Günther Kniewasser, BSc

Holistic End-to-End Panoptic Segmentation Network with Interrelations

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Computer Science

submitted to

Graz University of Technology

Supervisor
Dipl.-Ing. Dr.techn. Peter M. Roth
Institute of Computer Graphics and Vision

Advisor
Dipl.-Ing. Alexander Grabner, BSc
Institute of Computer Graphics and Vision

Graz, Austria, May 2020

Abstract

To provide a complete 2D scene segmentation, Panoptic Segmentation (PS) unifies the tasks of semantic and instance segmentation. For this purpose, existing approaches independently address semantic and instance segmentation and merge their outputs in a heuristic fashion. However, this simple fusion has two limitations in practice. First, the system is not optimized for the final objective in an end-to-end manner. Second, the mutual information between the semantic and the instance segmentation tasks is not fully exploited. To overcome these limitations, we present a novel end-to-end trainable architecture that generates a full pixel-wise image labeling with resolved instance information. Additionally, we introduce interrelations between the two subtasks by providing instance segmentation predictions as feature input to our semantic segmentation branch. This inter-task link eases the semantic segmentation task and increases the overall panoptic performance by providing segmentation priors. We evaluate our method on the challenging Cityscapes dataset for semantic understanding of urban street scenes and show significant improvements compared to previous *PS* architectures.

Kurzfassung

Um eine vollständige 2D-Szenensegmentierung zu ermöglichen, vereinigt *PS* die Aufgaben der Semantischen und der Instanz-Segmentierung. Zu diesem Zweck adressieren bestehende Ansätze die Semantische und die Instanz-Segmentierung unabhängig voneinander und führen ihre Ausgaben heuristisch zusammen. Diese einfache Verschmelzung hat jedoch in der Praxis zwei Einschränkungen. Erstens ist das System nicht durchgehend für das Endziel optimiert. Zweitens wird die gegenseitige Information zwischen den Semantischen und den Instanz-Segmentierungsaufgaben nicht vollständig genutzt. Um diese Einschränkungen zu überwinden, stellen wir eine neuartige, durchgehend trainierbare Architektur vor, die eine vollständige pixelweise Bildbeschriftung mit aufgelöster Instanzinformation erzeugt. Zusätzlich führen wir Wechselbeziehungen zwischen den beiden Teilaufgaben ein, indem wir Instanz-Segmentierungsvorhersagen als Feature-Input für unseren Semantischen Segmentierungszweig bereitstellen. Diese Verknüpfung zwischen den beiden Teilaufgaben erleichtert die Semantische Segmentierungsaufgabe und erhöht die gesamte panoptische Leistung durch die Bereitstellung von Segmentierungs-Prioritäten. Wir evaluieren unsere Methode anhand des anspruchsvollen Cityscapes-Datensatzes für das semantische Verständnis städtischer Straßenszenen und zeigen signifikante Verbesserungen im Vergleich zu früheren *PS*-Architekturen.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

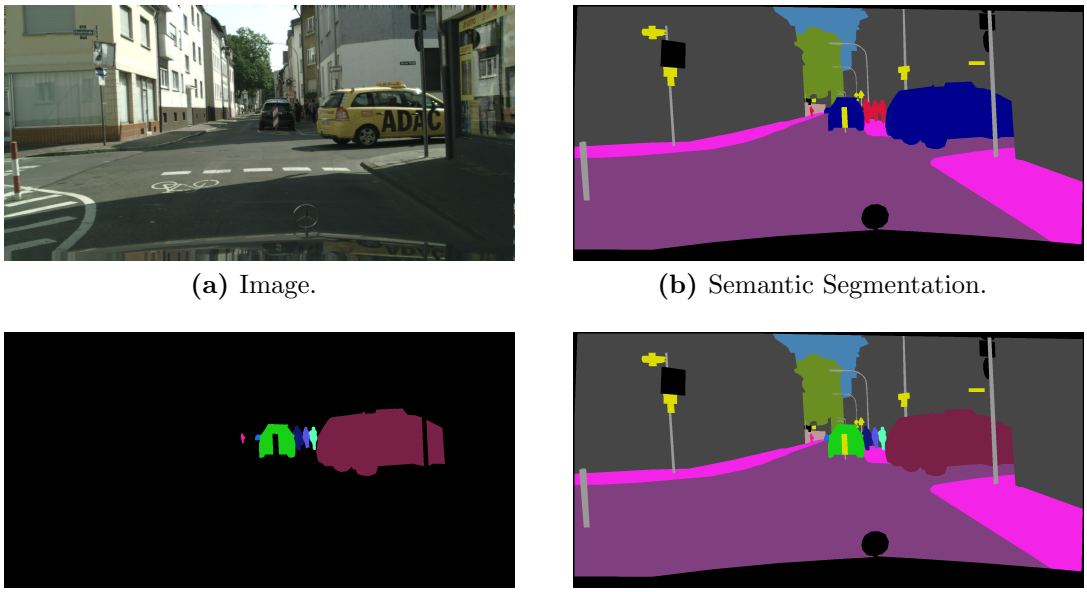
The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date	Signature

Contents

1 Introduction

Panoptic Segmentation (PS) addresses the task of complete 2D scene segmentation [44]. The goal of *PS* is to assign a class label to each pixel of an image while differentiating between instances within a common class, as shown in Figure 1.1. Thus, it can be seen as a unification of semantic segmentation [9, 62, 74] and instance segmentation [31, 45, 54, 58]. *PS* is a new and active research topic with applications in augmented reality, robotics, and medical imaging [21, 69, 96].



(a) Image. **(b)** Semantic Segmentation. **(c)** Instance Segmentation. **(d)** Panoptic Segmentation.

Figure 1.1: The goal of Panoptic Segmentation is to assign a class label to each pixel of an image while differentiating between instances within a common class, e.g., separating individual *cars* and *humans*. To illustrate this, for a given image (a), we show *ground truth* for Semantic Segmentation (b), Instance Segmentation (c), and the unified task of Panoptic Segmentation (d).

To predict a *PS* of an image, recent approaches perform three tasks [43]. First, they perform semantic segmentation to identify regions of uncountable *stuff* classes like *road* or *sky* (see Figure 1.1b). Second, they perform instance segmentation to detect individual instances of countable *things* classes like *cars* or *humans* (see Figure 1.1c). Third, they merge the outputs of these two tasks into a single panoptic prediction (see Figure 1.1d).

However, this strategy has two limitations in practice. First, because the panoptic output is generated using heuristics, the system cannot be optimized for the final objective in an end-to-end manner. Second, because semantic and instance segmentation are addressed independently, mutual information and similarities between the two tasks are not fully exploited.

To overcome these limitations, we propose a *holistic* neural network architecture for *PS* that supports end-to-end training and exploits inter-task relations, as illustrated in Figure 1.2. Our network directly generates the target panoptic output and entangles the semantic and instance segmentation tasks to improve the panoptic performance. In

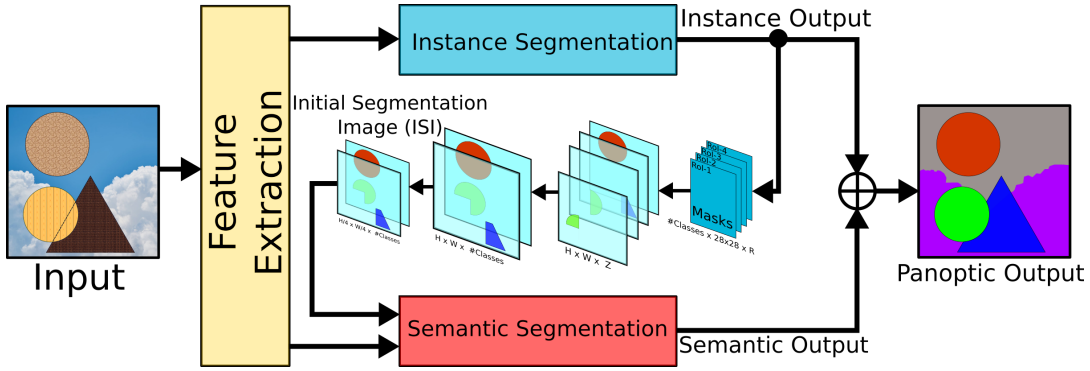


Figure 1.2: Overview of our proposed end-to-end panoptic segmentation network with task interrelations. We perform semantic and instance segmentation on top of a shared feature extraction backbone, provide instance segmentation predictions as additional feature input to our semantic segmentation branch, and merge semantic and instance segmentation predictions using differentiable operations. In this way, we exploit mutual information between the tasks and support end-to-end training.

particular, we perform semantic and instance segmentation on top of a shared feature extraction backbone. To take advantage of mutual information between the two tasks, we provide instance segmentation predictions as additional feature input to our semantic segmentation branch. For this purpose, we gather predicted instance masks into an Initial Segmentation Image (ISI) which represents a coarse semantic segmentation for *things* classes. In this way, we exploit a segmentation prior which eases the semantic segmentation by entangling the two previously disjoint subtasks.

Finally, we use differentiable operations instead of heuristics to combine individual semantic and instance segmentation results and to generate a full pixel-wise image labeling with resolved instance information internally. As a result, our entire system can be trained in an end-to-end manner using a panoptic objective.

The benefits of our approach are threefold. First, using a single network for *PS* reduces the model size and the training time compared to two disjoint semantic and instance segmentation networks. Second, our inter-task link between semantic and instance segmentation increases the panoptic performance, because it avoids the computation of redundant information in the two subtask branches. Third, end-to-end training using a panoptic objective increases the overall system accuracy and removes the necessity to manually select thresholds for a heuristic combination of results.

To demonstrate the benefits of our approach, we evaluate it on a synthetically generated TOY dataset consisting of geometric shapes and on the challenging Cityscapes dataset [14] for semantic understanding of urban street scenes. We benchmark our approach against different methods using the common evaluation metrics Intersection Over Union (IoU), Mean Average Precision (mAP), and Panoptic Quality (PQ) to asses the semantic, instance, and panoptic segmentation performance respectively. Our experiments show that both end-to-end training and inter-task relations improve the panoptic performance in practice. This quantitative improvement is also reflected in qualitative examples, where our method creates more accurate label transitions between individual classes while being less sensitive to speckle noise in semantically coherent regions.

The rest of this work is structured as follows. In Chapter 2, we provide preliminaries that are required to understand the contribution of this thesis and discuss related work. In Chapter 3, we present our *holistic* neural network architecture for *PS* with inter-task relations. In Chapter 4, we conduct a detailed experimental evaluation of our approach. Finally, we summarize our results and draw conclusions in Chapter 5.

2

Preliminaries and Related Work

Contents

In this chapter, we provide preliminaries that are required to understand the contribution of this thesis and discuss related work. We start with an introduction to deep learning in Section 2.1, which became a fundamental building block in computer vision. We present different neural network architectures (Section 2.1.2), activation functions (Section 2.1.5), optimization methods (Section 2.1.3), and loss functions (Section 2.1.4) that are used in our approach. Finally, in Section 2.2, we give an overview of how object recognition systems evolved with deep learning.

2.1 Deep Learning

In recent years, deep learning has significantly improved the state of the art in disciplines like speech recognition [64] and image processing [33]. Especially in the field of computer vision, deep learning systems outperform complex hand-crafted processing pipelines because they learn features that are relevant for the task from raw data. In contrast to engineered models, deep models learn a hierarchy of abstract features from possibly millions of matching input and output data pairs. While deep learning techniques have been known for decades [26, 48], the availability of large supervised datasets and advances in computational hardware made them successful in practice in recent years. In the next sections, we describe the building blocks of deep Artificial Neural Networks (ANNs).

2.1.1 Artificial Neurons

Artificial neurons are the fundamental structures of an *ANN*. They portray a mathematical model that is loosely inspired by biological neurons and synapses in the human brain. An artificial neuron, as shown in Figure 2.1, computes a weighted sum between a set of inputs $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ with corresponding weights $\mathbf{w} = \{w_1, w_2, \ldots, w_n\}$. This weighted sum and the added bias b yield a linear transformation z. To solve more complex problems a non-linearity is introduced into the equation by applying an activation function $\sigma(\cdot)$ to z. Popular activation functions are monotonically increasing, continuous and piecewise differentiable. They range from piecewise linear to sigmoid functions and are described in more detail in Section 2.1.5.

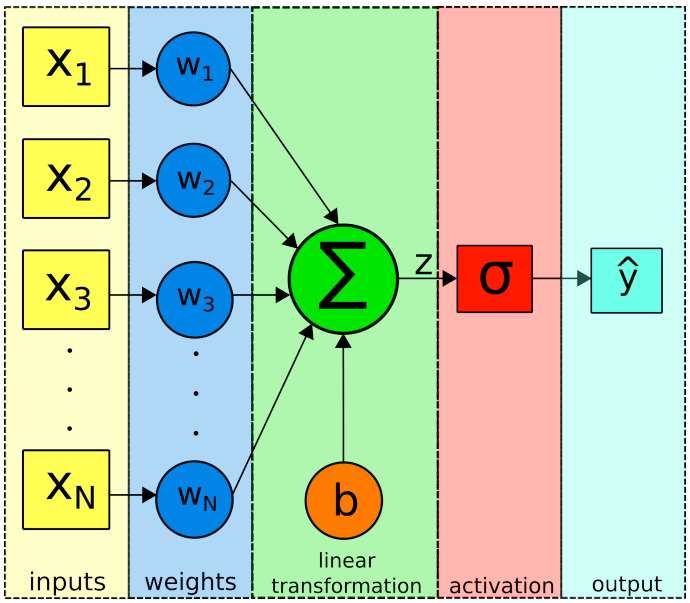


Figure 2.1: Illustration of an artificial neuron. A linear transformation z is computed as a weighted sum of inputs $\{x_1, \ldots, x_N\}$ and weights $\{w_1, \ldots, w_N\}$ and the addition of the bias b. The activation function $\sigma(\cdot)$ calculates the non-linear output $\hat{y}$ from z.

Formally, an artificial neuron computes an output $\hat{y}$ as

$$\hat{y} = \sigma\left(\sum_{i}^{N} w_i \cdot x_i + b\right) = \sigma(z) \ . \tag{2.1}$$

Eq (2.1) can also be rewritten as a dot product between the weight vector $\mathbf{w}$ and the input vector $\mathbf{x}$:

$$\hat{y} = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right) \ . \tag{2.2}$$

The earliest known artificial neuron is the *McCulloch-Pitts-Neuron* first proposed in 1943 by McCulloch and Pitts in an attempt to find a simple model for nervous activity in the brain [65]. This neuron model is also known as a Threshold Logic Unit (TLU), since it

aggregates all inputs and depending on a defined threshold has an output of one or zero. This neuron is able to learn simple boolean algebra [65]. In 1958, Rosenblatt adapted this concept and changed the world of machine learning and artificial intelligence with the idea of perceptrons [75]. A perceptron is a single neuron with a step activation function:

$$\hat{y} = \sigma(\mathbf{x};\theta) = \begin{cases} 1 & \mathbf{w}^T\mathbf{x} + b > 0 \\ 0 & \text{else} \end{cases}, \tag{2.3}$$

where $\theta = \{\mathbf{w}, b\}$ is the set of learnable parameters and $\mathbf{x}$ is the input vector. The parameters θ are optimized using one of the earliest supervised learning algorithms based on the Hebbian theory of neuroscience [34]. The parameters are incrementally adjusted based on the difference between the target value and the neuron prediction outcome $\sigma(\mathbf{x};\theta)$. The perceptron is used as a binary classifier with a guaranteed convergence for linearly separable data. Nowadays, the parameter updates are based on gradient techniques which are described in more detail in Section 2.1.3.1.

2.1.2 Artificial Neural Networks

Although a single neuron can be used for many tasks, it has limited expressive power. For example, a single neuron cannot predict multidimensional outputs nor implement a hierarchy of non-linear transformations. *ANNs* overcome these limitations by forming a structured network of interconnected artificial neurons.

ANNs are one of the main concepts in machine learning. As the name suggests, *ANNs* are mathematical models that are inspired by the complex network structures in the human brain. *ANNs* have input neurons, hidden neurons, and output neurons. In many cases, these neurons are organized in layers. Figure 2.2 illustrates the structure of a simple *ANN* with an input layer, a hidden layer, and an output layer. If each neuron of a layer is connected to all neurons of the previous layer, the layers are referred to as fully-connected (see Figure 2.2).

An *ANN* models a function $\hat{\mathbf{y}} = f(\mathbf{x};\theta)$ for a given input $\mathbf{x}$ and a set of parameters θ. In contrast to a single artificial neurons, the output $\hat{\mathbf{y}}$ can be multidimensional, the parameters θ comprise weights and biases of all artificial neurons of the *ANN*, and the function $f(\cdot)$ can be arbitrarily complex. The universal approximation theorem states that it is possible to approximate any continuous function by a network with a single hidden layer with a finite number of artificial neurons up to a desired precision [15, 30, 36]. Although it is in theory possible to approximate any function with a single hidden layer, a large number of neurons in the hidden layer is required in practice. However, research shows that *ANNs* with multiple hidden layers empirically requires a smaller number of neurons to approximate the same function and often generalize better to previously unseen data [33, 78]. The structure of an *ANN*, which is also referred to as architecture, can

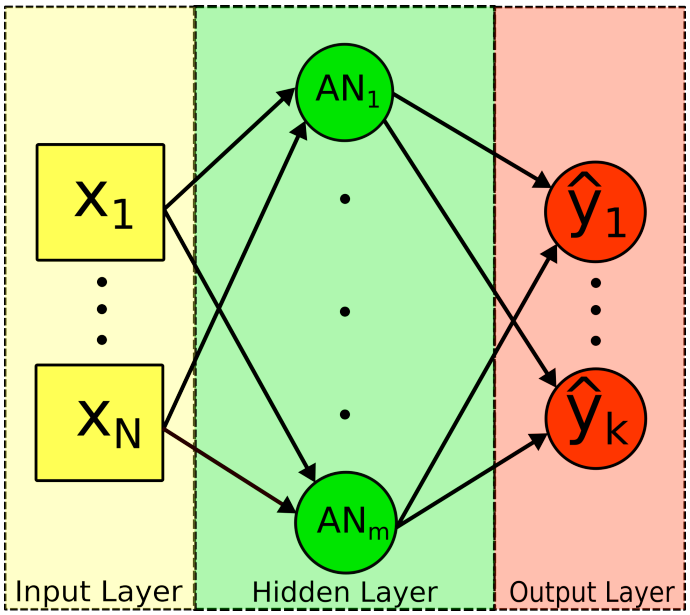


Figure 2.2: Illustration of a simple *ANN*. The depicted network has an input layer with n neurons, a hidden layer with m neurons, and an output layer with k neurons. Its layers are fully-connected in a feed-forward manner. In practice, the structure of the network and the number of neurons are chosen depending on the task.

be designed arbitrarily depending on the task. In practice, many *ANNs* show common architecture patterns that can be used to categorize them. For example, many *ANNs* are Feed-forward Neural Networks (FNNs) [95]. *FNNs* do not have recurrent connections nor form any cycles (see Figure 2.2). Their computational graph is directed and acyclic. This structure eases the computation gradients during training [76] which makes them a popular choice for many computer vision tasks [26].

In contrast to *FNNs*, Recurrent Neural Networks (RNNs) [42, 67] form cycle connections and have recurrent states. This principle of loops in the computational graph introduce a memory effect that allows *RNNs* to process inputs with variable length [35]. This structure is important for applications that process input sequences like speech recognition [64], text-to-speech [11], or video analysis [19]. However, in this work, we focus on *FNNs* because we are dealing with single image inputs. Next, we describe certain types of *FNNs* used in our approach.

2.1.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [49] are a special type of *ANNs* that employ layers with a sparse connection pattern which implement a convolution operation. Opposed to fully-connected layers, which lose spatial input information by computing a global weighted sum of all inputs, convolutional layers compute a local weighted sum only for neighboring inputs. This connection pattern is inspired by the organization of receptors in the first

stages of the human visual cortex [38], where only a few specific neurons respond to changes within a local region called the receptive field.

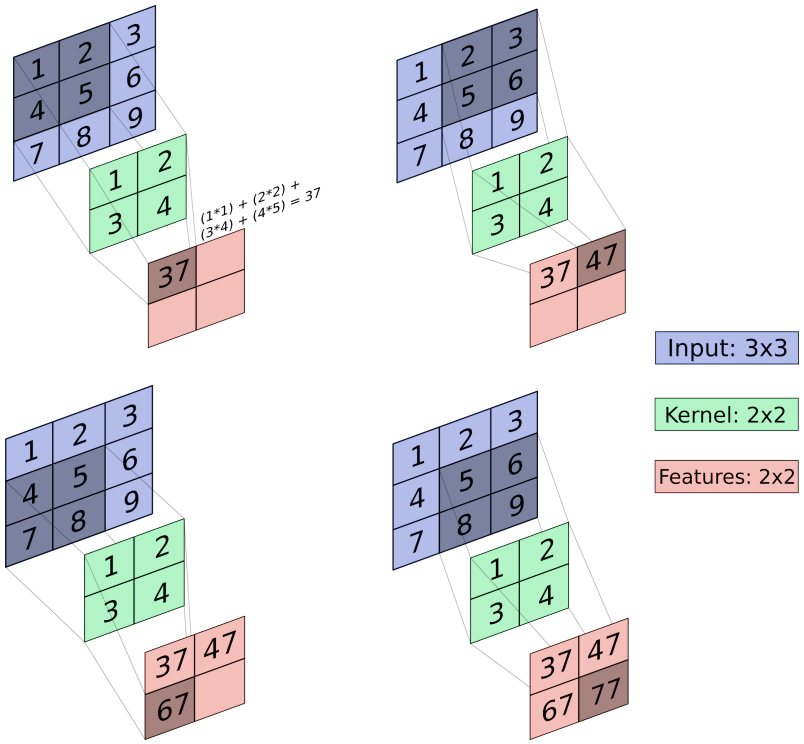


Figure 2.3: Visual example of a convolutional layer applying a 2×2 kernel to a 3×3 input.

Formally, a convolutional layer performs a filter operation

$$F_{l+1}(i,j) = \sigma\Big(\sum_{m=1}^{M}\sum_{n=1}^{N} F_l(i - \frac{M}{2} + m, j - \frac{N}{2} + n)K(m,n)\Big)\,, \tag{2.4}$$

where the feature map F_{l+1} is computed by sliding the kernel K over the input map F_l and computing the weighted sum for each position (i,j). In this case, K is a filter kernel with size M and N and σ is an activation function. A visual example for a convolution layer applying a 2×2 kernel to a 3×3 input is shown in Figure 2.3. A convolution layer has several hyperparameters:

Kernel size: The kernel size determines the number of learnable parameters, the receptive field, and the output feature dimensionality. For two-dimensional input data like images, kernels are commonly declared using a square and odd $n \times n$ window, such as 3×3 or 7×7, and an input and output feature dimensionality N_{in} respectively N_{out}, such as 64 or 128. The total number of learnable parameters is given by $N_{\text{in}} \cdot n \cdot n \cdot N_{\text{out}}$.

Kernel Stride: The kernel stride describes how the filter moves across the input. The stride affects on the spatial dimensions of the output feature map. Larger strides result in feature maps with a smaller spatial resolution, because the filter is applied at fewer positions. An example image with a stride of 2 is shown in Figure 2.4. In practice, a stride of 1 is used most commonly.

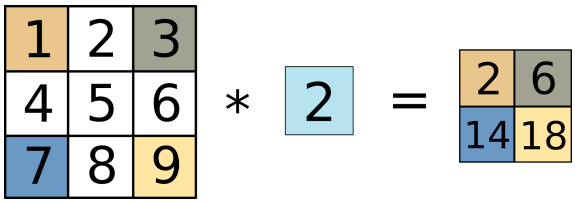


Figure 2.4: Example of a 1×1 kernel with stride 2. Strides larger than 1 can be used to reduce the spatial input size.

Padding: Padding controls the border handling of the convolution. Due to the filter operation over the entire image, edge cases at the spatial input border need to be addressed. A common solution to this problem is to pad the input with zeros around the border. This is referred to as zero-padding which is illustrated in Figure 2.5. In practice, zero-padding is often performed in a way that the kernel can be applied at every input position and the spatial input and output size are equal.

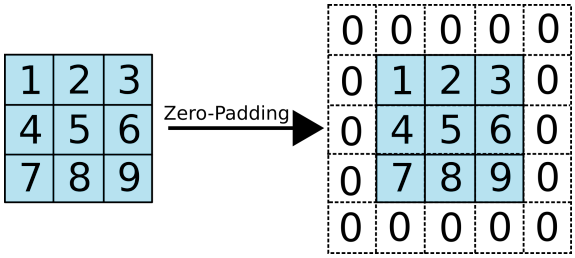


Figure 2.5: Example of zero-padding. The input is artificially augmented by adding zeros around the spatial border.

Due to the sparse local connection pattern, the filter operation of a convolutional layer can be computed efficiently even for inputs with large spatial dimensions. Because only a local region in the input contributes to the output at each position (see Figure 2.3), the computation for a single output position is only dependent on the spatial kernel size and not on the spatial input size. Besides, the computations for individual positions can be parallelized to increase the computation speed.

Another key concept of *CNNs* is weight sharing. In contrast to fully-connected layers that have different weights for each neuron, convolutional layers share the same kernel weights for all neurons. This significantly decreases the number of weights and increases generalization. Also, convolutional layers can be applied to inputs with arbitrary spatial

dimensions which is not possible for fully-connected layers. In addition, the features computed by a convolutional layer are translation equivariant. This means that if the input spatially shifts in a certain direction, the corresponding features also spatially shift by the same amount. Thus, convolutional layers are important for location-aware feature extraction [78] which enables applications like object detection [24].

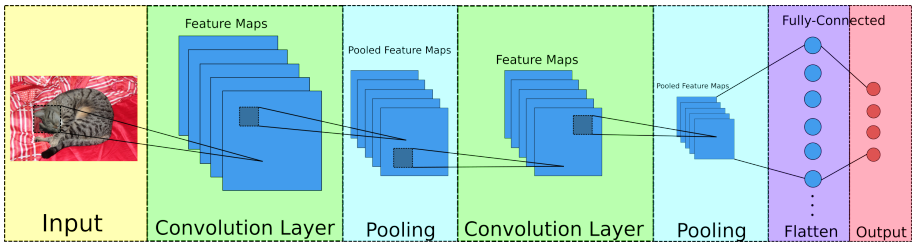


Figure 2.6: Illustration of a simple *CNN*. Kernels are convolved over the input image to produce feature maps. To increase the receptive field and reduce memory consumption, pooling is performed. Stacking multiple convolutional layer on top each other efficiently learn a hierarchy of features. Finally, the computed feature maps are flattened and processed by a fully-connected layer before predicting the output.

By stacking multiple convolutional layers on top of each other, *CNNs* efficiently learn a hierarchy of features. An example of a simple *CNN* is shown in Figure 2.6. While early layers extract simple local features like edges in images, later and deeper layers extract more complex features with larger spatial extend like faces in images [94]. Consecutive convolutional layers increase the respective field of computed features while having a small number of parameters and being computationally efficient. To further increase the effectiveness of kernels and compute features with larger receptive field without increasing the number of convolutional layers or the kernel size and therefore, the number of weights, *CNNs* often employ pooling layers. Pooling reduces the spatial size of feature maps using the max or average operation. This increases the receptive field and reduces the memory consumption at the expense of an information loss. Empirically, the benefit of features with a larger receptive field sometimes outweighs the information loss resulting from the pooling compression in terms of final performance [78].

CNNs have established themselves as the primary tool for image processing and pattern recognition due to their ability to compactly approximate complex functions from data pairs. In recent years, *CNNs* significantly improved the state of the art in tasks like image classification [33], super-resolution [17], image denoising [97], optical character recognition [93], optical flow [79], and 3D pose estimation [28].

2.1.2.2 Fully Convolutional Network

A *FCN* [62] is a special type of *CNN* which only employs convolutional layers, as can be seen in Figure 2.7. In addition to standard convolutional layers, *FCNs* often employ deconvolution layers [62] also known as transpose convolutional layers. These layers are

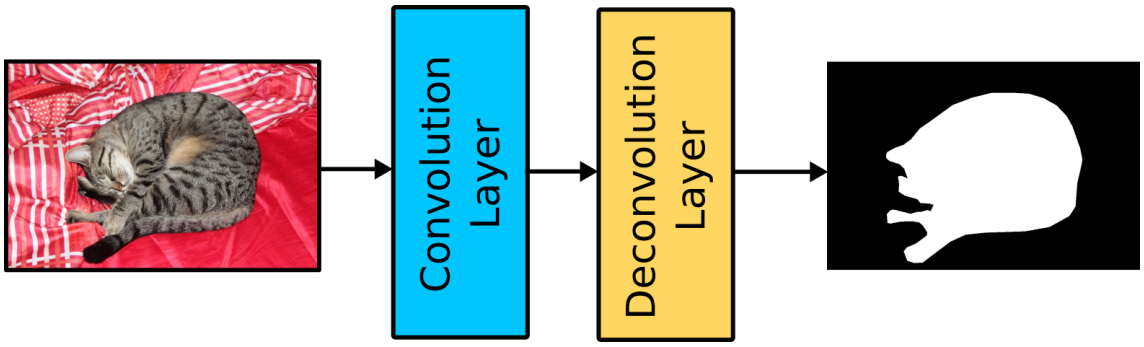


Figure 2.7: Illustration of a simple Fully Convolutional Network (FCN). The depicted network only employs layers that perform convolutions and deconvolutions. In this way, the *FCN* is able to make predictions for arbitrarily sized input images.

commonly used for upsampling, since they are more flexible than traditional interpolation methods because they provide learnable weights. Transpose convolutions also slide a kernel over their input but aim at decompressing data. An example of a transposed convolution is illustrated in Figure 2.8.

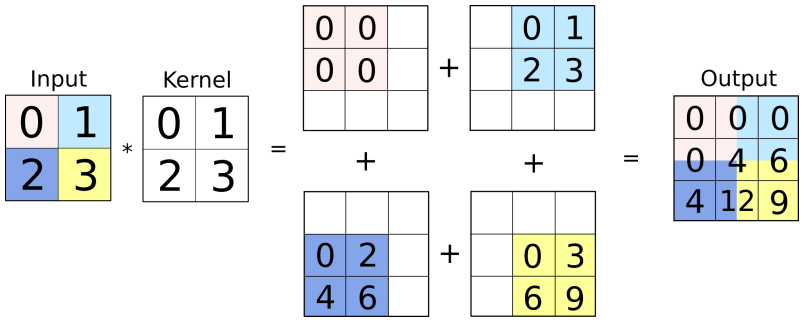


Figure 2.8: Example of a transpose convolution with a 2×2 input and a 2×2 kernel. The upsampling yields a 3×3 output. Transpose convolutions increase the spatial output size by sliding learnable kernels over the input.

Because *FCNs* only employ layers that perform convolutions and deconvolutions they do not require a specific spatial input resolution. As long as the input dimensionality is correct, the spatial dimension can be arbitrary. Thus, *FCNs* are a popular choice for tasks that require dense outputs at the same spatial resolution as the input image like semantic segmentation [74], super-resolution [17], and depth estimation [47].

2.1.2.3 Region-based Convolutional Neural Network

Region-based Convolutional Neural Networks (R-CNNs) are a recently introduced extension to *CNNs* that make multiple predictions for sub-regions of the input instead of a single prediction for the entire input. In this way, *R-CNNs* are able to deal with multiple objects in a single image, for example. This is not only important for object detection [24]

but also for other tasks that implicitly require object detection to perform higher-level tasks such as 3D pose estimation [28] or 3D reconstruction [25].

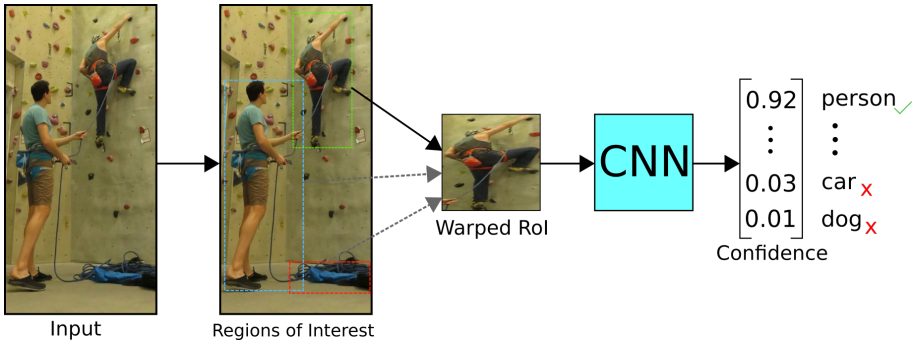


Figure 2.9: Example of a Region-Based CNN: Individual regions of interest are fed into a CNN to classify the region.

A simple example for an *R-CNN* which first finds Regions of Interest (RoIs) and then uses a *CNN* to classify the individual regions can be seen in Figure 2.9.

2.1.2.4 Multi-task Neural Network (MNN)

Multi-task Neural Networks (MNNs) use multiple independent network branches to perform independent predictions that possibly contribute to the same output. Each sub-network shares the same input but performs unique tasks without interaction between modules. Multi-task learning is a common approach to reduce the number of network parameters, increase training speed, and improve generalization. An illustration of such a *MNN* can be seen in Figure 2.10.

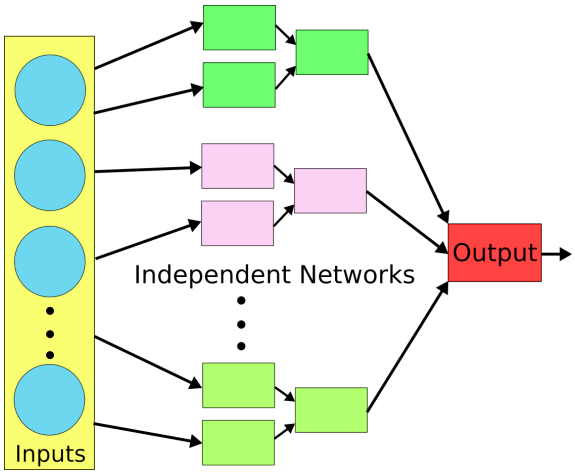


Figure 2.10: Example of a Multi-task Neural Network: Independent sub-networks with a shared input perform different tasks that contribute to a joint output.

2.1.3 Optimization Methods

In this section, we discuss optimization methods that are used to train *ANNs*. Optimization describes the process of finding a solution with respect to a certain objective. In the context of supervised deep learning, optimization is used to minimize the error between a prediction $\hat{\mathbf{y}} = f(\mathbf{x};\theta)$ made by an *ANN* and the ground truth $\mathbf{y}$. In order to measure the error an objective function is defined, which is often referred to as loss function $L(\mathbf{y},\hat{\mathbf{y}})$ [26]. The loss function quantizes the error between a prediction $\hat{\mathbf{y}}$ and the ground truth $\mathbf{y}$. Thus, in terms of training *ANNs*, the goal of optimization is to find a set of parameters θ which minimize a defined loss function given data pairs consisting of an input $\mathbf{x}$ and its associated ground truth output $\mathbf{y}$.

2.1.3.1 Backpropagation

ANNs implement highly non-linear and non-convex [26] functions. As a consequence, there is no closed-form solution to compute optimal parameters θ for a certain loss function. However, backpropagation [76] makes it possible to compute the gradient of a loss function $L(\mathbf{y},\hat{\mathbf{y}})$ with respect to the parameters θ for given data pairs $\{\mathbf{x},\mathbf{y}\}$. By adjusting the parameters θ in the direction of the negative gradient, the loss function can be minimized in an iterative way.

In particular, backpropagation [76] refers to the process of computing gradients for all parameters θ. Backpropagation makes use of the chain rule to calculate the gradient for each parameter. The algorithm uses dynamic programming to efficiently compute gradients and to avoid redundant calculations of intermediate terms.

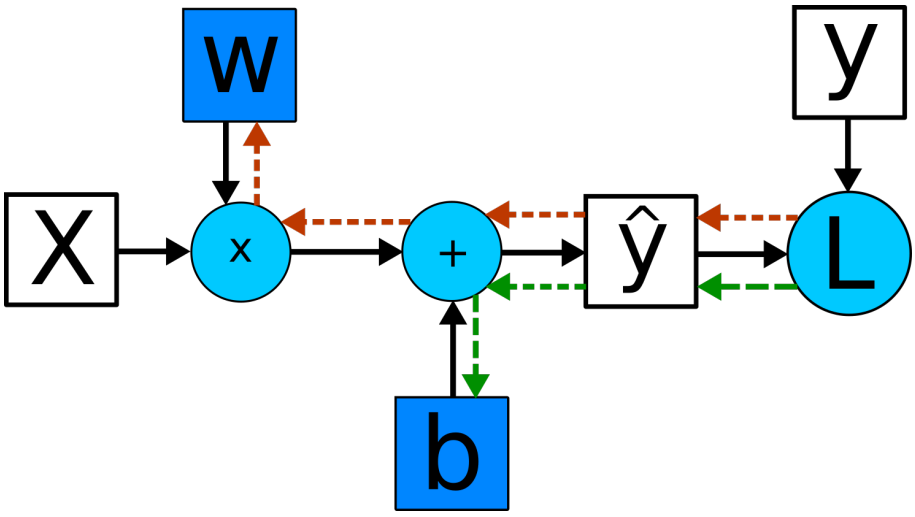


Figure 2.11: Example of backpropagation for a simple *ANN*. The dashed arrows show the gradient flow of backpropagation to compute gradients for w (orange line) and b (green line) starting from the loss L.

Figure 2.11 shows an example of backpropagation for a simple *ANN*. The computational graph implements a one-dimensional linear regression. Gradients for the individual parameters are computed as

$$\begin{aligned} \frac{\partial L}{\partial w} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w} = \delta \cdot \frac{\partial \hat{y}}{\partial w} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b} = \delta \cdot \frac{\partial \hat{y}}{\partial b}, \end{aligned} \tag{2.5}$$

where $\delta = \frac{\partial L}{\partial \hat{y}}$ describes the shared intermediate term that is only computed once thanks to dynamic programming.

2.1.3.2 Gradient Descent

Backpropagation only computes gradients for the parameters θ. In order to actually optimize the parameters, an iterative optimization technique is applied. Gradient descent is an iterative gradient-based approach to minimize a loss function

$$L\left(\mathbf{y}, f(\mathbf{x}; \theta^{k+1})\right) \leq L\left(\mathbf{y}, f(\mathbf{x}; \theta^{k})\right) \quad \forall k > 0 \tag{2.6}$$

such that the next set of parameters θ^{k+1} yields a smaller loss than the current set of parameters θ^k. Gradient descent minimizes a function by following the curve from a random starting point towards the steepest descent. Unless the function is *convex* it is not guaranteed that the global minimum is attained. The mathematical formulation of gradient descent is

$$\theta^{k+1} = \theta^k - \eta \cdot \nabla_\theta L\left(\mathbf{y}, f(\mathbf{x}; \theta^k)\right), \tag{2.7}$$

where η is the step size also called learning rate and $\nabla_\theta L(\cdot)$ is the gradient of the loss function with respect to θ. To compute gradient directions given an entire dataset instead of a single sample, the gradients of individual samples are averaged.

2.1.3.3 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a variation of gradient descent presented above. The main difference to gradient descent is that the gradient calculated in each iteration is not computed from the entire dataset but only from a randomly selected subset of samples. In gradient descent all samples are considered to compute a gradient. This is computationally expensive if the number of samples is large. Thus, *SGD* only considers a randomly selected subset of all training samples. This subset is called a minibatch and can be as small as a single sample. Typically, samples are randomly selected without replacing them in the selection pool to make sure all samples are seen during training.

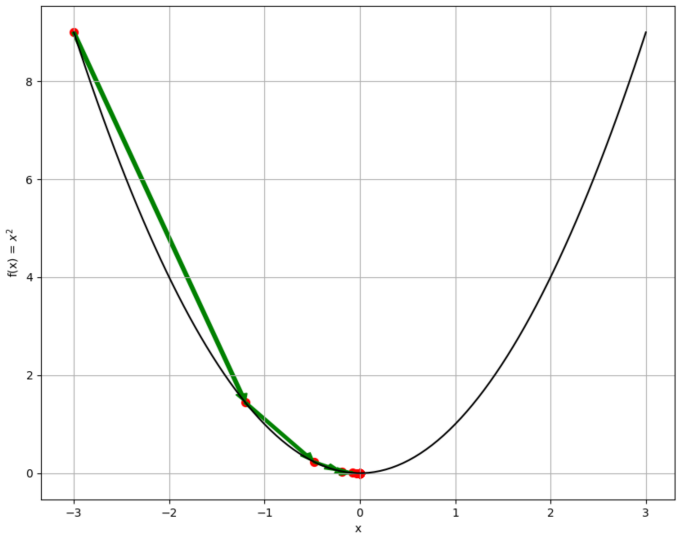


Figure 2.12: Example of gradient descent on the function $\mathbf{f(x) = x^2}$. We choose a random starting point at $\mathbf{x = -3}$ and a learning rate of $\eta = \mathbf{0.3}$. The function moves towards the global optimum at $x = 0$.

SGD has three major advantages. First, the computation of a single update is efficient. Second, stochastic updates avoid getting stuck in local optimization minima. Finally, *SGD* reduces overfitting and increases generalization compared to gradient descent.

2.1.3.4 Stochastic Gradient Descent with Momentum

SGD with momentum has similarities to momentum in physics. A problem that arises in using *SGD* is the oscillation towards convergence, which happens because of the variance in each minibatch. To overcome this limitation, a momentum term β is added that results in an exponentially weighted average of the gradients. This forces *SGD* to consider past gradients and, therefore, smooths the update directions. In each iteration, a new update direction V_{new} is computed as

$$V_{\text{new}} = \beta \cdot V_{\text{old}} + (1 - \beta) \cdot \nabla_\theta L, \tag{2.8}$$

where V_{old} is the previous update direction and $\nabla_\theta L$ is the current gradient. The momentum term β regulates the tradeoff between past and new update directions. A value of $\beta = 0$ will result in the standard *SGD* update rule. A typical range for β is $[0.8, 0.999]$. The actual parameter update follows the same principle as the previously described optimization methods:

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \cdot V_{\text{new}} \quad . \tag{2.9}$$

2.1.4 Loss Functions

Loss functions quantize the error between a prediction and the ground truth to measure the quality of predictions. Different types of data require different loss functions. For many problems, multiple loss functions are applicable but each loss function has unique properties that impact the final trained model. In this section, we present different loss functions that are used in our work.

2.1.4.1 Squared Loss

The squared loss [26], also known as L2 loss, computes the squared error

$$L_{\mathrm{L2}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2} D(\mathbf{y}, \hat{\mathbf{y}})^2 \tag{2.10}$$

of a distance $D(\cdot)$ between the ground truth output $\mathbf{y}$ and the predicted output $\hat{\mathbf{y}}$. For one-dimensional outputs, $D(\cdot)$ can be the difference between the output scalars $D(y, \hat{y}) = y - \hat{y}$, for example. For multi-dimensional outputs, $D(\cdot)$ can be the Euclidean distance between the output vectors $D(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|_2$, for example. This loss function has a unique global minimum for convex problems and is often used for regression tasks. In practice, however, *ANNs* implement non-convex functions. Additionally, the squared loss is not robust to outliers or mislabeled data pairs. The factor $\frac{1}{2}$ is used as a scaling constant that eases the computation of the gradient and does not influence the optimization minimum. To compute a loss for multiple samples, the mean of individual squared losses is computed. This is known as the Mean Squared Error (MSE).

2.1.4.2 Absolute Loss

The absolute loss [41], also known as L1 loss, computes the absolute error

$$L_{\mathrm{L1}}(\mathbf{y}, \hat{\mathbf{y}}) = |D(\mathbf{y}, \hat{\mathbf{y}})| \tag{2.11}$$

of a distance $D(\cdot)$ between the ground truth output $\mathbf{y}$ and the predicted output $\hat{\mathbf{y}}$. Unlike the squared loss, the absolute loss is more robust to outliers because the computed loss grows linearly with the error. As a consequence, the gradient for every error is the same positive or negative constant. In this way, the gradients of samples with large error do not outweigh the gradients of samples with small error during the optimization. However, this also reduces the stability of the solution. To compute a loss for multiple samples, the mean of individual absolute losses is computed. This is known as the Mean Absolute Error (MAE).

2.1.4.3 Smooth L1 Loss

The smooth L1 loss [23] is a combination of the L1 loss and L2 loss. Formally, this loss function is computed as

$$L_{\text{smooth L1}}(\mathbf{y}, \hat{\mathbf{y}}) = \begin{cases} \frac{1}{2} D(\mathbf{y}, \hat{\mathbf{y}})^2 & \text{if} \quad |D(\mathbf{y}, \hat{\mathbf{y}})| < 1 \\ |D(\mathbf{y}, \hat{\mathbf{y}})| - \frac{1}{2} & \text{else} \end{cases} . \tag{2.12}$$

The smooth L1 loss behaves like the L1 loss if the distance $D(\cdot)$ is larger than 1 and like the L2 loss otherwise. In this way, the unique global minimum property of the L2 loss is exploited while the influence of outliers is reduced using the L1 loss for large errors. Figure 2.13 illustrates this effect and compares the L1, L2, and smooth L1 loss. The smooth L1 loss is used in many object detection methods such as Faster R-CNN [73] to optimize regression tasks robustly.

Figure 2.13: Comparison between the L1, L2, and smooth L1 loss for the interval $[-3, 3]$.

2.1.4.4 Cross Entropy Loss

The cross-entropy loss [4] is one of the most used loss functions for classification tasks. It originates from the mathematical field of information theory. There, the cross entropy describes the average amount of bits needed to identify an event drawn from the estimated probability distribution q rather than from the true distribution p, where both distributions have the same set of events. In the context of learning, the cross-entropy is used as a loss

function for multi-class classification problems. Formally, the cross-entropy loss computes

$$L_{\text{Cross Entropy}}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{c=1}^{C} y_c \cdot log(\hat{y}_c) \ , \tag{2.13}$$

where $\mathbf{y}$ is a one-hot vector of length C that represents the ground truth class probability distribution and $\hat{\mathbf{y}}$ is a vector of length C that represents the predicted class probability distribution. In this case, C is the number of classes, while y_c and $\hat{y}_c$ represent the ground truth and predicted probability for a certain class.

However, the output of a *ANN* is typically not a valid class probability distribution in which, for example, all class probabilities sum to one and there are no negative values. To overcome this limitation, a softmax normalization

$$\hat{y}_c = \frac{\exp(\tilde{y}_c)}{\sum_{i=1}^{C} \exp(\tilde{y}_i)} \tag{2.14}$$

is applied to all output values. In this case, $\hat{y}_c$ is a normalized output and $\tilde{y}_c$ is the corresponding raw output. This normalization makes it possible to interpret each element $\hat{y}_c$ of $\hat{\mathbf{y}}$ as the probability $p(c|\mathbf{x})$ of a certain class c given the input $\mathbf{x}$.

In a binary classification setup, a single output neuron with a sigmoid non-linearity (see Section 2.1.5.2) can be used instead of a softmax normalization for two output neurons. In this case, the cross entropy simplifies to

$$L_{\text{Binary Cross Entropy}}(y, \hat{y}) = -\big(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\big) \ , \tag{2.15}$$

where $y \in \{0, 1\}$ is the ground truth class probability for the first class and $\hat{y}$ is the predicted class probability for the first class.

2.1.5 Activation Functions

Activation functions have a fundamental role in *ANNs*. They apply a non-linear transformation to a linear combination of inputs which enables *ANNs* to approximate arbitrary functions. In this section, we briefly discuss activation functions used in this work.

2.1.5.1 Rectified Linear Unit (ReLU)

The most popular activation function is the Rectified Linear Unit (ReLU)

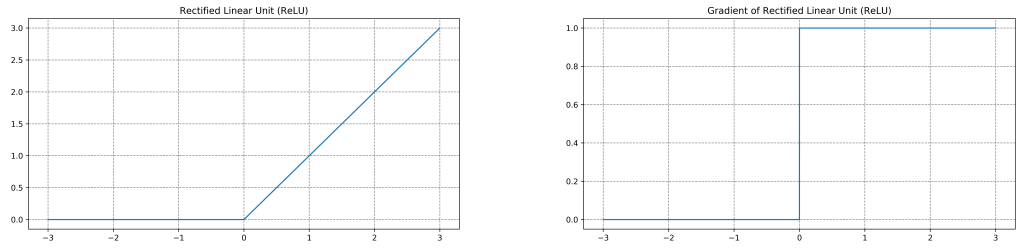
$$\sigma(x) = \max(0, x) \ . \tag{2.16}$$

This function is simple, thus, it is a common choice in many *ANNs*. The *ReLU* maps negative inputs to zero and leaves positive inputs unchanged, as shown in Figure 2.14a.

This result yields a piecewise-linear convex function. The derivative of the *ReLU* function

$$\sigma'(x) = \begin{cases} 1 & \text{if x} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

is a constant depending on the input. It is one for positive inputs and zero otherwise. The *ReLU* is not differentiable at the input $x = 0$. However, this is no issue in practice as one of the subgradients is simply selected in this case. The *ReLU* does not saturate for large inputs but is unbounded. Neurons with the *ReLU* activation function can end up in inactive states due to squashing negative inputs to zero. An illustration of the *ReLU* function and its derivative is shown in Figure 2.14.



(a) **(b)**

Figure 2.14: Illustration of the *ReLU* activation function and its derivative in the interval $[-3, 3]$.

2.1.5.2 Sigmoid

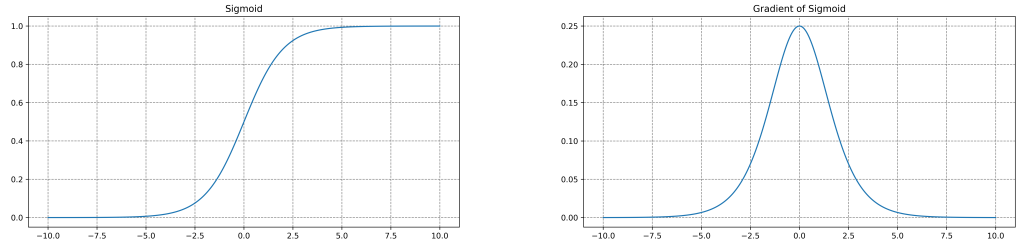
The sigmoid activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.18}$$

is another popular activation function that maps values from $\mathbb{R}$ to the interval $[0, 1]$. It is derived from probability theory and the Bayes' theorem. The derivative of the sigmoid function can be computed using the sigmoid function itself

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x)) \ . \tag{2.19}$$

The sigmoid function is smooth and continuously differentiable. It is centered and symmetric around input zero. The sigmoid function is approximately linear for inputs around zero and saturates for large positive or negative inputs. This saturation causes small gradients for inputs with large absolute value. An illustration of the sigmoid function and its derivative is shown in Figure 2.15.



(a) **(b)**

Figure 2.15: Illustration of the sigmoid activation function and its derivative in the interval $[-10, 10]$.

2.1.5.3 Hyperbolic Tangent Function

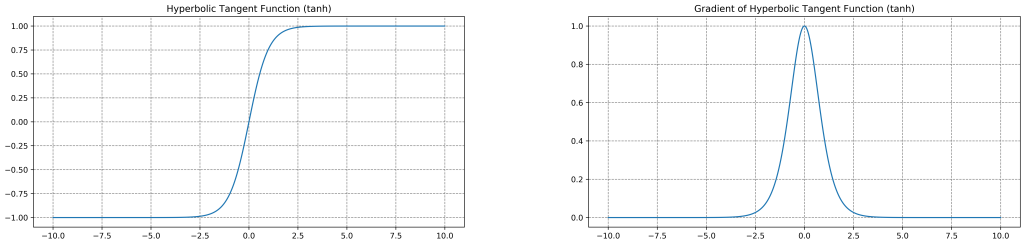
The hyperbolic tangent activation function

$$\sigma(x) = tanh(x) = \frac{sinh(x)}{cosh(x)} \tag{2.20}$$

is similar to the previously described sigmoid function. The main difference to the sigmoid function is that input values from $\mathbb{R}$ are mapped to the interval $[-1, 1]$. Similar to the sigmoid function, the derivative of the hyperbolic tangent activation function can be computed using the function itself:

$$\sigma'(x) = 1 - tanh^2(x) \ . \tag{2.21}$$

The saturation properties of the hyperbolic tangent activation function are similar to the sigmoid activation function. An illustration of the hyperbolic tangent function and its derivative is shown in Figure 2.16.



(a) **(b)**

Figure 2.16: Illustration of the hyperbolic tangent activation function and its derivative in the interval $[-10, 10]$.

2.2 Object Recognition

Humans are great at recognizing objects. We recognize a multitude of different objects within milliseconds, even despite challenging conditions like partial occlusions, scale changes, rotations, and translations. In this way, we achieve a high-level understanding of images that enables us to perform complex tasks like safely driving a car through a crowded street. Thus, the goal of object recognition approaches is to match and surpass human-level performance in terms of understanding objects in images.

Object recognition is an important field in computer vision [61, 63]. It is an umbrella term for techniques to find and identify objects in images. As a consequence, the object recognition approach has many applications in practice, for example, augmented reality [27], robotics [18] and medical imaging [50].

The prevalent strategy to understand objects with machines is to extract features from images and make a prediction based on these features, as shown in Figure 2.17. The idea of feature extraction is to compute a different representation of the input data that makes it easier to perform a certain task. Thus, feature extraction is an essential aspect of many object recognition approaches across different tasks.

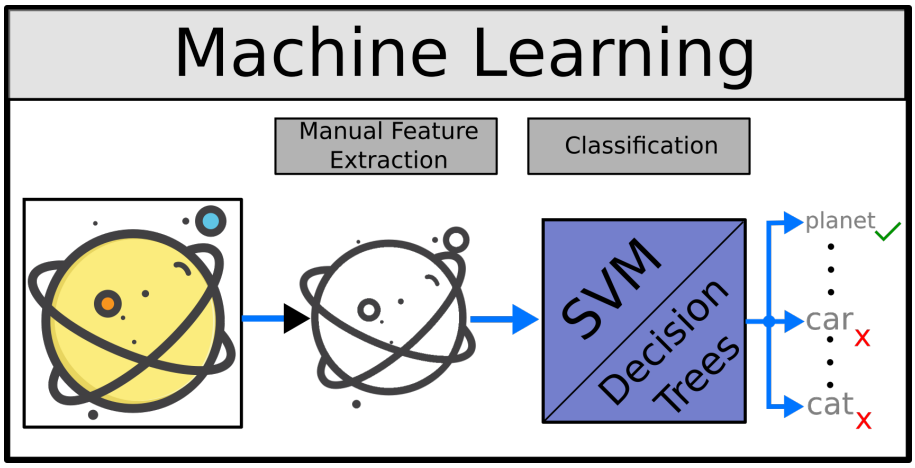
In this context, traditional machine learning approaches extract hand-crafted features and use a learning algorithm such as support vector machines [80] or decision trees [70] on top of these features to make a prediction (see Figure 2.17a). Early approaches for different object recognition tasks used hand-crafted feature extractors like SIFT [63], SURF [3], or HOG [16]. However, these features are not optimized for a certain task and, thus, the performance of traditional machine learning approaches is limited.

In contrast, deep learning approaches simultaneously learn a feature representation and make a prediction (see Figure 2.17b). In this way, they exploit the power of *CNNs* to compute features that are optimized for the specific task. In recent years, deep learning approaches greatly advanced the state-of-the-art in many object recognition tasks and significantly outperformed traditional machine learning approaches [46]. Today, *CNNs* are the prevailing tool used in object recognition systems.

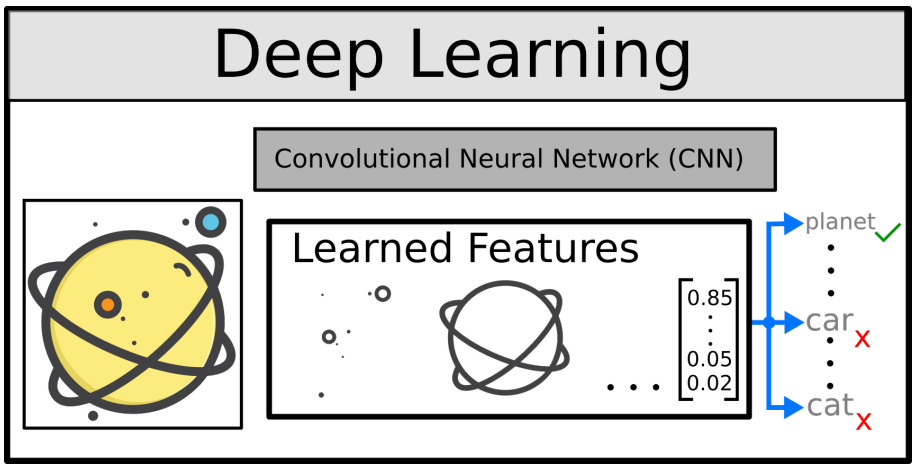
In the following sections, we describe different object recognition tasks and show how different tasks finally issue into Panoptic Segmentation (PS). In addition, we present existing methods to address these tasks focusing on recent approaches based on *CNNs* and discuss the relation of these works to our approach.

2.2.1 Object Classification

Object classification addresses the problem of predicting the class of an object in an image. Classification predictions are typically expressed as a list of individual class probabilities, where the most likely one is chosen as the final class label. Object classification has many applications, such as face recognition [87], image retrieval [81], and traffic sign recognition [77]. An example of object classification is shown in Figure 2.17.



(a) Traditional machine learning approaches extract hand-crafted features and use a learning algorithm such as support vector machines or decision trees on top of these features to make a prediction.



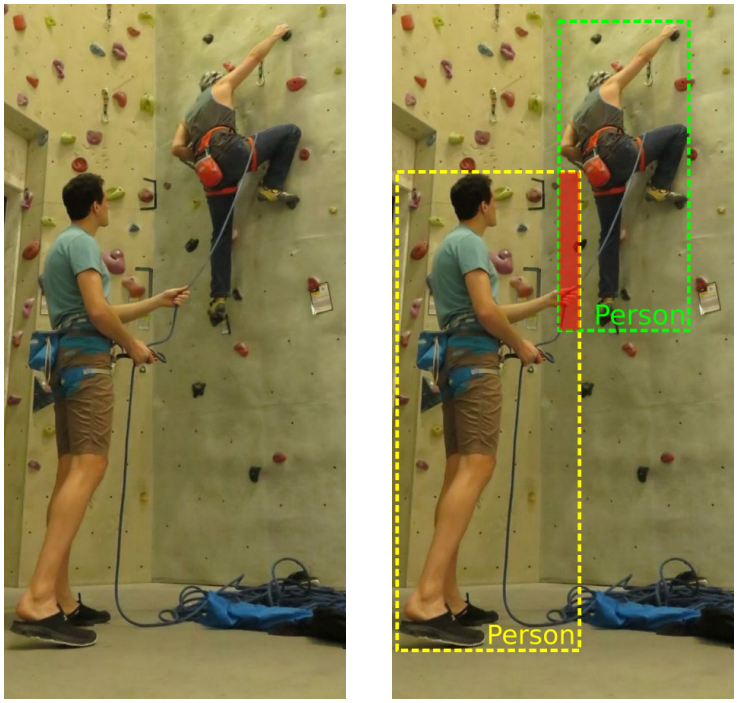
(b) Deep learning approaches simultaneously learn a feature representation and make a prediction, which results in significantly improved performance.

Figure 2.17: Comparison between (a) traditional machine learning and (b) deep learning for image feature based object recognition.

In recent years, many breakthroughs have been achieved in the field of object classification thanks to deep learning. In fact, the revolution of deep learning-based computer vision started with the successful application of *CNNs* to object classification [46]. Since then, many popular deep learning concepts that are found across different disciplines today have first been proposed for the task of object classification [33, 78, 82]. Due to the availability of very large supervised datasets this task has been a driving force in deep learning research.

2.2.2 Object Detection

Object detection aims at predicting the presence as well as the 2D location of objects in images. Detections are typically expressed as 2D bounding boxes with an accompanied class label for each bounding box, as shown in Figure 2.18. 2D bounding boxes are marked by four values (x, y, w, h) that present the 2D location and the 2D dimensions. Object detection has many applications in computer vision, including surveillance [37], face detection [87], robotics [90], and image analysis [5, 40, 81]. Moreover, object detection is often a first step to solve higher-level tasks like 3D object pose estimation [28] or 3D object reconstruction [25].



(a) Image. **(b)** Object Detection.

Figure 2.18: An example for object detection. Object detection identifies the presence of objects by predicting a 2D bounding box and a class label for each detected object. Bounding boxes of detected objects might overlap because objects often do not have perfectly rectangular shapes.

Images can contain many different objects. Thus, many approaches to object detection apply object classification on small *RoIs* of an image. Traditional machine learning approaches use exhaustive search algorithms and segmentation techniques to find possible *RoIs*. State-of-the-art networks for object detection use region-based approaches like Region Proposal Network (RPN) to find multiple *RoIs*.

Early approaches to object detection using sliding window techniques [16, 20, 87] to classify hand-crafted features computed from *RoIs* cropped from an image individually. Deep learning approaches significantly improved the state of the art in object detection be-

cause they learn features that are optimized for the task. Deep object detection methods can be categorized into two-stage detectors, including R-CNN [24], Fast R-CNN [23], Faster R-CNN [73], and SPPNet [32], and single shot detectors like YOLO [71] and SSD [60]. Two-stage detectors achieve higher accuracy, while single-shot detectors provide real-time object detection.

In the following, we briefly describe the evolution of *R-CNNs* for object detection as our approach builds on region-based computations.

R-CNN: The first *R-CNN* approach, which is called R-CNN [24], combines a selective search method [86] with a conventional *CNN*, as shown in Figure 2.19. This approach generates multiple *RoIs* across the image by clustering similar pixels. For each of these regions, a *CNN* performs a class prediction and a bounding box regression. Since the used *CNN* requires a fixed-size input, i.e., 227×227, each *RoI* is rescaled to this size without preserving the aspect ratio.

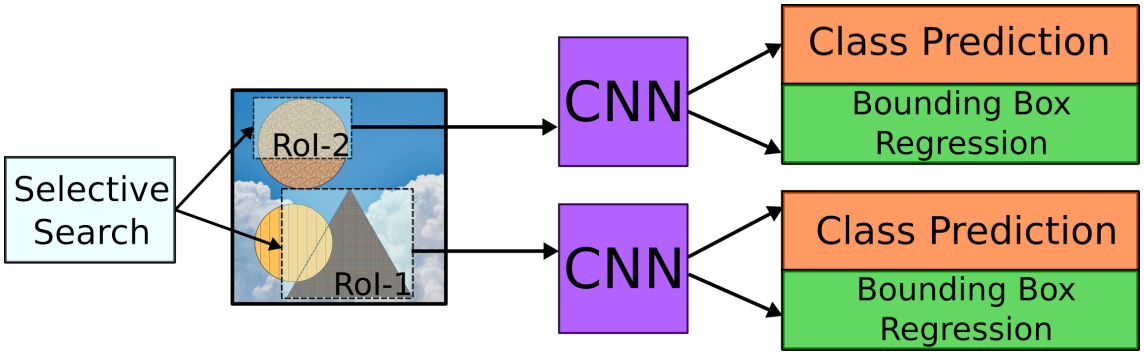


Figure 2.19: Schematic of R-CNN [24]. A class-agnostic selective search algorithm identifies *RoIs*. Each *RoI* is processed by the same CNN, as illustrated by the network color, to perform a class prediction and a bounding box regression.

Fast R-CNN: The biggest limitation of R-CNN [24] is the fact that the computationally expensive *CNN* is evaluated for each detected *RoI*. As a consequence, features in overlapping *RoIs* are unnecessarily recomputed in each *CNN* forward pass. Thus, Fast R-CNN [23], illustrated in Figure 2.20, uses a single *CNN* forward pass to extract features for the entire image and only applies a light-weight *ANN* on top of these features for each detected *RoI*. For this purpose, a new *RoI* pooling operation is employed that samples features from an associated region of the full-size feature map and transform them into a fixed-size sub-feature map. This is necessary because the light-weight *ANN* that performs the class prediction and bounding box regression requires a fixed input size.

Faster R-CNN: To increase the accuracy compared to Fast R-CNN [23], Faster R-CNN [73] replaces the selective search with a *RPN* that detects *RoIs*, as shown in Figure 2.21. This approach has multiple benefits. First, the system can be trained end-to-end.

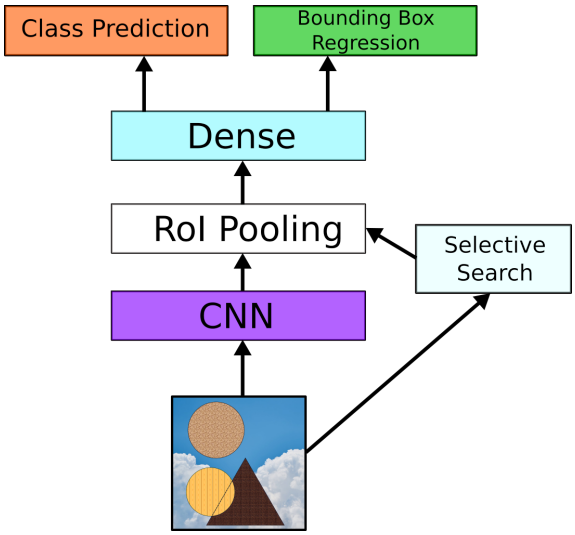


Figure 2.20: Schematic of Fast R-CNN [23]. In contrast to R-CNN [24], Fast R-CNN only performs a single computationally expensive *CNN* forward pass to compute features for the entire image. Next, *RoIs* are sampled from this full-size feature map instead of sampling from the image. Finally, a light-weight *ANN* performs a class prediction and a bounding box regression for each *RoI*.

Second, *RoIs* are detected in the feature map, where they are actually sampled from. Finally, the *RPN* gives more control over the number of *RoIs*.

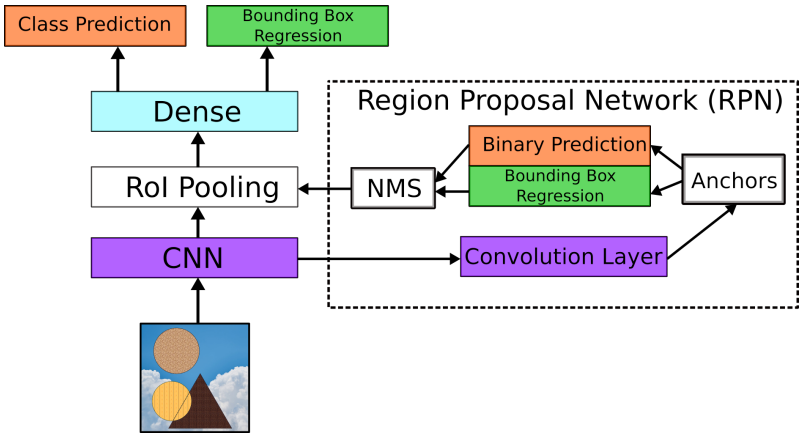


Figure 2.21: Schematic of Faster R-CNN [73]. Compared to Fast R-CNN [23], the selective search is replaced by a *RPN* that detects *RoIs* to improve the accuracy.

2.2.3 Instance Segmentation

Instance segmentation takes the idea of object detection a step further. Instead of only computing a 2D bounding box and a class label for each detected object, a detailed binary mask which takes recognizes the shape of the found object is predicted. An example for

(a) Ground Truth. **(b)** Instance Segmentation.

Figure 2.22: An example for instance segmentation. In contrast to object detection, each detected object is recognized with a binary mask.

instance segmentation is shown in Figure 2.22, where the two persons from our previous object detection example in Figure 2.18 are now recognized with masks.

Instance segmentation models can be categorized into top-down [31, 54, 59] and bottom-up [45, 55, 58, 85] approaches. One of the most popular top-down frameworks is Mask R-CNN [31], which is a combination of the object detection network Faster R-CNN [73] and an additional mask segmentation branch. In particular, an additional *FCN* (Section 2.1.2.2) on top of Faster R-CNN [73] is employed to produce instance segmentation masks. Since our work builds on the Mask R-CNN framework, we describe its components in more detail:

Mask R-CNN: Mask R-CNN [31] generalizes the Faster R-CNN [73] approach and presents an extensible framework that supports higher-level tasks like instance mask predictions on top of object detection, as demonstrated in Figure 2.23. In this context, different light-weight *ANNs* are applied on top of sub-feature maps to address various tasks besides class prediction and bounding box regression with a single network. Moreover, Mask R-CNN [31] replaces the *RoI* pooling operation introduced in Fast R-CNN [23] with a *RoI* align operation that uses bilinear interpolation as opposed to nearest-neighbor interpolation to extract sub-feature maps with reduced quantization errors.

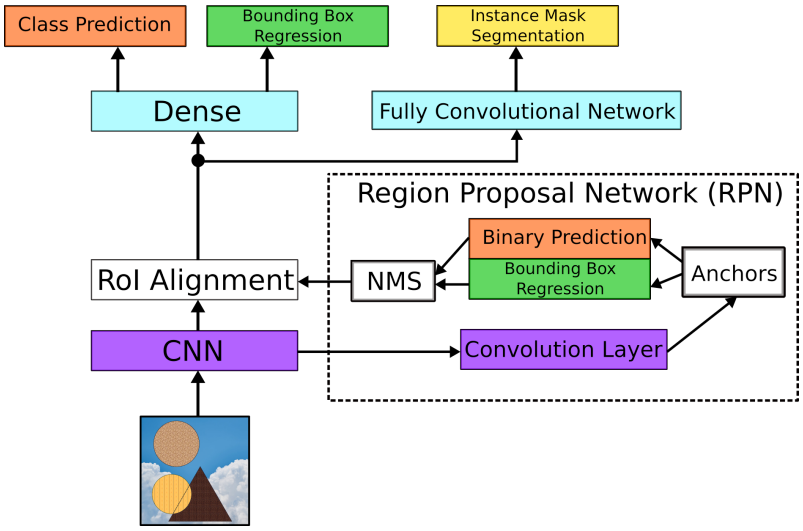


Figure 2.23: Schematic of Mask R-CNN [31]. Compared to previous *R-CNN* approaches, quantization errors due to the RoI pooling operation are reduced using a novel RoI align operation. Additionally, different light-weight *ANNs* are applied on top of sub-feature maps to address various tasks besides class prediction and bounding box regression, e.g., instance mask prediction, with a single network.

Feature Pyramid Network: A feature pyramid network is a feature extraction backbone with a bottom-up and top-down architecture that has lateral connections to build rich feature maps from multi-scale features [56]. *CNNs* like ResNet [33] build the bottom-up feature pyramid, where early stages detect low-level features and later stages compute high-level features. Features of individual stages are then combined in a multi-scale feature map in a top-down manner. Similar concepts of fusing information from multiple scales can be found in many areas of computer vision [62, 74].

Region Proposal Network: *RPNs* have first been introduced in Faster R-CNN [73] and detect *RoIs* in feature maps. In Mask R-CNN, a *RPN* receives the convolutional feature map computed from a Feature Pyramid Network (FPN) and provides region proposals as an output. In this case, the *RPN* computes regions based on a set of anchors that span the entire image. For this purpose, anchors with five different scales and three different aspect-ratios are designed. For each anchor, a class-agnostic objectness score is predicted and depending on this score the region is discarded or kept. Additionally, the *RPN* outputs four delta bounding box regression values: $\Delta x_{center}, \Delta y_{center}, \Delta_{height}, \Delta_{width}$ which refine the given anchor to the final region proposal.

RoI Alignment: *RoI* alignment solves a quantization problem occurring when pooling features for a proposed region for a global feature map. This pooling is necessary because a fully-connected network that classifies the region proposals expects a fixed-sized input. Since *RoIs* can have any size they have to be rescaled consistently. In Fast R-CNN [23] an

operation called *RoI* pooling based on nearest-neighbor interpolation is used to rescale the feature map. In Mask R-CNN, the Roi alignment operation based on linear interpolation is used for rescaling to reduce the quantization error. This allows *RoI* alignment to capture information more accurately compared to *RoI* pooling, which results in improved overall object detection quality.

2.2.4 Semantic Segmentation

Semantic segmentation is an important task in computer vision. It describes the process of labeling an image in a semantic meaningful way by associating each pixel with a class label, as shown in Figure 2.24. Semantic segmentation combines information about *stuff* and *things* into one prediction.

(a) Ground Truth. **(b)** Semantic Segmentation.

Figure 2.24: An example of semantic segmentation. Each pixel is associated with a class label.

Most semantic segmentation approaches are based on *FCN* (see Section 2.1.2.2) [62] that can take an arbitrarily sized image and output a pixel-wise prediction. State-of-the-art semantic segmentation networks can be categorized into an encoder-decoder structure [2, 74] and a dilated convolution structure [8, 9].

Encoder-decoder networks have an hourglass structure that first compresses and then decompresses information. The encoder part downsamples the input information using the convolutions and the pooling layer. Thus, it reduces the spatial resolution and generates

high-level feature maps. The decoder part upsamples this compressed representation back to the original input size creating the full pixel-wise semantic segmentation output.

Dilated convolutions support the exponential expansion of the receptive field without pooling or large filter kernels [92]. Atrous convolutions make it possible to capture multi-scale context by using different atrous rates. In this way, features with large receptive fields are calculated without reducing the spatial input size or applying many regular convolutions.

2.2.5 Panoptic Segmentation

Instance segmentation makes it possible to recognize objects of countable *things* classes at the pixel level, as shown in Figure 2.22. However, simply observing the instance segmentation output, there is no context information about the surrounding of objects. Without the visual clues of the ground truth image, we would have a hard time figuring out what is going on in this particular image. Clearly, we need more information about the background *stuff* classes to understand the scene.

In contrast, semantic segmentation provides a dense pixel-level labeling that combines information about *stuff* and *things* classes into one prediction, as shown in Figure 2.24. The semantic segmentation output now gives us the contextual information that there are persons on the ground (green area) and on the wall (orange area). However, semantic segmentation cannot distinguish between multiple instances within a common class, e.g., both persons in this image get the same class id. Although it is rather easy to separate the two instances in this case, additional post-processing is necessary. In practice, objects belonging to the same class might be close to each other or occlude each other. In this case, the output of semantic segmentation creates a single class blob that cannot be separated anymore. If we want to understand the complete image both semantic segmentation and instance segmentation need to be joined.

PS [44] addresses this problem of complete 2D scene segmentation by not only assigning a class label to each pixel of an image but also differentiating between instances within a common class. Thus, it can be seen as a unification of semantic segmentation [9, 62, 74] and instance segmentation [31, 45, 54, 58], as shown in Figure 2.25. *PS* combines all the background and foreground information of semantic segmentation and instance segmentation. *PS* is a new and active research area with applications in augmented reality, robotics, and medical imaging [21, 69, 96].

Fusing semantic and instance information has a rich history in computer vision [83, 84]. However, only recently Kirillov et al. formalized the task of *PS* and introduced a panoptic quality (PQ) metric to assess the performance of complete 2D scene segmentation in an interpretable and unified manner [44]. This formalization and the availability of large datasets with corresponding annotations [57] motivated research on *PS* lately.

Early approaches to *PS* use two highly specialized networks for semantic segmentation [9, 62, 74] and instance segmentation [31, 55, 59, 85] and combine their predictions

(a) Ground Truth. **(b)** Panoptic Segmentation.

Figure 2.25: An example of panoptic segmentation. Each pixel is associated with a class label. Additionally, instances belonging to the same class are separated.

heuristically [1]. Instead, recent methods address the two segmentation tasks with a single network by training a multi-task system that performs semantic and instance segmentation on top of a shared feature representation [43]. This reduces the number of parameters, the computational complexity, and the time required for training.

To improve panoptic quality, newer approaches propose a differentiable fusion of semantic and instance segmentation instead of a heuristic combination. In this way, they learn to combine the individual predictions and optimize directly for the final objective in an end-to-end manner. For example, UPSNet [89] introduces a parameter-free merging technique to generate panoptic outputs using a single network.

Another strategy to improve accuracy is to exploit mutual information and similarities between semantic and instance segmentation network branches. In this context, AUNet [53] incorporates region proposal information as an attention mechanism in the semantic segmentation branch. In this way, the semantic segmentation focuses more on *stuff* classes and less on *things* classes, which are eventually replaced by predicted instance masks. TASCNet [52] enforces L2-consistency between predicted semantic and instance segmentation masks to exploit mutual information. SOGNet [91] addresses the overlap-

[1] COCO 2018 Panoptic Segmentation Task. `http://cocodataset.org/index.htm#panoptic-leaderboard`. Accessed: 31.01.2020

ping issue of instances using a scene graph representation which computes a relational embedding for each object based on geometry and appearance.

Similar to our approach, IMP [22] uses predicted instance segmentation masks as additional input for the semantic segmentation branch. Compared to our approach, they use a different normalization technique and combine instance masks using the max operator instead of averaging them.

3

Holistic End-to-End Panoptic Segmentation Network with Interrelations

Contents

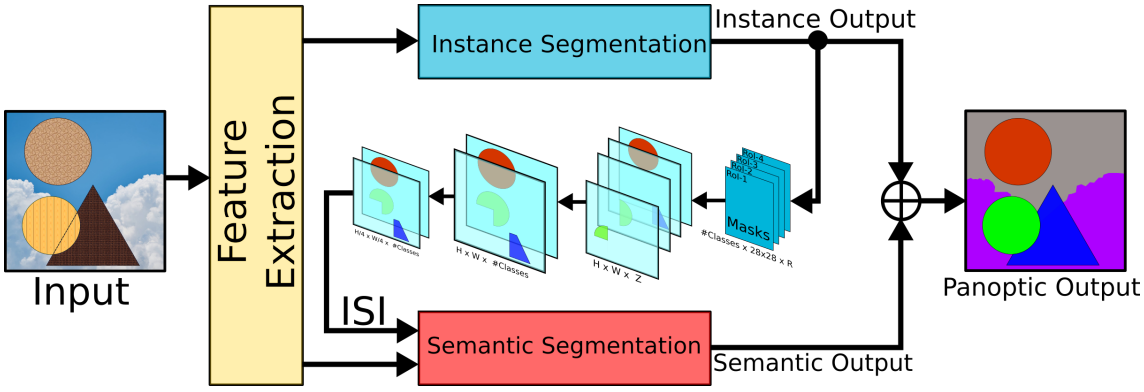


Figure 3.1: Overview of our proposed end-to-end panoptic segmentation network with task interrelations. We perform semantic and instance segmentation on top of a shared feature extraction backbone, provide instance segmentation predictions as additional feature input to our semantic segmentation branch, and merge individual predictions using differentiable operations. In this way, we exploit mutual information between the tasks and support end-to-end training.

In this chapter, we introduce our end-to-end trainable Panoptic Segmentation (PS) network with inter-task relations, which is illustrated in Figure 3.1. For this purpose, we first present our end-to-end trainable architecture, which combines semantic and instance segmentation predictions in a differentiable way, in Section 3.1. Then, we introduce our interrelations module, which provides instance segmentation predictions as additional feature input to our semantic segmentation branch, in Section 3.2.

3.1 End-to-End Panoptic Architecture

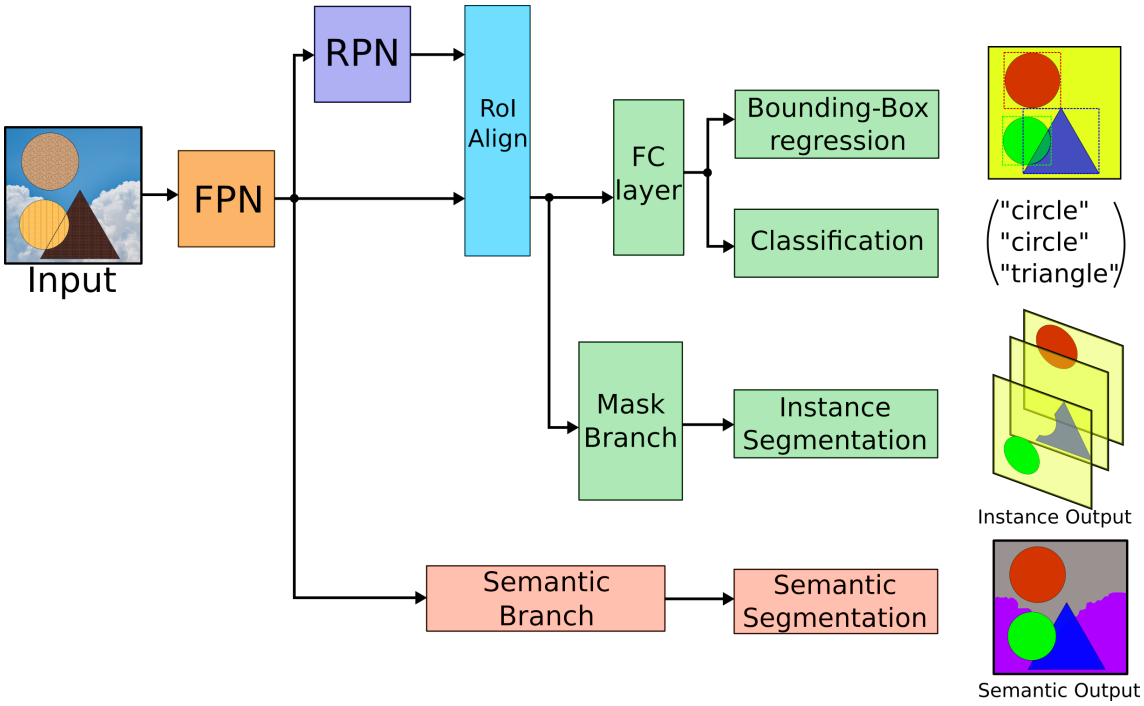


Figure 3.2: Our network architecture builds upon Panoptic Feature Pyramid Networks [43]. This work extends the generalized Mask R-CNN framework [31], which performs object detection and instance segmentation, with a semantic segmentation branch. However, to generate a panoptic output, individual predictions are combined using heuristics in [43]. In contrast, we propose a differentiable fusion.

Our network architecture builds upon Panoptic Feature Pyramid Networks [43]. Like many recent panoptic segmentation methods, this approach extends the generalized Mask R-CNN framework [31] with a semantic segmentation branch, as shown in Figure 3.2. This results in a multi-task network that predicts a dense semantic segmentation in addition to sparse instance segmentation masks. For our implementation, we use a shared ResNet-101 [33] feature extraction backbone with a Feature Pyramid Network [56] architecture to obtain a multi-level representation with combined low- and high-level features [1]. These features serve as shared input to our semantic and instance segmentation branches, as shown in Figure 3.4.

For the semantic segmentation branch, we process each stage of the feature pyramid $\{P2, P3, P4, P5\}$ by a series of upsampling modules. These modules consists of 3×3 convolutions, batch normalization [39], ReLU [29], and $2\times$ bilinear upsampling. Because the individual stages have different spatial dimensions, we process each stage by a different

[1] Abdulla, W. H. (2017). Mask R-CNN for Object Detection and Instance Segmentation on Keras and TensorFlow. `https://github.com/matterport/Mask_RCNN`.

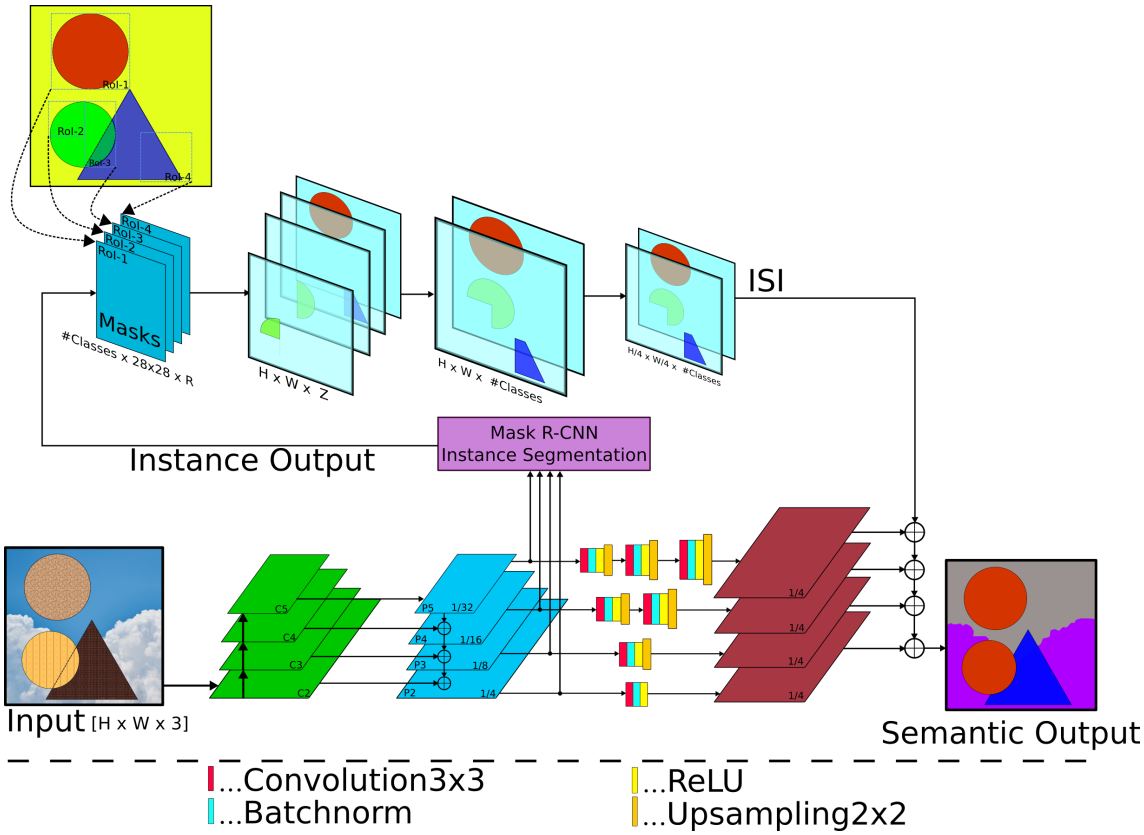


Figure 3.3: Illustration of our semantic segmentation branch. The ResNet-101 FPN backbone combines four stages of low-level and high-level features. For each stage we use 3×3 convolutions, batch normalization, ReLU and 2×2 upsampling until we reach the desired size of $H/4 \times W/4$. These outputs are combined with the additional prior knowledge of our instance segmentation branch in the form of an initial segmentation image (ISI) to produce the final pixel-wise semantic segmentation prediction.

number of upsampling modules to generate $H/4 \times W/4 \times 128$ feature maps. In this case, H and W are the spatial input image dimensions and the feature dimensionality is set to 128 as in [43]. For $P5$, which has the lowest spatial resolution but the highest level of feature abstraction, we use three upsampling modules. For $P2$, which has the highest spatial resolution but the lowest level of feature abstraction, we only use a single Conv-BN-ReLU block but no bilinear upsampling. The resulting outputs of all stages are concatenated and processed using a final 1×1 convolution to reduce the channel dimension to the desired number of classes.

For our instance segmentation branch, we implemented a Mask R-CNN [31]. We use a Region Proposal Network (RPN) to detect Regions of Interest (RoIs), perform non-maximum suppression, execute *RoI* alignment, and predict 28×28 binary masks as well as class probabilities for each detected instance. In order to combine the semantic and instance segmentation outputs of our network, we use an internal differentiable fusion

instead of external heuristics. For this purpose, we first select the most likely class label for each detected instance using a differentiable

$$\text{soft argmax}(\mathbf{z}) = \sum_{i}^{N} \lfloor \frac{e^{z_i \cdot \beta}}{\sum_{k}^{N} e^{z_k \cdot \beta}} \rceil \cdot i \tag{3.1}$$

operation [7], where N is the number of *things* classes, β is a large constant, and $\mathbf{z}$ is a vector of predicted class logits with length N. Using β in the exponent in combination with the round function allows us to squash all non-maximum values to zero. In this way, we approximate the traditional non-differentiable *argmax* function, which allows us to backpropagate gradients through this operation. This approximation holds as long as there are no duplicate maximum logit values in $\mathbf{z}$. However, even if two or more logits share the same maximum value, we can detect and resolve the edge case by adding a small amount of random noise to the predictions.

Additionally, the large constant β in the exponent can cause overflows. To ensure numerical stability, we, thus, subtract the maximum logit value from $\mathbf{z}$ such that the largest logit becomes zero while the remaining logits become negative an converge to zero:

$$\lim_{x \to -\infty} e^x = 0 \,. \tag{3.2}$$

Next, we resize the predicted 28×28 mask logits for each detected instance according to its predicted 2D bounding box size and place them in empty canvas layers at the predicted 2D location, as shown in Figure 3.4 (top right). Additionally, we merge the canvas layers for regions of interest with the same object id and high mask Intersection Over Union (IoU). To merge canvas layers, we add all combinable canvas layers and normalize each spatial location by the number of contributing regions.

The resulting canvas collection from the instance segmentation branch is then concatenated with the *stuff* class logits of the semantic segmentation branch to generate our panoptic output, as illustrated in Figure 3.4 (bottom). The pixel-wise panoptic segmentation output is attained by applying a softmax layer on top of the stacked semantic and instance segmentation information. The shape of the final output is $H \times W \times$ (*# stuff classes* + *# detected instances*). For *stuff* classes, the output is a class ID. For *things* classes, the output is an instance ID. The corresponding class ID for each instance can be gathered from our instance segmentation output.

During training, it is important to reorder the detected instances to match the order of the ground truth instances. For this purpose, we use a ground truth instance ID lookup table. Additionally, to support training with multiple images per mini batch, we fix the number of instances that can be detected. The number of maximum instances per image, thus, becomes a hyper-parameter. If the number of detected instances is smaller than our maximum number of instances hyper-parameter, our instance canvas collection is padded with empty canvas layers.

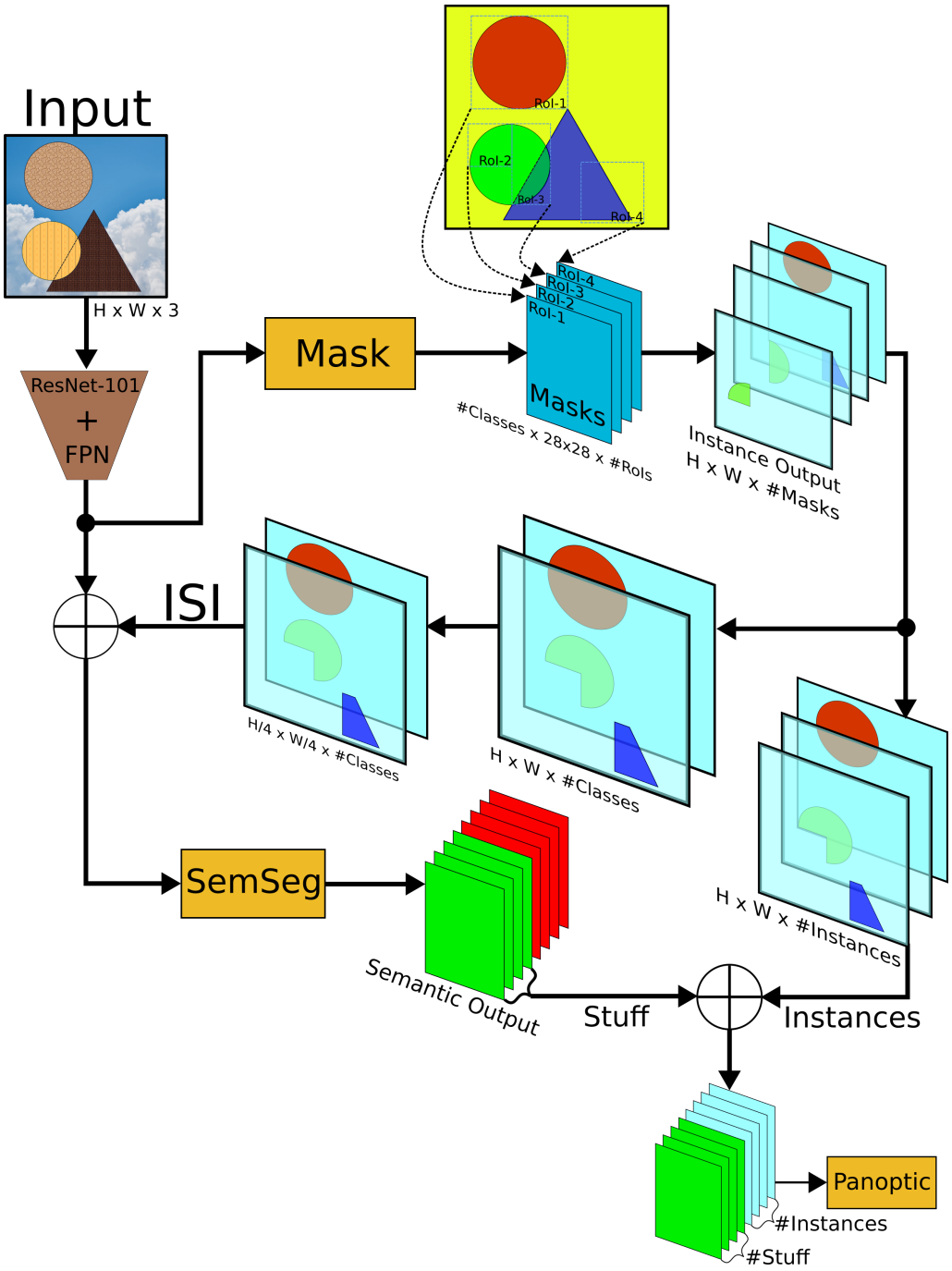


Figure 3.4: Detailed illustration of our proposed panoptic segmentation architecture. We extend the Mask R-CNN framework [1] with a semantic segmentation branch. These branches use instance segmentation predictions as additional feature input in the form of an initial segmentation image (ISI). In this way, the semantic segmentation branch exploits prior knowledge from already recognized instances. Finally, we merge *stuff* predictions form semantic segmentation and *things* predictions from instance segmentations in a differentiable way. This results an end-to-end trainable panoptic segmentation network with interrelations.

3.2 Inter-task Relations

In Section 3.1, we presented a differentiable way to combine semantic and instance segmentation predictions. This formulation allows us to join the outputs of our two branches internally for end-to-end training. However, it also allows us to provide instance predictions as additional feature input to our semantic segmentation branch, as shown in Figure 3.4.

For this purpose, we first evaluate our instance segmentation branch and build an instance canvas collection as described in Section. 3.1. Next, we merge canvas layers of instances that belong to the same class using weighted average and insert empty canvas layers for missing or undetected classes. In this way, we generate an Initial Segmentation Image (ISI) which represents a coarse semantic segmentation for *things* classes.

To exploit this segmentation prior in our semantic segmentation branch, we downsample our *ISI* to $(H/4 \times W/4 \times \#\ \textit{things classes})$ and concatenate it with the output of our semantic segmentation upsampling modules, as shown in Figure 3.3. Next, we apply four network blocks consisting of 3×3 convolution, batch normalization, and ReLU followed by a single 1×1 convolution, batch normalization, and ReLU block to reduce the channel dimension to the number of classes. Finally, we use bilinear upsampling to obtain semantic segmentation logits at the original input image dimensions. A softmax layer followed by an argmax operation can be used to compute a semantic segmentation from these logits.

By exploiting the segmentation prior given by *ISI*, the upsampling modules of our semantic segmentation branch focus more on the prediction of *stuff* classes and boundaries between individual classes instead of *things* classes. This is a huge advantage compared to disjoint semantic and instance segmentation branches where redundant predictions are performed in the semantic segmentation branch. As a consequence, this link between the individual tasks increases the panoptic performance of our system.

4 Evaluation and Results

Contents

In this chapter, we evaluate our proposed approach. For this purpose, we describe the different evaluation metrics for semantic, instance, and panoptic segmentation in Section 4.1. In Section 4.2, we cover different forms of data augmentation applied to images during training. Finally, we present the results on a synthetically generated TOY dataset consisting of geometric shapes and on the challenging Cityscapes dataset [14] for semantic understanding of urban street scenes in Section 4.3.

To provide an unbiased evaluation, we compare four different approaches with an increasing level of entanglement between semantic and instance segmentation. All methods use the same backbone, training protocol, and hyper-parameters:

Semantic + Instance: This approach uses two different networks based on a ResNet-101 [33] backbone which independently performs semantic and instance segmentation. A heuristic is used to combine the individual results [44].

Panoptic FPN: This method is a reimplementation of Panoptic Feature Pyramid Networks [43] with a ResNet-101 [33] backbone. In contrast to *Semantic + Instance*, the semantic and instance segmentation branches use a single shared feature representation. The results, however, are still merged heuristically.

HPS: Our holistic panoptic segmentation network (HPS) extends *Panoptic FPN* as described in Section 3.1. Our network internally builds the panoptic segmentation output using differentiable operations which enables us to optimize for the final objective.

HPS + ISI: This method augments our *HPS* with inter-task relations between the semantic and instance segmentation branches by using an initial segmentation image (ISI), as introduced in Section 3.2.

4.1 Metrics

In this section, we describe the Intersection Over Union (IoU), Mean Average Precision (mAP), and Panoptic Quality (PQ) metrics which are used to measure performance on the tasks of semantic segmentation, instance segmentation, and panoptic segmentation respectively.

4.1.1 Intersection over Union (IoU)

The IoU is a simple evaluation metric that is used in many computer vision tasks to measure accuracy. In the field of object detection, for example, it is used to determine the exactness of a predicted bounding box compared to the given ground truth. In this work, we adopt the IoU to measure semantic segmentation accuracy. Instead of calculating the IoU with bounding boxes, as shown in Figure 4.1, we compute it at pixel-level segments:

$$IoU = \frac{TP}{TP + FP + FN} \,. \tag{4.1}$$

In this case, the confusion matrix for an image is computed from True Positives (TPs), False Positives (FPs), and False Negatives (FNs) of all pixels. IoU is the ratio of TP pixels divided by the sum of pixels, as shown in Equation 4.1.

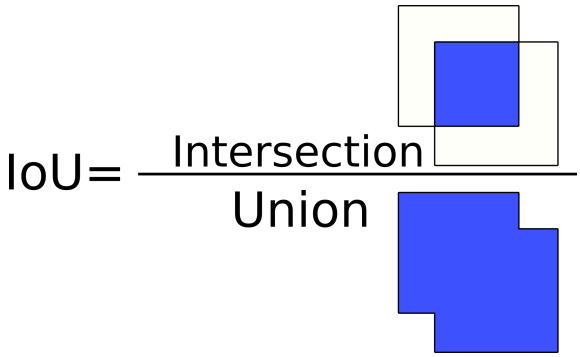


Figure 4.1: Illustration of IoU: The area of intersection is divided by the area of union. This metric measures the accuracy in a range $[0\% - 100\%]$, where 0% has no overlap between ground truth and prediction and 100% has the exact overlap.

The intersection consists only of pixels of a class, which are found in both the ground truth mask and the prediction mask. The union consists of all pixels found in the ground truth mask and the prediction mask. The IoU metric is computed for each class separately and then averaged over all classes to get the Mean Intersection Over Union (mIoU).

4.1.2 Mean Average Precision (mAP)

The mAP is a popular evaluation metric for the task of instance segmentation. It involves the computation of the area under the precision/recall curve, see Eq (4.4) and Figure 4.2.

Precision measures the accuracy of a prediction. It is calculated by dividing TP objects by the total number of classified objects, as shown in Eq (4.2). The decision, whether an object is classified as positive or negative depends on the selected IoU (Section 4.1.1) threshold.

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

Recall measures how well objects can be retrieved. It is calculated by dividing TP objects by the total amount of instances, as shown in Eq (4.3). Recall starts at a minimum value of zero and increases over time:

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

$$AP = \int_0^1 p(r)dr, \tag{4.4}$$

AP computes the area under a curve given by precision over recall, through integrating it from zero to one, see Eq (4.4). The area is first rectified, as shown in Figure 4.2, by shifting each precision value to the maximum of the current recall level. In order to obtain mAP the sum of individual Average Precisions per class is divided by the total number of classes:

$$mAP = \frac{1}{C} \cdot \sum_{i}^{C} AP_i \quad . \tag{4.5}$$

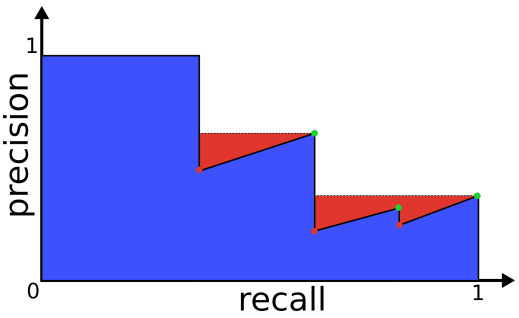


Figure 4.2: Illustration of mAP: with an example curve. **Recall** measures how well objects can be retrieved from a given image. **Precision** measures the accuracy of the prediction. The average precision is given by the area under the precision/recall curve. Since precision can have a "zig-zag" pattern through increasing accuracy the curve is oftentimes interpolated, where each precision value is set to the maximum precision at this recall level.

4.1.3 Panoptic Quality (PQ)

The PQ [44] is a recently introduced evaluation metric for panoptic segmentation. It measures semantic segmentation and instance segmentation quality using a simple and easily interpretable formula:

$$PQ = \frac{\sum_{(p,q)\in TP} IoU(p,q)}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}, \tag{4.6}$$

where the IoU is calculated between a predicted segment p and the ground truth segment q. The term $\frac{1}{2}|FP| + \frac{1}{2}|FN|$ is added as a penalization for segments without matching pairs. Segments are matched when the IoU between them is larger than 0.5. Eq (4.6) can be split into two parts:

$$PQ = \underbrace{\frac{\sum_{(p,q)\in TP} IoU(p,q)}{|TP|}}_{segmentation\ quality\ (SQ)} \cdot \underbrace{\frac{|TP|}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}}_{recognition\ quality\ (RQ)}, \tag{4.7}$$

which define PQ as the product between Segmentation Quality (SQ) and Recognition Quality (RQ). PQ is insensitive to class imbalance, by averaging over all classes. The evaluation splits ground truth and prediction into three sets $\{TP, FP, FN\}$ of unique segments, as shown in Figure 4.3.

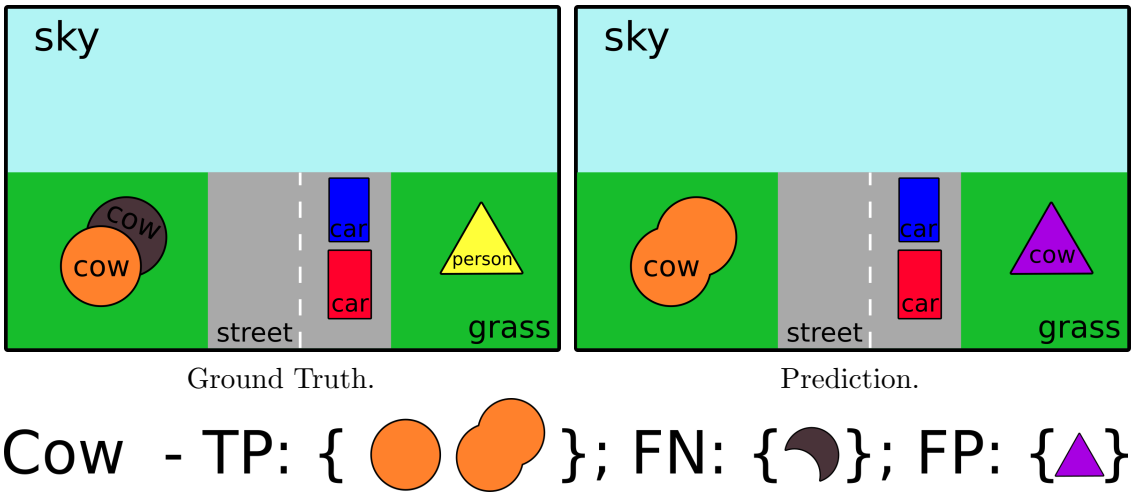


Figure 4.3: Example of ground truth and prediction for panoptic segmentation. Pairs of matching colored segments have an IoU > 0.5. In this example, the cow class prediction yields the three sets TP, FN, and FP.

4.2 Data Augmentation

Data augmentation describes the process of modifying the input data before feeding it to a network. This synthetically increases the training data size without actually acquiring more samples. Data augmentation improves image classification accuracy [68] and regularizes possible overfitting issues with small datasets. In this work, we apply four different data augmentation methods to our input images.

4.2.1 Horizontal Flipping

Image. Horizontally flipped image.

Figure 4.4: Data augmentation: Horizontal flipping applied to an example image from the Cityscapes dataset.

Horizontal flipping, as shown in Figure 4.4, is a simple data augmentation method that flips the image around its y-axis. For each 2D location (x, y), the rigid transformation

$$\begin{pmatrix} x\prime \\ y\prime \end{pmatrix} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \tag{4.8}$$

is applied. We apply this transformation at random with a chance of 50% to flip. The horizontal flip is a very useful method because the content of natural images can face in the other direction as well and still be valid. For example, cars can drive on the left or right side of the street. Note that vertical flips would result in invalid images such as sky at the bottom and upside-down cars. Hence, it would create unrealistic images that to not improve the quality of trained models.

4.2.2 Random Rotation

Random rotations are rigid transformations that warp an image by a random angle around some origin. Each 2D location (x, y) is rotated around the origin, as shown in Figure 4.5, by first shifting the point to the origin then rotating the point and finally shifting it back

Image. Rotated image (10 degrees).

Figure 4.5: Data augmentation with a **Rigid transformation: Random rotation** applied to an example Cityscapes image.

to get (x', y'). Mathematically, this is formulated as

$$\begin{bmatrix} x' + origin_x \\ y' + origin_y \end{bmatrix} = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \begin{bmatrix} x - origin_x \\ y - origin_y \end{bmatrix} . \tag{4.9}$$

We use random rotations in the range of $\theta \in [-10, 10]$ degrees.

4.2.3 Brightness Augmentation

Image. Image with increased brightness $[H, S, V + 60]$.

Figure 4.6: Data augmentation: **Brightness augmentation** applied to an example Cityscapes image.

This augmentation method describes the element-wise addition of a scalar to the intensity value of each pixel. For this process, we transform our image from RGB to HSV color space, then add a scalar value drawn from a normal distribution to the **V** channel, and transform the modified image back to the RGB space. Augmented images, as shown in Figure 4.6, appear to be brighter or darker. This is a useful augmentation for our dataset since images can be taken at different times of the day.

Image. Image with added noise.

Figure 4.7: Data augmentation: **Random noise** applied to an example Cityscapes image.

4.2.4 Random Noise

This augmentation method describes the process of adding random Gaussian noise to each pixel. Random noise can improve the robustness and generalization of a trained network [88]. The additional noise forces the network to learn the underlying structure of objects and not only memorize specific pixel patterns since the input is constantly changing. An example of added random noise can be seen in Figure 4.7.

4.3 Results

In this section, we quantitatively and qualitatively evaluate our proposed approach and discuss results. We perform experiments on a synthetically generated TOY dataset consisting of geometric shapes and on the challenging Cityscapes dataset [14] for semantic understanding of urban street scenes.

4.3.1 TOY-Shapes

The TOY-Shapes dataset is a synthetically generated dataset consisting of 500 training and 100 validation images created by us as a sanity check for our approach. The dataset contains four different classes (1. Square, 2. Circle, 3. Triangle, 4. Half-circle), where the first two belong to *stuff* and the other two belong to *things*. We provide the semantic segmentation, instance segmentation, and panoptic segmentation masks for every image. The resolution of our input images is $[512 \times 256 \times 3]$. Example images with accompanied ground truth annotations for semantic segmentation, instance segmentation, and panoptic segmentation can be seen in Figure 4.8.

Images for TOY-Shapes can be generated on the fly, however, we generate all 500 training and 100 validation images once before training. This is useful, since creating all the images on the fly when a new minibatch for training has to be composed causes a massive bottleneck on the data loading pipeline. Each image contains a randomly colored background and up to 10 objects in different locations.

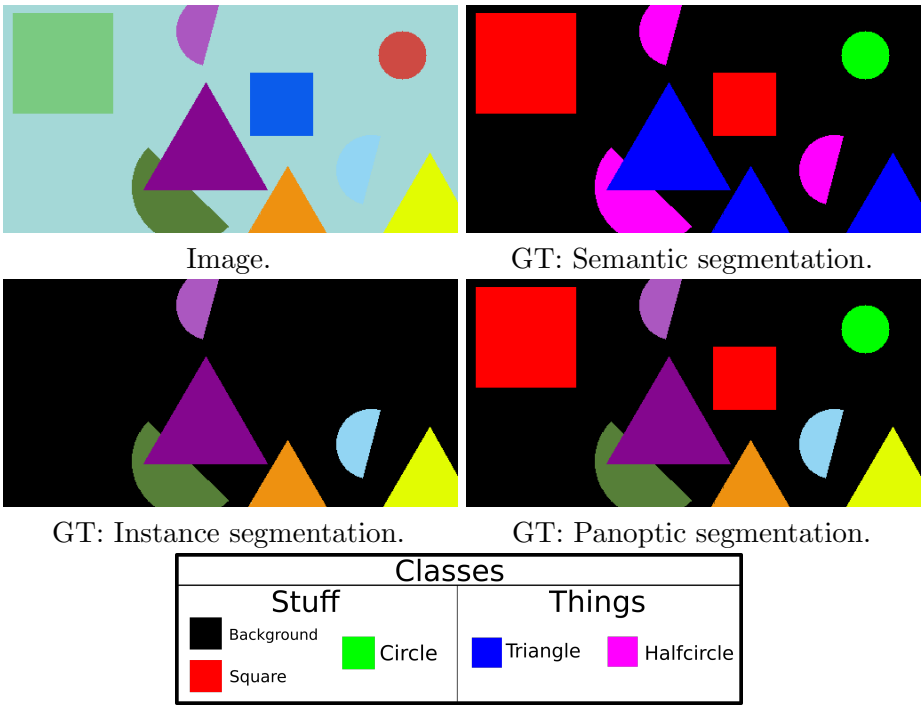


Figure 4.8: Example from the TOY-Shapes dataset. We show an image with corresponding ground truth annotations for semantic segmentation, instance segmentation, and panoptic segmentation. The color palette is showing the respective color for semantic segmentation. Note that in the ground truth annotations instances are shown with different colors to make it easier to differentiate between them.

We provide the following data for training:

Semantic segmentation: The ground truth semantic segmentation is provided as an image of size $[H \times W \times 1]$ and has a traditional mapping of classes, where the last dimension has a value range of $[1, 4]$.

Instance segmentation: The ground truth instance segmentation is provided as an image of size $[H \times W \times M]$, where M is the number of instances in the image. For each instance, a binary $[H \times W]$ mask segments the instance and occlusions are resolved by considering later masks to be on top.

Panoptic segmentation: The ground truth panoptic segmentation is provided as an image of size $[H \times W \times 1]$. The last dimension is the instance id in the range $[1, 10]$. Note that we provide a second ground truth image for easier evaluation since we adopt the evaluation scripts of the Cityscapes dataset [13]. This augments the class information with the additional instance id. Hence, the object class is multiplied by 1000 and the current instance count is added. For example, the third triangle which is drawn in the image would have the number 3002. This format supports up to 1000 objects per class.

4.3.1.1 Evaluation Setup

We use Stochastic Gradient Descent (SGD) with momentum $\beta = 0.9$ and a minibatch size of 6 split on two GPUs (TITAN X with 12GB). At the start of each epoch, the dataset is randomly shuffled and split into 84 batches without selecting an image again until the whole dataset was used at least once. Each image in a minibatch is randomly modified with data augmentation methods, as presented in Section 4.2. Since random rotations can rotate objects out of scope of the image, we have two options. First, remove all the missing masks and clean up the training data. Second, perform repeated runs of data augmentation on the original image from the dataset until a data augmentation run is valid. We use option number two.

The Region Proposal Network (RPN) uses the following scales $(16, 32, 64, 128, 256)$ to generate anchors that cover the entire image as well as small regions, where each scale corresponds to one level of the Feature Pyramid Network (FPN). Each level has 3 aspect-ratios $(0.5, 1, 2)$ per anchor, which makes a total of 32736 anchors for our images. We train our networks in a three-stage learning process. In the first stage, we initialize our Mask R-CNN [31] with weights pre-trained for instance segmentation on COCO [57] and use a learning rate of $\eta = 0.001$. In this stage, we freeze all layers except our network heads and train the network for 50 epochs. In the second stage, we decrease the learning rate by a factor of 5 and train all layers above ResNet-101 layer 4 for another 25 epochs. In the last stage, we reduce the learning rate again by a factor of 5 and fine-tune the entire network for 25 epochs. In total, the networks train for 100 epochs in a period between 6 and 12 hours, as shown in Figure 4.9. The idea of this multi-stage learning is that the pre-trained weights of our feature extractor generalize well and we only have to learn the additional new tasks on top. In the last few iterations, we train the entire network but use a small learning rate such that the network does not overfit to the small dataset while still fine-tuning all weights.

We train the four methods *Semantic + Instance*, *Panoptic FPN*, *HPS*, and *HPS + ISI* described at the beginning of chapter 4 once before evaluation. The hyper-parameters were selected based on the best testing performance on *Panoptic FPN* [43], which acts as the baseline for this thesis. *Semantic + Instance* and *Panoptic FPN* do not have a single panoptic segmentation output. Hence, we use the simple heuristic described in the original paper to merge the semantic segmentation and instance segmentation outputs [44].

4.3.1.2 Training Times

First, we compare the training times of different approaches. The obvious training time reduction achieved by combining two separate neural networks (*Semantic + Instance*) into a single network (*Panoptic FPN*, *HPS*, and *HPS + ISI*) can be seen in Figure 4.9. The training time is nearly halved when using a shared backbone. Merging the information of semantic segmentation and instance segmentation internally with our methods, as described in Chapter 3, only adds 8.4 minutes for *HPS* to the total training time.

Semantic Segmentation: IoU

Class	Semantic + Instance	Panoptic FPN	HPS	HPS + ISI
Square	96.0	95.8	95.9	**96.8**
Circle	**95.6**	95.3	95.5	95.3
Triangle	94.6	94.6	94.6	**95.2**
Halfcircle	**92.3**	91.4	92.0	91.2
mIoU	**94.6**	94.3	94.5	**94.6**

Table 4.1: Quantitative *IoU* results on the TOY-Shapes dataset. The results show that for such a simple dataset there are no significant differences between the individual methods. This can also be accounted to the small size of the dataset.

Using additional inter-task relations only adds 33 minutes to the total training time for *HPS + ISI*.

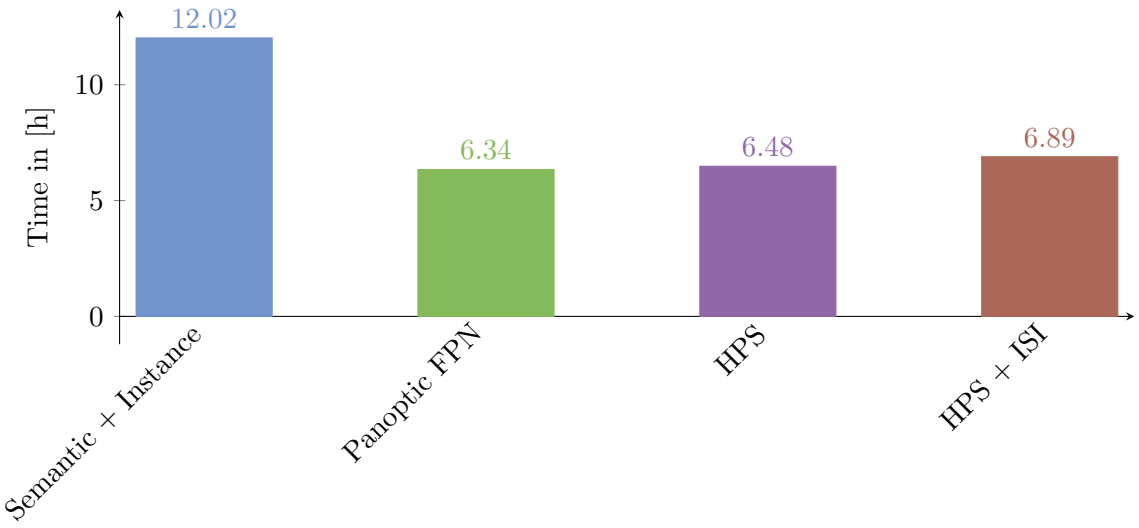


Figure 4.9: TOY-Shapes dataset training time comparison. We can see that training a single neural network with a shared backbone reduces training time by nearly half as expected. Adding additional complexity for the generation of *ISI* does not increase the time significantly.

4.3.1.3 Semantic Segmentation

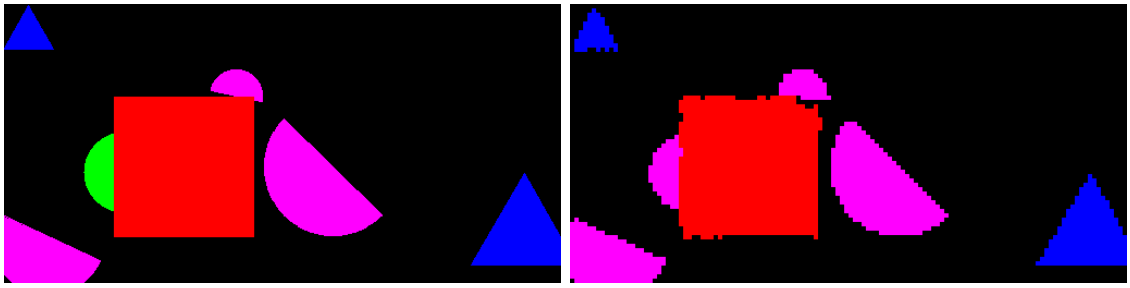
In this section, we discuss the semantic segmentation performance of different approaches. The corresponding results under the *IoU* metric (see Section 4.1.1) are shown in Table 4.1.

The accuracy drops in cases of ambiguous shapes, such as half-circles and circles, which happen to look the same when half of the objects are occluded.

The results on the TOY-shapes dataset show no significant differences between the individual methods. We hypothesize that the four types of shapes have distinct charac-

teristics which makes it easy for the network to learn how to distinguish between them. For example, squares have perpendicular corners, triangles have sharp angles, half-circles and circles have round boundaries. We can see that the *IoU* score for circular shapes (half-circles and circles) is lower than for the other shapes since there is more ambiguity between them.

The average score for all four methods is approximately 94.5%. This high accuracy can be explained by the simple structures and homogeneously colored areas without textures in the images. Although the dataset is simple, we do not reach higher accuracy because occlusions make it hard for the network to figure out the right class of the object in many cases.



GT: Semantic Segmentation. Prediction: Semantic Segmentation.

Figure 4.10: Illustration of occlusion issues: The *green circle* on the left side is occluded by the *red square*, hence the network does predict a *half-circle* on the right.

One such example can be seen in Figure 4.10, which shows that objects can sometimes overlap in a way that they look like another shape. In this case, the circle has an overlaying square, which makes it look like a half-circle. In these instances, not even humans could clearly tell the correct class without further clues. On the one hand, we can see that the results on the triangle class improve slightly using *HPS + ISI* because of the additional prior information of the instance segmentation branch. On the other hand, the performances on the half-circle shapes are slightly lower since the prior information is wrong in many cases due to the previously described ambiguity in the case of overlapping shapes. Finally, we show a few visual examples of our results for all four methods in Figure 4.11.

4.3.1.4 Instance Segmentation

In this section, we discuss the instance segmentation performance of different approaches. The corresponding results under the *mAP* metric (see Section 4.1.2) are shown in Table 4.2.

The results are similar as in the case of semantic segmentation (see Section 4.3.1.3). There are no significant differences between all four evaluated methods. Again, we argue that this simple dataset is too easy to solve for the complex networks. Thus, no method outperforms the others.

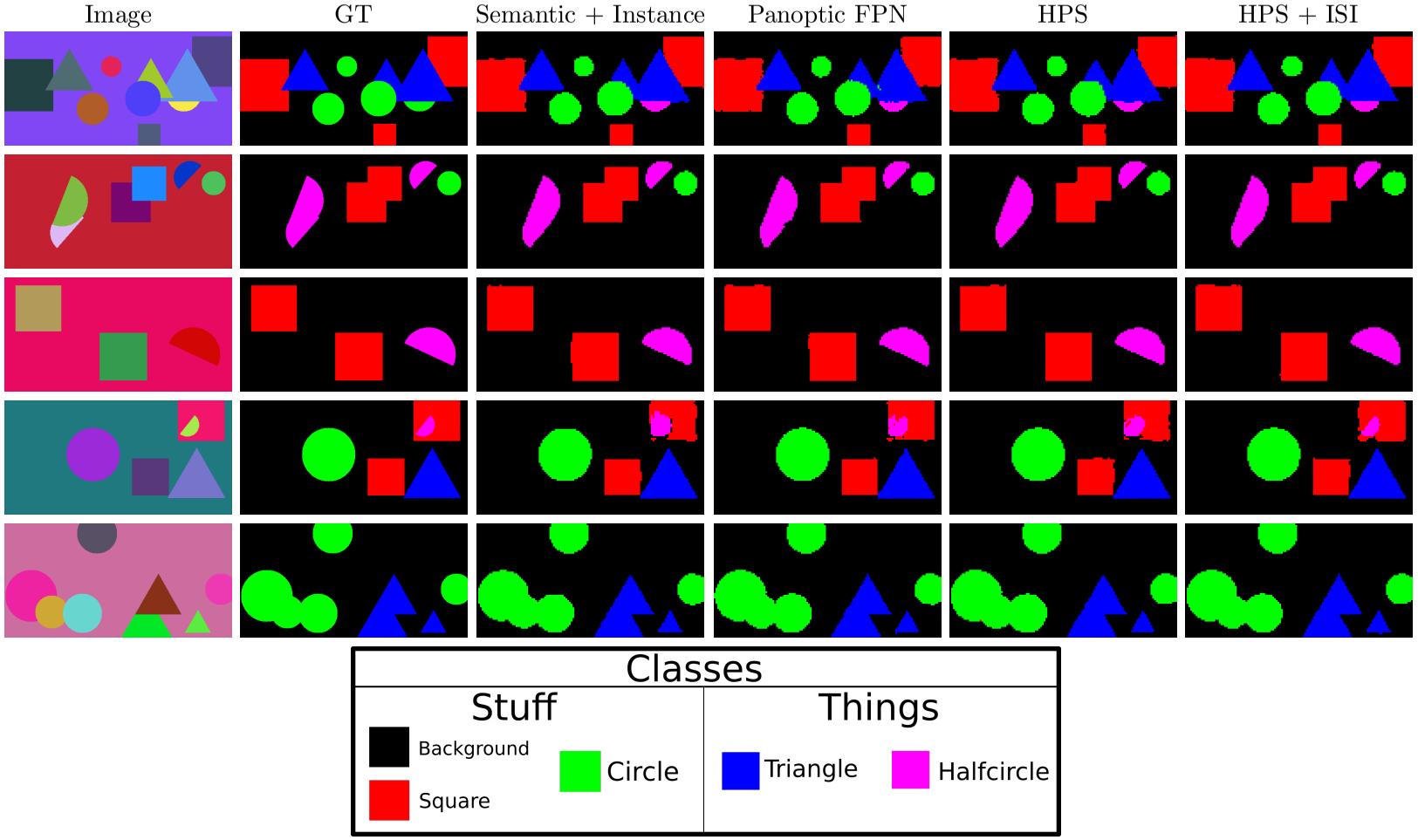


Figure 4.11: Qualitative results on the TOY-Shapes dataset. The results show that there is no significant difference between evaluated methods even on ambiguous objects like halfcircles and circles (1st row. Sometimes, our *HPS + ISI* predicts more accurate semantic label transitions (3rd row). Overall the dataset is not complex enough to see a significant differences between the methods. **Best viewed in digital zoom.**

Mean Average Precision: mAP

Class	Semantic + Instance		Panoptic FPN		HPS		HPS + ISI	
	AP	**AP@50**	**AP**	**AP@50**	**AP**	**AP@50**	**AP**	**AP@50**
Triangle	88.2	98.1	88.6	97.2	**89.6**	**99.0**	89.4	99.0
Halfcircle	87.0	94.9	**89.5**	**97.0**	89.2	95.2	87.5	94.8
mAP	87.6	96.5	89.1	97.1	**89.4**	**97.1**	88.4	96.9

Table 4.2: Quantitative mAP results on the TOY-Shapes dataset. The results show that for such a simple dataset we do not gain any significant improvements in the instance segmentation metric.

However, the reported scores for the *mAP* are only around 88%. This is due to the difficulty of large occlusions in the TOY-Shapes dataset, as shown in Figure 4.12. In the case of Region-based Convolutional Neural Network (R-CNN) such as Mask R-CNN [31] anchor boxes might fail to detect small or highly occluded objects. Another limitation of this instance segmentation strategy is that only one object can be found per region. This means that multiple objects which are close to each other are often only detected as one object. Finally, we show a qualitative examples of our results for all four methods in Figure 4.13.

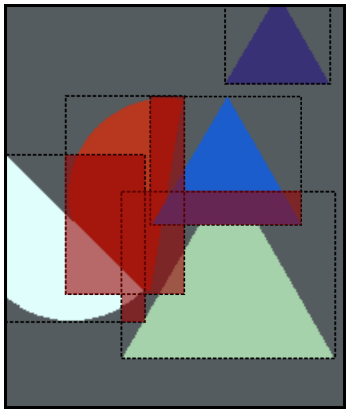


Figure 4.12: Illustration of the overlap problem. The disadvantage of *R-CNN* approaches is the single object per box. In case of occlusion and many overlaps between bounding boxes some objects might be missed.

4.3.1.5 Panoptic Segmentation

In this section, we discuss the panoptic segmentation performance of different approaches. The corresponding results under the *PQ* metric (see Section 2.2.5) are shown in Tables 4.3 and 4.4.

We observe that for all methods the predictions for squares and triangles have a higher score than the predictions for more ambiguous shapes such as half-circles and circle.

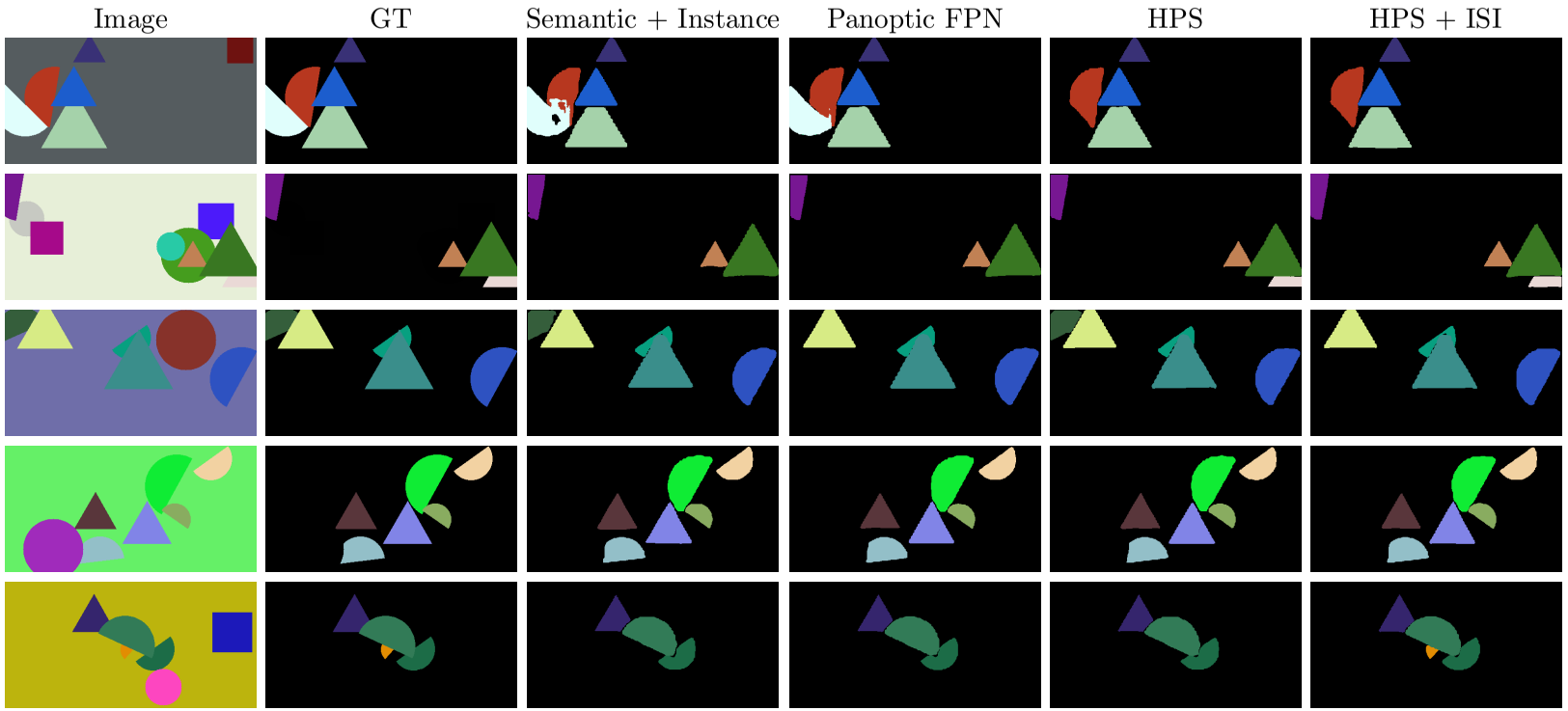


Figure 4.13: Qualitative results on the TOY-Shapes dataset. All four methods use the same instance segmentation branch. Thus, there are little differences in pure instance segmentation performance. Overall the dataset is not complex enough to see a significant differences between the methods. **Best viewed in digital zoom.**

Another observation is that the *PQ* increases with increasing entanglement between semantic and instance segmentation in different methods, as shown in Table 4.3. *Semantic + Instance* and *Panoptic FPN* are not optimized for the panoptic segmentation task. Their predictions for semantic segmentation and instance segmentation are merged heuristically [44] to produce a panoptic output. In contrast, our methods (*HPS* and *HPS + ISI*) use a single end-to-end trainable network, which significantly improves the achieved *PQ*.

One major issue in the first two methods (*Semantic + Instance* and *Panoptic FPN*) is the threshold selection for instance masks. In particular, the instance segmentation masks are generated using a sigmoid nonlinearity followed by a thresholding function. If the predicted value of a pixel is larger than 0.5 this pixel becomes active. In the case of overlapping regions, this causes confusion and the network does not know which mask should be chosen since both have the same value after thresholding. Additionally, pixels at transitions between instances are often predicted incorrectly as background because the thresholding reduced the mask probability of instances to zero in uncertain areas, as shown in Figure 4.14. Our methods (*HPS* and *HPS + ISI*) resolve this issue by letting the network learn to address overlapping issues on its own, as described in Section 3.1.

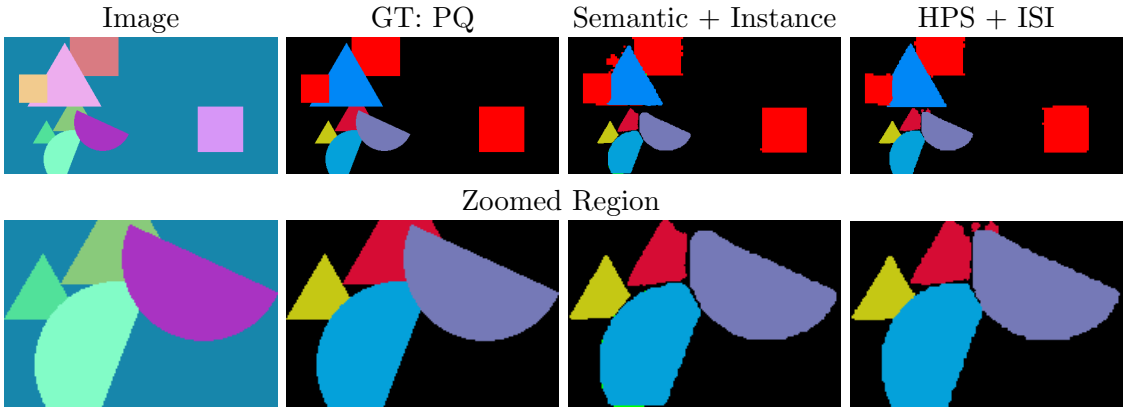


Figure 4.14: Example of edge improvements for overlapping objects. *HPS + ISI* resolves the overlapping issue of object better than *Semantic + Instance*.

Additionally, using inter-task relations in the form of an Initial Segmentation Image (ISI), as described in Section 3.2 provides an effective segmentation prior and increases the overall panoptic quality for *HPS + ISI*. Finally, we show qualitative examples of our results for all four methods in Figure 4.15.

4.3.2 Cityscapes

The Cityscapes dataset [14] is a challenging dataset for semantic understanding of urban street scenes. It contains 5000 fine annotated images with large diversity among locations (50 cities), several seasons (spring, summer, fall) and good/medium weather conditions. The resolution of all images is $[2048 \times 1024 \times 3]$. Cityscapes has 30 classes grouped

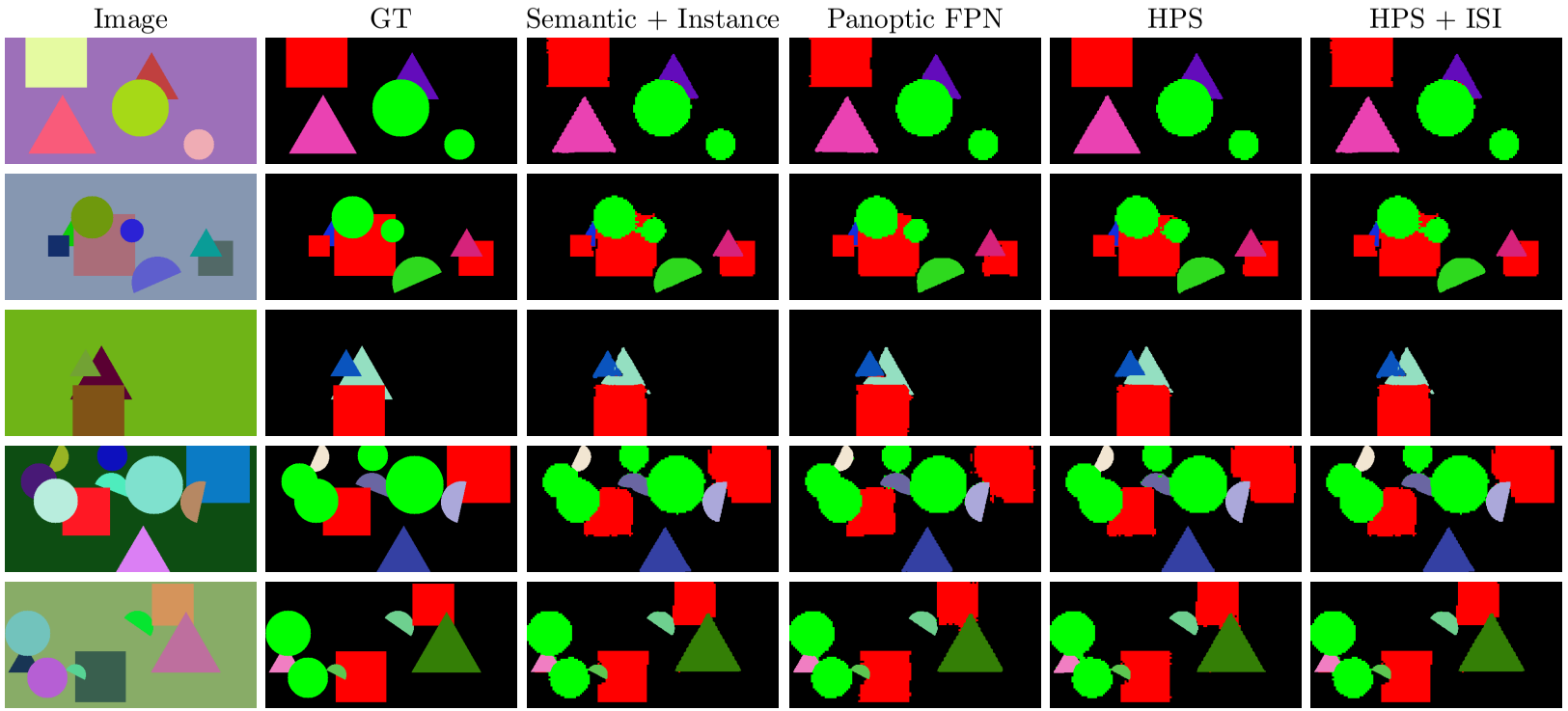


Figure 4.15: Qualitative results on the Toy-Shapes dataset. As the quantitative results suggest, each approach improves the image quality more. The most improvement can be seen in occluded areas with many overlapping objects. Overall the dataset is not complex enough and the objects are spread to much such that we do not see a significant change across the approaches. **Best viewed in digital zoom.**

Panoptic Quality: PQ

Class	Semantic + Instance			Panoptic FPN			HPS			HPS + ISI		
	PQ	SQ	RQ	PQ	SQ	RQ	PQ	SQ	RQ	PQ	SQ	RQ
Square	87.4	95.3	91.8	91.4	95.5	95.7	92.6	95.4	97.1	**95.5**	**96.2**	**99.3**
Circle	89.0	96.1	92.7	89.6	96.0	93.3	89.7	96.2	93.3	**92.9**	**95.9**	**96.9**
Triangle	91.8	93.5	98.2	92.4	94.6	97.7	95.6	96.5	99.1	**96.0**	**96.9**	**99.1**
Halfcircle	90.2	94.7	95.2	91.6	94.9	96.5	**92.7**	96.5	**96.0**	92.3	**96.6**	95.6

Table 4.3: Quantitative results on the TOY-Shapes dataset. The results show that a shared feature backbone reduces overfitting compared to two disjoint networks (*Semantic + Instance* vs *Panoptic FPN*). Also, generating the final panoptic output internally and training the system end-to-end increases the performance (*Panoptic FPN* vs *HPS*). Finally, using inter-task relations in the form of an initial segmentation image (ISI) provides an effective segmentation prior and increases the overall panoptic quality (*HPS* vs *HPS + ISI*).

Method	PQ	SQ	RQ	PQ^{Things}	SQ^{Things}	RQ^{Things}	PQ^{Stuff}	SQ^{Stuff}	RQ^{Stuff}
Semantic + Instance	89.6	94.8	94.4	90.6	94.1	96.6	88.2	95.6	92.2
Panoptic FPN	91.2	95.2	95.7	92.0	94.7	97.1	90.4	95.7	94.4
HPS	92.6	96.1	96.3	94.1	96.5	**97.5**	91.1	95.8	95.5
HPS + ISI	**94.2**	**96.4**	**97.7**	**94.1**	**96.7**	97.3	**94.2**	**96.0**	**98.0**

Table 4.4: Quantitative results on the TOY-Shapes dataset. The results show that a shared feature backbone reduces overfitting compared to two disjoint networks (*Semantic + Instance* vs *Panoptic FPN*). Also, generating the final panoptic output internally and training the system end-to-end increases the performance (*Panoptic FPN* vs *HPS*). Finally, using inter-task relations in the form of an initial segmentation image (ISI) provides an effective segmentation prior and increases the overall panoptic quality (*HPS* vs *HPS + ISI*).

into 8 different categories. It provides ground truth for semantic segmentation, instance segmentation, and panoptic segmentation. Each image has many dynamic objects and different backgrounds. The 5000 images are split into 2975 training samples, 500 validation samples, and 1525 test samples.

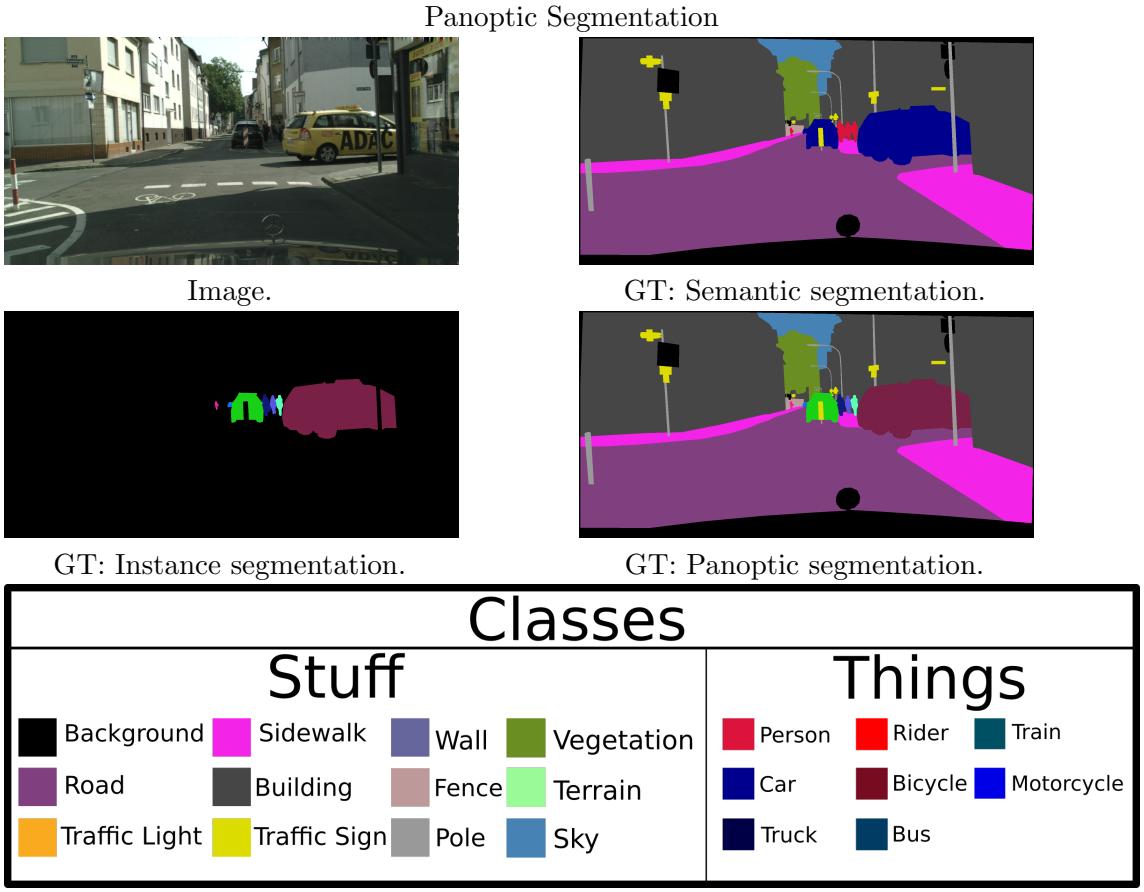


Figure 4.16: Example from the Cityscapes dataset. We show an image with corresponding ground truth annotations for semantic segmentation, instance segmentation, and panoptic segmentation. The color palette is showing the respective color for semantic segmentation. Note that in the ground truth annotations instances are shown with different colors to make it easier to differentiate between them.

Cityscapes has an online benchmark submission system that evaluates the prediction accuracy on the test data. Since the test image annotations are not public, we only use the validation images to evaluate our approach. In this work, we use 19 of the available 30 classes, where 11 are considered as *stuff* and 8 as *things*. An example of images with accompanying ground truth annotations for semantic segmentation, instance segmentation, and panoptic segmentation can be seen in Figure 4.16.

4.3.2.1 Evaluation Setup

The pre-processing performed for Cityscapes images is similar as for the TOY-Shapes images described in Section 4.3.1.1. We use *SGD* with momentum (see Section 2.1.3.3) $\beta = 0.9$ and a minibatch size of 6 split on two GPUs (TITAN X with 12GB). At the start of each epoch, the dataset is randomly shuffled and split into 500 batches without selecting an image again until the whole dataset was used at least once. Each image in a minibatch is randomly modified with data augmentation methods, as presented in Section 4.2. Because our computational resources are limited, we restrict the maximum number of instances per image to 30 and excluded samples with more instances from training and evaluation. In this way, we use 2649 of 2975 training images ($\approx 89\%$) and 415 of 500 publicly available validation images ($\approx 83\%$). Additionally, we reduce the spatial image resolution from 2048×1024 to 1024×512. We could downsample the images even more to increase the minibatch size, but this would result in images like Figure 4.17, where small structures such as poles vanish.

The *RPN* uses the following scales $(16, 32, 64, 128, 256)$ to generate anchors that cover the entire image as well as small regions, where each scale corresponds to one level of the *FPN*. Each level has 3 aspect-ratios $(0.5, 1, 2)$ per anchor, which makes a total of 65472 anchors for our images. We train our networks in a three-stage learning process. In the first stage, we initialize our Mask R-CNN [31] with weights pre-trained for instance segmentation on COCO [57] and use a learning rate of $\eta = 0.001$. In this stage, we freeze all layers except our network heads and train the network for 50 epochs. In the second stage, we decrease the learning rate by a factor of 5 and train all layers above ResNet-101 layer 4 for another 25 epochs. In the last stage, we drop the learning rate again by a factor of 5 and fine-tune the entire network for 25 epochs. In total, the networks trains for 100 epochs in a time span between 20 and 37 hours, as shown in Figure 4.18.

We train the four methods *Semantic + Instance*, *Panoptic FPN*, *HPS*, and *HPS + ISI* described at the beginning of chapter 4 once before evaluation. The hyper-parameters were selected based on the best testing performance on *Panoptic FPN* [43], which acts as the baseline for this thesis. *Semantic + Instance* and *Panoptic FPN* do not have a single panoptic segmentation output. Hence, we use the simple heuristic described in the original paper to merge the semantic segmentation and instance segmentation outputs [44].

4.3.2.2 Training Time Comparison

First, we compare the training times of different approaches. The time reduction achieved by combining two separate neural networks (*Semantic + Instance*) into a single network (*Panoptic FPN*, *HPS*, and *HPS + ISI*) can be seen in Figure 4.18. Similar as for the TOY-Shapes dataset (see Section 4.3.1.2), the training time is nearly halved, when using a shared backbone. Merging the information of semantic segmentation and instance segmentation internally with our methods, as described in Chapter 3, only adds 60 minutes for *HPS* to the total training time. Using additional inter-task relations only adds 66 minutes to the

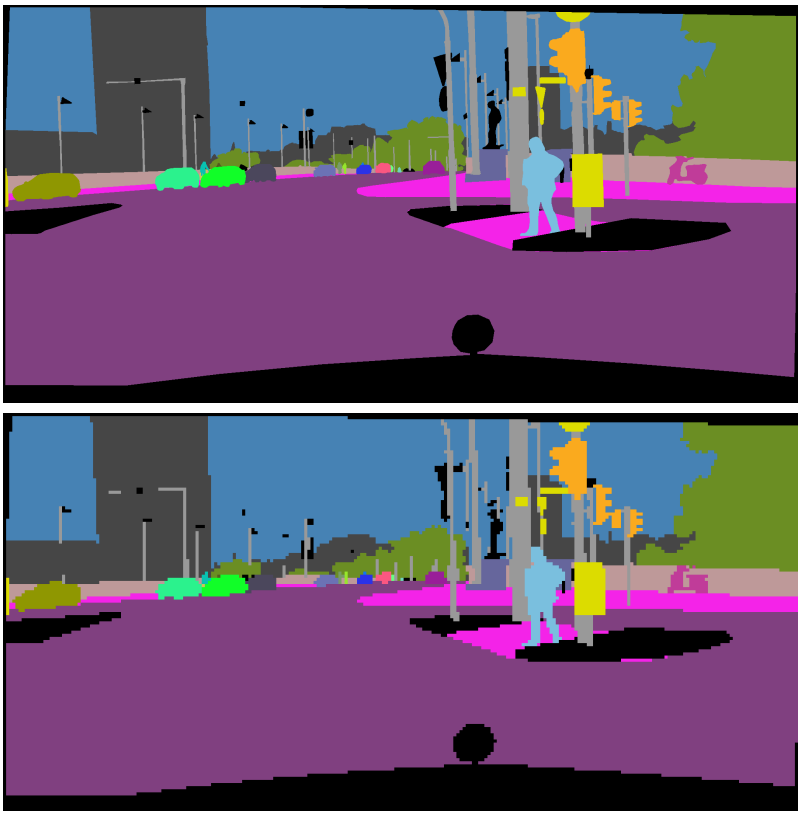


Figure 4.17: Example of a 8× downsampled Cityscape image. **Top:** is the original [2048×1024×3] image. **Bottom:** is the 8× nearest-neighbor downsampled [256 × 128 × 3] image. On the left side, we can see vanishing structures of small objects, such as the pole.

total training time for *HPS + ISI*. Compared to the TOY-Shapes dataset, the Cityscapes dataset has more images and a larger number of maximum instances per image, i.e., 30 instead of 10. Thus, training requires more time.

4.3.2.3 Semantic Segmentation

In this section, we discuss the semantic segmentation performance of different approaches. The corresponding results under the *IoU* metric (see Section 4.1.1) are shown in Table 4.5.

In contrast to the simple TOY-Shapes dataset results, the challenging Cityscapes dataset is more complex. Thus, we see differences in the performance of individual methods. We observe that our *HPS + ISI* method is more useful on this complex dataset than on the much simpler TOY-Shapes dataset. We significantly improve the *IoU* score for many *things* classes and also slightly improve the performance of many *stuff* classes. The additional prior information of the instance segmentation branch in the form of an *ISI* increases the semantic segmentation task on this dataset. We gain the largest accuracy improvement for classes that are closely related, such as *bus*, *truck*, or *train*. We also

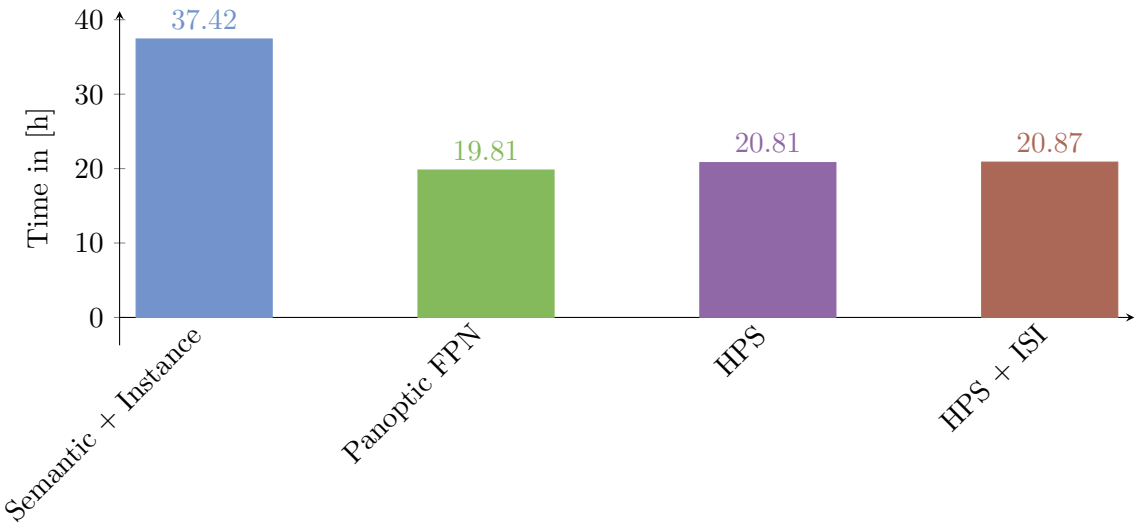


Figure 4.18: Cityscapes dataset training time comparison. We can see that training a single neural network with a shared backbone reduces training time by nearly half as expected. Adding additional complexity for the generation of *ISI* does not increase the time significantly.

observe improvements for predictions of the *rider* class which is the only class that does not occur in the COCO dataset [57].

Finally, we show qualitative examples of our results for all four evaluated methods in Figure 4.19.

4.3.2.4 Instance Segmentation

In this section, we discuss the instance segmentation performance of different approaches. The corresponding results under the *mAP* metric (see Section 4.1.2) are shown in Table 4.6.

We observe that the shared backbone in *Panoptic FPN* slightly increases the *mAP* compared to two disjoint networks *Semantic + Instance*. Our methods (*HPS* and *HPS + ISI*) that add more intelligence to the semantic segmentation branch do not improve the instance segmentation predictions. In this work, we focus on an end-to-end trainable network for panoptic segmentation and improve the semantic segmentation branch with additional interrelations. Hence, we do not expect large changes in pure instance segmentation performance.

Due to the rectangular region proposals in Mask R-CNN [31], we observe the same issues for occluded objects as already discussed for the TOY-Shapes dataset in Section 4.3.1.4. However, the Cityscapes dataset is far more complex and presents real-world scenarios, where even more objects overlap, occlude each other, and appear in different sizes. This makes it hard to find the right anchor box sizes and results in low *mAP*. Finally, we show qualitative examples of our results for all four evaluated methods in Figure 4.20.

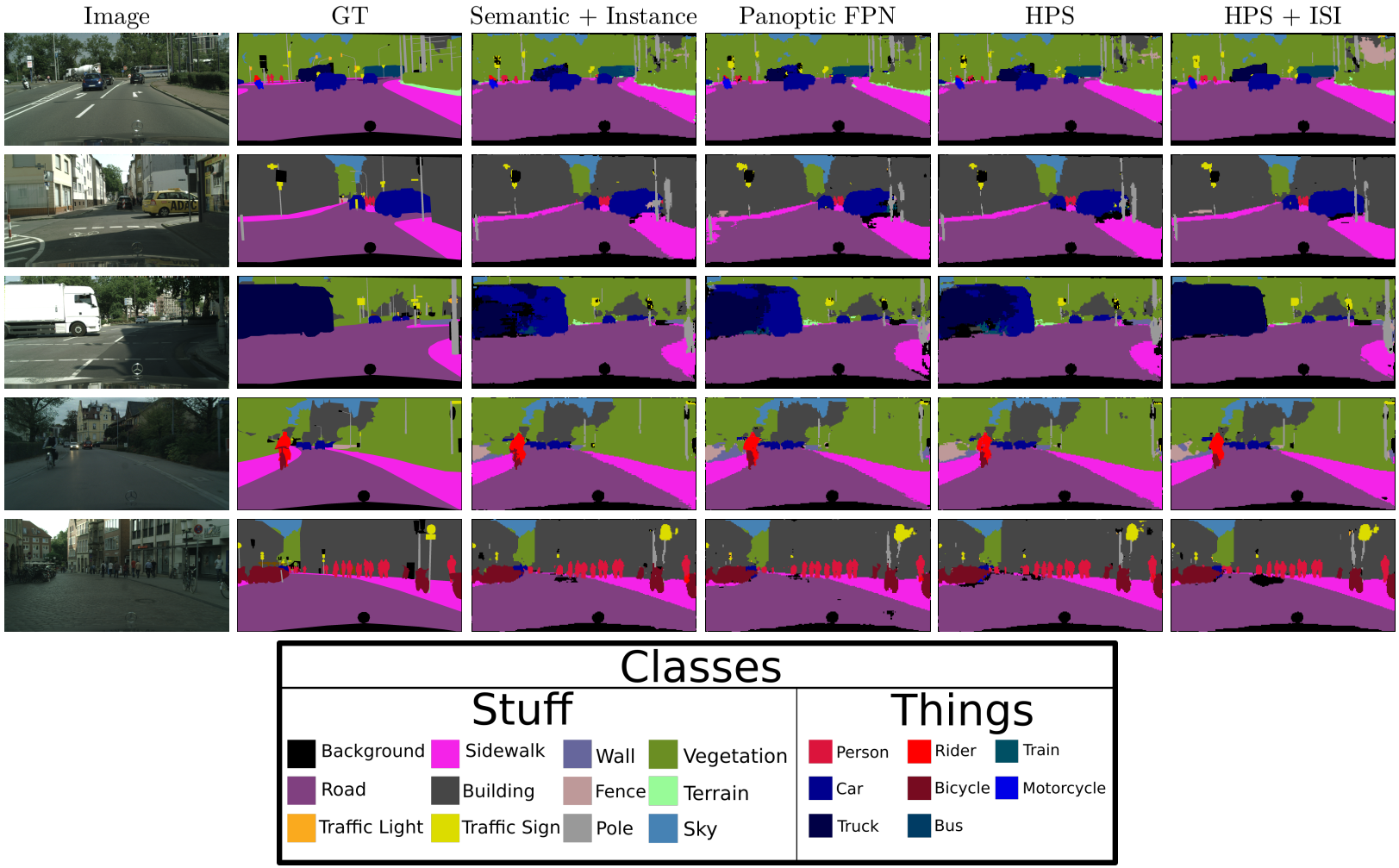


Figure 4.19: Qualitative results on the Cityscapes dataset. Compared to *Panoptic FPN*, *HPS + ISI* is less sensitive to speckle noise in semantically coherent regions. Additionally, we reduce confusion between classes with similar semantics. **Best viewed in digital zoom.**

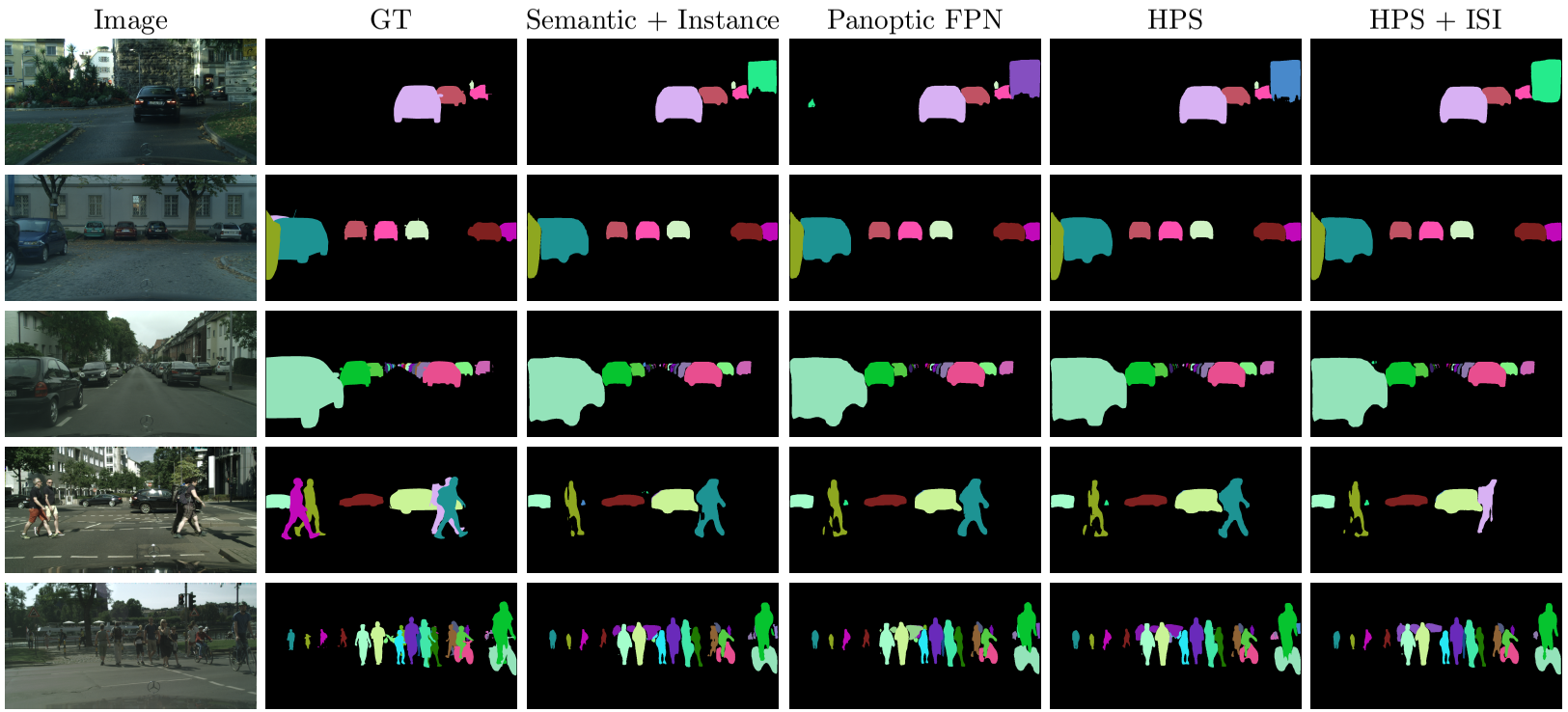


Figure 4.20: Qualitative results on the Cityscapes dataset. The results show variance in the instance segmentation branch, which finds different Regions of Interest (RoIs) depending on the used approach. From the visual inspection and the quantitative results, we conclude that the single network structure has a minor impact on the instance segmentation. Overall our approach does not change the instance segmentation branch, hence we did not expect an improvement on this metric. **Best viewed in digital zoom.**

Semantic Segmentation: IoU

Class	Semantic + Instance	Panoptic FPN	HPS	HPS + ISI
Road	95.1	93.6	95.0	**95.3**
Sidewalk	69.2	67.8	68.9	**69.8**
Building	86.3	86.0	86.3	**86.5**
Wall	41.5	38.7	39.9	**42.1**
Fence	35.2	34.7	**35.9**	34.4
Pole	36.3	35.3	37.4	**37.6**
Traffic Light	**46.6**	44.8	45.7	45.9
Traffic Sign	50.4	48.7	51.1	**52.2**
Vegetation	87.6	87.5	87.7	**87.8**
Terrain	48.7	47.6	**50.0**	49.5
Sky	90.2	90.2	**90.4**	90.1
Person	65.2	65.2	65.2	**68.9**
Rider	37.3	37.1	34.5	**51.1**
Car	89.2	89.1	89.1	**91.2**
Truck	54.8	49.1	49.1	**65.6**
Bus	68.9	67.5	67.2	**83.8**
Train	53.1	47.7	48.5	**73.3**
Motorcycle	**50.8**	45.0	48.5	50.7
Bicycle	59.6	59.2	58.9	**61.7**
mIoU	61.4	60.4	60.5	**65.1**

Table 4.5: Quantitative results on the Cityscapes dataset. The results show that our *HPS + ISI* method significantly improves the performance of *things* classes compared to other methods.

4.3.2.5 Segmentation Panoptic

In this section, we discuss the panoptic segmentation performance of different approaches. The corresponding results under the PQ metric (see Section 2.2.5) are shown in Tables 4.7 and 4.8. In addition, to the PQ, we show the SQ and the RQ for all classes, *things* (Th) classes only, and *stuff* (St) classes only. Since PQ is a measurement of semantic (SQ) and instance (RQ) segmentation quality an improvement in either part will increase the accuracy of the overall system.

Interestingly, *Semantic + Instance* performs worse than *Panoptic FPN*. We hypothesize that this is because the number of training images in Cityscapes is rather low. Thus, the shared feature backbone of *Panoptic FPN* acts as a regularizer which reduces overfitting compared to two individual full networks on this dataset.

Next, *HPS* improves upon *Panoptic FPN* across all metrics and classes, because we optimize for the final panoptic segmentation output. Our system minimizes a panoptic loss in addition to the semantic and instance segmentation losses which provides better guidance for the network. In this way, we do not rely on the heuristic merging of subtask

Mean Average Precision: mAP

Class	Semantic + Instance		Panoptic FPN		HPS		HPS + ISI	
	AP	**AP@50**	**AP**	**AP@50**	**AP**	**AP@50**	**AP**	**AP@50**
Person	18.8	42.8	**19.4**	**43.8**	19.0	42.7	19.0	43.5
Rider	12.7	38.0	**14.4**	**41.0**	13.0	38.2	12.5	37.5
Car	38.9	61.3	**39.3**	**61.8**	39.3	61.4	38.1	60.6
Truck	22.3	36.5	25.0	36.7	**25.7**	**40.0**	24.2	39.6
Bus	39.1	56.5	38.1	53.3	43.6	61.6	**43.9**	**62.6**
Train	19.5	33.3	**27.6**	**57.4**	27.4	57.0	26.1	52.1
Motorcycle	8.37	24.3	8.36	25.6	**9.83**	**29.7**	8.19	20.6
Bicycle	9.32	30.6	**9.77**	**32.1**	9.70	30.9	9.40	31.1
mAP	21.1	40.4	22.7	44.0	**23.4**	**45.2**	22.6	43.4

Table 4.6: Quantitative results on the Cityscapes dataset. The results show small improvements in accuracy for using a shared backbone structure (*Semantic + Instance* vs. *Panoptic FPN*). The accuracy on the instance segmentation task does not improve further with our methods (*HPS* and *HPS + ISI*) because all methods use the same instance segmentation branch architecture.

predictions but directly generate the desired output internally which results in improved accuracy in practice. Finally, *HPS + ISI* outperforms all other methods because it additionally leverages inter-task relations. Compared to *Panoptic FPN, HPS + ISI* improves *PQ* by +4% relative from 41.9 to 43.5. Providing instance segmentation predictions as additional feature input for the semantic segmentation branch gives a segmentation prior. By exploiting this prior, the semantic segmentation branch can focus more on the prediction of stuff classes and boundaries between individual classes which results in improved accuracy across all metrics.

This quantitative improvement is also reflected qualitatively, as shown in Figure 4.22. We observe that *HPS + ISI* handles occlusions more accurately (1^{st} row) and resolves overlapping issues on its own while being less sensitive to speckle noise in semantically coherent regions (2^{nd} row). Thanks to our end-to-end training and inter-task relations, we predict more accurate semantic label transitions (3^{rd} row) and reduce confusion between classes with similar semantic meaning like *bus* and *car* (4^{th} row). Finally, we show qualitative examples of our results for all four evaluated methods in Figure 4.21.

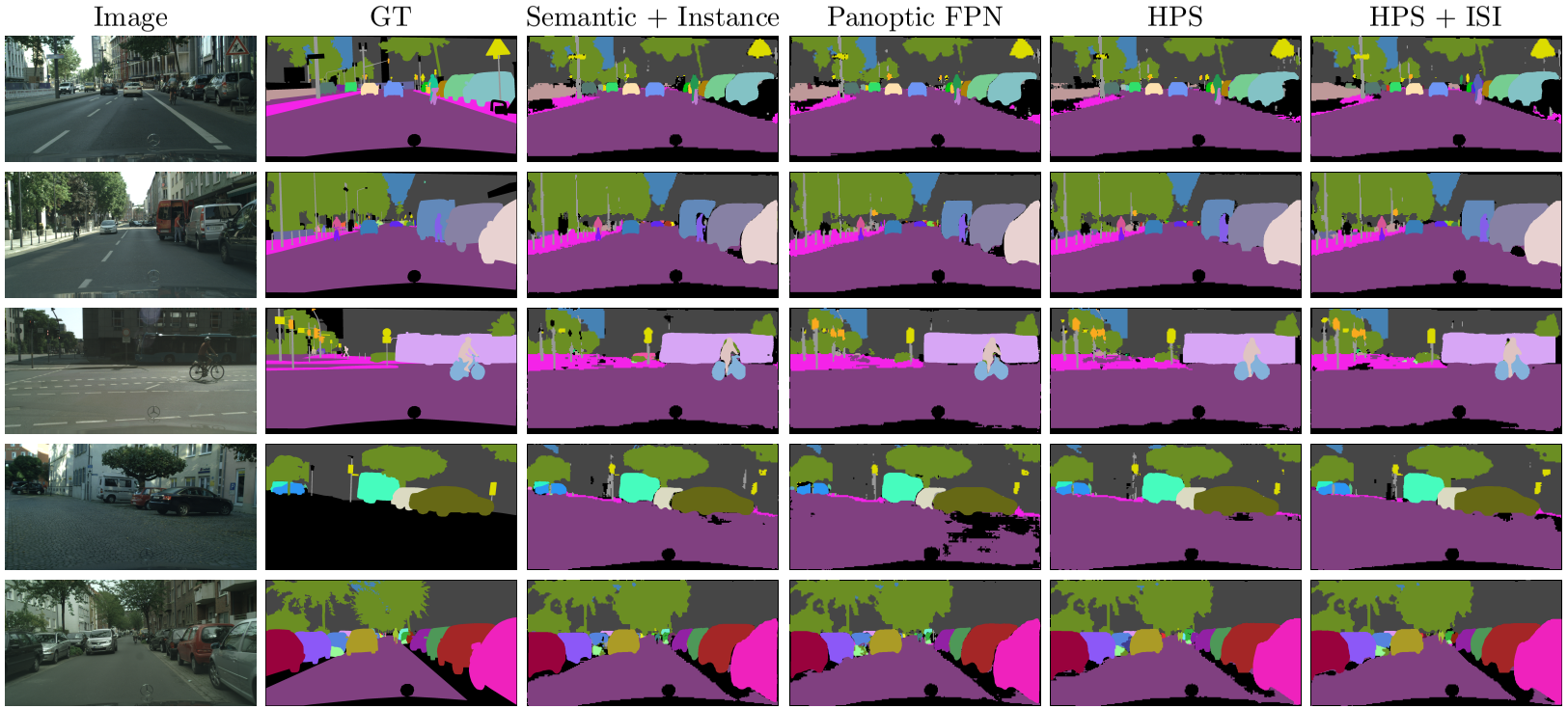


Figure 4.21: Qualitative results on the Cityscapes dataset. Compared to *Panoptic FPN*, *HPS + ISI* handles occlusions more accurately and is less sensitive to speckle noise in semantically coherent regions. Additionally, we predict more accurate semantic label transitions. Both our end-to-end training as well as inter-task relations increase panoptic quality. **Best viewed in digital zoom.**

Class	Panoptic Quality: PQ											
	Semantic + Instance			Panoptic FPN			HPS			HPS + ISI		
	PQ	SQ	RQ	PQ	SQ	RQ	PQ	SQ	RQ	PQ	SQ	RQ
Road	94.7	94.8	99.8	92.7	93.5	99.1	94.7	94.9	99.9	**95.0**	**95.1**	**99.9**
Sidewalk	54.5	74.6	73.0	53.0	74.2	71.5	55.4	**75.0**	73.9	**57.2**	74.7	**76.5**
Building	78.0	82.3	**94.7**	77.4	82.3	94.1	78.4	**83.0**	94.5	**78.4**	82.9	94.5
Wall	9.26	**68.4**	13.5	7.86	65.4	12.0	9.24	65.7	14.1	**11.0**	68.2	**16.1**
Fence	8.06	64.1	12.5	8.45	64.4	13.1	8.94	**65.8**	13.6	**10.3**	64.6	**15.9**
Pole	6.36	57.0	11.1	6.06	55.5	10.9	**7.86**	56.6	**13.9**	7.6	**57.4**	13.2
Traffic Light	13.8	**62.3**	22.2	13.1	60.8	21.5	13.5	62.2	21.8	**15.1**	61.9	**24.4**
Traffic Sign	22.7	64.9.7	35.0	20.8	64.3	32.4	24.4	**66.1**	37.0	**28.6**	64.9	**44.1**
Vegetation	82.8	84.9	97.4	82.4	84.7	97.2	83.1	**85.5**	97.2	**83.3**	85.4	**97.4**
Terrain	16.3	67.4	24.2	14.2	66.1	21.5	16.0	68.1	23.5	**16.8**	**68.8**	**24.5**
Sky	77.0	**87.4**	88.0	77.3	87.0	88.9	77.2	87.1	88.6	**77.6**	86.9	**89.3**
Person	38.5	72.4	53.2	39.6	72.6	54.5	**40.1**	73.2	**54.8**	39.7	**73.5**	54.0
Rider	34.3	67.6	50.8	**35.9**	67.8	**52.9**	35.1	67.8	51.7	34.3	**68.0**	50.4
Car	54.8	80.2	68.2	55.4	80.3	69.0	56.1	**81.1**	69.2	**56.1**	80.8	**69.5**
Truck	39.8	82.7	39.8	41.2	**84.4**	48.8	40.7	82.0	49.7	**43.7**	82.7	**52.9**
Bus	56.4	85.9	65.6	57.6	86.4	66.6	61.2	85.9	71.3	**63.1**	**86.9**	**72.6**
Train	39.9	79.9	50.0	**52.5**	76.4	**68.7**	50.7	78.3	64.7	48.9	**80.6**	60.6
Motorcycle	28.7	69.1	41.6	30.8	68.4	44.9	**32.4**	**70.0**	**46.3**	29.0	69.6	41.6
Bicycle	30.3	65.4	46.3	30.8	65.4	47.0	31.3	**67.9**	46.2	**31.6**	66.6	**47.4**

Table 4.7: Quantitative results on the Cityscapes dataset. The results show that a shared feature backbone reduces overfitting compared to two disjoint networks (*Semantic + Instance* vs *Panoptic FPN*). Also, generating the final panoptic output internally and training the system end-to-end increases the performance (*Panoptic FPN* vs *HPS*). Finally, using inter-task relations in the form of an initial segmentation image (ISI) provides an effective segmentation prior and increases the overall panoptic quality (*HPS* vs *HPS + ISI*).

Method	PQ	SQ	RQ	PQ^{Things}	SQ^{Things}	RQ^{Things}	PQ^{Stuff}	SQ^{Stuff}	RQ^{Stuff}
Semantic + Instance	41.4	74.3	52.4	40.3	75.4	53.0	42.1	73.5	51.9
Panoptic FPN	41.9	73.7	53.4	43.0	75.2	56.6	41.2	72.5	51.1
HPS	42.9	74.5	54.3	**43.4**	75.7	**56.7**	42.6	73.6	52.5
HPS + ISI	**43.5**	**74.7**	**54.9**	43.2	**76.1**	56.1	**43.7**	**73.7**	**54.1**

Table 4.8: Quantitative results on the Cityscapes dataset. The results show that a shared feature backbone reduces overfitting compared to two disjoint networks (*Semantic + Instance* vs *Panoptic FPN*). Also, generating the final panoptic output internally and training the system end-to-end increases the performance (*Panoptic FPN* vs *HPS*). Finally, using inter-task relations in the form of an initial segmentation image (ISI) provides an effective segmentation prior and increases the overall panoptic quality (*HPS* vs *HPS + ISI*).

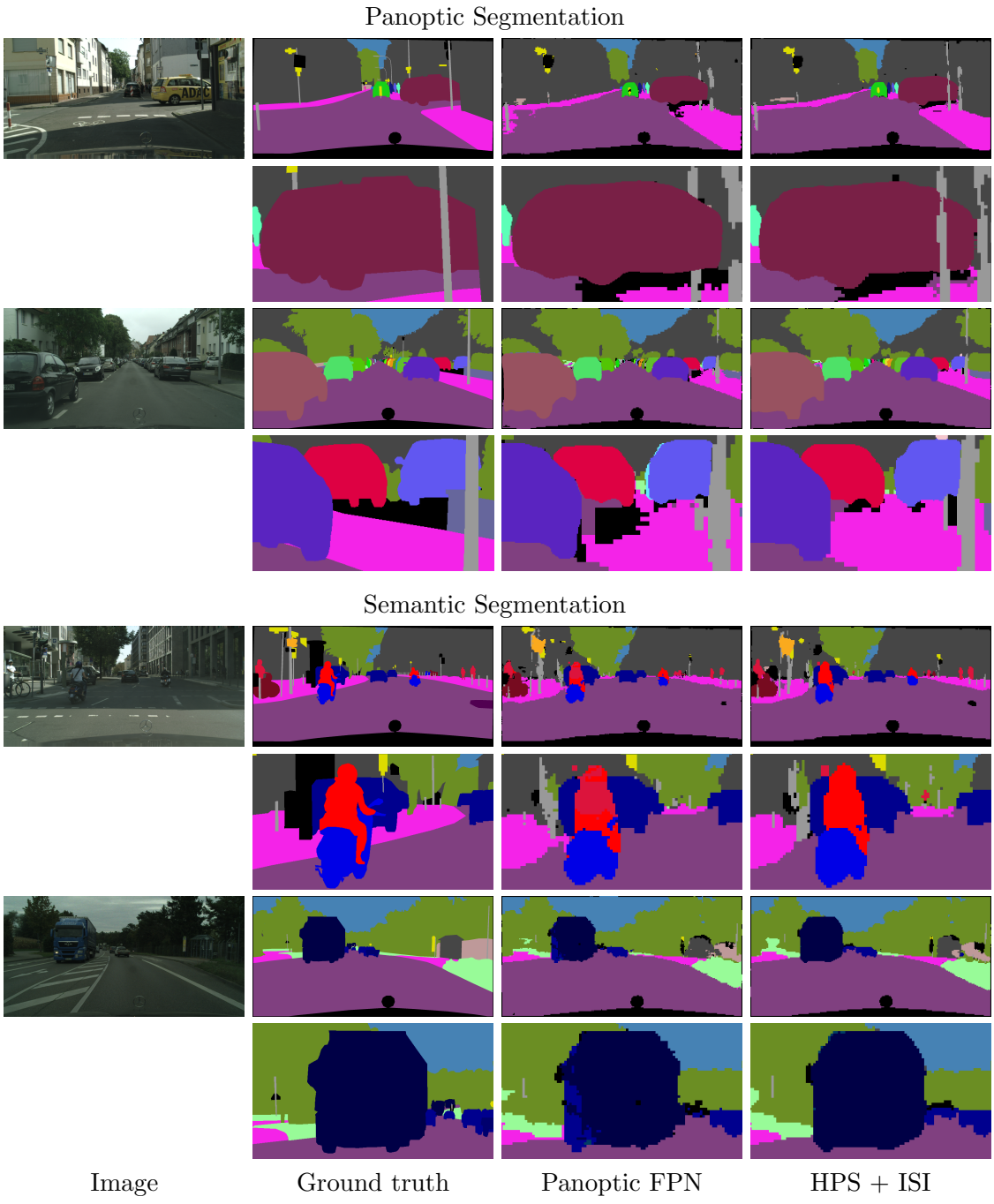


Figure 4.22: Qualitative results on the Cityscapes dataset. Compared to *Panoptic FPN*, *HPS + ISI* handles occlusions more accurately (1st row) and is less sensitive to speckle noise in semantically coherent regions (2nd row). Additionally, we predict more accurate semantic label transitions (3rd row) and reduce confusion between classes with similar semantic meaning like *rider* and *person* or *bus* and *car* (4th row). Both our end-to-end training as well as inter-task relations increase panoptic quality. **Best viewed in digital zoom.**

5 Conclusion

Panoptic Segmentation (PS) is an important but challenging problem with high relevance in practice. *PS* addresses the task of complete 2D scene segmentation by assigning a class label to each pixel of an image while differentiating between instances within a common class. Thus, it can be seen as a unification of semantic segmentation and instance segmentation. In recent years the task of *PS* gained popularity because the Panoptic Quality (PQ) [44] metric made it possible to assess the performance of complete 2D scene segmentation in an unified manner and publicly available datasets like Microsoft COCO [57], Mapillary Vistas [66], and Cityscapes [14] added the panoptic segmentation challenge to their benchmarks.

Many *PS* methods approach the task by independently addressing semantic and instance segmentation and generating the panoptic output using heuristics. However, this strategy has two limitations. First, the system cannot be optimized for the final objective in an end-to-end manner. Second, mutual information between the semantic and instance segmentation tasks is not fully exploited. To overcome these limitations, we propose a single end-to-end trainable network architecture and directly optimize our system using a panotpic objective. Moreover, we present a way to share mutual information between the two subtasks by providing instance segmentation predictions as additional feature input for our semantic segmentation branch. This inter-task link allows us to exploit a segmentation prior and improve the overall panoptic quality.

Our results show that our proposed holistic *PS* method achieves significant improvements compared to previous architectures with only a negligible increase in computational complexity during training. Quantitatively, we show that both end-to-end training and inter-task relations result in improved *PQ* in practice. Qualitatively, our method creates more accurate label transitions between individual classes while being less sensitive to speckle noise in semantically coherent regions. Our approach performs especially well on complex datasets like Cityscapes [14] that contain images with many objects at different scales and difficult occlusion scenarios. In these cases, our method achieves significantly

improved PQ because our network learns to resolve overlapping issues between neighboring classes on its own. In many cases, our approach does not only compute more accurate PS predictions but also improved semantic and instance segmentation predictions.

Our experiments show that our method performs well for large objects and regions but struggles to make accurate predictions for small objects and regions. This can be accounted to the fact that we process images at reduced spatial resolution due to GPU memory limitations. Additionally, there is a bias in the optimization towards minimizing the error on large objects and regions because these occupy more pixels and, thus, contribute stronger to the loss. Moreover, one limitation of our approach is that the our employed instance segmentation is based on Region-based Convolutional Neural Networks (R-CNNs) that can only detect a single object per region. In many cases, small or strongly occluded objects are not detected because only one prominent instance is detected in each region.

In the Cityscapes [14] dataset, we found some minor flaws that are considered as label noise in this work but should be explicitly addressed in future projects. First, many images contain transparent subregions like car windows or glass walls of bus stations. These subregions belong to regions of a specific classes but actually show content that is behind the labeled class. According to the ground truth annotation, the see-through region should be labeled as the foreground region itself but our network predicts the visible region behind. Second, there are ambiguous regions that show subregions of other classes. For example, there are posters and t-shirt that show printed persons or cars which our method detects as distinct object instances.

A possible future research direction and a way to resolve some of the issues described above is to add depth sensor information as additional input to our approach. In this way, transparent regions could be disambiguated. Another promising future research direction is to exploit semantic segmentation predictions as additional feature input to the instance segmentation branch to improve instance detection accuracy. Additionally, instead of discarding the *things* predictions of the semantic segmentation branch, they could be taken into account during the generation of the final panoptic output to improve the prediction. Finally, our work showed that sharing information between semantic and instance segmentation indeed improves the panoptic performance and marks a first step towards fully entangled PS in the future.

List of Figures

List of Tables

A

List of Acronyms

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
FCN	Fully Convolutional Network
FN	False Negative
FNN	Feed-forward Neural Network
FP	False Positive
FPN	Feature Pyramid Network
IoU	Intersection Over Union
ISI	Initial Segmentation Image
mAP	Mean Average Precision
MNN	Multi-task Neural Network
MSE	Mean Squared Error
PQ	Panoptic Quality
PS	Panoptic Segmentation
R-CNN	Region-based Convolutional Neural Network
ReLU	Rectified Linear Unit
ResNet	Residual Neural Network
RNN	Recurrent Neural Network
RoI	Region of Interest
RPN	Region Proposal Network
RQ	Recognition Quality
SGD	Stochastic Gradient Descent
SQ	Segmentation Quality
TLU	Threshold Logic Unit
TP	True Positive

Bibliography

[1] Abdulla, W. H. (2017). Mask R-CNN for Object Detection and Instance Segmentation on Keras and TensorFlow. `https://github.com/matterport/Mask_RCNN`. (page 37)

[2] Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495. (page 29)

[3] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). SURF: Speeded up Robust Features. In *European Conference on Computer Vision*, pages 404–417. (page 22)

[4] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer. (page 18)

[5] Blaschke, T. (2010). Object Based Image Analysis for Remote Sensing. *Journal of Photogrammetry Engineering and Remote Sensing*, 65(1):2–16. (page 24)

[6] Bolya, D., Zhou, C., Xiao, F., and Lee, Y. J. (2019). YOLACT: Real-Time Instance Segmentation. In *International Conference on Computer Vision*, pages 9157–9166. (page)

[7] Brachmann, E., Krull, A., Nowozin, S., Shotton, J., Michel, F., Gumhold, S., and Rother, C. (2017). Dsac-Differentiable Ransac for Camera Localization. In *Conference on Computer Vision and Pattern Recognition*, pages 6684–6692. (page 36)

[8] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2017a). Deeplab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848. (page 29)

[9] Chen, L.-C., Papandreou, G., Schroff, F., and Adam, H. (2017b). Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv:1706.05587*. (page 1, 29, 30)

[10] Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. (2018). Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In *European Conference on Computer Vision*, pages 801–818. (page)

[11] Chen, S.-H., Hwang, S.-H., and Wang, Y.-R. (1998). An RNN-Based Prosodic Information Synthesizer for Mandarin Text-to-Speech. *Transactions on Speech and Audio Processing*, 6(3):226–239. (page 8)

[12] Çiçek, Ö., Abdulkadir, A., Lienkamp, S. S., Brox, T., and Ronneberger, O. (2016). 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. In *Medical Image Computing and Computer-Assisted Intervention*, pages 424–432. (page)

[13] Cordts, M. and Omran, M. (2016). The Cityscapes Dataset Evaluation Scripts. `https://github.com/mcordts/cityscapesScripts`. Accessed: 2020-03-11. (page 46)

[14] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Conference on Computer Vision and Pattern Recognition*, pages 3213–3223. (page 3, 39, 45, 53, 71, 72)

[15] Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals and Systems*, 2(4):303–314. (page 7)

[16] Dalal, N. and Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. In *Conference on Computer Vision and Pattern Recognition*, pages 886–893. (page 22, 24)

[17] Dong, C., Loy, C. C., and Tang, X. (2016). Accelerating the Super-Resolution Convolutional Neural Network. In *European Conference on Computer Vision*, pages 391–407. (page 11, 12)

[18] Eitel, A., Springenberg, J. T., Spinello, L., Riedmiller, M., and Burgard, W. (2015). Multimodal Deep Learning for Robust RGB-D Object Recognition. In *International Conference on Intelligent Robots and Systems*, pages 681–687. (page 22)

[19] Fan, Y., Lu, X., Li, D., and Liu, Y. (2016). Video-Based Emotion Recognition Using CNN-RNN and C3D Hybrid Networks. In *International Conference on Multimodal Interaction*, pages 445–450. (page 8)

[20] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2009). Object Detection with Discriminatively Trained Part-Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645. (page 24)

[21] Feng, D., Haase-Schuetz, C., Rosenbaum, L., Hertlein, H., Duffhauss, F., Glaeser, C., Wiesbeck, W., and Dietmayer, K. (2019). Deep Multi-Modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges. *arXiv:1902.07830*. (page 1, 30)

[22] Fu, C.-Y., Berg, T. L., and Berg, A. C. (2019). IMP: Instance Mask Projection for High Accuracy Semantic Segmentation of Things. In *International Conference on Computer Vision*, pages 5178–5187. (page 32)

[23] Girshick, R. (2015). Fast R-CNN. In *International Conference on Computer Vision*, pages 1440–1448. (page 18, 25, 26, 27, 28)

[24] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Conference on Computer Vision and Pattern Recognition*, pages 580–587. (page 11, 12, 25, 26)

[25] Gkioxari, G., Malik, J., and Johnson, J. (2019). Mesh R-CNN. In *International Conference on Computer Vision*, pages 9785–9795. (page 13, 24)

[26] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. (page 5, 8, 14, 17)

[27] Gordon, I. and Lowe, D. G. (2006). What and Where: 3D Object Recognition with Accurate Pose. In *Toward Category-Level Object Recognition*, pages 67–82. Springer. (page 22)

[28] Grabner, A., Roth, P. M., and Lepetit, V. (2018). 3D Pose Estimation and 3d Model Retrieval for Objects in the Wild. In *Conference on Computer Vision and Pattern Recognition*, pages 3022–3031. (page 11, 13, 24)

[29] Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., and Seung, S. H. (2000). Digital Selection and Analogue Amplification Coexist in a Cortex-Inspired Silicon Circuit. *Nature*, 405(6789):947–951. (page 34)

[30] Hassoun, M. H. et al. (1995). *Fundamentals of Artificial Neural Networks*. MIT Press. (page 7)

[31] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask R-CNN. In *International Conference on Computer Vision*, pages 2961–2969. (page 1, 27, 28, 30, 34, 35, 47, 51, 58, 60)

[32] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1904–1916. (page 25)

[33] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 770–778. (page 5, 7, 11, 23, 28, 34, 39)

[34] Hebb, D. O. (1995). *The Organization of Behavior: A Neuropsychological Theory*. Psychology Press. (page 7)

[35] Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780. (page 8)

[36] Hornik, K. (1991). Approximation Capabilities of Multilayer Feedforward Networks. *Neural Networks*, 4(2):251–257. (page 7)

[37] Huang, K., Wang, L., Tan, T., and Maybank, S. (2008). A Real-Time Object Detecting and Tracking System for Outdoor Night Surveillance. *Pattern Recognition*, 41(1):432–444. (page 24)

[38] Hubel, D. H. (1982). Exploration of the Primary Visual Cortex, 1955–78. *Nature*, 299(5883):515–524. (page 9)

[39] Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167*. (page 34)

[40] Jain, A. K. and Vailaya, A. (1996). Image Retrieval Using Color and Shape. *Pattern Recognition*, 29(8):1233–1244. (page 24)

[41] Janocha, K. and Czarnecki, W. M. (2017). On Loss Functions for Deep Neural Networks in Classification. *arXiv:1702.05659*. (page 17)

[42] Jordan, M. I. (1986). Attractor Dynamics and Parallelism in a Connectionist Sequential Machine. *Conference on Cognitive Science Society*, pages 531–546. (page 8)

[43] Kirillov, A., Girshick, R., He, K., and Dollár, P. (2019a). Panoptic Feature Pyramid Networks. In *Conference on Computer Vision and Pattern Recognition*, pages 6399–6408. (page 2, 31, 34, 35, 39, 47, 58)

[44] Kirillov, A., He, K., Girshick, R., Rother, C., and Dollár, P. (2019b). Panoptic Segmentation. In *Conference on Computer Vision and Pattern Recognition*, pages 9404–9413. (page 1, 30, 39, 42, 47, 53, 58, 71)

[45] Kirillov, A., Levinkov, E., Andres, B., Savchynskyy, B., and Rother, C. (2017). InstanceCut: From Edges to Instances with MultiCut. In *Conference on Computer Vision and Pattern Recognition*, pages 5008–5017. (page 1, 27, 30)

[46] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105. (page 22, 23)

[47] Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F., and Navab, N. (2016). Deeper Depth Prediction with Fully Convolutional Residual Networks. In *International Conference on 3D Vision*, pages 239–248. (page 12)

[48] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324. (page 5)

[49] LeCun, Y., Haffner, P., Bottou, L., and Bengio, Y. (1999). Object Recognition with Gradient-Based Learning. In *Shape, Contour and Grouping in Computer Vision*, pages 319–345. Springer. (page 8)

[50] Li, H., Deklerck, R., De Cuyper, B., Hermanus, A., Nyssen, E., and Cornelis, J. (1995). Object Recognition in Brain CT-Scans: Knowledge-Based Fusion of Data From Multiple Feature Extractors. *IEEE Transactions on Medical Imaging*, 14(2):212–229. (page 22)

[51] Li, H., Xiong, P., An, J., and Wang, L. (2018a). Pyramid Attention Network for Semantic Segmentation. *arXiv:1805.10180*. (page)

[52] Li, J., Raventos, A., Bhargava, A., Tagawa, T., and Gaidon, A. (2018b). Learning to Fuse Things and Stuff. *arXiv:1812.01192*. (page 31)

[53] Li, Y., Chen, X., Zhu, Z., Xie, L., Huang, G., Du, D., and Wang, X. (2019). Attention-Guided Unified Network for Panoptic Segmentation. In *Conference on Computer Vision and Pattern Recognition*, pages 7026–7035. (page 31)

[54] Li, Y., Qi, H., Dai, J., Ji, X., and Wei, Y. (2017). Fully Convolutional Instance-Aware Semantic Segmentation. In *Conference on Computer Vision and Pattern Recognition*, pages 2359–2367. (page 1, 27, 30)

[55] Liang, X., Lin, L., Wei, Y., Shen, X., Yang, J., and Yan, S. (2017). Proposal-Free Network for Instance-Level Object Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2978–2991. (page 27, 30)

[56] Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017). Feature Pyramid Networks for Object Detection. In *Conference on Computer Vision and Pattern Recognition*, pages 2117–2125. (page 28, 34)

[57] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, L. C. (2014). Microsoft COCO: Common Objects in Context. In *European Conference on Computer Vision*, pages 740–755. (page 30, 47, 58, 60, 71)

[58] Liu, S., Jia, J., Fidler, S., and Urtasun, R. (2017). SGN: Sequential Grouping Networks for Instance Segmentation. In *International Conference on Computer Vision*, pages 3496–3504. (page 1, 27, 30)

[59] Liu, S., Qi, L., Qin, H., Shi, J., and Jia, J. (2018). Path Aggregation Network for Instance Segmentation. In *Conference on Computer Vision and Pattern Recognition*, pages 8759–8768. (page 27, 30)

[60] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). SSD: Single Shot Multibox Detector. In *European Conference on Computer Vision*, pages 21–37. (page 25)

[61] Logothetis, N. K. and Sheinberg, D. L. (1996). Visual Object Recognition. *Annual Review of Neuroscience*, 19(1):577–621. (page 22)

[62] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully Convolutional Networks for Semantic Segmentation. In *Conference on Computer Vision and Pattern Recognition*, pages 3431–3440. (page 1, 11, 28, 29, 30)

[63] Lowe, D. G. (1999). Object Recognition From Local Scale-Invariant Features. In *International Conference on Computer Vision*, pages 1150–1157. (page 22)

[64] Mammone, R. J. (1994). *Artificial Neural Networks for Speech and Vision*, volume 4. Chapman & Hall. (page 5, 8)

[65] McCulloch, W. S. and Pitts, W. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133. (page 6, 7)

[66] Neuhold, G., Ollmann, T., Bulo, S. R., and Kontschieder, P. (2017). The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes. In *International Conference on Computer Vision*, pages 4990–4999. (page 71)

[67] Pearlmutter, B. A. (1989). Learning State Space Trajectories in Recurrent Neural Networks. *Neural Computation*, 1(2):263–269. (page 8)

[68] Perez, L. and Wang, J. (2017). The Effectiveness of Data Augmentation in Image Classification Using Deep Learning. *arXiv:1712.04621*. (page 43)

[69] Petrovai, A. and Nedevschi, S. (2019). Multi-Task Network for Panoptic Segmentation in Automated Driving. In *Intelligent Transportation Systems Conference*, pages 2394–2401. (page 1, 30)

[70] Quinlan, R. J. (1986). Induction of Decision Trees. *Machine Learning*, 1(1):81–106. (page 22)

[71] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *Conference on Computer Vision and Pattern Recognition*, pages 779–788. (page 25)

[72] Redmon, J. and Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. In *Conference on Computer Vision and Pattern Recognition*, pages 7263–7271. (page)

[73] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems*, pages 91–99. (page 18, 25, 26, 27, 28)

[74] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. (page 1, 12, 28, 29, 30)

[75] Rosenblatt, F. (1958). The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6):386. (page 7)

[76] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Representations by Back-Propagating Errors. *Nature*, 323(6088):533–536. (page 8, 14)

[77] Sermanet, P. and LeCun, Y. (2011). Traffic Sign Recognition with Multi-Scale Convolutional Networks. In *International Joint Conference on Neural Networks*, pages 2809–2813. (page 22)

[78] Simonyan, K. and Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556*. (page 7, 11, 23)

[79] Sun, D., Yang, X., Liu, M.-Y., and Kautz, J. (2018). Pwc-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume. In *Conference on Computer Vision and Pattern Recognition*, pages 8934–8943. (page 11)

[80] Suykens, J. A. and Vandewalle, J. (1999). Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, pages 293–300. (page 22)

[81] Swets, D. L. and Weng, J. J. (1996). Using Discriminant Eigenfeatures for Image Retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):831–836. (page 22, 24)

[82] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going Deeper With Convolutions. In *Conference on Computer Vision and Pattern Recognition*, pages 1–9. (page 23)

[83] Tighe, J., Niethammer, M., and Lazebnik, S. (2014). Scene Parsing with Object Instances and Occlusion Ordering. In *Conference on Computer Vision and Pattern Recognition*, pages 3748–3755. (page 30)

[84] Tu, Z., Chen, X., Yuille, A. L., and Zhu, S.-C. (2005). Image Parsing: Unifying Segmentation, Detection, and Recognition. *International Journal of Computer Vision*, 63(2):113–140. (page 30)

[85] Uhrig, J., Cordts, M., Franke, U., and Brox, T. (2016). Pixel-Level Encoding and Depth Layering for Instance-Level Semantic Labeling. In *German Conference on Pattern Recognition*, pages 14–25. (page 27, 30)

[86] Uijlings, J. R., Van De Sande, K. E., Gevers, T., and Smeulders, A. W. (2013). Selective Search for Object Recognition. *International Journal of Computer Vision*, 104(2):154–171. (page 25)

[87] Viola, P. and Jones, M. (2001). Rapid Object Detection Using a Boosted Cascade of Simple Features. In *International Conference on Computer Vision*, pages 511–518. (page 22, 24)

[88] Xie, J., Xu, L., and Chen, E. (2012). Image Denoising and Inpainting with Deep Neural Networks. In *Advances in Neural Information Processing Systems*, pages 341–349. (page 45)

[89] Xiong, Y., Liao, R., Zhao, H., Hu, R., Bai, M., Yumer, E., and Urtasun, R. (2019). UPSNet: A Unified Panoptic Segmentation Network. In *Conference on Computer Vision and Pattern Recognition*, pages 8818–8826. (page 31)

[90] Yang, B., Luo, W., and Urtasun, R. (2018). Pixor: Real-Time 3D Object Detection From Point Clouds. In *Conference on Computer Vision and Pattern Recognition*, pages 7652–7660. (page 24)

[91] Yang, Y., Li, H., Li, X., Zhao, Q., Wu, J., and Lin, Z. (2019). SOGNet: Scene Overlap Graph Network for Panoptic Segmentation. *arXiv:1911.07527*. (page 31)

[92] Yu, F. and Koltun, V. (2015). Multi-Scale Context Aggregation by Dilated Convolutions. *arXiv:1511.07122*. (page 30)

[93] Yuan, A., Bai, G., Jiao, L., and Liu, Y. (2012). Offline Handwritten English Character Recognition Based on Convolutional Neural Network. In *International Workshop on Document Analysis Systems*, pages 125–129. (page 11)

[94] Zeiler, M. D. and Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In *European Conference on Computer Vision*, pages 818–833. (page 11)

[95] Zell, A. (1994). *Simulation Neuronaler Netze*, volume 1. Addison-Wesley. (page 8)

[96] Zhang, D., Song, Y., Liu, D., Jia, H., Liu, S., Xia, Y., Huang, H., and Cai, W. (2018a). Panoptic Segmentation with an End-to-End Cell R-CNN for Pathology Image Analysis. In *Medical Image Computing and Computer-Assisted Intervention*, pages 237–244. (page 1, 30)

[97] Zhang, K., Zuo, W., Chen, Y., Meng, D., and Zhang, L. (2017). Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155. (page 11)

[98] Zhang, Z., Liu, Q., and Wang, Y. (2018b). Road Extraction by Deep Residual U-Net. *Geoscience and Remote Sensing Letters*, 15(5):749–753. (page)

[99] Zhao, H., Shi, J., Qi, X., Wang, X., and Jia, J. (2017). Pyramid Scene Parsing Network. In *Conference on Computer Vision and Pattern Recognition*, pages 2881–2890. (page)