Technische Universität Braunschweig

Institut für Nachrichtentechnik

Seminar Paper

Joint Training Methods of Visible and Amodal Semantic Segmentation

Yu-Chuan Cheng
Matrikelnummer: 4998248

17.08.2022

Technische Universität Braunschweig
Institute for Communications Technology
Schleinitzstraße 22 · 38106 Braunschweig

Prüfer: Prof. Dr.-Ing. Tim Fingscheidt
Betreuer: Jasmin Breitenstein M.Sc.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Seminar Paper mit dem Titel

„Joint Training Methods of Visible and Amodal Semantic Segmentation“

selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Braunschweig, 17.08.2022

Yu-Chuan Cheng

Contents

List of Tables

List of Figures

List of Abbreviations

CNN Convolutional neural network

FCN Fully convolutional networks

NN Neural Network

FC Fully connected

ANN Artificial Neural Network

MTL Multi-task learning

SGD stochastic gradient descent

1 Introduction

A crucial part of computer vision is semantic picture segmentation. It is seen as a vital activity that may aid in a thorough knowledge of the scene, objects, and people since it predicts dense labels for each and every pixel in the picture. Recent deep convolutional neural network (CNN) development has made significant strides in semantic segmentation. The intricate model design for these networks' depth and breadth, which must incorporate several operations and factors, is primarily responsible for their efficacy. Semantic segmentation with CNNs primarily makes use of fully convolutional networks (FCNs). Nowadays, it is common knowledge that improving result accuracy nearly always entails increasing the number of processes, particularly for pixel-level prediction tasks like semantic segmentation. However, visual perception is a key component of autonomous driving, and the development and dissemination of intelligent cars depend on the dependability of this capability. Visual perception itself entails a variety of tasks, including object recognition, instance segmentation, and semantic segmentation, for example.

1.1 Motivation

Since the bulk of today's autonomous driving relies on public data, I wish to coordinate this work with visible datasets. Amodal perception refers to the perception of information that is redundant or common across many senses (e.g., auditory, visual, tactile). Amodal information contains modifications along the three fundamental stimulation-related dimensions of time, place, and intensity. Based on this cut-and-paste-generated amodal [BF22], various neural networks (HRNet) [Sun+19b] are afterwards validated to compare and explore with their initial baseline (ERFNet) [Rom+18]. We anticipate that by using various semantic segmentation models, we may get a deeper comprehension of this dataset.

1.2 Structure of the work

Chapter 2 provides an outline of the primary work foundations. It discusses the fundamentals of neural networks in Section 2.1, convolutional neural networks and some basic training ideas in Section 2.2, segmentation using fully convolutional networks in Section 2.3, the introduction of HRNet with ERFNet in Section 2.4,

the notion of multi-task learning in Section 2.5, and why and how should we apply data augmentation in Section 2.6. Since our dataset is a derivative of the cityscape dataset, we will introduce it by describing how it was made and what the amodal dataset is. The fourth chapter's main themes are the introduction of the experimental technique, the exact parameter, and the establishment of data augmentations. The experimental results will be assessed in the fifth chapter, which will also compare the advantages and disadvantages of the two different modules. The results of the experiment are briefly summarized in Chapter 6, which concludes the article.

2 Literature Overview

2.1 Neural Networks

An “Artificial Neural Network” is a computing system that aims to replicate the neural connections in the human nervous system. The term “neural“ is the adjective form of “neuron”, and “network” signifies a structure like a graph. Other names for artificial neural networks include “neural networks ” and “artificial neural systems”. Artificial Neural Networks are often referred to as “Artificial Neural Network (ANN)” or simply “Neural Network (NN)” [Gur97]. A system has to have a labeled, directed graph structure with each node performing a straightforward calculation in order to qualify as a NN. According to graph theory, a directed graph consists of a collection of nodes (also known as vertices) and a set of connections (also known as edges) that connect pairs of nodes. An illustration of such a NN graph may be seen in Figure 2.1. Each node does a straightforward calculation.

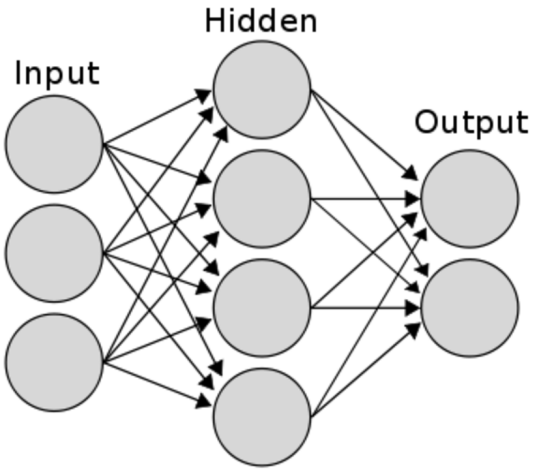


Figure 2.1 A simple neural network structure. The network is shown with inputs. Each link sends a signal over the network’s covert levels. In a final function, the output class label is calculated.

The outcome of the calculation is then sent from one node to the next through each link, which is tagged with a weight indicating how much the signal has been amplified or reduced. The signal is amplified by several connections with significant, positive weights, showing that the signal is crucial for classifying data. Others have negative weights, which weaken the signal and indicate that the node’s output is

less significant in the final categorization. If a system has a graph structure with connection weights that can be changed using a learning algorithm, we refer to it as an artificial neural network [GBC15].

2.2 Convolutional neural network

In order to interpret data with a grid pattern, such as photos, CNN is a form of deep learning model. CNN [Yam+18]was created with the animal visual cortex in mind and is intended to automatically and adaptively learn spatial hierarchies of characteristics, from low- to high-level patterns. Convolution, pooling, and fully linked layers are the three kinds of layers or "building blocks"that make up a standard CNN. Convolution and pooling layers in orders one and two do feature extraction, whereas a fully connected layer in order three translates the retrieved features into the output, such as classification.
Here is a description of CNN (Figure 2.2). **Convolution layers**, **pooling layers**, and **fully connected layers** are some of the building pieces that make up a CNN. A learnable parameter, such as kernels and weights, is updated in accordance with the loss value through backpropagation with the gradient descent optimization algorithm. A model's performance under specific kernels and weights is calculated with a loss function through forwarding propagation on a training dataset. Rectified linear unit, or ReLU.
In CNN, which is made up of a stack of mathematical operations, including convolution, a specific kind of linear operation, a convolution layer is crucial. Since a feature may appear anywhere in a digital image, the pixel values are stored in a two-dimensional (2D) grid, or array of numbers (Figure 2.3), and a small grid of parameters called the kernel, an optimizable feature extractor, is applied at each image position. This makes CNNs extremely effective for image processing. Extracted characteristics may gradually and hierarchically grow more sophisticated as one layer feeds its output into the next layer. Training is the process of minimizing the difference between outputs and ground truth labels using optimization techniques like backpropagation and gradient descent, among others. It involves improving parameters such as kernels.

In the training portion, these three types of layers will determine the eventual number of parameters that must be trained. Table 2.1 provides a basic summary.

2.2.1 Convolution

Convolution is a specific sort of linear operation used for feature extraction in which a tiny array of numbers, known as a kernel, is applied to the input, which is a larger array of numbers known as a tensor. At each point of the tensor, an element-wise product between each element of the kernel and the input tensor is

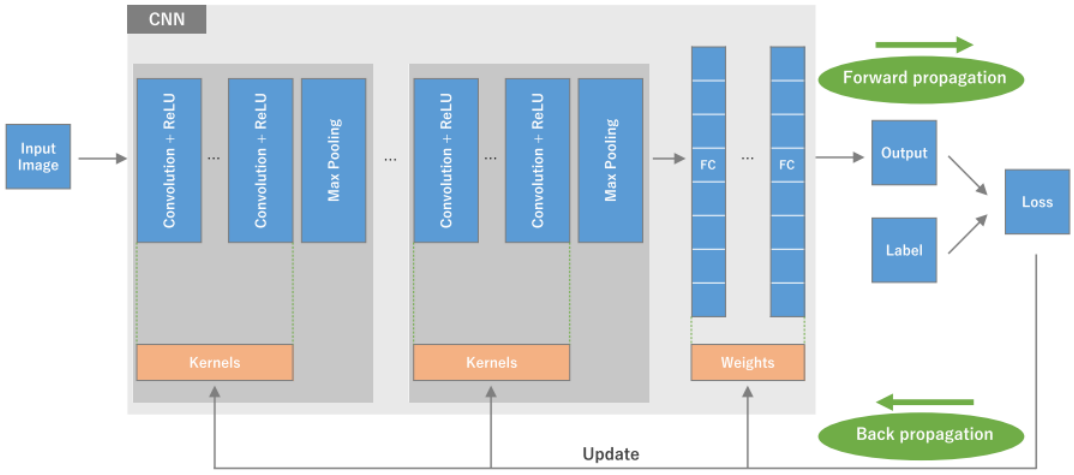


Figure 2.2 A description of the architecture of a CNN and how it is trained. Source:[Yam+18]

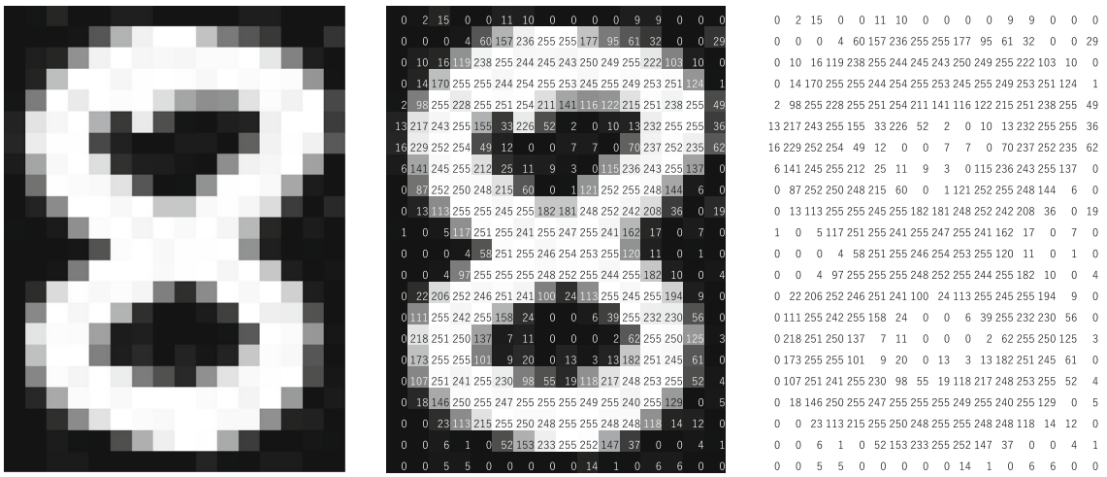


Figure 2.3 A computer sees an image as an array of numbers. The source image was downloaded via http://yann.lecun.com/exdb/mnist

computed and added to generate the output value in the corresponding place of the output tensor, referred to as a feature map (Figure 2.4). This technique is repeated by applying numerous kernels to generate an arbitrary number of feature maps, each of which represents a different property of the input tensors; various kernels may therefore be regarded as distinct feature extractors. Size and number of kernels are two critical hyperparameters that determine the convolution procedure. The former is often 3x3, but may also be 5x5 or other values. The latter is arbitrary and affects the depth of output feature maps.

The convolution described above prevents the center of each kernel from overlapping the outermost element of the input tensor and minimizes the height and breadth of the output feature map relative to the input feature map. To overcome this problem, padding, often zero padding, is a method wherein rows and columns of

Table 2.1 A list of the convolutional neural network's parameters and hyperparameters. source: [Yam+18]

	Parameters	Hyperparameters
Convolution layer	Kernels	Kernel size, number of kernels, stride, padding
Pooling layer	None	Pooling method, filter size, stride, padding
Fully connected layer	Weights	Number of weights, activation function

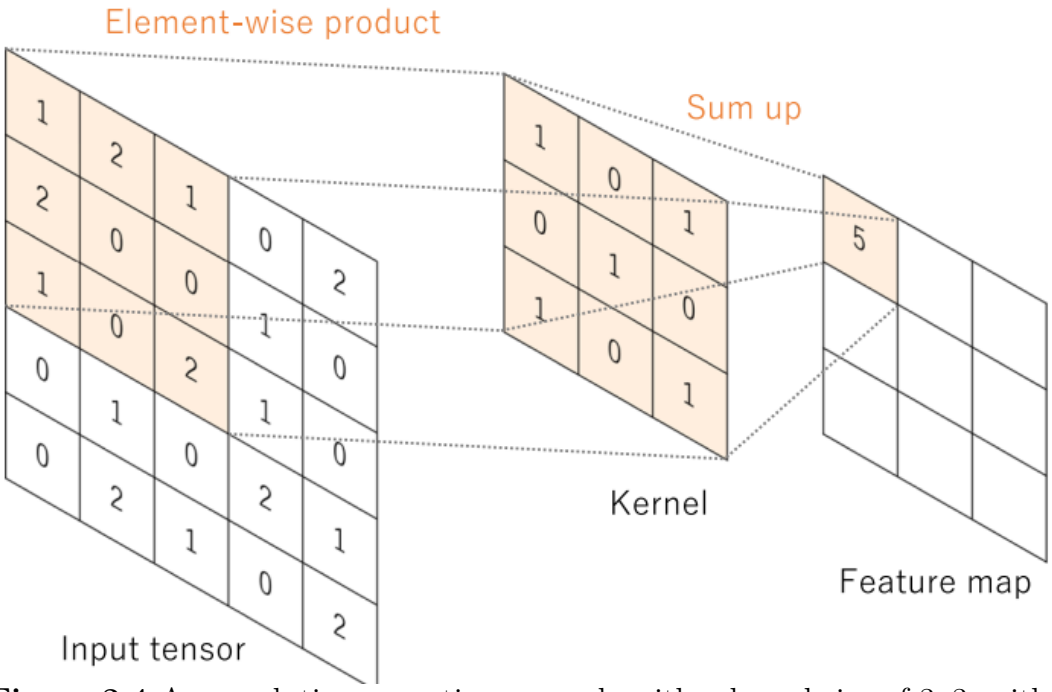


Figure 2.4 A convolution operation example with a kernel size of 3x3, without padding, and a stride of 1. source: [Yam+18]

zeros are added to either side of the input tensor in order to fit the center of a kernel on the outermost element and maintain the same in-plane size throughout the convolution procedure (Figure 2.5). Modern CNN designs often utilize zero padding to preserve in-plane dimensions so that additional layers may be applied. Without zero padding, the size of each subsequent feature map would decrease after the convolution process. A stride is a distance between two consecutive kernel points, which also determines the convolution procedure. In order to accomplish downsampling of the feature maps, a stride greater than 1 is sometimes employed. Pooling is an alternate approach for doing downsampling.

Weight sharing is the defining characteristic of a convolution process; kernels are shared across all picture places. Weight sharing gives convolution processes the following characteristics: **a.** allowing local feature patterns retrieved by kernels to remain translation-invariant when kernels traverse all picture locations and identify learned local patterns, **b.** Learning spatial hierarchies of feature patterns

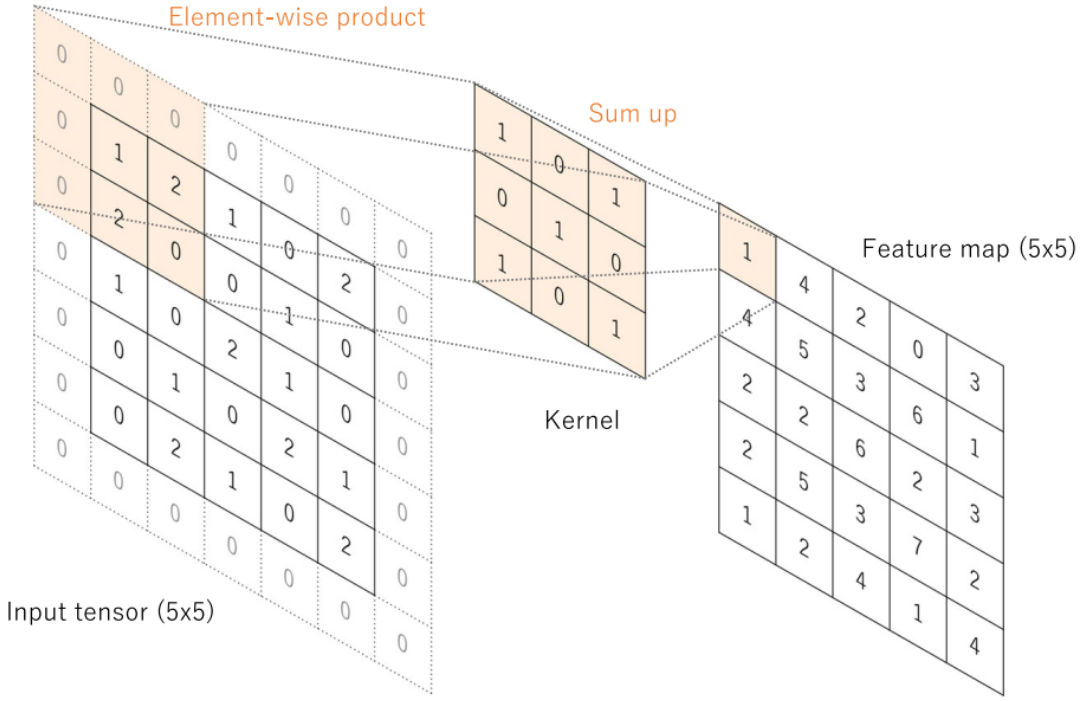


Figure 2.5 A convolution procedure with zero padding to preserve in-plane dimensions. source: [Yam+18]

via downsampling in combination with a pooling operation, so collecting an ever-increasing field of view, and **c.** enhancing model performance by decreasing the number of parameters to be learned compared to fully connected neural networks

2.2.2 Pooling

A pooling layer offers a common downsampling procedure that decreases the in-plane dimensionality of the feature maps to provide translation invariance to tiny shifts and distortions and to minimize the number of future learnable parameters. Notably, none of the pooling layers have a learnable parameter, although filter size, stride, and padding are hyperparameters. We often employ maximum pooling, however, there are alternative pooling methods, such as average pooling. In this thesis, ERFNet uses maximum pooling while HRNet uses global average pooling.

Max Pooling. max pooling extracts patches from input feature maps, outputs the biggest value in each patch and discards the remainder of the data. (Figure 2.6). This reduces by a factor of 2 the in-plane dimension of feature maps. Unlike the width and height dimensions, the depth dimension of feature maps does not vary.

Global average pooling. A global average pooling is an extreme sort of downsampling in which a feature map with the dimensions height x width is downsampled into a 1 x 1 array by averaging all the elements in each feature map while retaining the depth of feature maps. The use of global average pooling has two advantages: a. Minimizing the number of learnable parameters. b. Allowing the CNN to take inputs of variable size.

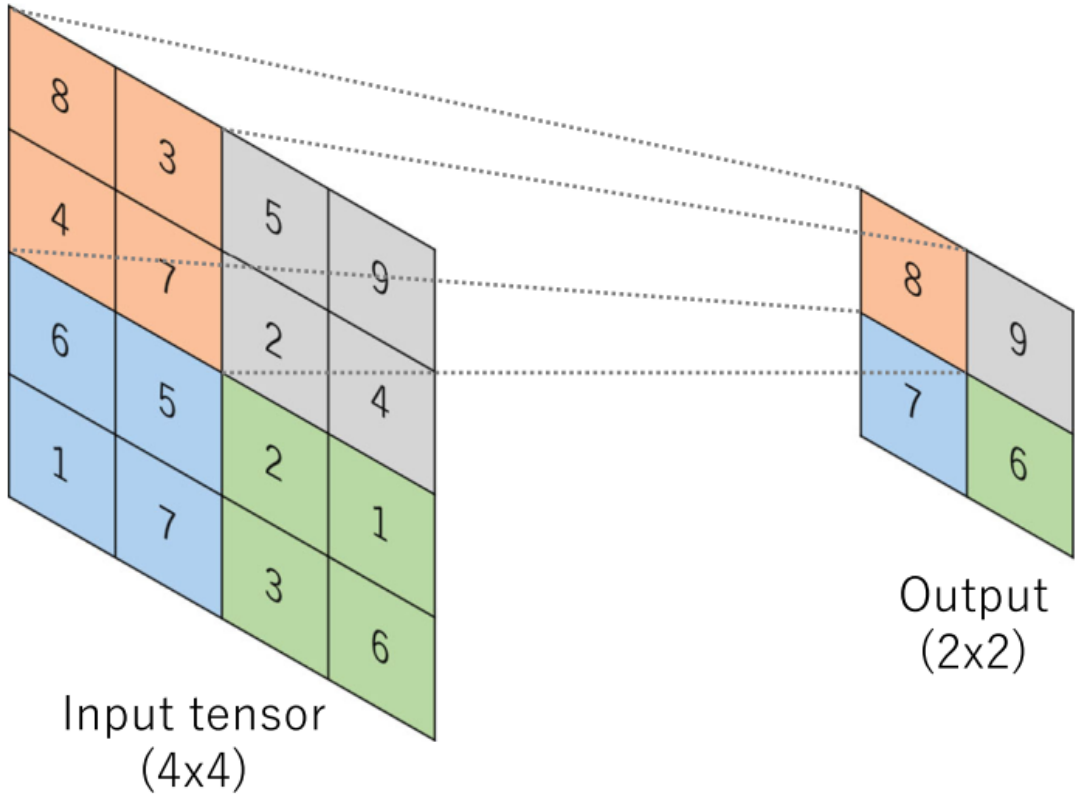


Figure 2.6 An illustration of maximum pooling with a filter size of 2x2 With no padding and a stride of 2. source: [Yam+18]

2.2.3 Fully linked layers

Typically, the output feature maps of the last convolution or pooling layer are flattened, i.e., transformed to a one-dimensional array or vector, and linked to one or more dense layers, also known as fully connected layers, in which each input is coupled to each output by a learnable weight. After the features are retrieved by the convolution layers and downsampled by the pooling layers, they are mapped by a subset of fully connected layers to the network's final outputs, such as the probabilities for each class in classification tasks. Typically, the final fully linked layer has the same number of output nodes as classes.[Yam+18] Following each completely linked layer comes a nonlinear function, such as ReLU.

Table 2.2 A collection of frequently used final layer activation functions for diverse applications. source: [Yam+18]

Task	Last layer activation function
Binary classification	Sigmoid
Multiclass single-class classification	Softmax
Multiclass multiclass classification	Sigmoid
Regression to continuous values	Identity

2.2.4 Last layer activation function

Typically, the activation function used for the last completely linked layer differs from the others. For each task, the proper activation function must be determined. A multiclass classification task activation function is a softmax function that normalizes output real values from the final fully connected layer to target class probabilities, where each value ranges from 0 to 1 and all values total to 1.[Yam+18] Table 2.2 summarizes typical selections for the last layer activation function for different sorts of jobs and Figure 2.7 provides an overview of these three activation functions.

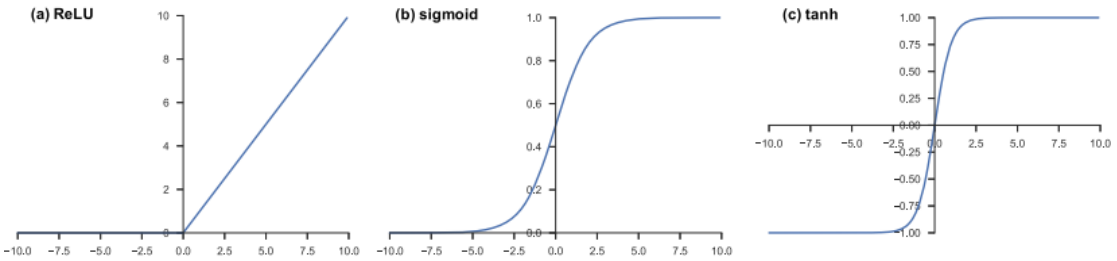


Figure 2.7 Typical activation functions used to neural networks. Source: [Yam+18]

2.2.5 Training

Training a network involves discovering kernels in convolution layers and weights in fully connected layers that minimize discrepancies between output predictions and ground truth labels on a training dataset.[Yam+18] **Loss function** and **gradient descent** optimization techniques play crucial roles in the backpropagation algorithm, which is often used to train neural networks. A loss function calculates a model's performance under given kernels and weights using forward propagation on a training dataset, and kernels and weights are modified based on the loss value using an optimization process called backpropagation and gradient descent.

Loss function. A loss function, also known as a cost function, assesses the compatibility between the network's forwarding propagation output predictions and

provided ground truth labels. Cross entropy is a frequent loss function for multiclass classification, although the mean squared error is widely used for regression to continuous values. One of the hyperparameters that must be selected according to the provided tasks is the kind of loss function.

Gradient descent. Gradient descent is a typical optimization approach that repeatedly adjusts the network's learnable parameters, i.e. kernels and weights, in order to minimize loss. The gradient of the loss function indicates the direction in which the function's rate of growth is steepest, and each learnable parameter is updated in the opposite direction of the gradient using an arbitrary step size set by a hyperparameter known as the learning rate. Mathematically, the gradient is a partial derivative of the loss with respect to each learnable parameter, and a single parameter update is defined as follows:

$$w := w - \alpha \cdot \frac{\partial L}{\partial w}$$

w represents each learnable parameter, represents the learning rate, and L represents the loss function (Figure 2.8). Notably, the learning rate α is one of the most critical hyperparameters that must be established before training begins. Due to memory restrictions, gradients of the loss function with respect to the parameters are generated using a subset of the training dataset known as a mini-batch and then applied to parameter updates. This approach is known as mini-batch gradient descent, also known as stochastic gradient descent (SGD), and the size of the mini-batch is a hyperparameter. In addition, other enhancements to the gradient descent method, such as SGD with momentum, RMSprop, and Adam, have been suggested and are frequently used, however, the specifics of these algorithms are outside the scope of this article. [Yam+18]

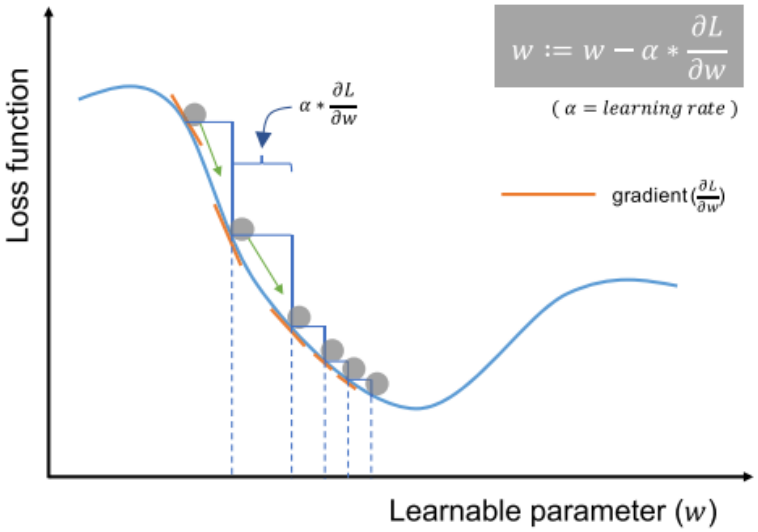


Figure 2.8 Gradient descent. source: [Yam+18]

2.3 Fully Convolutional Networks for Semantic Segmentation

A neural network that exclusively uses convolution and downsampling or upsampling operations is known as a full convolution network (FCN). An FCN is just a CNN with fewer completely linked layers. Without additional equipment, a trained end-to-end, pixels-to-pixels FCN on semantic segmentation outperforms the state-of-the-art. Existing networks' fully convolutional iterations forecast dense outputs from arbitrary-sized inputs. [SLD17] By using backpropagation and dense feedforward computing, learning and inference are both carried out one full picture at a time. Pixel-wise prediction and learning are made possible in nets with subsampled pooling by in-network upsampling layers.
It may be broken down into three sections:

- **Semantic Segmentation by Downsampling.** In classification, a shrunk input picture typically travels through fully connected (FC) layers and convolution layers before producing one predicted label for the input image. If the picture is not shrunk, we convert the FC layers into convolutional layers, and the result will not be a single label. Instead, the output is less than the size of the original picture (due to the max pooling). We can determine the pixel-wise output if we upsample the result.
- **Upsampling by Deconvoluting.** The technique of convolution reduces the output size. Deconvolution is the term used when upsampling is desired in order to increase the output size. Deconvolution is not the opposite of convolution, despite the name's common misconception. Transposed convolution and up-convolution are other names for it. When using the fractional stride, it is also known as fractional stride convolution.
- **Fusing the Outputs.** Following the completion of the following convolution layers, the output size is tiny. To make the output the same size as the input picture, upsampling is then performed. But it also deteriorates the output label map's quality. This is due to the fact that digging deeper allows for the acquisition of deep features while simultaneously causing the loss of spatial position information. As a result, the output from shallower layers contains more geographic data. Both may be combined to improve the outcome (Figure 2.9). Therefore, element-wise addition should be used to fuse the output. The boosting/ensemble strategy employed in AlexNet, VGGNet, and GoogLeNet, where they combine the data from numerous models to increase prediction accuracy, is really quite similar to this fusing procedure. However, in this instance, it is done for each individual pixel, and the pixels are added from the output of several model layers.

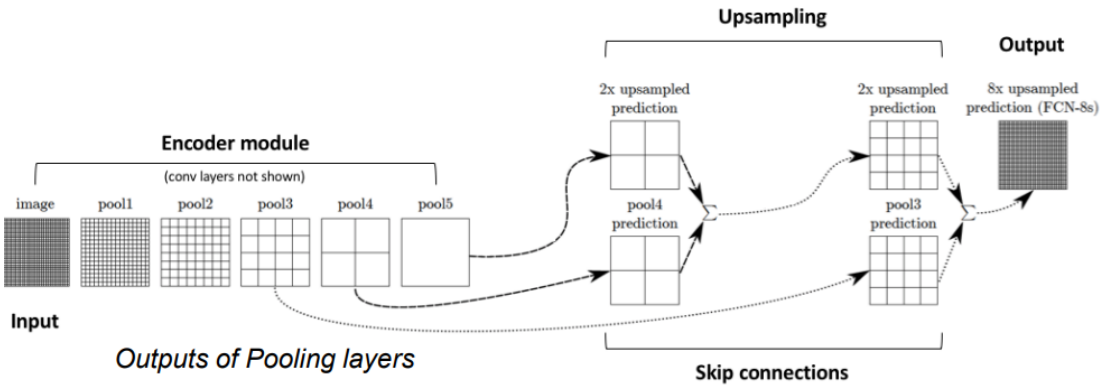


Figure 2.9 Fusing for FCN.

2.3.1 The difference between FCN and CNN

Fully convolutional CNNs (FCNs) lack fully linked layers since all of their learnable layers are convolutional. Simply defined, the area of FCN and CNN involves replacing the last fully connected layer of CNN with a convolutional layer, with the result being a labeled image. The primary distinctions between an FCN and a CNN, which comprises a few Fully connected (FC) layers after a few convolutional layers, would mostly be in two areas:

- **Input image size.** You can use the network to process photos of almost any size even if your network doesn't have any fully linked layers. In topologies like AlexNet, you must provide input pictures of a certain size since only the fully linked layer demands inputs of that size.
- **Spatial information.** Because it is "fully linked,"meaning that all output neurons are coupled to all input neurons, fully connected layers often result in loss of spatial information. If you are working in a vast space of possibilities, this kind of design cannot be employed for segmentation (e.g. unconstrained real images [SLD17]). Although fully connected layers may still segment pictures if you are constrained to a relatively narrow area, such as a small number of item categories with little visual fluctuation, in which case the FC activations may suffice as a statistic [Kul+15] [Dos+17]. In the second scenario, both the item type and its spatial arrangement may be encoded using only the FC activations. Depending on the loss function and the FC layer's capacity, one or the other may occur.
- **Cost of computation.** CNN has poor computing efficiency due to the fact that neighboring pixel blocks are essentially duplicated and the convolution is performed individually for each pixel block, a process that is likewise mainly repeated. Additionally, CNN has a hefty storage overhead. For instance, the size of the image block utilized for each pixel is 5x5, the window is continually

sliding, and each sliding window is recognized and classed by the CNN; hence, the amount of storage space needed rises exponentially with the number and size of sliding windows. The sole difference between the fully connected layer and the convolutional layer is that neurons in the convolutional layer are only linked to a limited region in the input data and neurons in the convolutional column share parameters. However, neurons compute dot products in both sorts of layers, therefore their functional shape is the same. Therefore, the two may be transformed into one another. There exists a completely linked layer that can accomplish the same forward propagation function as any convolutional layer. The weight matrix is a massive matrix, with the exception of a few specified cells, all of which contain zeros. The majority of these blocks include identical components. In contrast, every layer with complete connectivity may be converted into a convolutional layer.

2.3.2 U-Net

Why bring up U-Net? It is because U-Net adopts a completely different feature fusion method from FCN with concatenation. Additionally, there are three benefits of using U-net. In other words, it computes an output at the pixel level after first fitting for segmentation. Since we are attempting to tackle segmentation issues, it should operate well as is. Also, its structure is simple. Finally, it is effective with less training data. Despite the fact that the original writers do not adequately describe this problem.

U-net is a well-known network used for semantic segmentation, or categorizing pixels of an image such that pixels belonging to the same class are assigned the same label, also known as pixel-wise or dense classification. A typical U-net is like Figure 2.10. Each blue rectangle represents a multichannel feature map. The number of channels is shown on the box's lid. The x-y dimension is shown in the bottom left corner of the box. White boxes indicate duplicated feature maps. The arrows represent the various operations.

Since ERFNet and HRNet both employ upsampling for improved results (HRNet is not classic upsampling refer to Figure 2.14), we would like to note the simplest upsampling approach from U-net for joining the idea that we will discuss in the next section. Because ERFNet and HRNet are our primary focus, we must understand the distinctions between U-net and FCN and the difference between ERFNet and U-net. FCN has just one upsampling layer, but U-net has numerous levels. Second, the original FCN implementation utilizes bilinear interpolation to upsample the convolved picture, which precludes the inclusion of a learnable filter, while U-Net employs skip connections, concatenates, and learnable weight filters in place of the fixed interpolation method. Last, the primary distinctions between U-net and

ERFNet are Factorized Residual Layers (which will be explained later) and the manner in which they upsample.

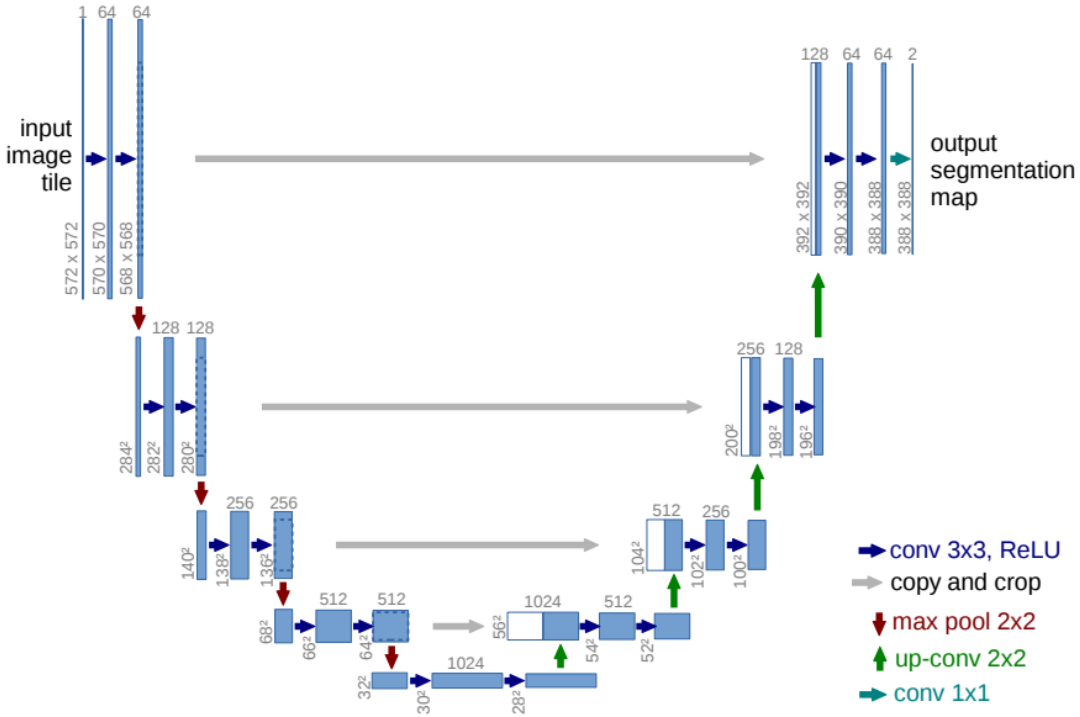


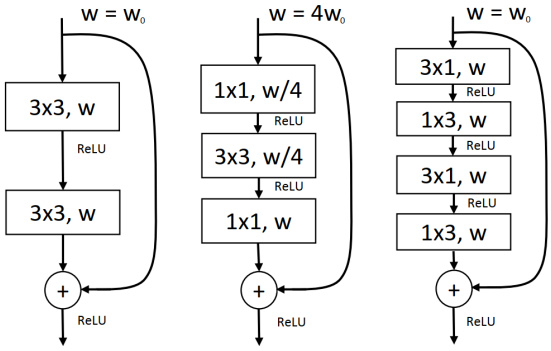
Figure 2.10 U-net architecture (The example for 32 x 32 pixels in the lowest resolution.) [RFB15]

2.4 Inroduction of ERFNet and HRNet

We've incorporated the FCN feature because we want to explain why ERFNet is one of the best methods for semantic segmentation. Then, get into the specifics. FCN is the first to employ the classification network VGG16 for end-to-end semantic segmentation; it also transforms the network into complete convolution and upsamples the output feature maps. Invoking these networks directly, however, leads to coarse pixel output, which affects pixel accuracy, due to the fact that increased downsampling is conducted in classification tasks to gain more context. In order to enhance these outputs, ERFNet has suggested fusing shallow feature maps with skip connections. We anticipate HRNet to have superior mIOU than ERFNet. Therefore, it is essential to understand these two models in further depth.

2.4.1 ERFNet

Factorized Residual Layers. The figure 2.11 The modules ResNet suggests are a and b, and both have comparable parameters and near accuracy. Bottleneck, however, uses less computational resources. This feature becomes more cost-effective and adaptable as the depth rises. Non-bottleneck modules, however, may achieve more accuracy, yet the bottleneck continues to have deterioration issues.



(a) Non-bottleneck (b) Bottleneck (c) Non-bottleneck-1D

Figure 2.11 Factorized Residual Layers. source: [Rom+18]

A revised version was thus suggested by the author, as shown in Figure 2.11 (c). The latest iteration only makes use of 1D convolution. There are two layers of neural networks on the left side of the Figure 2.12, made up of N neurons to M neurons, and the weights are

$$N \cdot 1 \cdot M = W.$$

However, if we add a layer before N and M, the weight changes to U and V. At this point, we can determine the weight parameter:

$$V = N \cdot 1 \cdot K$$

$$U = K \cdot 1 \cdot M$$

What is thus required for the two layers

$$K \cdot (M + N)$$

As long as K is small, the total weights will also be smaller.

This technique can approach the original effect as long as the number of kernels is minimal, but it also uses fewer parameters and less computation. The insertion of a nonlinear function between the two layers of the 1D convolution by the author should have theoretically improved learning capacity. Any job that requires the ResNet network may utilize the suggested module.

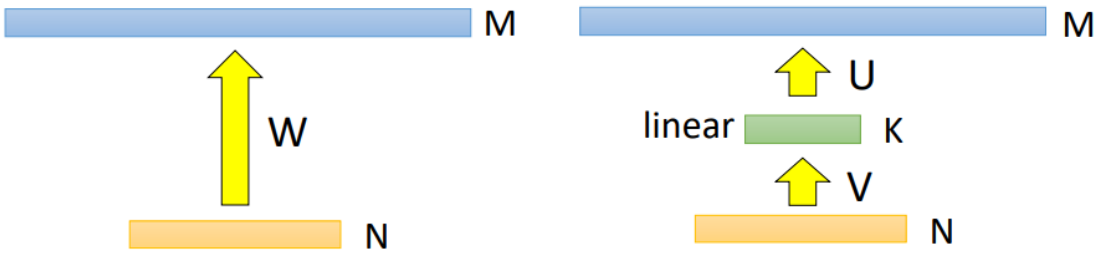
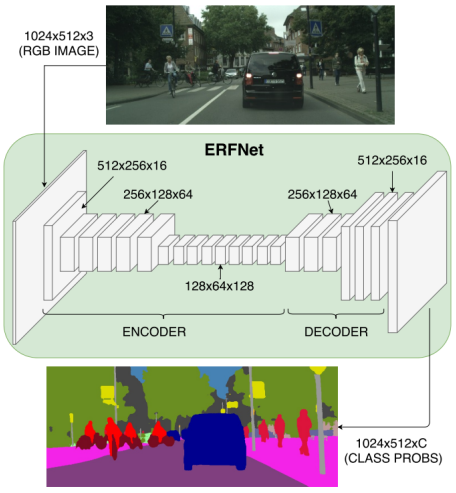


Figure 2.12 Elaboration of 1D convolution. **Source:** Li Hongyi Network compression course

Network architecture. ERFNet's network design is identical to ENet's previous encoder-decoder architecture. In contrast to the FCN design, this architecture requires the feature maps of many layers to be fused in order to provide a delicate output. The encoder-decoder design of ENet is exactly the same as the network architecture of ERFNet. In contrast to the FCN design, this architecture requires the fusion of the feature maps from many layers to provide a delicate output.

Downsampling decreases the amount of computation even if it produces an approximate outcome. The network uses 3 down-sampling and the early sampling mode of ENet, which is a combination of 2x2 max pooling and 3x3 convolution (strides 2). And to gain extra information, interleave the usage of hole convolutions on the suggested resnet block which shows to be more effective than utilizing bigger size convolutions. Dropout is another term that is used.

The main purpose of the upsampling component, which uses an ENet-like architecture, is to modify the fineness and match the input. The distinction is that the max-unpooling option of ENet is not used. [Rom+18] employs a straightforward stride 2 deconvolution layer. We can see the whole structure of ERFNet in Figure 2.13.



	Layer	Type	out-F	out-Res
ENCODER	1	**Downsampler block**	16	512x256
	2	**Downsampler block**	64	256x128
	3-7	5 x **Non-bt-1D**	64	256x128
	8	**Downsampler block**	128	128x64
	9	**Non-bt-1D** (dilated 2)	128	128x64
	10	**Non-bt-1D** (dilated 4)	128	128x64
	11	**Non-bt-1D** (dilated 8)	128	128x64
	12	**Non-bt-1D** (dilated 16)	128	128x64
	13	**Non-bt-1D** (dilated 2)	128	128x64
	14	**Non-bt-1D** (dilated 4)	128	128x64
	15	**Non-bt-1D** (dilated 8)	128	128x64
	16	**Non-bt-1D** (dilated 16)	128	128x64
DECODER	17	**Deconvolution** (upsampling)	64	256x128
	18-19	2 x **Non-bt-1D**	64	256x128
	20	**Deconvolution** (upsampling)	16	512x256
	21-22	2 x **Non-bt-1D**	16	512x256
	23	**Deconvolution** (upsampling)	C	1024x512

Figure 2.13 ERFNet architecture. Source:[Rom+18]

Short summary. According to [Pas+16] and [Rom+18],ENet's IoU on the Cityscapes data set was 58.3, whereas ERFNet's IoU was 69.7, an increase of 10 points in accuracy. The quantity of parameters in ERFNet is 10 times more than that of ENet, and its speed is about half that of ENet, according to experimental data from ERFNet. In general, real-time speed and accuracy of ERFNet are actually within acceptable bounds, despite the fact that it runs slowly and has a high number of parameters. The Resnet section has been enhanced in both publications. The Low-rank Approximation operation of ResNet in ERFNet, which seems to be often employed subsequently, is more significant.

2.4.2 HRNet

There are two primary categories of representations: low-resolution ones that are mostly used for picture categorization and high-resolution ones that are crucial for solving many other vision-related issues, such as semantic segmentation, human posture estimation, etc. The representation in which we are most interested in the latter [Sun+19b]. Two HRNet are present. We will provide illustrations in the next two paragraphs. In this thesis, we perform semantic segmentation for greater mIOU using HRNetV2.

HRNetV1. The basic architecture of HRNet is like the Figure 2.14. There are four phases. Parallel connections between high-to-low resolution convolutions are made in the second, third, and fourth phases. The parallel processing is shown by the channel map lines stacked one on top of the other. The red tiny channel maps show the lowest resolution, while the yellow channel map shows the maximum resolution. The fourth block handles four resolutions concurrently. A straightforward improvement on group convolution, multi-resolution group convolution performs a normal convolution over each subgroup of input channels across various spatial resolutions individually. We may observe a complete link to the multi-resolution group of the next stage, which resembles the Figure 2.14, at the conclusion of each stage. Multi-resolution convolution is the name of this section of the HRNet.

Conventional convolution neural networks, such as low-resolution networks that operate in series and enable a high resolution to be recovered from low resolution, are what make HRNet semantically robust and spatially exact. Additionally, HRNet's parallel method enables the maintenance of high resolution across the entire neural network, improving the accuracy of the representation. Finally, additional methods combine low-resolution and high-resolution representations. The high-to-low representations are strengthened semantically by HRNet's repetition of multi-resolution.

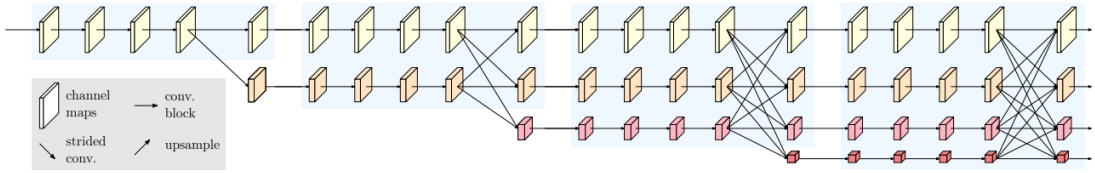
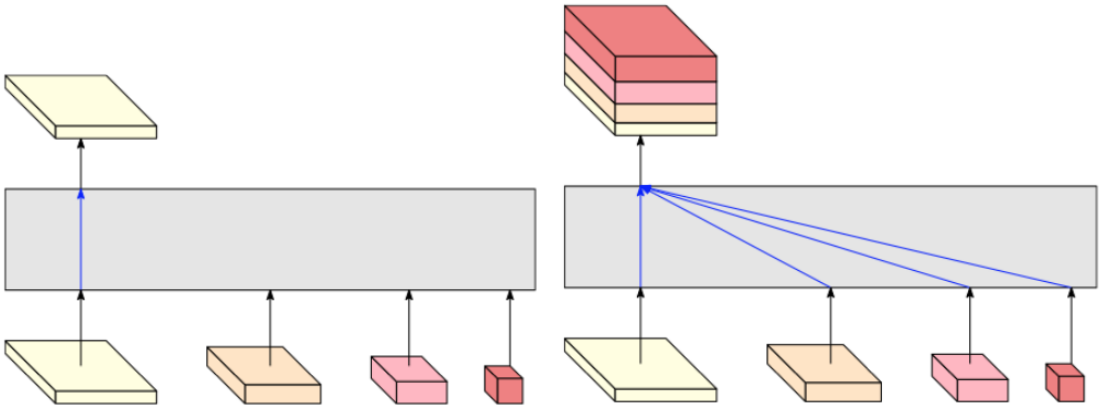


Figure 2.14 A simple example of a high-resolution network. Source: [Sun+19b]

HRNetV2. Basically, the whole architecture is pretty much as same as HRNetV1 except for concatenating all resulting subsets. The network in Figure 2.14 outputs the four-resolution representations at the bottom of each sub-figure; the gray box shows how the output representation is created from the input four-resolution representations. This model Figure 2.15 (b), known as HRNetV2, is mostly used for predicting segmentation maps and heatmaps of face landmarks.



(a) HRNetV1 **(b)** HRNetV2

Figure 2.15 Comparasion between HRNetV1 and HRNetV2. Source: [Sun+19b]

Short summary. In the realm of computer vision, HRNet, and its derivatives represent the cutting edge of numerous machine learning techniques. The serial approach used in the traditional convolution architecture has been replaced with a parallel design with multiple group convolutions by the authors. High resolutions were made possible by this design, which also enhances accuracy and the semantic relationship.

2.5 Multi-task learning

Multi-task learning (MTL) in the context of classification tries to enhance the performance of numerous classification problems by learning them together. A spam filter is one example, which can be thought of as several but related categorization jobs performed by various individuals. As a result, joint learning is another word

for MTL. All machine learning applications, including computer vision, voice recognition, and natural language processing, have effectively exploited MTL. MTL has been referred to by a variety of names, including joint learning, learning to learn, and learning with auxiliary activities. In general, we can tell that we are practicing MTL when we are improving more than one loss function (in contrast to single-task learning). In certain circumstances, it might be beneficial to openly think about what we're attempting to do in terms of MTL and to get insights from it. [Rud17]

2.5.1 Two MTL methods in Deep Learning

We'll now have a look at the two most popular methods for carrying out multi-task learning in deep neural networks to help put MTL's theories into practice. Multi-task learning in the context of deep learning is often carried out using either hard or soft parameter sharing of hidden layers.

Hard parameter sharing. As shown in Figure 2.16a, it is often implemented by preserving a number of task-specific output layers while sharing the hidden levels across all activities. Overfitting is considerably reduced by hard parameter sharing. [Bax04] actually shows that the danger of overfitting the shared parameters, or the output layers, is an order N less than the risk of overfitting the task-specific parameters. Intuitively, this makes sense that the more tasks we learn concurrently, the harder it is for our model to discover a representation that encompasses all of the tasks, and the less likely it is that we would overfit our initial task.

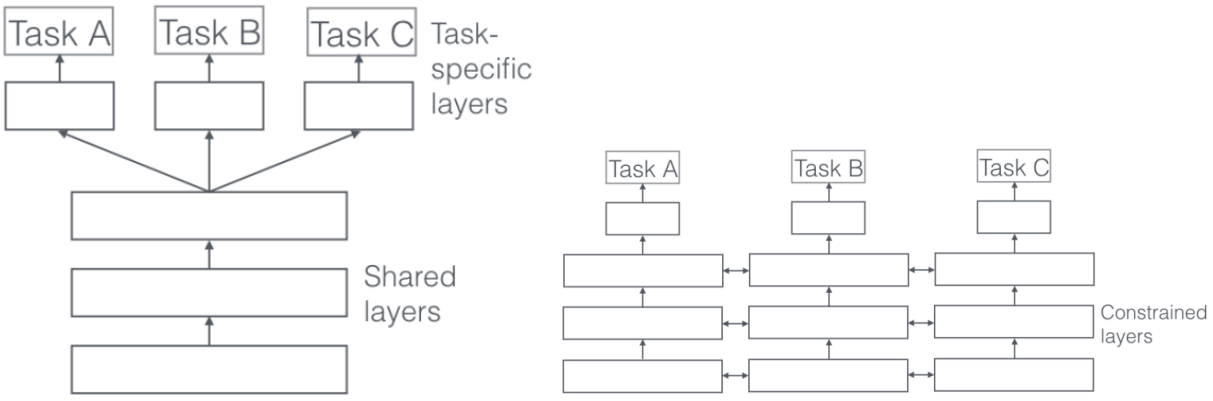
Soft parameter sharing. On the other hand, with soft parameter sharing, every job has a unique model with unique parameters. In order to encourage the model's parameters to be comparable, the distance between them is then regularized, as shown in Figure 2.16b. Regularization methods for MTL that have been created for other models have had a major influence on the constraints used for soft parameter sharing in deep neural networks.

2.5.2 The advantages of MTL

We'll suppose that we have two related tasks, α and β, which depend on a single hidden layer representation, Γ, to demonstrate the benefits. [Rud17]

Regularization. MTL adds an inductive bias to serve as a regularizer. As a result, it lowers both the likelihood of overfitting and the Rademacher complexity of the model or its capacity to fit random noise.

Attention focusing. It could be difficult for a model to differentiate between pertinent and irrelevant data if the job is very noisy or the data is sparse and high-dimensional. MTL may help the model focus on the factors that truly matter



(a) Hard Parameter Sharing **(b)** Soft Parameter Sharing

Figure 2.16 Comparasion between Hard Parameter and Soft Parameter. Source: [Rud17]

since future tasks will provide further evidence for the relevance or irrelevance of those traits.

Eavesdropping. A few attributes Θ are simple to learn for some task β but challenging for other tasks α. This may be due to the more intricate interactions between α and the features or because other features are making it more difficult for the model to learn Θ. We can enable the model to eavesdrop using MTL so that it can pick up on task β and learn Θ.[Rud17]

Implicit data enhancement. Our sample size for training our model is effectively increased using MTL. The goal of learning a suitable representation for task α that, ideally, avoids the data-dependent noise and generalizes effectively is to train a model on task α since all tasks are at least somewhat noisy. A model may acquire a more generic representation by learning two tasks at once since various tasks have distinct noise patterns. Learning just task α runs the danger of overfitting to task α, but learning both tasks α and β simultaneously allows the model to average the noise patterns to provide a better representation Γ.[Rud17]

2.6 Data Augmentation

Since we did some data augmentations on HRNet, we would like to introduce some basic ideas and explain why it works well on semantic segmentation. We will only pick some of these methods which we use in the experiment from [MTT19]. In general, the greater the quantity of training data, the greater the accuracy of the trained model and the greater the generalization capacity. A sophisticated deep learning convolution network performs badly on a little dataset. Deep learning's performance on the small dataset is neither greater nor worse to that of

Table 2.3 The effectiveness of semantic segmentation in Deeplabv3+ is affected differently by the two primary augmentation techniques. source: [MTT19]

Category	Mean-IOU
Original	73.28 %
Random flipping	87.24 %
Random cropping	91.00 %

standard approaches. Particularly for certain practical applications, data collection is difficult, and data labeling is time-consuming and costly. Training data enhancement thus plays the most significant function in practice. When training on tiny datasets, it is simple to overfit. Data Augmentation refers to boosting the amount of data samples via different artificial means. Many augmentation strategies have been suggested in prior studies. Methods of enhancement include flipping, rotating, cropping, separating R, G, and B color components, and adding noise, among others. A Generative Adversarial Networks model will be produced via a sophisticated augmentation. In this study, however, we shall just apply these fundamental methods. Numerous augmentation techniques have been successfully used in traditional semantic segmentation networks such as YOLO [Som+01].

These segmentation techniques may be categorized as either global or local augmentation. Global augmentation techniques modify every pixel of a picture using the same transform, such as color transformation, inversion, and projection. Included in local augmentations are random cropping, local pixel translation, and local region copy, which only affect the semantic object area annotated in the training dataset. Experiment findings indicate that all strategies may boost performance, and certainly, some are superior to others. Experimentally, using several data augmentations approaches to add more training data does not necessarily result in improved performance. To improve the performance of dataset augmentation, we must balance the performance and variety of each augmentation approach.

In this section, we would like to refer to [MTT19] for a better understanding of how the data augmentation improves our experiment and focus on the augmentation method that we apply in this study. Although the author use Deeplabv3+ with Resnet101 for feature extraction. Since they don't have enough 385 pictures for training and only 81 picture in the verification dataset, data augmentation helps them with different levels. We can see it as Table 2.3

2.6.1 Global Augmentation

Flipping. Flipping is now the most used way for augmenting data. After Alex Krizhevsky [KSH12] implemented it in ImageNet Classification in 2012, the code for flipping pictures is straightforward and can be used to the majority of issues.

Effectively enhance the model's performance. Becoming a common method of augmenting. There are three horizontal flips, vertical flips, and horizontal flips in the primary uses.

2.6.2 Local Augmentation

Cropping. One of the most popular techniques for data augmentation is random cropping. The network is made less sensitive to scales by cropping, which is similar to a magnification approach and enables it to recognize smaller objects. The centroid of the segmented object is calculated in the experiment of [MTT19] using the mask of the training-set picture. The fundamental point is the centroid. Different cropping frames are produced using various geometrical connections between the centroid and the two random integers. Different enhanced pictures correlate to various reduced boundaries.

For a better understanding of how the data augmentation enhances our experiment in this part, please refer to [MTT19]. Although for feature extraction the author uses Deeplabv3+ and Resnet101 and applied SSG data set (Figure 2.17). They need data augmentation since there aren't enough 385 images for training and only 81 images in the verification dataset. It may be seen as Table 2.3 and Table 2.4.

Figure 2.17 Original image with its label. source: [MTT19]

2.6.3 Section summary

The local augmentation approach and the global augmentation method have often not been compared in a way that stands out, yet both have a major impact on the

Table 2.4 The new data set's accuracy after being integrated in Deeplabv3+. source: [MTT19]

Category	Mean-IOU
Global Augmentation	87.92 %
Local Augmentation	86.31 %
Global Augmentation + Local Augmentation	87.98 %

system's capacity to generalize. The data augmentation method's ultimate purpose is to improve network performance. Usually, many augmentation techniques are used to supplement the data and cross-validate it. Local and global augmentation combined for accuracy of 87.92 percent and 86.31 percent, respectively, although not the highest accuracy. According to the result of [MTT19], we anticipate that random cropping and flipping will improve our results.

3 Amodal Cityscapes

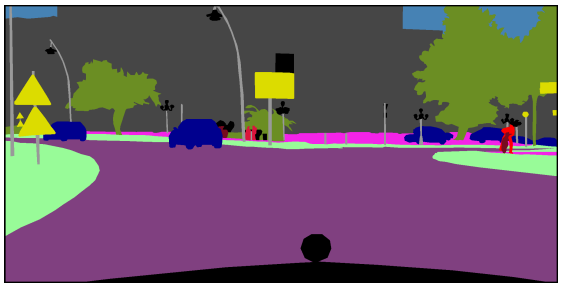
In this research, we primarily apply our approach to Amodal cityscapes; hence, we would want to briefly present it. Amodal perception is the capacity of humans to envisage the whole forms of obscured things. This makes it easier for humans to keep track of everything going on, particularly in busy environments. However, typical perception functions lack amodal perception skills and are consequently disadvantaged in occlusion circumstances. Numerous sorts of occlusions are often encountered in complex urban driving settings; consequently, investigating amodal perception for autonomous cars is a significant endeavor. Here we can have an overview of Amodal Cityscapes dataset. [BF22] (Figure 3.1)

3.1 Dataset splits

The author describes the general method by which a synthetic dataset for amodal semantic segmentation will be generated. In the article [BF22], the author generates an amodal version of the Cityscapes dataset. The Cityscapes collection includes 5000 photos of urban traffic scenes. The training set consists of 2,975 photos, the validation set of 500, and the test set of 1,525. Ground truth labels in the form of semantic and instance segmentation masks are accessible for both training and validation sets. These two subsets are used by the author to build our amodal dataset. In the following, the term target image refers to the picture into which the occluders, i.e. occluding instances, are pasted, whereas source images refer to the original images of the occluders.

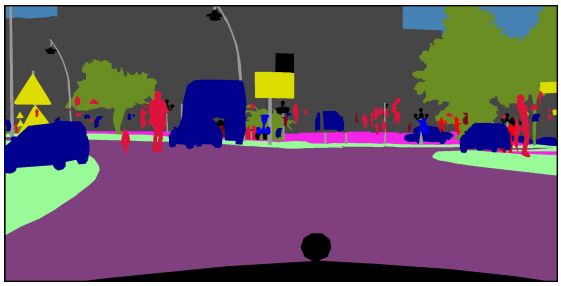
The final dataset includes training, validation, and test splits. As indicated in Table 3.1, the author selected 2900 photos from the original Cityscapes training set as target images for the training split $\mathcal{D}^{\text{train}}_{\text{amCS}}$. The same 2900 photos serve as source images for occluder extraction. However, per target image, the author select only occluders from other images, resulting in a total of $2900 - 1 = 2899$ source images with $36303 - N$ occluders available for pasting into this target image [BF22]. Here, 36303 represents the total number of instances from all source photos, while N represents the number of occurrences in the current target image. For both the validation and test splits, the author generates data in the same manner.

(a) Normal Cityscape image input



(b) Normal Cityscape image label

(c) Amodal Cityscape image input



(d) Amodal Cityscape image visible label

(e) Amodal Cityscape image invisible label

Figure 3.1 Normal and Amodal Cityscape comparison

3.2 Amodal semantic classes

The ERFNet serves as the foundation for the amodal semantic segmentation approach the author provide as a dataset (or challenge) baseline. They modify the normal softmax layer to predict unseen semantic categories. This strategy is based on classifying S semantic classes into four categories. Due to class absence, it will end up with 19 + 1 semantic classes in total (see Table 3.2).

3.3 Experimental evaluation

In this part, the author defines the metrics required to assess amodal semantic segmentation, as well as the Amodal Cityscapes job. The author then describes the

Table 3.1 Split of the Amodal Cityscapes dataset. source: [BF22]

	$\mathcal{D}_{\text{amCS}}^{\text{train}}$	$\mathcal{D}_{\text{amCS}}^{\text{val}}$	$\mathcal{D}_{\text{amCS}}^{\text{test}}$
images	2900	75	500
target images	2900	75	500
source images	2900 - 1	75 - 1	500 - 1
occluders availabel for pasting	36303 - N	832 - N	6438 - N

Table 3.2 Grouping of the semantic classes in the amodal Cityscapes dataset. source: [BF22]

Group name	Semantic classes
Static	road, sidewalk, building, wall, sky, terrain, fence, vegetation, **absence**
Traffic objects	traffic signs, traffic light, pole, **absence**
Person-like	person, rider, **absence**
Vehicle-like	car, truck , bus, train, bicycle, motorcycle, **absence**

training specifics of the baseline approach and evaluates it based on the specified challenge task.

Metrics. For assessment of amodal semantic segmentation on the obtained dataset, the author suggest the following configuration using evaluation metrics based on the generally used in semantic segmentation mean intersection over union (mIoU). In the case of author's baseline technique, the mIoU for the underlying ERFNet without the amodal alterations is also reported. Additionally, the author give two metrics that indicate the segmentation's quality. In order to assess amodal semantic segmentation algorithms, the author first provide $\text{mIoU} = \text{mIoU}^{\text{vis}}$ [BF22], which is computed on the visible portions of the pictures. The second statistic is the invisible mIoU:

$$\text{IOU}^{\text{inv}} = \frac{1}{\text{S}} \sum_{\text{s} \in \text{S}} \frac{\text{TP}_{\text{s}}^{\text{inv}}}{\text{TP}_{\text{s}}^{\text{inv}} + \text{FP}_{\text{s}}^{\text{inv}} + \text{FN}_{\text{s}}^{\text{inv}}}$$

Training Details. The network is trained using the $\mathcal{D}_{\text{amCS}}^{\text{train}}$ Amodal Cityscapes training set. Using the validation split $\mathcal{D}_{\text{amCS}}^{\text{val}}$, the author monitor the training. The amodal semantic segmentation is trained for 120 epochs using the Adam optimizer with an initial learning rate of 0.01 and exponential decay. The author choose the model with the highest performance on the validation set for testing on the test set [BF22].

Baseline. See Table 3.3 for the baseline from ERFNet and its Y-Net. The Y-Net achieves about the same performance as the original mIOU metric, as seen. However, Y-Net may considerably boost mIOU^{inv} from 5 percent to 43.32 percent. In the next sections, we assume HRNet will provide comparable outcomes.[BF22]

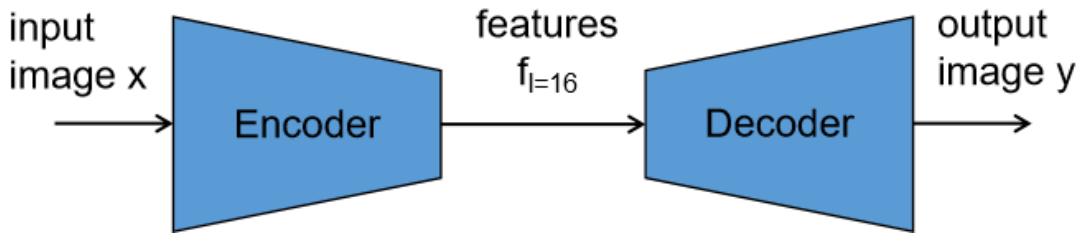
Table 3.3 ERFNet's performance in comparison to other ERFNet methods (the baseline for the Amodal Cityscapes Challenge) on the original and Amodal Cityscapes datasets. source: [JF22]

Method	$\mathcal{D}^{\text{test}}_{\text{amCS}}$		$\mathcal{D}^{\text{val}}_{\text{CS}}$	
	mIOU	mIOU^{inv}	mIOU	mIOU^{inv}
ERFNet	62.99	5.00	67.21	*
$\text{ERFNet}^{\text{am}}$	20.16	36.48	21.00	*
Y-ERFNet	**63.32**	**43.32**	**68.35**	*

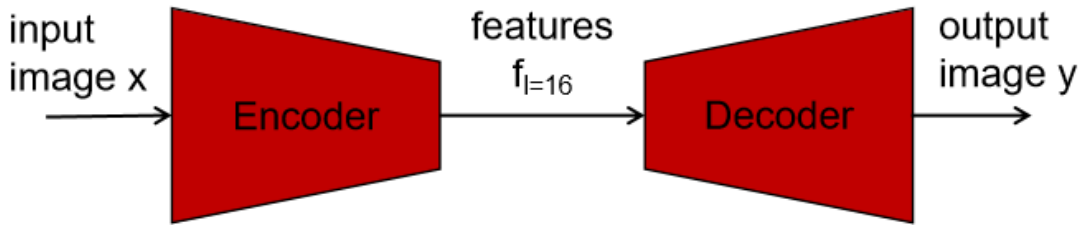
4 Experiment Setup

4.1 Training Configuration

With the exception of the model, we will use the dataset from [BF22] in almost the same environment. We use HRNet as the model since its semantic segmentation is state-of-the-art. The three methods of training resemble Figures 4.1 and 4.2. The network will be trained individually with visible and invisible labels before being trained using the MTL approach, which contains two decoders.



(a) HRNet



(b) $HRNet^{am}$

Figure 4.1 First part of experiment setup

The Amodal Cityscapes training set $\mathcal{D}^{\text{train}}_{\text{amCS}}$ is used to train the network. Utilizing the validation split $\mathcal{D}^{\text{val}}_{\text{amCS}}$, we keep an eye on the training. In the beginning, we utilize the HRNet configuration which has data augmentation from open-mmlab's repository `https://github.com/open-mmlab/mmsegmentation`. However, they use two GPUs to train the whole model, with two workers on each GPU. It indicates that in order to hypothetically get the same outcome with a single GPU, we need to multiply the original iterations (160,000) by 4 and divide the initial learning rate (0.01) by 4. Therefore, we train the amodal semantic segmentation over a period of 640,000 iterations using the SGD optimizer with an initial learning rate of 0.0025 and exponential decay. Open-mmlab used random scaling, random cropping, random flip, and normalizing to the Cityscapes dataset during data augmentation.

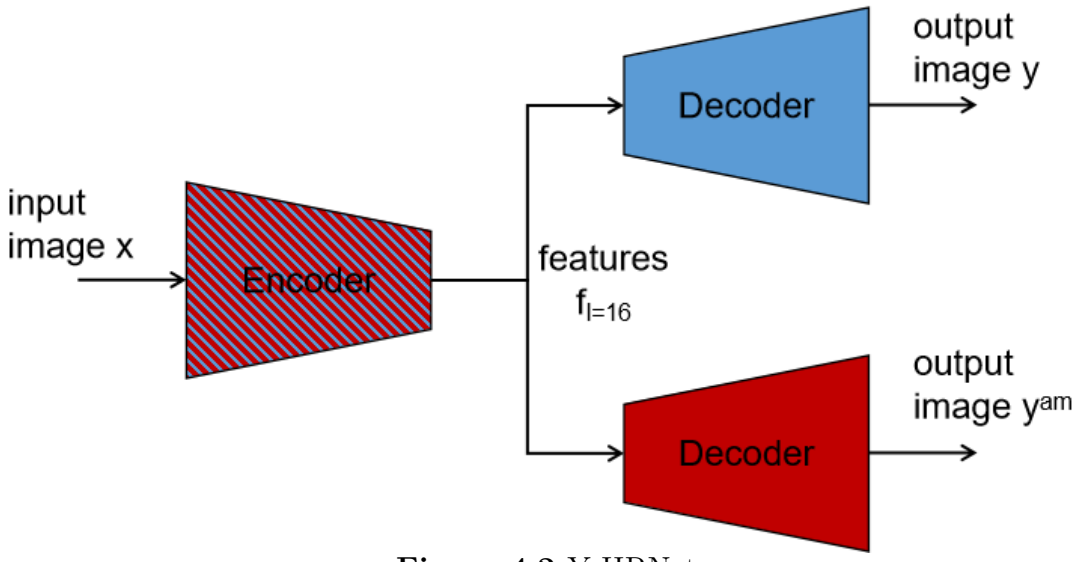


Figure 4.2 Y-HRNet

In addition, we give a comparison between with and without data augmentation, as the original ERFNet lacks this feature. Our data augmentations are random cropping and flipping with a probability of 0.5. For evaluation on the test set, the model with the highest performance on the validation set is selected.

5 Test Evaluation

5.1 Segmentation Results and Discussion

Improved data efficiency, less overfitting due to shared representations, and quick learning due to the use of auxiliary information are all benefits of MTL techniques. Choosing which activities should be taught jointly is a difficult issue in and of itself, and the simultaneous learning of several tasks poses significant design and optimization problems. As a consequence, we anticipate that Table 5.1's results without data augmentation will be less favorable than those that do. If we employ data augmentation, mIOU is obviously improved. But it still falls short of the typical Cityscapes dataset performance, which hovers around 80% in HRNet [Sun+19a].

Data augmentation. HRNet without data augmentation improves the visible portion of the $\mathcal{D}^{\text{test}}_{\text{amCS}}$ by around 2 percent, while HRNet_{DA} improves both the visible and invisible portions of the $\mathcal{D}^{\text{test}}_{\text{amCS}}$. MTL with augmentation improves in both mIOU as predicted, but its improvement on visible is not substantial. On the other hand, the MTL performance cannot even be made to increase observable mIOU without data augmentation. In the performance per class, it is clear that applying data augmentation would enhance HRNet's performance. Usually, it will increase the mIOU by 0.1 to 21% in visible condition, but only by 0.2 to 17% otherwise (Table 5.3, Table 5.4).

Per class performance. Using an amodal dataset, we solely compare ERFNet and HRNet here. Because we want to see the differences, HRNet_{DA} indicates

Table 5.1 Performance in mIOU(%) of the original HRNet and Amodal Cityscapes datasets compared to our proposed HRNet^{am}. Due to the absence of amodal ground truth, fields marked with * cannot be computed.

Method	$\mathcal{D}^{\text{test}}_{\text{amCS}}$		$\mathcal{D}^{\text{val}}_{\text{CS}}$	
	mIOU	mIOU^{inv}	mIOU	mIOU^{inv}
HRNet	**65.70**	5.35	**70.04**	*
HRNet^{am}	2.72	31.19	19.90	*
Y-HRNet	64.14	**38.49**	68.19	*
ERFNet	62.99	5.00	67.21	*
$\text{ERFNet}^{\text{am}}$	20.16	36.48	21.00	*
Y-ERFNet	**63.32**	**43.32**	**68.35**	*

Table 5.2 Performance with data augmentation in mIOU(%) of the original HRNet and Amodal Cityscapes datasets compared to our proposed HRNet^{am}.

Method	$\mathcal{D}^{test}_{amCS}$		$\mathcal{D}^{val}_{CS}$	
	mIOU	mIOU^{inv}	mIOU	mIOU^{inv}
$HRNet_{DA}$	67.79	4.32	73.04	*
$HRNet^{am}_{DA}$	3.03	39.84	2.37	*
$Y - HRNet_{DA}$	**68.41**	**45.45**	**74.73**	*
ERFNet	62.99	5.00	67.21	*
ERFNet^{am}	20.16	36.48	21.00	*
Y-ERFNet	**63.32**	**43.32**	**68.35**	*

that it was trained with augmented data. In four groups—static, traffic objects, person-like, and vehicle-like—we will discuss the performance. The most noticeable improvement in the static group for $HRNet_{DA}$, at least in terms of the visible part, is the wall iteam. Other than that, HRNET does not perform particularly well in static groups. Additionally, group persona and the vehicle are where much of the mIOU progress comes from. Last but not least, $HRNet_{DA}$ enhance group traffic object by around 5%. For the invisible portion, $HRNet_{DA}$ performs almost as well as ERFNet in group static. On the road label, it didn't advance or even regress, but on the fence label, it advanced by 14%. Additionally, only a little amount of progress was accomplished in group traffic object and person-like by $HRNet_{DA}$. Last but not least, it functions effectively as a group vehicle since it improved in each class, with the biggest increase reaching 14%.

Visual results. Based on the findings of HRnet (Figure 5.1), it is evident that the capacity to recognize traffic signs is subpar, although basic humanoid models and automobile model objects can be recognized with precision. The findings of ERFNet indicate that it has a very high capacity for recognition. Presumably, the findings of the two are also consistent with the general improvement of around 5% shown in the Table 5.2, which is not visible to human sight.

(a) ERFnet training inputs

(b) HRNet_{DA} training inputs

(c) ERFNet training label

(d) HRNet_{DA} training label

(e) ERFNet training outputs

(f) HRNet_{DA} training outputs

Figure 5.1 HRNet and ERFNet comparison

Table 5.3 Each class visible performance in mIOU(%)

		$\mathcal{D}^{\text{test}}_{\text{amCS}}$		
		ERFNet	HRNet	HRNet$_{DA}$
Static	Road	**97.35**	95.66	95.76
	Sidewalk	**80.11**	77.28	78.58
	Building	89.08	89.59	**90.66**
	Wall	39.33	38.93	**48.19**
	Fence	41.32	43.46	**50.60**
	Vegetation	90.03	**90.76**	90.66
	Terrain	57.81	**59.72**	50.75
	Sky	92.94	93.27	**93.81**
Traffic Object	Pole	55.75	56.52	**59.11**
	Traffic light	59.76	62.74	**63.97**
	Traffic sign	68.32	70.09	**72.00**
Person-like	Person	66.29	71.17	**71.41**
	Rider	30.34	29.91	**40.36**
Vehicle-like	Car	87.19	89.46	**90.30**
	Truck	47.76	48.79	**69.24**
	Bus	50.51	50.61	**63.28**
	Train	42.87	45.76	**56.85**
	Motorcycle	26.23	33.57	**42.12**
	Bicycle	68.71	71.31	**72.07**

Table 5.4 Each class invisible performance in mIOU(%)

		$\mathcal{D}^{\text{test}}_{\text{amCS}}$		
		ERFNet	HRNet	HRNet$_{DA}$
Static	Road	**85.87**	71.61	71.88
	Sidewalk	58.80	54.81	**60.48**
	Building	65.21	62.96	**68.85**
	Wall	**36.41**	31.25	36.29
	Fence	32.35	36.55	**46.74**
	Vegetation	**65.76**	60.89	64.90
	Terrain	**39.30**	35.38	34.84
	Sky	54.40	46.91	**59.35**
Traffic Object	Pole	34.22	26.48	**36.03**
	Traffic light	**6.32**	0.00	0.00
	Traffic sign	21.22	9.45	**28.42**
Person-like	Person	43.06	41.32	**47.29**
	Rider	24.02	20.55	**30.83**
Vehicle-like	Car	72.06	69.76	**74.20**
	Truck	30.37	26.08	**38.97**
	Bus	50.76	50.54	**54.29**
	Train	44.81	40.49	**57.74**
	Motorcycle	6.70	13.09	**16.38**
	Bicycle	**36.71**	33.16	36.07

6 Conclusion

In the last chapter, section 6.1 should include a summary of the work and findings, as well as a review of the initially defined objectives. Section 6.2 concludes with a discussion of issues that should be improved and altered for future projects. This covers functions that could not be implemented within the scope of this study.

6.1 Summary

The experiment's outcomes are categorized into three main points. This experiment illustrates, first and foremost, that data augmentation enhances mIOU in every manner. Moreover, in each category, the identification of traffic objects has improved the greatest, followed by the recognition of humanoid things and lastly static objects. Under regular HRNet operation (refer to open-lab), the mIOU enhancement is at least 5 percent more than that of ERFNet.

6.2 Future Prospect

Three areas of future study, in my opinion, have the potential to further develop the joint training methods of visible and amodal datasets.

Using different loss functions. Typically, we'll utilize cross-entropy loss as our loss function, however, this approach doesn't perform as well on the imbalance dataset as focused loss does. We may expect that focused loss would be the approach to enhance mIOU in this case since, for instance, our traffic object in the overall image would perceive it as imbalanced data. The formula for focused loss is given in the reference as Figure 6.1, and it is shown that focal loss performs better when the probability of the ground truth class is greater than 0.6. Setting $\gamma > 0$ lessens the relative loss for correctly categorized cases ($p_t > 0.5$) and increases the emphasis on difficult, incorrectly classified examples. [Lin+20]
Modify the training visible and invisible loss ratio. Since our default training ratio is often half, we may think about modifying it via trial and error to get better outcomes.
Make use of the dataset produced by other data generators. An alternate

method to create a more logical dataset that is more suited for cross-matching is to use the CARLA engine.

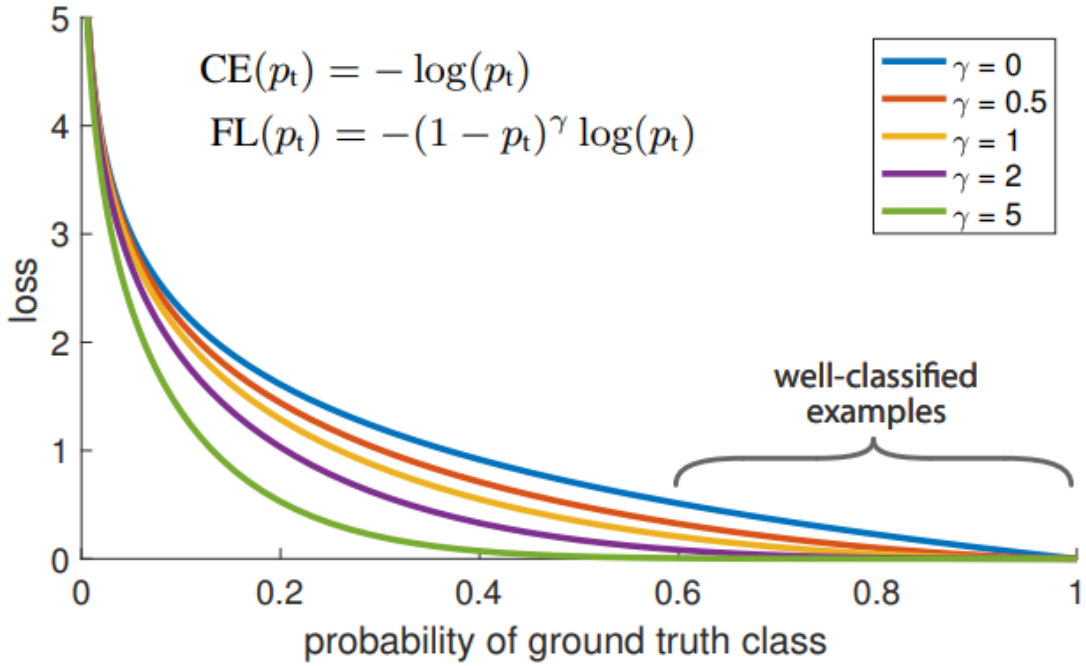


Figure 6.1 Focal Loss. Source: [Lin+20]

Reference

[Bax04] Jonathan Baxter. „A Bayesian/Information Theoretic Model of Learning to Learn via Multiple Task Sampling“. In: *Machine Learning* 28 (2004).

[BF22] Jasmin Breitenstein and T. Fingscheidt. „Amodal Cityscapes: A New Dataset, its Generation, and an Amodal Semantic Segmentation Challenge Baseline“. In: (2022).

[Dos+17] Alexey Dosovitskiy, Jost Tobias Springenberg, Maxim Tatarchenko, and Thomas Brox. „Learning to Generate Chairs, Tables and Cars with Convolutional Networks“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2017), pages 692–705.

[GBC15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. „Deep Learning“. In: *Nature* (2015).

[Gur97] Kevin N. Gurney. „An introduction to neural networks“. In: (1997).

[JF22] Jonas Löhdefink Jasmin Breitenstein and Tim Fingscheidt. „Joint Prediction of Amodal and Visible Semantic Segmentation for Automated Driving“. In: Braunschweig, Germany, 2022.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. „ImageNet classification with deep convolutional neural networks“. In: *Communications of the ACM* 60 (2012), pages 84–90.

[Kul+15] Tejas D. Kulkarni, William F. Whitney, Pushmeet Kohli, and Joshua B. Tenenbaum. „Deep Convolutional Inverse Graphics Network“. In: (2015).

[Lin+20] Tsung-Yi Lin, Priyal Goyal, Ross Girshick, et al. „Focal Loss for Dense Object Detection“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42 (2020), pages 318–327.

[MTT19] Rui Ma, Pin Tao, and Huiyun Tang. „Optimizing Data Augmentation for Semantic Segmentation on Small-Scale Dataset“. In: *Proceedings of the 2nd International Conference on Control and Computer Vision - ICCCV 2019* (2019).

[Pas+16] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. „ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation“. In: *ArXiv* (2016).

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. „U-Net: Convolutional Networks for Biomedical Image Segmentation“. In: (2015).

[Rom+18] Eduardo Romera, José M. Álvarez, Luis M. Bergasa, and Roberto Arroyo. „ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation“. In: *IEEE Transactions on Intelligent Transportation Systems* (2018).

[Rud17] Sebastian Ruder. „An Overview of Multi-Task Learning in Deep Neural Networks“. In: (2017).

[SLD17] Evan Shelhamer, Jonathan Long, and Trevor Darrell. „Fully Convolutional Networks for Semantic Segmentation“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2017), pages 640–651.

[Som+01] Ted Sommer, Bill Harrell, Matt Nobriga, et al. „California's Yolo Bypass: Evidence that flood control Can Be compatible with fisheries, wetlands, wildlife, and agriculture“. In: *Fisheries* 26 (2001), pages 6–16.

[Sun+19a] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. „Deep High-Resolution Representation Learning for Human Pose Estimation“. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019).

[Sun+19b] Ke Sun, Yang Zhao, Borui Jiang, et al. „High-Resolution Representations for Labeling Pixels and Regions“. In: (2019).

[Yam+18] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. „Convolutional neural networks: an overview and application in radiology“. In: *Insights into Imaging* 9 (2018), pages 611–629.