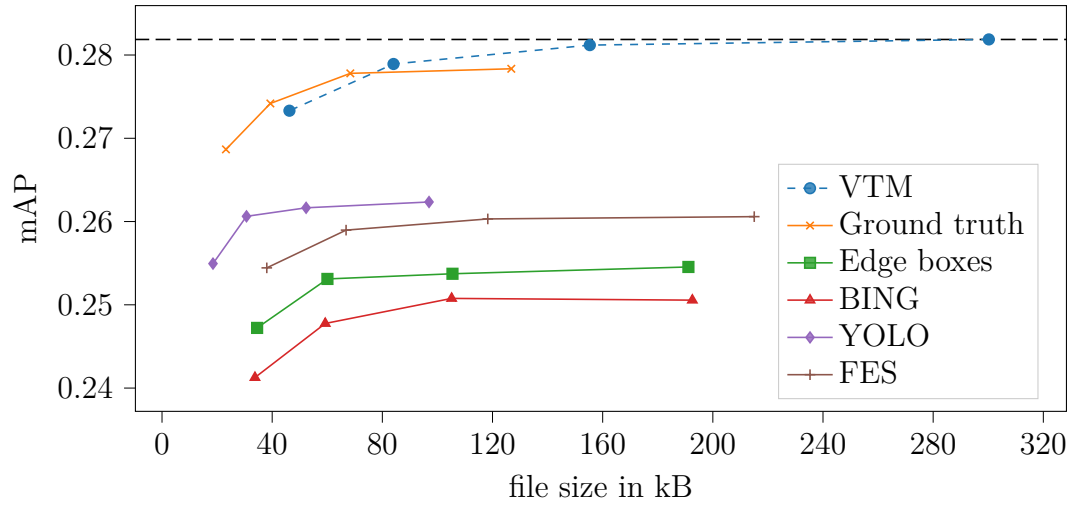


(b) QP 32 – 47

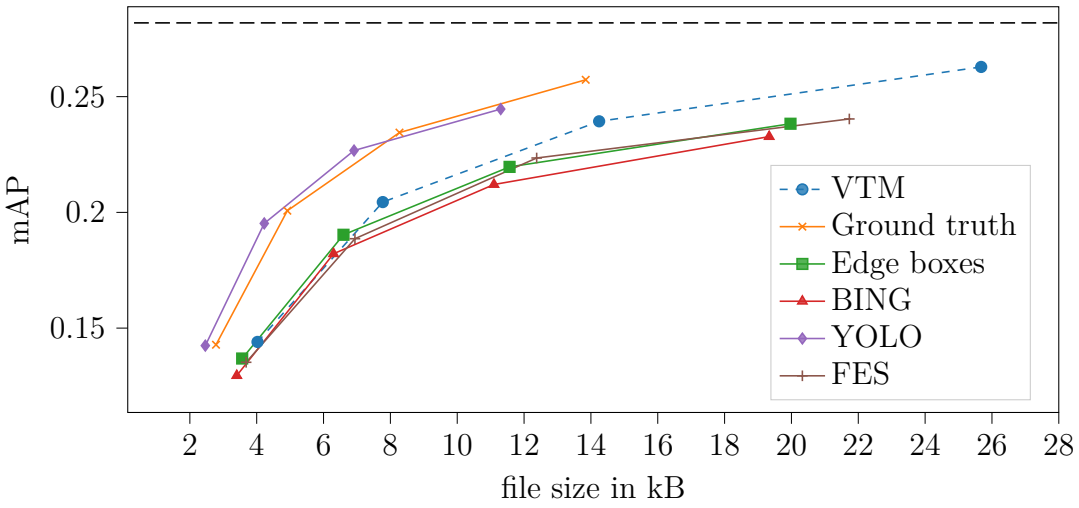
Figure 5.8: Rate-mAP curves for Faster R-CNN using the different saliency detectors and an averaging filter of size 5×5

Table 5.4: BD values for Faster R-CNN using the different saliency detectors and an averaging filter of size 5×5

	BD rate (QP 12 – 27)	BD mAP (QP 12 – 27)	BD rate (QP 32 – 47)	BD mAP (QP 32 – 47)
Ground truth	-15.95%	+0.02%	-14.17%	+1.06%
Edge boxes	-6.60%	+0.04%	-1.82%	+0.13%
BING	+57.72%	-0.28%	-1.56%	+0.10%
YOLO	-30.15%	+0.19%	-20.16%	+1.58%
FES	+39.43%	-0.24%	-0.49%	+0.03%



(a) QP 12 – 27



(b) QP 32 – 47

Figure 5.9: Rate-mAP curves for Faster R-CNN using the different saliency detectors and an averaging filter of size 15×15

Table 5.5: BD values for Faster R-CNN using the different saliency detectors and an averaging filter of size 15×15

	BD rate (QP 12 – 27)	BD mAP (QP 12 – 27)	BD rate (QP 32 – 47)	BD mAP (QP 32 – 47)
Ground truth	-10.75%	-0.02%	-33.49%	+2.82%
Edge boxes	-	-2.53%	+6.58%	-0.55%
BING	-	-2.93%	+14.81%	-1.05%
YOLO	-	-1.50%	-37.57%	+3.20%
FES	-	-1.94%	+12.10%	-0.82%

The same experiment has also been conducted with the filter size 15×15 instead of 5×5. The rate-mAP curves and BD values are depicted in Fig. 5.9 and Tab. 5.5, respectively. For QP 12 – 27, no BD rates could be calculated for YOLO, BING, FES and BING, because the differences of the mAPs are too large. One can observe for low QPs that there is a significant degradation of the mAP for YOLO, FES, Edge boxes and BING of up to -2.93% for BING. The reason for this is that if an object is not detected by the saliency model, then it is very likely that the network cannot detect the object as well, because the filtering is too strong for $N = 15 \times 15$. The results show that for larger filter sizes the CTU recall of the corresponding saliency models becomes more and more important.

5.3 Mask R-CNN

Analogously to Section 5.2, the results of Mask R-CNN for the different approaches are presented in this section.

5.3.1 Constant Quantization Parameter

The results of Mask R-CNN for different QPs are depicted in Fig. 5.10. Similar to Faster R-CNN, there is no big difference to the mAP in the uncompressed case for QPs < 17. Again, the mAP decreases for higher QPs.

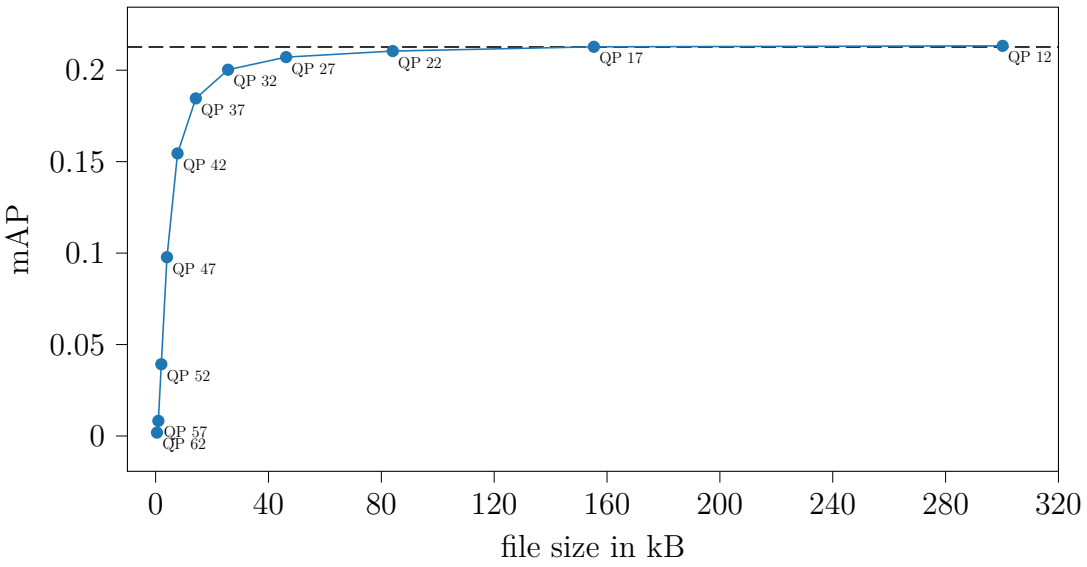
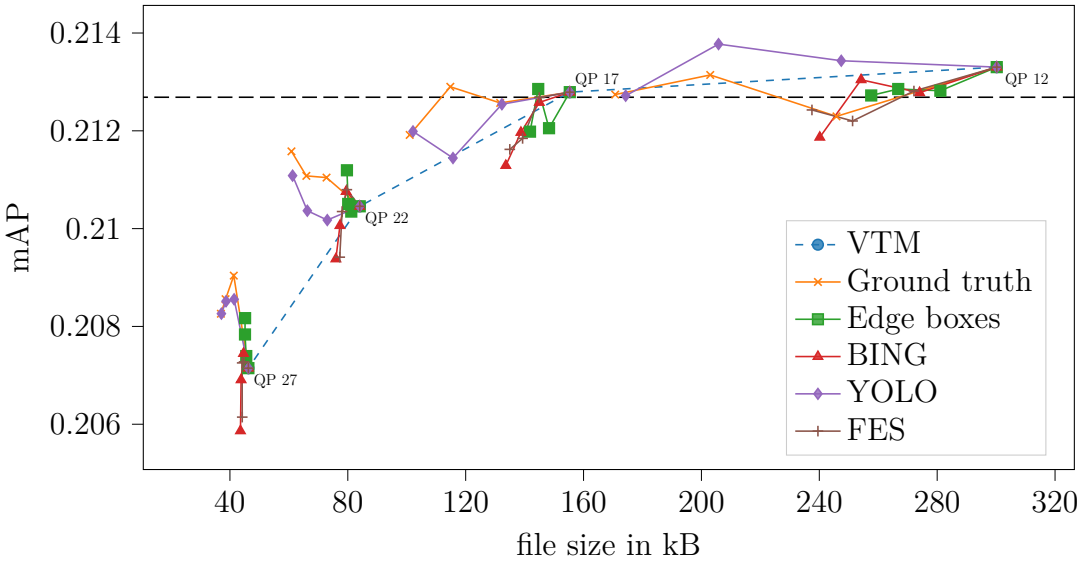
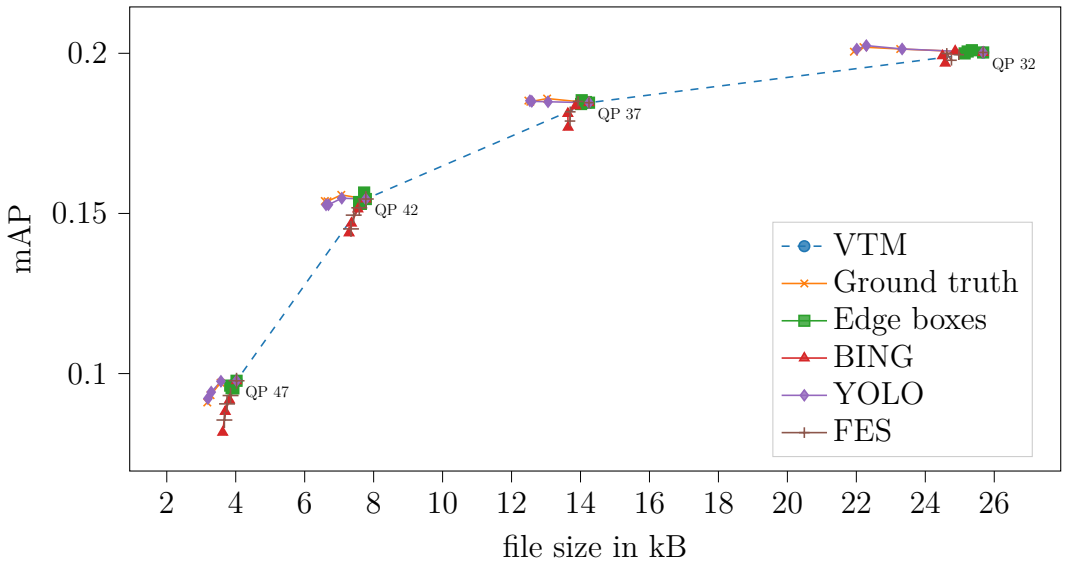


Figure 5.10: Performance of Mask R-CNN for different QPs. The dashed line represents the mAP for the uncompressed images



(a) QP 12 – 37



(b) QP 32 – 47

Figure 5.11: Result of the application of QPA with varying higher QPs for Mask R-CNN

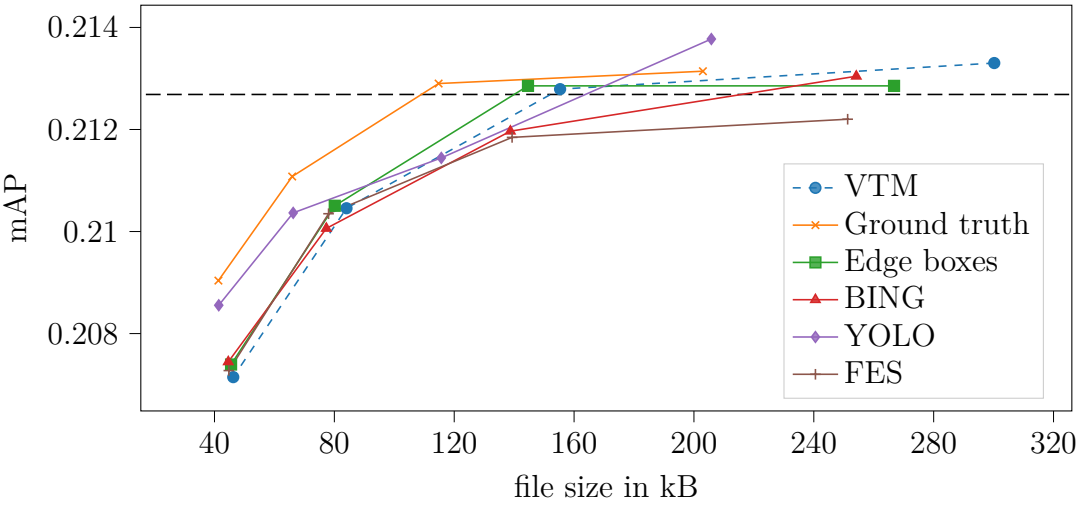
5.3.2 Quantization Parameter Adaptation

Analogously to Section 5.2.2, where the results for Faster R-CNN have been described, the outcome for Mask R-CNN using QPA is presented in this section. For each base QP 12 – 47, the corresponding next three higher QPs in steps of 5 are plotted in Fig. 5.11. The parameters used for each saliency model are equal to the ones that have been chosen for Faster R-CNN in Section 5.2.2. Again, the impact of higher background

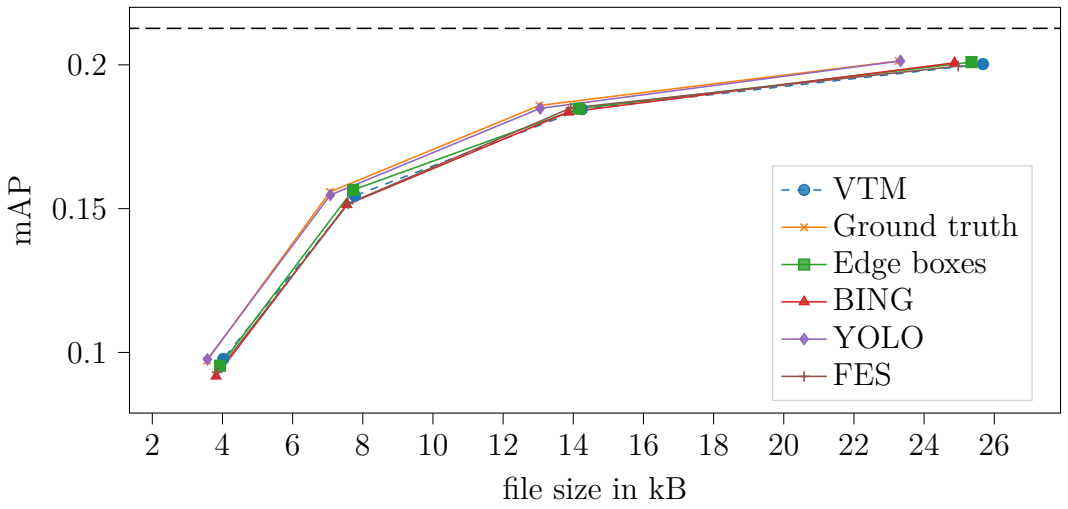
QPs on the mAP is negligible for the models ground truth and YOLO. Edge boxes, BING and FES tend to produce worse results for larger gaps between base QP and background QP.

In order to be able to compare the results of Faster R-CNN and Mask R-CNN, the same QP combinations that have been used for Faster R-CNN as shown in Tab. 5.1 are used for Mask R-CNN in the following. The rate-mAP curves for Mask R-CNN using QPA with the different saliency detectors is depicted in Fig. 5.12 and the corresponding BD values are listed in Tab. 5.6. Ground truth achieves the best results with BD rates of -31.71% for QP 12 – 27 and -11.34% for QP 32–47. The maximum improvement of the mAP +0.65% is also achieved by ground truth for QP 32 – 47. The models YOLO and Edge boxes also allow for a reduction of the file size with small improvements of the mAP over the whole range of QPs. The application of FES is only beneficial for low QPs. For high QPs, it performs worse than the unmodified VTM. BING performs worse than the unmodified VTM on average for both low and high QPs.

When comparing the results of Mask R-CNN with the ones of Faster R-CNN in Tab. 5.2, one can observe that the performance of the model ground truth gets better for Mask R-CNN. In the case of low QPs, for instance, the BD rate is more than doubled from -15.37% to -31.71% for ground truth. A reason for this is that compression artifacts have a higher influence on the performance of Mask R-CNN in comparison to Faster R-CNN due to the pixel-to-pixel behavior. YOLO performs worse for low QPs. The main reason for this is the outlier at QP = 17 that has a negative influence on the average BD rate. The BD rate of Edge boxes also improves significantly for Mask R-CNN from -1.04% to -7.71% in the case of low QPs. FES has a better performance as well for low QPs in terms of BD rate. The application of BING is neither useful for Faster R-CNN nor for Mask R-CNN. A better result may be achieved by increasing the number of maximum bounding boxes which consequently leads to an improved CTU recall.



(a) QP 12 – 27



(b) QP 32 – 47

Figure 5.12: Rate-mAP curves for Mask R-CNN using the different saliency detectors with QPA

Table 5.6: BD values for Mask R-CNN using the different saliency detectors with QPA

	BD rate (QP 12 – 27)	BD mAP (QP 12 – 27)	BD rate (QP 32 – 47)	BD mAP (QP 32 – 47)
Ground truth	-31.71%	+0.12%	-11.34%	+0.65%
Edge boxes	-7.71%	+0.01%	-2.82%	+0.13%
BING	+5.96%	-0.02%	+0.78%	-0.04%
YOLO	-8.51%	+0.04%	-9.91%	+0.57%
FES	-1.01%	-0.03%	+0.33%	+0.00%

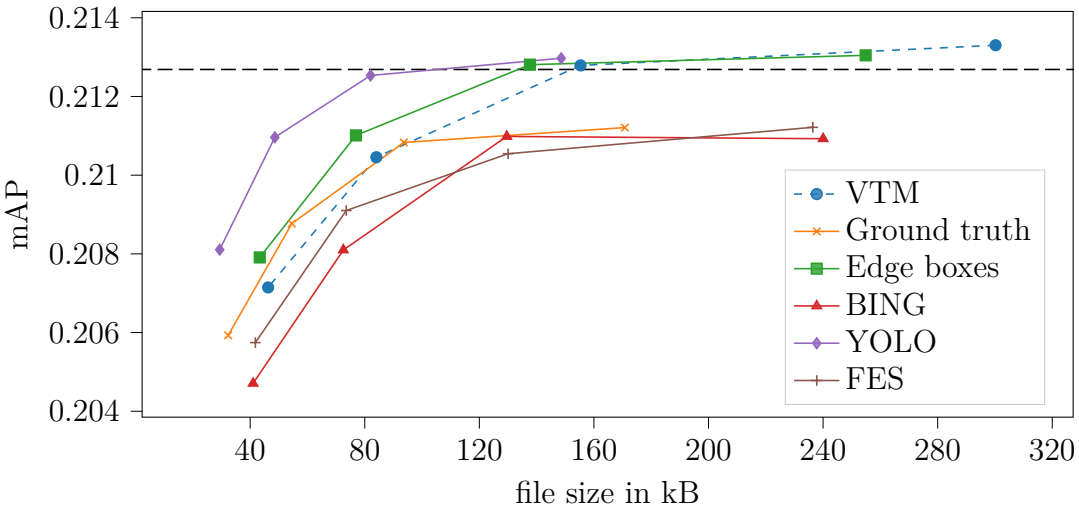
5.3.3 Filtering

The rate-mAP curves and BD values for the averaging filter-based approach for the different saliency models with $N = 5 \times 5$ are depicted in Fig. 5.13 and Tab. 5.7, respectively. The filtering operations were conducted in the same way as it has been described in Section 5.2.3.

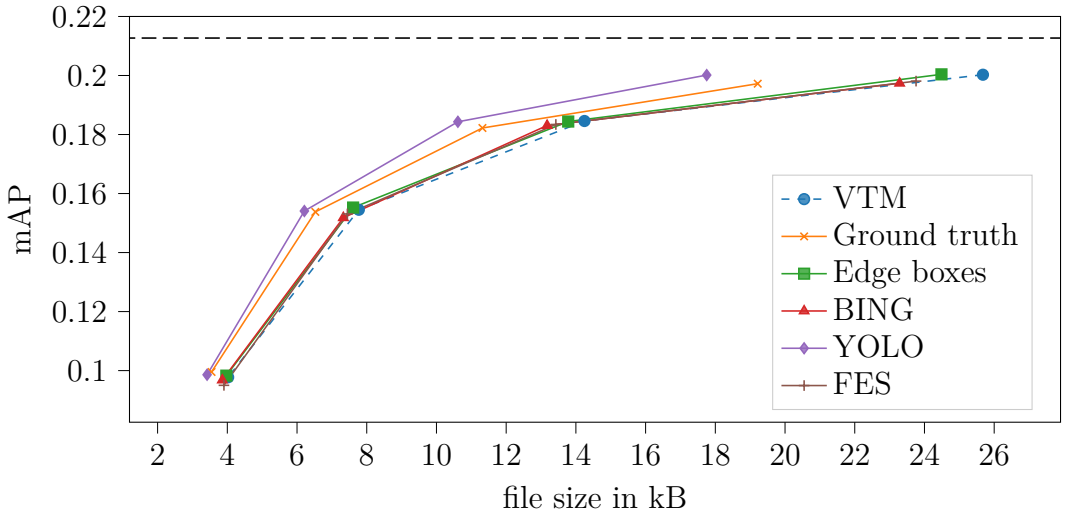
YOLO achieves the best results for both low and high QPs. Ground truth also allows for a rate reduction on average. However, the uncompressed level cannot be reached by ground truth. The second best result for low QPs is achieved by Edge boxes with a BD rate of -17.44%. For high QPs, Edge boxes perform worse than YOLO and ground truth but they are still the best among the saliency detectors that run on a CPU only. The application of BING and FES is not useful for low QPs, whereas for high QPs average rate savings of around -2% can be achieved.

When comparing the results with the ones for Faster R-CNN in Tab. 5.4, one can observe that the results for Mask R-CNN are better on average again, as it is also the case for QPA. Especially for low QPs there is mainly an improvement of the BD rates in comparison to Faster R-CNN. This can be attributed again to the fact that compression artifacts influence the performance of Mask R-CNN more badly than the performance of Faster R-CNN.

The results for the filter size 15×15 using Mask R-CNN as the final detector have also been simulated. The rate mAP curves are depicted in Fig. 5.14 and the corresponding BD values are listed in Tab. 5.8. It can be observed that for low QPs the larger filter size has a negative impact on the performance, so for low QPs the smaller filter size 5×5 should be chosen in practice. For high QPs, the performance of ground truth and YOLO can be improved further in comparison to $N = 5 \times 5$. The BD rate of YOLO, for instance, is improved significantly from -20.2% for $N = 5 \times 5$ to -33.73% for $N = 15 \times 15$. The BD mAP is also improved by more than 1 percentage point for ground truth and YOLO in comparison to $N = 5 \times 5$ for QP 32 – 47.



(a) QP 12 – 47

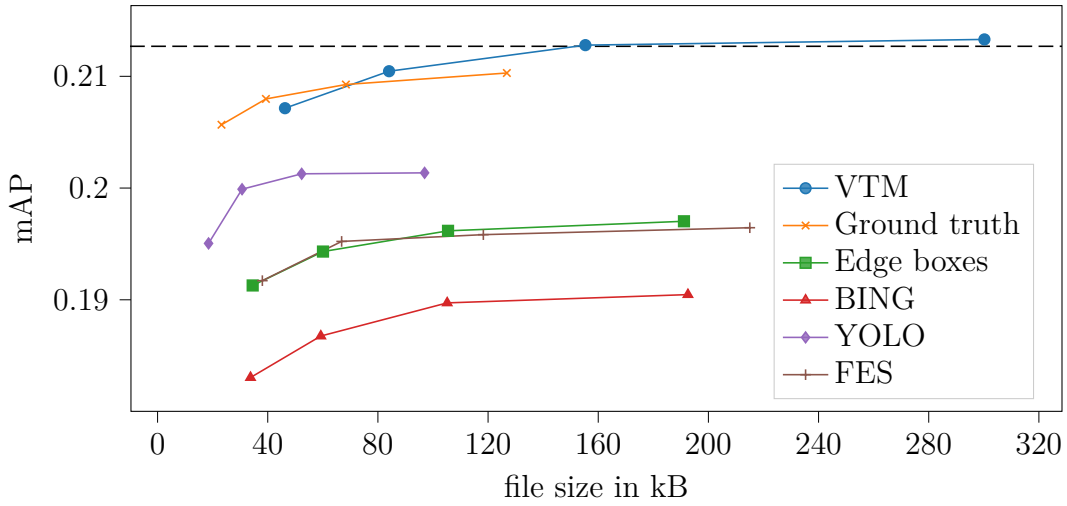


(b) QP 12 – 47

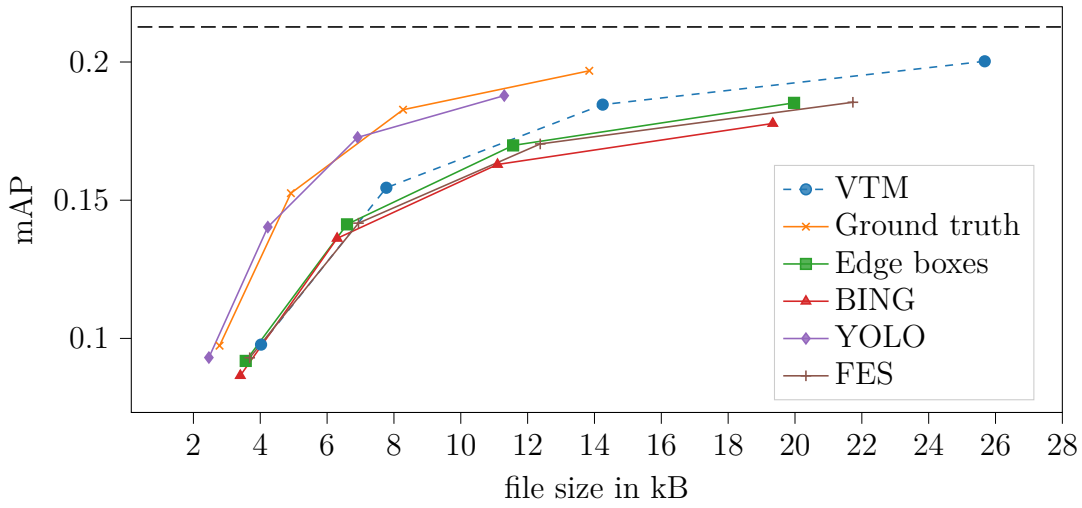
Figure 5.13: Rate-mAP curves for Mask R-CNN using the different saliency detectors and an averaging filter of size 5×5

Table 5.7: BD values for Mask R-CNN using the different saliency detectors and an averaging filter of size 5×5

	BD rate (QP 12 – 27)	BD mAP (QP 12 – 27)	BD rate (QP 32 – 47)	BD mAP (QP 32 – 47)
Ground truth	-6.51%	-0.03%	-14.51%	+0.92%
Edge boxes	-17.44%	+0.06%	-2.93%	+0.16%
BING	+29.60%	-0.17%	-2.05%	+0.13%
YOLO	-45.64%	+0.20%	-20.20%	+1.37%
FES	+25.88%	-0.13%	-2.14%	+0.10%



(a) QP 12 – 47



(b) QP 12 – 47

Figure 5.14: Rate-mAP curves for Mask R-CNN using the different saliency detectors and an averaging filter of size 15×15

Table 5.8: BD values for Mask R-CNN using the different saliency detectors and an averaging filter of size 15×15

	BD rate (QP 12 – 27)	BD mAP (QP 12 – 27)	BD rate (QP 32 – 47)	BD mAP (QP 32 – 47)
Ground truth	-7.03%	-0.05%	-34.61%	+2.62%
Edge boxes	-	-1.51%	+5.59%	-0.37%
BING	-	-2.20%	+10.44%	-0.75%
YOLO	-	-0.79%	-33.73%	+2.63%
FES	-	-1.54%	+10.35%	-0.62%

Chapter 6

Conclusion

In this work, a framework that can be used for saliency coding of video data for machine to machine communication has been presented. The first step in the framework is detecting regions in images that are likely to contain objects using algorithms like YOLO, BING, FES or Edge boxes. Afterwards, the information about the salient regions is used for further processing of the images. This processing can be conducted in two ways. The first approach is based on the perceptually optimized bit-allocation for block-based image and video coding used in VTM 6.1 which has actually been developed to match human perception better. In the course of this work, this approach was adapted such that it is useful for the application in computer-vision-based systems. CTUs that are likely to contain objects are coded with the base QP and the remaining CTUs are coded with a higher QP. In the second approach, averaging filters are used for the regions in the background. The area that is detected by the saliency models is not modified. Afterwards, the pre-processed images are coded with the unmodified VTM 6.1. In the next step of the framework, Faster R-CNN and Mask R-CNN are used to detect objects in the coded images. Finally, the performance of the networks is evaluated using mAP. The results show that saliency coding for machine to machine communication can lead to significant reductions of the file sizes while the impact on the mAP is negligible. With the second approach and a filter size of 5×5, the file size can be reduced by up to 30.15% using YOLO as a saliency detector and QP 12 – 27 for Faster R-CNN. Additionally, the mAP is improved by +0.19% for these parameters. For high QPs 32 – 47 and $N = 15 \times 15$, the results get even better with a BD rate of -37.57% and a BD mAP of +3.20% for the saliency model YOLO. When employing a model with lower complexity like Edge boxes, it is still possible to reduce the file size by -6.60% for QP 12 – 27 and -1.82% for QP 32 – 47. For Mask R-CNN, it is possible to achieve rate savings as well. Using the average filter-based approach, the best BD rate that can be achieved is -45.64% with the saliency model YOLO for QP 12 – 27. Additionally, the mAP can be improved by +0.20% for this constellation. When using Edge boxes, the BD rate is still -17.44% for QP 12 – 27 and -2.93% for QP 32–47 with

corresponding BD mAPs of +0.06% and +0.16%, respectively.

BING and FES also allow for rate savings for high QPs using $N = 5 \times 5$. For low QPs, however, the results are significantly worse than the results for Edge boxes, YOLO and ground truth.

The results show that Edge boxes is the best choice when a system with low computational power is used as it runs on a CPU only and has a runtime of 0.25 seconds per image. When a GPU is available, one should choose YOLO as it achieves the best performance by far among the saliency detectors.

Future work may include the improvement of the saliency models such that they achieve a higher CTU recall which leads to higher rate savings, as the results for ground truth show. This could be done, for instance, by tuning the parameters of the corresponding saliency models.

List of Figures

List of Tables

Bibliography

[1] M. Bramberger, A. Doblander, A. Maier, B. Rinner, and H. Schwabach, "Distributed embedded smart cameras for surveillance applications," *Computer*, vol. 39, no. 2, pp. 68–75, Feb. 2006.

[2] J. Nagi, F. Ducatelle, G. A. Di Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella, "Max-pooling convolutional neural networks for vision-based hand gesture recognition," in *Proc. IEEE International Conference on Signal and Image Processing Applications*, Nov. 2011, pp. 342–347.

[3] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer, "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Jul. 2017, pp. 129–137.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. 28th International Conference on Neural Information Processing Systems*, Dec. 2015, pp. 91–99.

[5] G. Garza, "Mask R-CNN for ship detection & segmentation," last accessed: 14.03.2020. [Online]. Available: https://towardsdatascience.com/mask-r-cnn-for-ship-detection-segmentation-a1108b5a083

[6] M.-M. Cheng, Z. Zhang, W.-Y. Lin, and P. Torr, "BING: Binarized normed gradients for objectness estimation at 300fps," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 3286–3293.

[7] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *Proc. European Conference on Computer Vision*, Sep. 2014, pp. 391–405.

[8] H. Rezazadegan Tavakoli, E. Rahtu, and J. Heikkilä, "Fast and efficient saliency detection using sparse sampling and kernel density estimation," in *Proc. 17th Scandinavian Conference on Image Analysis*, May 2011, pp. 666–675.

[9] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv:1804.02767*, Apr. 2018.

[10] J. Chen, Y. Ye, and S. H. Kim, "Algorithm description for versatile video coding and test model 5 (VTM 5)," *JVET document, JVET N-1002*, 2019.

[11] C. R. Helmrich, S. Bosse, M. Siekmann, H. Schwarz, D. Marpe, and T. Wiegand, "Perceptually optimized bit-allocation and associated distortion measure for block-based image or video coding," in *Proc. Data Compression Conference*, Mar. 2019, pp. 172–181.

[12] R. Jain, R. Kasturi, and B. G. Schunck, *Machine vision*. McGraw-Hill, 1995.

[13] Y.-Y. Chen, Y.-H. Lin, C.-C. Kung, M.-H. Chung, and I.-H. Yen, "Design and implementation of cloud analytics-assisted smart power meters considering advanced artificial intelligence as edge analytics in demand-side management for smart homes," *Sensors*, vol. 19, no. 9, p. 2047, May 2019.

[14] M. A. Nielsen, *Neural networks and deep learning*. Determination press, 2015.

[15] A. Arnx, "First neural network for beginners explained (with code)," last accessed: 14.03.2020. [Online]. Available: https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf

[16] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. 27th International Conference on Machine Learning*, Jun. 2010, pp. 807–814.

[17] K. M. Fauske, "Example: Neural network," last accessed: 14.03.2020. [Online]. Available: http://www.texample.net/tikz/examples/neural-network/

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[19] A. Godbole, "Deep learning book: Chapter 9— convolutional networks," last accessed: 14.03.2020. [Online]. Available: https://medium.com/inveterate-learner/deep-learning-book-chapter-9-convolutional-networks-45e43bfc718d

[20] S. Nayak, "Understanding AlexNet," last accessed: 14.03.2020. [Online]. Available: https://www.learnopencv.com/understanding-alexnet/

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. 25th International Conference on Neural Information Processing Systems*, Dec. 2012, pp. 1097–1105.

[22] Vinzza, "Drawing a convolution with tikz," last accessed: 14.03.2020. [Online]. Available: https://tex.stackexchange.com/questions/437007/drawing-a-convolution-with-tikz?rq=1

[23] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, Apr. 2017.

[24] P.-L. Pröve, "An introduction to different types of convolutions in deep learning," last accessed: 14.03.2020. [Online]. Available: https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

[25] "Max-pooling / pooling," last accessed: 14.03.2020. [Online]. Available: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

[26] R. Gandhi, "R-CNN, Fast R-CNN, Faster R-CNN, YOLO — object detection algorithms." [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[27] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 580–587.

[28] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, Sep. 2013.

[29] R. Girshick, "Fast R-CNN," in *Proc. IEEE International Conference on Computer Vision*, Dec. 2015, pp. 1440–1448.

[30] J. Rey, "Faster R-CNN: Down the rabbit hole of modern object detection," last accessed: 14.03.2020. [Online]. Available: https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/

[31] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. European Conference on Computer Vision*, Sep. 2014, pp. 818–833.

[32] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, Sep. 2014.

[33] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE International Conference on Computer Vision*, Oct. 2017, pp. 2961–2969.

[34] B. Chaudhari, "How does ROTAlign work in Mask-RCNN?" last accessed: 14.03.2020. [Online]. Available: https://www.quora.com/How-does-ROTAlign-work-in-Mask-RCNN

[35] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," in *Proc. 28th International Conference on Neural Information Processing Systems*, Dec. 2015, pp. 2017–2025.

[36] A. Rosebrock, "YOLO object detection with OpenCV," last accessed: 14.03.2020. [Online]. Available: https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/

[37] ——, "Intersection over union (IoU) for object detection," last accessed: 14.03.2020. [Online]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[38] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Apr. 2016, pp. 3213–3223.

[39] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[40] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[41] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Proc. European Conference on Computer Vision*, Apr. 2014, pp. 740–755.

[42] M. Wien, *High efficiency video coding*. Springer, 2015.

[43] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[44] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Transactions on Computers*, vol. 100, no. 1, pp. 90–93, Jan. 1974.

[45] W. K. Chen, *The electrical engineering handbook*. Elsevier, 2004.

[46] K. R. Rao and P. Yip, *Discrete cosine transform: algorithms, advantages, applications*. Academic press, 2014.

[47] V. Sze, M. Budagavi, and G. J. Sullivan, *High efficiency video coding (HEVC)*. Springer, 2014.

[48] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H. 264/AVC video compression standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 620–636, Aug. 2003.

[49] C. Fu, E. Alshina, A. Alshin, Y. Huang, C. Chen, C. Tsai, C. Hsu, S. Lei, J. Park, and W. Han, "Sample adaptive offset in the HEVC standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1755–1764, Dec. 2012.

[50] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," VCEG-M33, ITU-T SG16/Q6, Tech. Rep., 2001.

[51] P. Dollár and C. L. Zitnick, "Structured forests for fast edge detection," in *Proc. IEEE International Conference on Computer Vision*, Dec. 2013, pp. 1841–1848.

[52] B. Alexe, T. Deselaers, and V. Ferrari, "Measuring the objectness of image windows," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2189–2202, Nov. 2012.

[53] K. Mikolajczyk and C. Schmid, "Scale & affine invariant interest point detectors," *International Journal of Computer Vision*, vol. 60, no. 1, pp. 63–86, Oct. 2004.

[54] M.-M. Cheng and S. Zheng, "Objectness proposal generator with BING," last accessed: 14.03.2020. [Online]. Available: https://github.com/torrvision/Objectness

[55] S. Hare, A. Saffari, and P. H. S. Torr, "Efficient online structured output learning for keypoint-based object tracking," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2012, pp. 1894–1901.

[56] S. Zheng, P. Sturgess, and P. H. S. Torr, "Approximate structured output learning for constrained local models with application to real-time facial feature detection and tracking on low-power devices," in *Proc. 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition*, Apr. 2013, pp. 1–8.

[57] Y.-C. Chen, "A tutorial on kernel density estimation and recent advances," *Biostatistics & Epidemiology*, vol. 1, no. 1, pp. 161–187, Apr. 2017.

[58] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve." *Radiology*, vol. 143, no. 1, pp. 29–36, May 1982.

[59] A. Valberg, *Light vision color*. John Wiley & Sons, 2005.

[60] Y. Shoham and A. Gersho, "Efficient bit allocation for an arbitrary set of quantizers (speech coding)," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 9, pp. 1445–1453, Sep. 1988.

[61] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 74–90, Nov. 1998.

[62] T. Wiegand and B. Girod, "Lagrange multiplier selection in hybrid video coder control," in *Proc. International Conference on Image Processing*, Oct. 2001, pp. 542–545.

[63] L. Galteri, M. Bertini, L. Seidenari, and A. Del Bimbo, "Video compression for object detection algorithms," in *Proc. 24th International Conference on Pattern Recognition*, Aug. 2018, pp. 3007–3012.

[64] "Edge boxes demo," last accessed: 14.03.2020. [Online]. Available: https://github.com/opencv/opencv_contrib/blob/master/modules/ximgproc/samples/edgeboxes_demo.py

[65] A. Rosebrock, "OpenCV saliency detection," last accessed: 14.03.2020. [Online]. Available: https://www.pyimagesearch.com/2018/07/16/opencv-saliency-detection/

[66] H. Rezazadegan Tavakoli, "FES," last accessed: 14.03.2020. [Online]. Available: https://github.com/hrtavakoli/FES

[67] G. Nishad, "YOLO-object-detection," last accessed: 14.03.2020. [Online]. Available: https://github.com/Garima13a/YOLO-Object-Detection

[68] "Boost," last accessed: 14.03.2020. [Online]. Available: https://www.boost.org/users/history/version_1_71_0.html

[69] S. Akramullah, *Digital video concepts, methods, and metrics: quality, compression, performance, and power trade-off analysis*. Apress, 2014.

[70] "Tensorflow," last accessed: 14.03.2020. [Online]. Available: https://www.tensorflow.org/

[71] "Tensorflow detection model zoo," last accessed: 14.03.2020. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

[72] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2016, pp. 770–778.

[73] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. IEEE Press, 2001.

[74] S. Sahoo, "Residual blocks — building blocks of ResNet," last accessed: 14.03.2020. [Online]. Available: https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec

[75] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, Inception-ResNet and the impact of residual connections on learning," in *Proc. Thirty-first AAAI Conference on Artificial Intelligence*, Feb. 2017, p. 4278–4284.

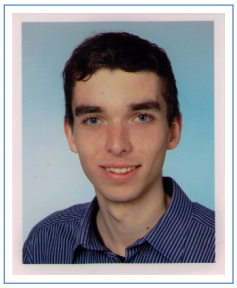
[76] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2015, pp. 1–9.

[77] M. Cordts and M. Omran, "The cityscapes dataset," last accessed: 14.03.2020. [Online]. Available: https://github.com/mcordts/cityscapesScripts

[78] "Image filtering," last accessed: 14.03.2020. [Online]. Available: https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html

Felix Fleckenstein

Obere Schmiedgasse 22
90403 Nürnberg
✉ *felix_fleckenstein@gmx.de*



Personal Information

Birthday	January 9, 1995
Place of birth	Hammelburg

Education

School

2001-2013 **General qualification for university entrance**, *Grundschule (primary school) and Wolfgang-Borchert-Gymnasium (grammar school) Langenzenn.*

University

2013-2017 **Bachelor of Science**, *Friedrich-Alexander University Erlangen-Nürnberg*, Electrical Engineering.
Topic of Bachelor's thesis: "Estimation of Chrominance Sub-Sampling Artifacts Perception for Screen Content Coding Applications"

since 2017 **Master of Science**, *Friedrich-Alexander University Erlangen-Nürnberg*, Electrical Engineering.
Topic of Master's thesis: "Saliency Coding of Video Data for Machine to Machine Communication"

Skills

OS	Windows, Linux	**Office**	LaTeX, Microsoft Office
IT	C/C++, Java, Python	**Other**	Matlab, VHDL, Altium Designer

Work Experience

2017 **Internship**, *Schuster Elektronik GmbH*, Herzogenaurach, 10 weeks.

2017–2019 **Student assistant**, *Chair of Multimedia Communication and Signal Processing*, Friedrich-Alexander University Erlangen-Nürnberg, 18 months.