Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Image segmentation for detecting objects specific to urban environment

Students' Scientific Conference Report

Author:

Gábor Révy

Advisor:

dr. Gábor István Hullám

2019

Contents

Kivonat **i**

Abstract **ii**

1 Basics of convolutional neural networks **1**

1.1 Neural networks . . . 1

1.2 Basic layers . . . 1

1.2.1 Fully connected layer . . . 1

1.2.2 Convolutional layer . . . 2

1.3 Activation functions . . . 3

1.3.1 Sigmoid . . . 4

1.3.2 ReLU . . . 4

1.3.3 Softmax . . . 5

1.4 Batch Normalization . . . 5

2 Previous works **6**

2.1 Simple Fully Convolutional Networks . . . 6

2.1.1 Encoder . . . 6

2.1.2 Decoder . . . 6

2.2 U-Net . . . 7

2.2.1 Encoder . . . 7

2.2.2 Decoder . . . 7

2.3 Feature Pyramid Network . . . 7

2.4 Mask R-CNN . . . 8

2.5 DeepLabv3+ . . . 9

2.5.1 Atrous convolution . . . 10

2.5.2 Depthwise separable convolution . . . 10

2.5.3 Encoder . . . 11

2.5.3.1 Feature extractor . . . 11

2.5.3.2 Atrous separable pyramid pooling . . . 11
2.5.4 Decoder . . . 12
2.6 HRNet . . . 12
2.6.1 Architecture . . . 12
2.6.1.1 Feature extractor . . . 13
2.6.1.2 Classifier . . . 13

3 Implementation **15**
3.1 Framework . . . 15
3.2 Data generator . . . 15
3.3 Defining networks . . . 15
3.3.1 DeepLabV3+ . . . 16
3.3.2 HRNet . . . 16
3.3.3 Training . . . 17

4 Dataset **18**
4.1 Cityscapes . . . 18
4.2 BDD100K . . . 18
4.3 Labeling policy . . . 18

5 Evaluation **20**
5.1 Benchmark . . . 20
5.1.1 Intersection over union . . . 20
5.1.2 Confusion matrix . . . 20
5.2 Results . . . 21
5.2.1 Overall results . . . 21
5.2.2 Comparison of results of different methods . . . 22
5.3 Comparison of results using different training datasets . . . 29
5.4 Comparison of results using BDD as evaluation dataset . . . 37

6 Conclusion **41**

Acknowledgements **43**

Bibliography **44**

Appendix **46**
A.1 Confusion matrices . . . 46

Kivonat

A grafikus kártyák számítási teljesítményének növekedésével, továbbá az ezeken alkalmazott algoritmusok kifinomultabbá válásával az elmúlt években jelentősen fejlődött a számítógépes látás. A klasszikus képfeldolgozási módszerek mellett egyre szélesebb körben terjedtek el a mély neurális hálózat alapú megoldások. Ezen módszerek jellemzője, hogy megfelelő tanító adathalmaz rendelkezésre állása esetén a minták alapján képesek - bizonyos korlátok között - bármilyen leképezést megtanulni. A gépi képfeldolgozás egyik fontos területe a képszegmentálás, amely során egy képet úgy osztunk részekre, hogy egy-egy rész egy adott objektumtípushoz tartozzon. Így tehát a kép minden egyes pixeléhez rendelünk egy címkét, ami megmondja, hogy az adott pixel milyen típusú objektumhoz tartozik. Városi környezetben jellemző címkék például: gyalogos, járda, út, bicikli, autó vagy közlekedési tábla. Ezen feladat automatizált megoldása nagyon sok előnyt jelentene, főleg, ha megfelelően rövid idő alatt képes az algoritmust lefuttatni egy számítógép. A városi környezetben megtalálható objektumok felismerésére kialakított képszegmentációs algoritmusok egy ígéretes felhasználási területe a vezetést támogató rendszerek, melyek egyre több autóban vannak jelen. Ezen eszközök egy részéhez ugyanis elengedhetetlen, hogy a rendszer szemantikailag értelmezni tudja az autót körülvevő objektumokat.
Célom egy olyan mély neurális hálózat alapú rendszer elkészítése, mely képes egy városi közlekedési környezetre jellemző képet szegmentálni. Ehhez először áttekintem a tématerülethez kapcsolódó releváns szakirodalmat, valamint megvizsgálok egypár korszerű, jó teljesítményt nyújtó módszert. Mindezek alapján létrehozok egy képszegmentáló rendszert és implementálom az azt tanító algoritmust. Ezt követően kiértékelem az egyes variánsokat a Cityscapes és a BDD100K adathalmazokon különböző teljesítménymutatók, mint például intersection over union alapján.

Abstract

Computer vision has evolved significantly in recent years with the improvements in computational performance of graphics cards and the sophistication of the algorithms executed on them. In addition to classical image processing methods, deep neural network based solutions have become increasingly widespread. These methods are characterized by the ability to learn, within certain limits, any mapping based on the samples, provided that an appropriate learning dataset is available. An important field of image processing is image segmentation, in which an image is divided into parts such that each part belongs to a particular object type. So, we assign a label to each pixel of the image that tells us what type of object that pixel belongs to. Labels specific to urban environments include: pedestrian, sidewalk, road, bicycle, car, or traffic sign. An automated solution to this task would have many benefits, especially if the algorithm could be executed in a reasonably short amount of time by a computer. One promising application of image segmentation algorithms for recognizing objects in urban environments is driving assistance systems, which are present in more and more cars. For some of these tools, it is essential that the system can semantically interpret the objects around the car.
My goal is to create a deep neural network based system that can segment the image of an urban traffic environment. In order to do this, I first review the relevant literature on the topic and examine a couple of state-of-the-art, well-performing methods. Based on that, I create an image segmentation system and implement the algorithm that trains it. Then I evaluate each variant on the Cityscapes and BDD100K datasets based on various performance indicators, such as intersection over union.

Chapter 1

Basics of convolutional neural networks

1.1 Neural networks

Neural networks are function approximators. This means that they can approximate the mapping between their input and output. To achieve this we have to train the network on a bunch - usually thousands - of examples. The most common training method is gradient descent. First, we have to define a loss function that measures the difference between the expected and the predicted output. During training, we want to minimize this loss value by computing the gradient of the loss function with respect to each parameter and modifying them.

1.2 Basic layers

Neural networks consist of layers, at least two of them: an input layer, optional hidden layer(s) and an output layer. The first layer serves as input for the model. The following layers get the input from the previous layer, transform it and send it to the next layer. The last layer serves as the output of the network.

1.2.1 Fully connected layer

Fully connected layers are essential components of neural networks. The universal approximation theorem has been proven for a neural network containing a single hidden layer. However, in this case, the width of the layer would be very large. In practice, it seems that complex functions can be learned better using deep neural networks than a single hidden layer-network with the same number of neurons. There have been many false "proofs" for this theory in literature [5], but all have holes.

Each layer consists of nodes. The nodes in fully connected networks are commonly called as neurons, referring to the biological analogy. A neuron first takes the weighted sum of all the outputs from the neurons of the previous layer as shown in Figure 1.1 (1.1).

$$s_i^l = \sum_{j=1}^{n} w_{j,i}^l a_j^{l-1} + b_i^l \tag{1.1}$$

s_i^l is the weighted sum of the ith neuron in the lth layer and $w_{j,i}^l$ weights the a_j^{l-1} value where j is the index of the neuron from the previous $(l-1\text{th})$ layer and there are n layers in the previous layer. b_i^l is the bias value, which is another degree of freedom helping the network to fit better by making it possible to shift the activation function.

Then, a non-linear function is applied to it (1.2), often referred to as the activation function.

$$o_i^l = a(s_i^l) \tag{1.2}$$

This is the output of the ith neuron in the lth layer. Parameters b and w are learnable through gradient descent [1].

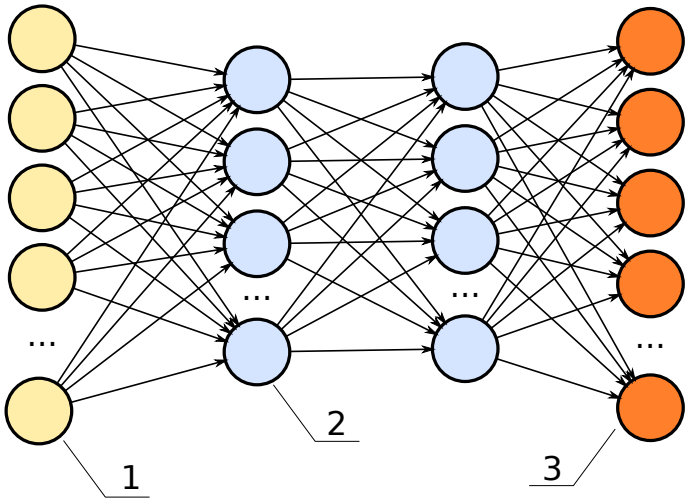

Figure 1.1: A neural network containing fully connected layers. All the nodes in the given layer are connected with all the nodes from the previous layer. 1 - input layer, 2 - hidden layers, 3 - output layer, source:[1]

1.2.2 Convolutional layer

Fully connected layers are used for a variety of problems but they have a serious disadvantage: when applied to image processing, the parameter space explodes. For example, we have a (medium size) $500 \times 500 \times 3$ RGB image so we put 64 neurons in the second layer. This results in $500 \cdot 500 \cdot 3 \cdot 64 = 48000000$ trainable parameters without bias!

The solution to this problem comes from classical image processing. It adapts to the properties of images:

- relation of pixels to each other has semantic meaning,
- elements are shift-invariant,

allowing to reduce the number of the parameters needed. The main idea behind convolutional layers is the kernel which is a small matrix we slide over the image (or in the hidden layer over the feature map). The kernel values are multiplied by the pixel values under

[1] https://commons.wikimedia.org/wiki/File:Neural_network_bottleneck_achitecture.svg

them and summed for each position of the kernel in the image as shown in Figure 1.2. Mathematically this can be formulated as:

$$o_z^l(x,y) = \sum_c \sum_{a,b} y_c^{l-1}(x \cdot s + a, y \cdot s + b) \cdot w^l(a,b,c,z) + b_z^l, \tag{1.3}$$

where $o_z^l(x,y)$ is the non-activated output value in the lth layer's zth channel in the pixel position (x,y). y_c^{l-1} is the output of the $l-1$th layer's cth channel. s is the stride we use during computation: after the matrix multiplication between the kernel and the image the kernel is shifted by this value. w and b are the weight and bias values.

The next step is to activate each value in the matrix using a non-linear function (1.4).

$$y_z^l(x,y) = a(o_z^l(x,y)) \tag{1.4}$$

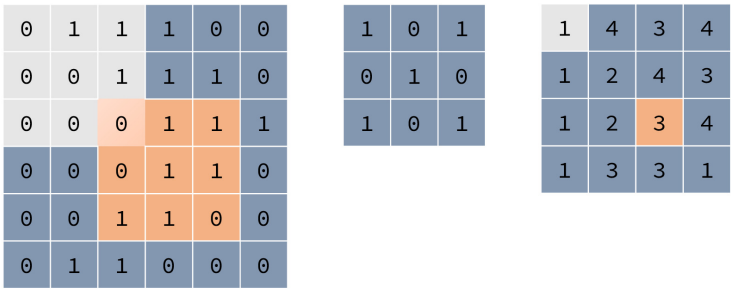

Figure 1.2: Convolution process. The initial matrix, the kernel, and the result matrix formed by convolution.

For example, let us assume that we have the $500 \times 500 \times 3$ RGB image and we would like to have a 250×250 feature map with 64 channels in the second layer using a 3×3 kernel, then we need $3 \cdot 64$ kernels of size 3×3. This results in $3 \cdot 64 \cdot 3 \cdot 3 = 1728$ parameters. Even if we need multiple layers, this solution scales better in terms of the number of parameters.

Visualizing the first hidden layer's activations, basic image processing step results can be observed, such as detecting corners, edges, sharpening, blurring. Going deeper, patterns get more complex indicating that these layers learn more abstract information, recognizing objects not only on a given image but generalized for object classes. That is one of the reasons why this type of deep neural network became so widespread.

1.3 Activation functions

If linear activation function is applied to the layers' values, then no matter how many hidden layers in the neural network is stacked, the final output is still a linear combination of the input data. For neural networks to function as universal approximators a non-linear function must be used.

1.3.1 Sigmoid

Nowadays, sigmoid function is used in binary classification problems to activate the output layer. However, sometimes this is also used in case of segmentation. If we have objects which can be classified into more than one classes, we can use sigmoid activation. Its output is between 0 and 1 ((1.5)). We can interpret this value as a certainty about a class.

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)} \tag{1.5}$$

Earlier, sigmoid activation function was also used to activate hidden layers. As *deep* neural networks became widespread, this caused a problem. The derivative function of sigmoid is the following:

$$\text{sigmoid'}(x) = \text{sigmoid}(x) \cdot (1 - \text{sigmoid}(x)) \tag{1.6}$$

Figure 1.3 shows the graph of the sigmoid and its derivative.

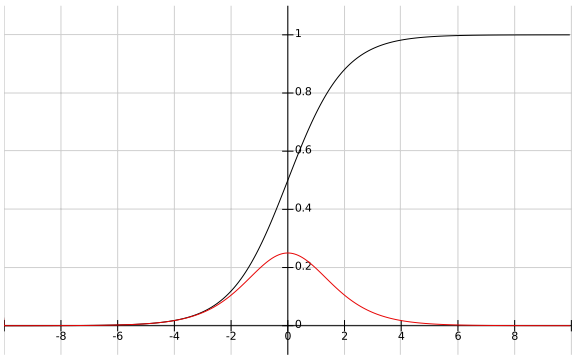

Figure 1.3: Sigmoid function and its derivative.

At very high and very low values sigmoid function gets saturated meaning that the corresponding gradients become almost zero. Gradient descent uses the chain rule to back-propagate the error and compute the gradient with respect to each weight. This means that many derivative values are multiplied to get the gradient of the weights in the lower layers. Multiplying many almost-zero values results in almost zero gradient, which leads to a very slow learning process. This phenomenon is called gradient vanishing.

1.3.2 ReLU

The solution for the vanishing gradient problem is using rectified linear unit (ReLU) activation function in the hidden layers. The ReLU function is defined as:

$$\text{relu}(x) = \max(0, x) \tag{1.7}$$

The derivative of the ReLU function is:

$$\text{relu'}(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{otherwise} \end{cases} \tag{1.8}$$

At this point, the reason why this solution works is observable: when the derivative is back-propagated there will be no vanishing of the error signal as it is multiplied by ones (in most cases). In the networks I examine ReLU activation is used in the hidden layers precisely because of its aforementioned property.

1.3.3 Softmax

Softmax activation function is applied always to the last layer of the network. It is used for multi-class classification problems, like segmentation. The ratio of the exponential of one logit and the sum of exponentials of all logits is computed so it sums up to one (1.9). In other words it turns logits into a probability distribution over the different classes.

$$\text{softmax}(x_n) = \frac{\exp x_n}{\sum_i \exp x_i} \tag{1.9}$$

1.4 Batch Normalization

Batch normalization is a widely used technique enabling deep neural networks to train faster and more stable by rescaling the output of a layer. It was introduced by Sergey Ioffe and Christian Szegedy [9] from Google research lab with the aim of reducing "internal covariate shift". ICS refers to the change in the distribution of layer inputs caused by updating previous layers. Such continuous change has a negative impact on training. The results show that the method works, however, the exact reason is not clarified completely. There are some theorems aiming to clarify this reason, for example, [15].

Batch normalization transforms activation distributions to have zero mean and a unitary variance by controlling the mean and variance of the layers' outputs [9]:

$$BN(y_j)^{(b)} = \gamma \cdot \left(\frac{y_j^{(b)} - \mu(y_j)}{\sigma(y_j)} \right) + \beta, \tag{1.10}$$

where $y_j^{(b)}$ denotes the j^{th} output on the b^{th} input of the batch, μ and σ are the mean and standard deviation of the output computed over the batch. Two learned parameters are introduced: β and γ, controlling the mean and standard deviation of the output.

Chapter 2

Previous works

To be familiar with the previous works I've started my thesis with a literature review on deep learning methods for segmentation. In parallel let me introduce the basic idea of the segmentation networks.

2.1 Simple Fully Convolutional Networks

Simple Fully Convolutional Networks (*FCN*) are very popular. J. Long et al. [12] have been the first group to develop a Fully Convolutional Network (FCN) (containing only convolutional layers) trained end-to-end for image segmentation. By simple FCNs I refer to the bottleneck architecture as shown in Figure 2.1. Generally speaking this is the main idea behind the semantic segmentation networks. Information has to flow through a more compact representation forcing the network to interpret the image and convert it to that smaller format.

One of the biggest advantages of the FCNs is that once the kernels are learned, it can be used for any input resolution - with some restrictions. In my case, I trained my networks on 320×640 pixel images. One of my networks uses downsampling by 32. This means that in order to enable appropriate upsampling from the bottleneck, input resolution must be divisible by 32.

2.1.1 Encoder

The first part of these networks is an encoder. Using 2-dimensional convolution, average-pooling and max-pooling layers a so-called feature map is made. This is the semantically interpreted format of the input image which has much smaller resolution than the input image. In later models, other types of convolution layers appeared, for example, dilated convolution or strided convolution layers. In parallel, pooling layers disappeared.

2.1.2 Decoder

The second part is the decoder. This stack of layers recovers the information using 2-dimensional convolution and upsampling layers to the original resolution. On the last layer, a logical activation function - usually softmax - is applied. Its value is calculated for every pixel of the image indicating which class it belongs to.

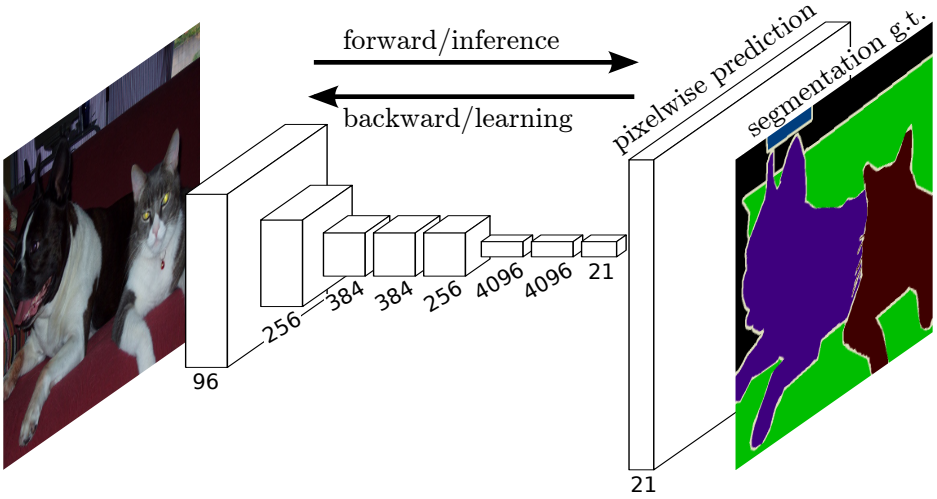

Figure 2.1: A simple fully convolutional network for image segmentation. source: [12]

2.2 U-Net

The U-Net [14] is a type of FCN. It could reach good results in biomedical imaging in 2015.

2.2.1 Encoder

It has an encoder called contractive path. It consists of repeated application of two 3×3 convolution layers, each followed by ReLU activation and a 2×2 max-pooling layer with a stride of 2 for downsampling as shown in Figure 2.2. At each downsampling step the number of feature channels is doubled.

2.2.2 Decoder

In the decoder part - called expanding path - every step consists of an upsampling layer, a convolutional layer which halves the number of the feature map channels, a concatenation with the corresponding layer from the contractive path and two 3×3 convolutions with ReLU activation. At the end a 1×1 convolution is applied to reduce the number of the channels to the number of the classes.
Here, the most important architectural solution is that the second part looks like the mirror image of the first part as can be seen in Figure 2.2. This makes it possible to concatenate layers with same resolution from the encoder to the decoder.

2.3 Feature Pyramid Network

The Feature Pyramid Network [11] is used in object detection and segmentation frameworks. It is based on the same basic underlying principle as the U-Net, with a slight difference in implementation. The encoder part - called bottom-up pathway - is processing the input image with convolutional and pooling layers. After the bottleneck, the decoder part - called top-down pathway - is upsampling the feature map stage-by-stage

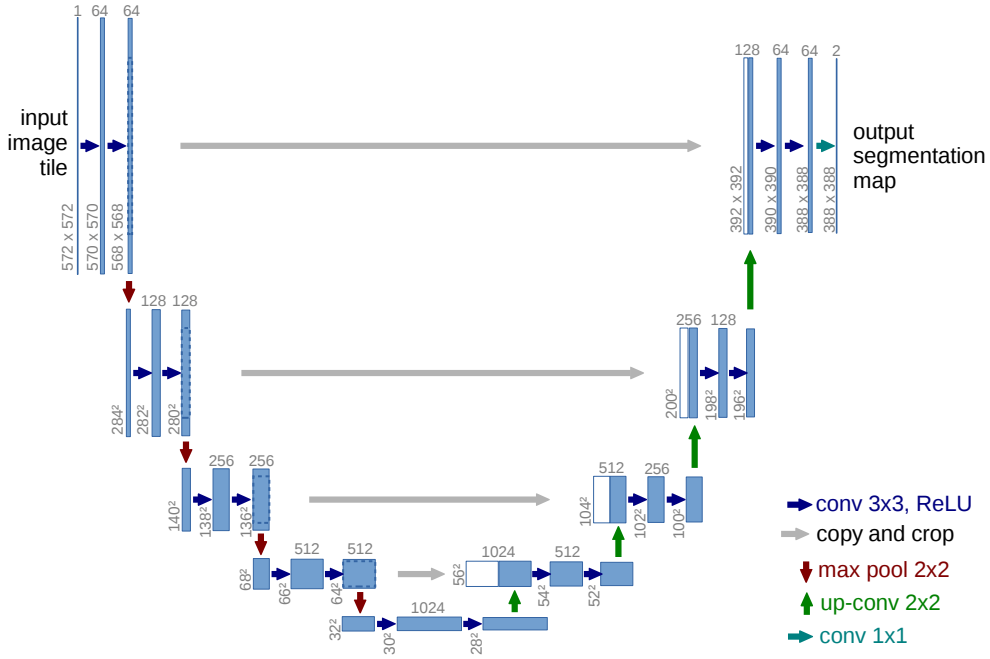

Figure 2.2: The architecture of U-Net. The most notable feature is that there are connections between the corresponding layers of the U-shape, which have similar size. source: [14]

merging channel-reduced feature maps from the bottom-up pathway by element-wise addition as shown in Figure 2.3. Finally, 3×3 convolution is applied on each stage merged feature maps to reduce the aliasing effect of the upsampling. For image segmentation, a 5×5 and a 7×7 MLP is used to predict object masks with different size over the objects.

2.4 Mask R-CNN

In 2017, the Facebook AI Research (FAIR) group have released the Mask R-CNN [7] model beating other systems on some of the Common Objects in Context (COCO) [10] challenges. This segmentation system builds on top of Faster R-CNN [13], which is an object detection model. First, I briefly present the architecture of Fast R-CNN.
First, a ResNet 101 [6] architecture backbone extracts features from the image. This acts as the input for the next part, the Region Proposal Network (RPN). RPN is a small convolutional network sliding over the feature map. It extracts *objectness scores* for *anchor boxes*. Objectness score is basically a [0, 1] value indicating if there's an object in the anchor box. An anchor box is a predefined-size box which is used to predict *bounding boxes* by scaling them. A bounding box is a rectangle closely enclosing the object. The regions obtained from the RPN are then fed into the RoI to bring the regions to the same size and then predict class labels and bounding boxes using a fully connected network. Here is the main difference between Mask R-CNN and Faster R-CNN: in Mask R-CNN, segmentation masks are also predicted at this last step using a convolutional network as shown in Figure 2.4.

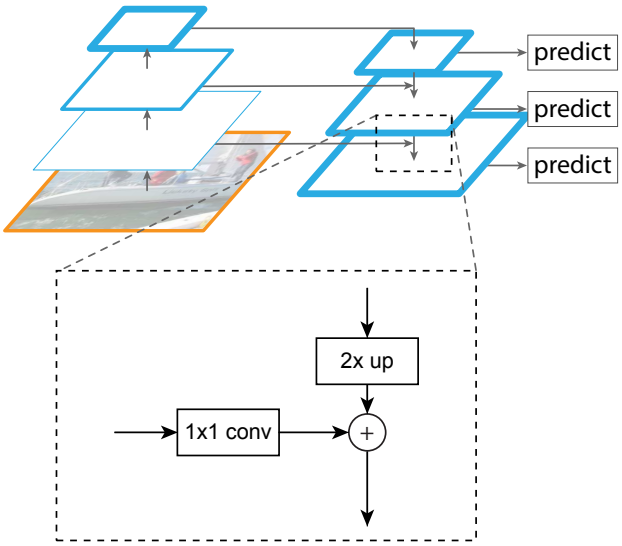

Figure 2.3: The architecture of Feature Pyramid Network. Two pyramids can be observed: the encoding (bottom-up pathway) and the decoding (top-down pathway) pyramids and the lateral connections between them. source: [11]

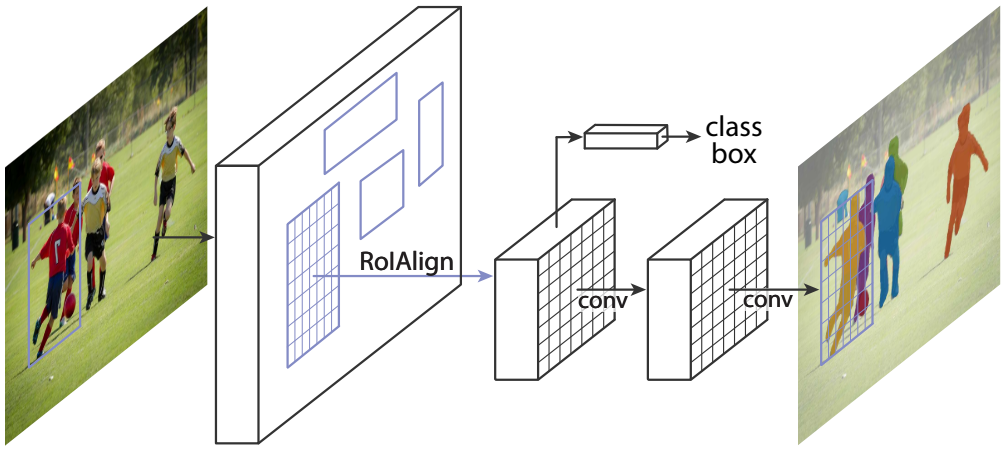

Figure 2.4: The architecture of Mask R-CNN. Bounding box, class and segmentation mask are predicted. source: [7]

2.5 DeepLabv3+

DeepLabv3+[2] was designed by software engineers of the Google Research group. This is also encoder-decoder structured but has more extra features which make it more accurate. In Figure 2.5 you can see the overall structure of the whole network. In the next sections, I present its architecture and the architectural features built in it.

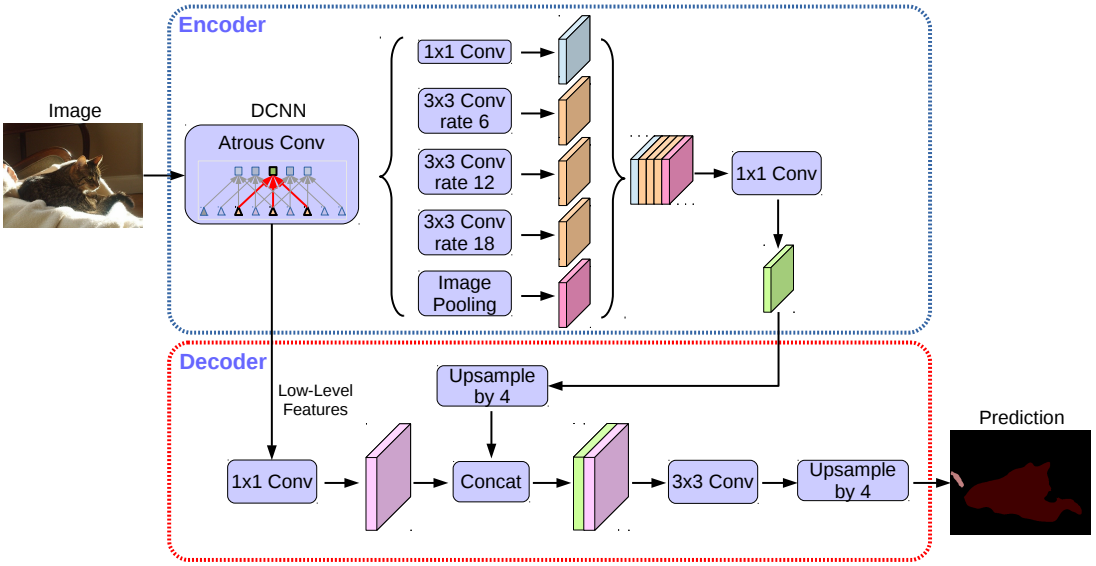

Figure 2.5: The architecture of DeepLabv3+. source: [2]

2.5.1 Atrous convolution

Atrous convolution (also called dilated convolution) is a type of convolution that has been described in [17]. The standard convolution can be computed as follows:

$$(F * k)(\mathbf{p}) = \sum_{\mathbf{s}+\mathbf{t}=\mathbf{p}} F(\mathbf{s})k(\mathbf{t}) \tag{2.1}$$

When applying dilated convolution we have a new parameter l, called the dilation factor.

$$(F *_l k)(\mathbf{p}) = \sum_{\mathbf{s}+l\mathbf{t}=\mathbf{p}} F(\mathbf{s})k(\mathbf{t}) \tag{2.2}$$

As can be seen, there are "holes" between the points the kernel is applied to. The connection between standard and atrous convolution is when dilation rate $l = 1$ as shown in Figure 2.6. The motivation for the usage of this type of convolution is the expansion of the receptive field without loss of resolution or coverage. The receptive field is a square of exponentially increasing size, while the number of parameters grows linearly.

2.5.2 Depthwise separable convolution

First, let's take a look at the hyperparameter number of the standard convolution. Let us suppose, we have a $height_{inp} \times width_{inp} \times channel_{inp}$ size input layer and the next layer has the size of $height_{next} \times width_{next} \times channel_{next}$. If we use standard convolution we have $kernel_{height} \times kernel_{width} \times channel_{inp} \cdot channel_{next}$ number hyperparameters to optimize and a lot of multiplications increasing the evaluation time.

A depthwise separable convolution separates the standard convolution process into 2 parts: a depthwise convolution and a pointwise convolution. This means fewer hyperparameters and less evaluation time. Staying with the previous example, first we apply depthwise convolution by using a kernel of size of $kernel_{height} \times kernel_{width} \times channel_{inp}$ to get a

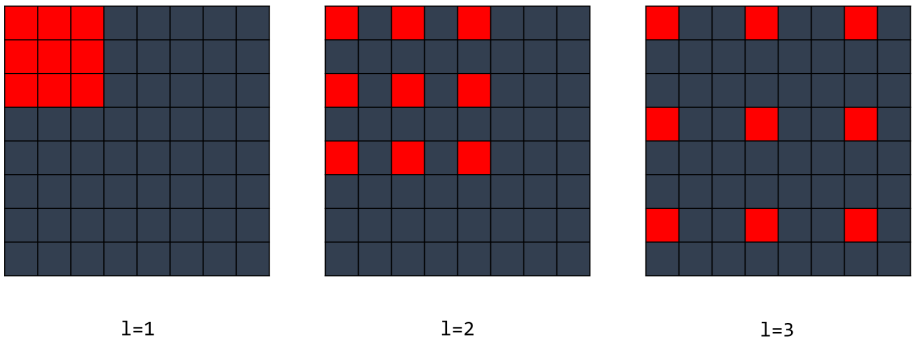

Figure 2.6: Dilated convolution with different dilation rates: l=1 (standard convolution), l=2, l=3.

result of a size of $height_{next} \times width_{next} \times channel_{inp}$. Then, we apply pointwise convolution to change the channel size: this convolution has a kernel of the size of $1 \times 1 \times channel_{inp} \cdot channel_{next}$.

Depthwise separable convolutions are popular in mobile nets (neural networks for low power devices), for example in MobileNet [8], where it is optimized for computational complexity. In Xception [3] network separable convolution was used to enhance Inception-v3, thus outperform it.

2.5.3 Encoder

The encoder of the DeepLabv3+ model consists of two parts: a feature extractor and an atrous separable pyramid pooling.

2.5.3.1 Feature extractor

The feature extractor of DeepLabv3+ has two variants: with Xception backbone or with ResNet-101. I implemented the one with Xception backbone because it reached better results.

The Xception has three big blocks: the entry, the middle and the out flow. Each flow contains a different number of residual blocks with separable convolution layers and a max-pooling layer in the end. These max-pool layers have been replaced with atrous separable convolution layers using a stride value of 2 in the DeepLabv3+ model as it can be seen in Figure 2.7.

2.5.3.2 Atrous separable pyramid pooling

The second part of the encoder is the atrous separable feature pooling. This means, that atrous separable convolution is applied here using several $(6, 12, 18)$ dilation rate values on different branches. This allows it to recognize objects of different size. A 1×1 convolution and global pooling are also applied to the Xception's output. All five branches' output is upsampled to the same size. After concatenating them, a 1×1 convolution is applied to reduce the number of channels.

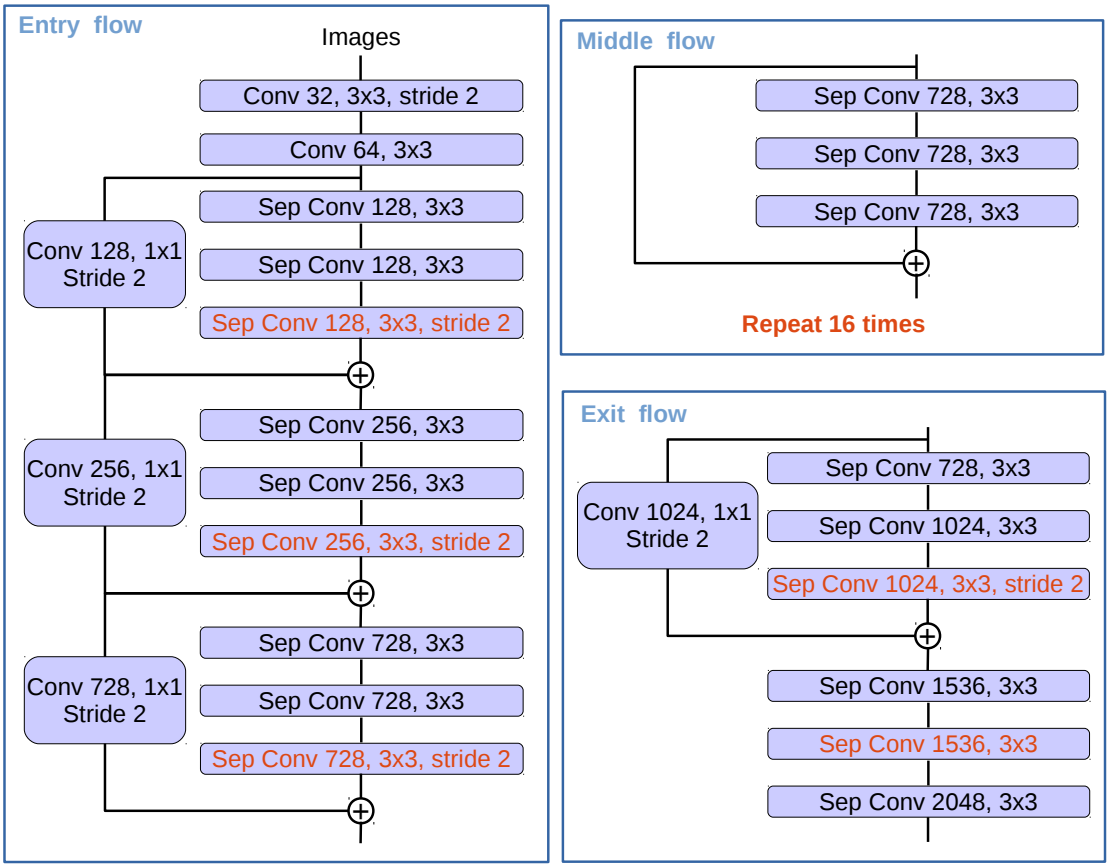

Figure 2.7: Modified Xception network in DeepLabv3+: max-pool layers replaced with strided atrous separable convolution. source: [2]

2.5.4 Decoder

The decoder part starts in two branches. One of its inputs is the output of a low-level feature layer from the entry flow of the feature extractor. It's a simple skip connection (referred to as hypercolumn in the paper), which is necessary to recover object segmentation details. It is connected to the upsampled output of the atrous separable pyramid pooling. After upsampling and some 3×3 convolution layers, we get the prediction of the network.

2.6 HRNet

High-resolution [16] network was designed by the Microsoft Research for several applications, for example pose estimation, object detection, facial landmark detection, keypoint detection and pixel segmentation.

2.6.1 Architecture

The main idea behind the structure differs from the previously presented ones'. Whereas the former are essentially encoder-decoder-structured, this one doesn't follow that line.

Instead of downsampling to low resolution representation and then recovering the high-resolution representation from it by continuously upsampling and using skipping connections, this network maintains the full-resolution stream that carries the high-resolution information as shown in Figure 2.8. Each unit in the downsample and upsample subnetworks exchanges information with the full-resolution stream (and also each other), thus replacing the skip connections.

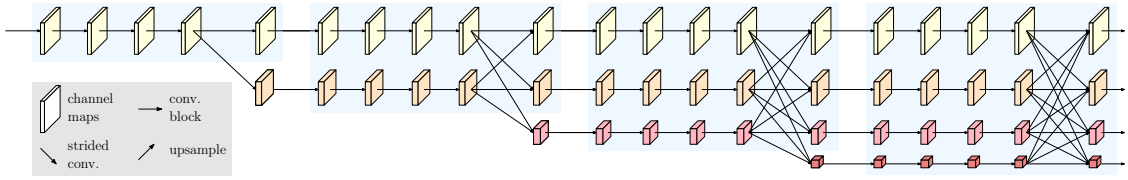

Figure 2.8: The architecture of the HRNet's feature extractor. Instead of the encoder-decoder structure, the full-resolution stream is maintained. source: [16]

The segmentation network consists of two main parts: the feature extractor and the classifier.

2.6.1.1 Feature extractor

HRNet was designed to have several applications so it has several variants. This is the part that is common to all of them. It consists of four stages. The first stage starts after two strided 3×3 convolution layers which reduce the resolution by 4. Each stage starts with a transition from the previous layer. This means that the corresponding resolution streams are simply connected and a new stream is created with half the resolution of the smallest resolution in the previous stage and twice the width of it. After it, the first stage contains 4 bottleneck blocks. A bottleneck block includes a 1×1 convolution to create a bottleneck with a width of 64 followed by a 3×3 and a 1×1 recovering the width and adding to the input using a residual connection.

Each stage ends with a fusion block. The fusion block serves the purpose of mixing the information of the different resolution stream's output. To this end, stream's outputs are converted to the size of the corresponding stream using strided convolution for downsampling and bilinear interpolation for upsampling as Figure 2.9 shows. These converted outputs are then added.

The 2nd, 3rd and 4th stage contain 4 multi-resolution blocks instead of the bottleneck block. This block contains 2 of 3×3 convolutions and a residual connection using summation.

2.6.1.2 Classifier

The output of the feature extractor connects to the classifier. The classifier contains 2 multi-resolution blocks, but there's no transition before them thus the number of different resolution streams remains 4. The output of the streams is concatenated and channel-reduced through a 1×1 convolution. Finally, the resulting feature map is upscaled by 4 to the input resolution and softmax activation is applied.

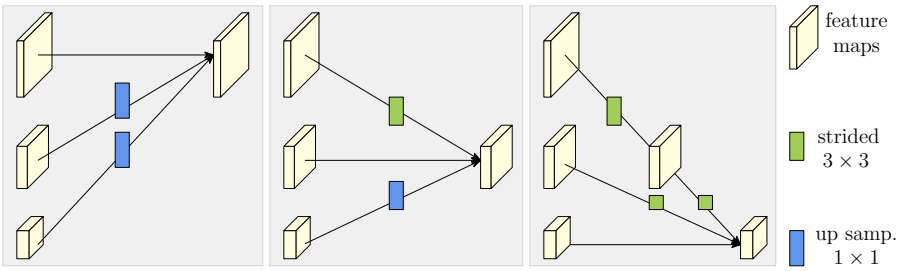

Figure 2.9: The operation of the function block. Different resolution feature maps are converted to the same size by (repeated) 3×3 strided convolution (downsampling) and bilinear interpolation with 1×1 convolution (upsampling). source: [16]

Chapter 3

Implementation

After examining the different networks we chose two networks with my supervisor. The main aspects of selection were (1) good performance on the images taken in an urban environment and (2) publicly available architecture. We chose DeepLabV3+ and HRNet based on the current state of the applied methods table on the webpage of the Cityscapes[1] benchmarks.

3.1 Framework

To implement the different algorithms Python 3 was used. Keras[2] library provided a rich environment for defining various neural networks and their training algorithms.

3.2 Data generator

After investigating the labeling policy I created a dataset generator. This makes it possible to generate training, validation and test data on multiple threads while training or evaluating the network on the GPU in parallel. On the one hand, a batch of input images is read in sequence, resized to input size and normalized by rescaling pixel values from [0; 255] to [0; 1]. On the other hand, a batch of label images is read, resized and encoded to one-hot tensors pixel-by-pixel.

After some training sessions I realized that reading from the hard drive is a bottleneck while training. So while generating the batches in the first epoch, images are also cached in the memory. This method is a little bit memory-intensive, but the batch generation is more than 10 times faster because in the further epochs reading from the drive is not needed. With this solution, GPU utilization can be kept constantly above 90%.

3.3 Defining networks

Defining the network with the Keras Functional API is very spectacular. It allows defining multi-output models, directed acyclic graphs, or models with shared layers. A layer is

[1] `https://cityscapes-dataset.com/benchmarks/#pixel-level-results`
[2] `https://keras.io`

defined by its type and symbolic tensor is returned which is the output of the layer and the input of the next layer. It is very useful when defining residual connection or concatenation.

3.3.1 DeepLabV3+

As far as the exact architecture is concerned, I set out from the paper and the original implementation[3]. I could have downloaded the ready-made model but implementing it for myself allowed me to get to know it deeper and also allowed further experimentation concerning both the structure and the parameters. Also, I think, an implementation in Keras is readable for a wider audience. The problem with the paper was that its purpose is to get a comprehensive picture of the system, not to go into all details. Thus I had to investigate the original code written in the TensorFlow[4] framework which can not be read easily.

First, the feature extractor was implemented. Since the 2-dimensional convolution layer was used frequently along with batch normalization and ReLU activation a function was defined to shorten code and make it more clear. This is also made possible by the use of symbolic tensors. An exception block function was also made for the same purpose. Concatenation was useful when defining atrous separable pyramid pooling because at the end different branches must be concatenated. I wanted the network to be input resolution-independent. The only layer preventing it was the global pooling layer which takes a feature map as input and returns the average of the values on it. After the pooling, different branches in the pyramid pooling must be converted to the same resolution. But without knowing the input resolution the upscaling factor after the global pooling layer can't be defined. Thus two versions of the network were created: the one with global pooling layer is input resolution-dependent and the other one without it is input resolution-independent. For a model to be input resolution-independent means that it can be trained and used in inference time with different resolution images.

An interesting thing is that batch normalization already includes the functionality of the bias on the layer it is applied to. When implementing the layer, this means, that bias should be disabled as it has no impact on the output and takes up space.

3.3.2 HRNet

As in the previous case, I set out from the paper and the original implementation[5]. The description of the paper is quite detailed, however I had to examine the original implementation thoroughly. Its code is written in the Pytorch[6] framework which was new to me.

Defining this network requires caution but is simple. First, the basic units were described: the bottleneck block, the transition, the fusion and the multi-resolution blocks. Implementing the fusion is the most complex part, since information from each stream must be fused into each other resulting in a confusing graph. Then larger components, i.e. the feature extractor part and the classifier part were implemented and merged into the model.

[3] https://github.com/tensorflow/models/tree/master/research/deeplab
[4] https://tensorflow.org/
[5] https://github.com/HRNet/HRNet-Semantic-Segmentation
[6] https://pytorch.org/

3.3.3 Training

Keras also makes the implementation of the training easier since gradient computation is handled by the backend. One just needs to specify the loss function. In both of the models categorical cross-entropy was used as loss function. Training using various optimizers have been tested: Adam, SGD, and RMSprop.

Several callback functions were also defined. These functions are called by Keras during training with a given frequency allowing to log changes or respond to changes. Model checkpointer is used to save the model if it improved based on the loss computed on the validation dataset. Although it helps to avoid overfitting by saving only when validation loss improved, training would run further without better results. Early stopping solves this problem by stopping the training if no improvement is observed after a given number of epochs. An epoch is a training unit when error is calculated and the parameters are updated for each sample of the dataset. The "reduce learning rate on plateau" callback reduces the learning rate by a given factor once learning stagnates. A custom callback was made to visualize the improvement of the network. At the end of each epoch, a batch of images is fed into the network to predict segmentation mask so that training can be followed not only by the loss and accuracy values but also visually. TensorBoard[7] was used to track and compare loss and accuracy changes during the training sessions. TensorBoard provides a graphical user interface and plots graphs for the different metrics which is available from the browser.

[7] `https://tensorflow.org/tensorboard`

Chapter 4

Dataset

In my work I used the Cityscapes [4] and the Berkeley DeepDrive BDD100K [18] dataset.

4.1 Cityscapes

Cityscapes is a well known open source database which is recorded during the span of several months, covering spring, summer, and fall in 50 cities of Germany and neighboring countries. It contains 5000 fine and 20000 coarsely annotated images focusing on semantic understanding of urban street scenes. I used the coarse dataset containing roughly labeled images. Then I trained the network on the fine dataset containing the same images with fine annotations. In both cases the dataset contains 2975 train and 500 validation images. Finally, I used the Cityscapes video sequences to illustrate the results.
There are 34 class definitions into which the elements can be classified. The labeling format is very simple, each pixel has a label consisting of a class number from 0 to 33 saved in a PNG image, see Figure 4.1.

4.2 BDD100K

The Berkeley Deep Drive BDD100K dataset is mainly captured from the different areas of the US road infrastructure and highway traffic signs, and it is comparable to the Cityscapes dataset. This dataset also includes object detection, lane detection, drivable area, and semantic instance segmentation datasets. For my work I used 7000 images for training and 1000 images for validation.

4.3 Labeling policy

At first I used all the 34 classes from the Cityscapes dataset but then I also wanted to use the BDD100K dataset. They claimed to be compatible with Cityscapes labeling, however they were not entirely correct. There's another labeling in Cityscapes called trainId. This means that one can choose the classes to train on and all the other classes will be labeled as "out of region of interest" *(roi)*. BDD100K was using this labeling policy choosing 19 of the original label types. So I had to convert the labels to make them compatible between the two data sets in order to compare or use them together. This resulted in a labeling which contains 15 classes and 1 out of roi class.

Figure 4.1: Sample images from the coarse and the fine dataset of Cityscapes.

Chapter 5

Evaluation

5.1 Benchmark

5.1.1 Intersection over union

A common metric for evaluating object detection and segmentation networks is intersection over union (IoU) also referred to as the Jaccard index. Pixels common between the ground truth and prediction masks divided by the total number of pixels present across both masks are measured for each class as can be seen in Figure 5.1.

$$IoU = \frac{intersection}{union} = \frac{predicted\,pixels \cap ground\,truth\,pixels}{predicted\,pixels \cup ground\,truth\,pixels}$$

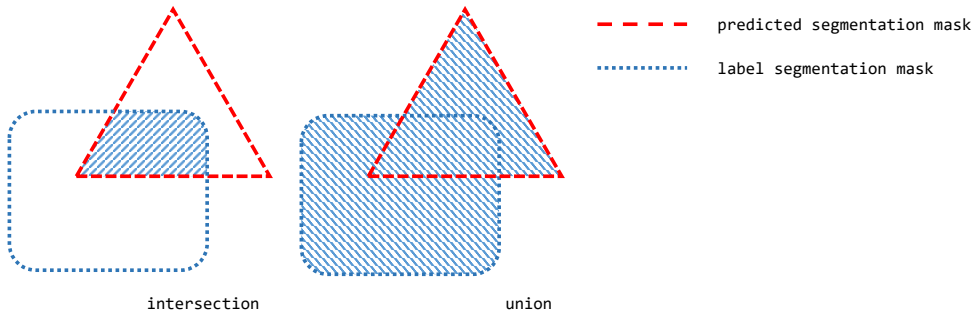

Figure 5.1: Intersection and union of predicted and label segmentation masks.

5.1.2 Confusion matrix

Confusion matrices are also computed to illustrate the results. This is a simple, pixel-level metric. The row of the confusion matrix is indexed by the labeled class of the pixel and the column is indexed by the predicted class of the pixel. For every pixel in the image of the test dataset, 1 is added to the corresponding cell of the confusion matrix.

5.2 Results

After training the networks were evaluated. Since the test datasets originally used in the paper are not available, validation datasets were created and used for this purpose. Three type of architectures were evaluated: HRNet, DeepLabv3+ and the modified DeepLabv3+ without the global pooling layer. All three networks were trained on the Cityscapes fine, on the Cityscapes coarse and on the BDD dataset resulting in 9 trained models. All the trained networks were evaluated on the Cityscapes fine and the BDD dataset which means 18 evaluations in total. Intersection over union metrics and a confusion matrix were computed for each case.

5.2.1 Overall results

To get an overview mean intersection over union (mIoU) was computed as shown in table 5.1. This means that IoU (as a result of evaluation) was averaged over the classes for each of the networks. These values are not expressive in themselves, but they allow the comparison of the networks using a single metric.

Method	**Training set**	**Test set**	**mIoU**
HRNet	Cityscapes - Fine	Cityscapes - Fine	**0.4900**
		BDD	0.1719
	Cityscapes - Coarse	Cityscapes - Fine	0.3281
		BDD	0.1438
	BDD	Cityscapes - Fine	0.3506
		BDD	0.3981
DeepLabv3+	Cityscapes - Fine	Cityscapes - Fine	**0.4906**
		BDD	0.2213
	Cityscapes - Coarse	Cityscapes - Fine	0.3438
		BDD	0.1531
	BDD	Cityscapes - Fine	0.3238
		BDD	0.3894
DeepLabv3+ w/o global pooling	Cityscapes - Fine	Cityscapes - Fine	**0.5206**
		BDD	0.2312
	Cityscapes - Coarse	Cityscapes - Fine	0.3381
		BDD	0.1537
	BDD	Cityscapes - Fine	0.3569
		BDD	0.4294

Table 5.1: The mean intersection over union depending on the network, the training dataset and the evaluation dataset.

As can be observed, the best result was reached with the modified DeepLabv3+ trained and evaluated on the Cityscapes fine dataset. However, when using the BDD dataset for the evaluation, modified DeepLabv3+ gives much worse results. This is due to two main reasons. The first is that although different datasets (training and validation) and early stopping were used, overfitting with respect to the training dataset is still likely. The reason is that there are similar pictures in the training and the validation datasets in the Cityscapes database even though they were recorded in different cities. The other possible reason is the difference between the Cityscapes and BDD datasets. This is caused by the usage of a different camera, camera position, area and thus the distribution of the classes.

In the next sections, the results of the evaluations will be detailed from different points of view. First, results of different methods are compared using the same dataset (the Cityscapes fine dataset). Then results, obtained by training the same network (the modified DeepLabv3+) on the various databases are described. Here the evaluation takes place on the same database (on the Cityscapes fine dataset). Finally, the results of the evaluation on the BDD database are detailed. Here, the modified DeepLabv3+ was used for the test, which was trained on BDD and Cityscapes fine. The other results are not detailed here, but the corresponding confusion matrices are included in the appendix.

5.2.2 Comparison of results of different methods

For the sake of comparability of the various methods, results are reported (see Figure 5.2) below under the same conditions (meaning same training, validation and evaluation set). For this purpose Cityscapes fine dataset was chosen.

Figure 5.2 shows the accuracy and loss over the training steps. Here, accuracy refers to the ratio of the number of well-labeled pixels and the number of pixels. A step means evaluating a batch of samples. The validation accuracy and loss values are computed step-by-step over the whole validation dataset after each epoch.

Learning curves are very similar. The only slight difference is that *DeepLabv3+ without global pooling layer* was trained for a few more epochs, i.e. the accuracy of the other two methods did not improve after 60.000 validation steps. All three methods reached an accuracy of 0.85 after approximately 30.000 validation steps. Afterwards the accuracy for *HRnet*, *DeepLabv3+*, and *DeepLabv3+ w/o global pooling* oscillated between (0.86,0.8874),(0.8537 0.8873), and (0.85,0.8881) respectively. In terms of validation loss, the loss of HRnet varies to the largest extent between (0.3808, 0.5516).

Based on the intersection over union values shown in Figure 5.3 the modified DeepLabv3+ network seems to be the most accurate. It performs reasonably better in case of "person", "traffic sign", "traffic light" instances. The only class where HRNet performed better is "train". Interestingly, this class has the smallest number of instances in the dataset. HRNet outperformed the original DeepLabv3+ at several classes, for example, "motorcycle", "train", "truck", "sky" and "pole".

Figure 5.4, 5.5 and 5.6 show the confusion matrix of the various networks trained and evaluated on the Cityscapes fine dataset. The rows of the matrix represent the true label of an object, and the columns represent the predicted labels. The values in the diagonal show the ratio of correctly classified objects. For example, in case of the "sidewalk" object in the second row of Figure 5.5, the rate of "sidewalk" objects correctly classified is 0.76, whereas some of the "sidewalk" objects is classified as "road" 0.15 or "other" 0.04.

The confusion matrices indicate that certain objects are detected with high accuracy by all three methods (*DeepLabv3+*, *modified DeepLabv3+*, *HRNet*) such as "road" (0.97, 0.95, 0.93 rates respectively for the three methods), "building/wall/fence" (0.92, 0.90, 0.89), "vegetation" (0.91, 0.94, 0.93), "sky" (0.96,0.95,0.95), and "car" (0.93,0.92,0.90). The classification of other objects such "person" are less accurate (0.72, 0.76, 0.77), and in some cases it is missclassified as "building" (0.13, 0.08, 0.10). Furthermore, objects that appeared relatively less frequently in the training data set are often missclassified by all three methods, such as "truck" objects: as "truck" (0.11, 0.27, 0.28), as "bus" (0.23, 0.34, 0.03), as "car" (0.37, 0.19, 0.32), as "building" (0.17, 0.10, 0.23). Note that there are differences between the methods as the modified *DeepLabv3+* tends to classify a "truck" as a "bus", whereas in case of the other two it is more likely that it is classified as a

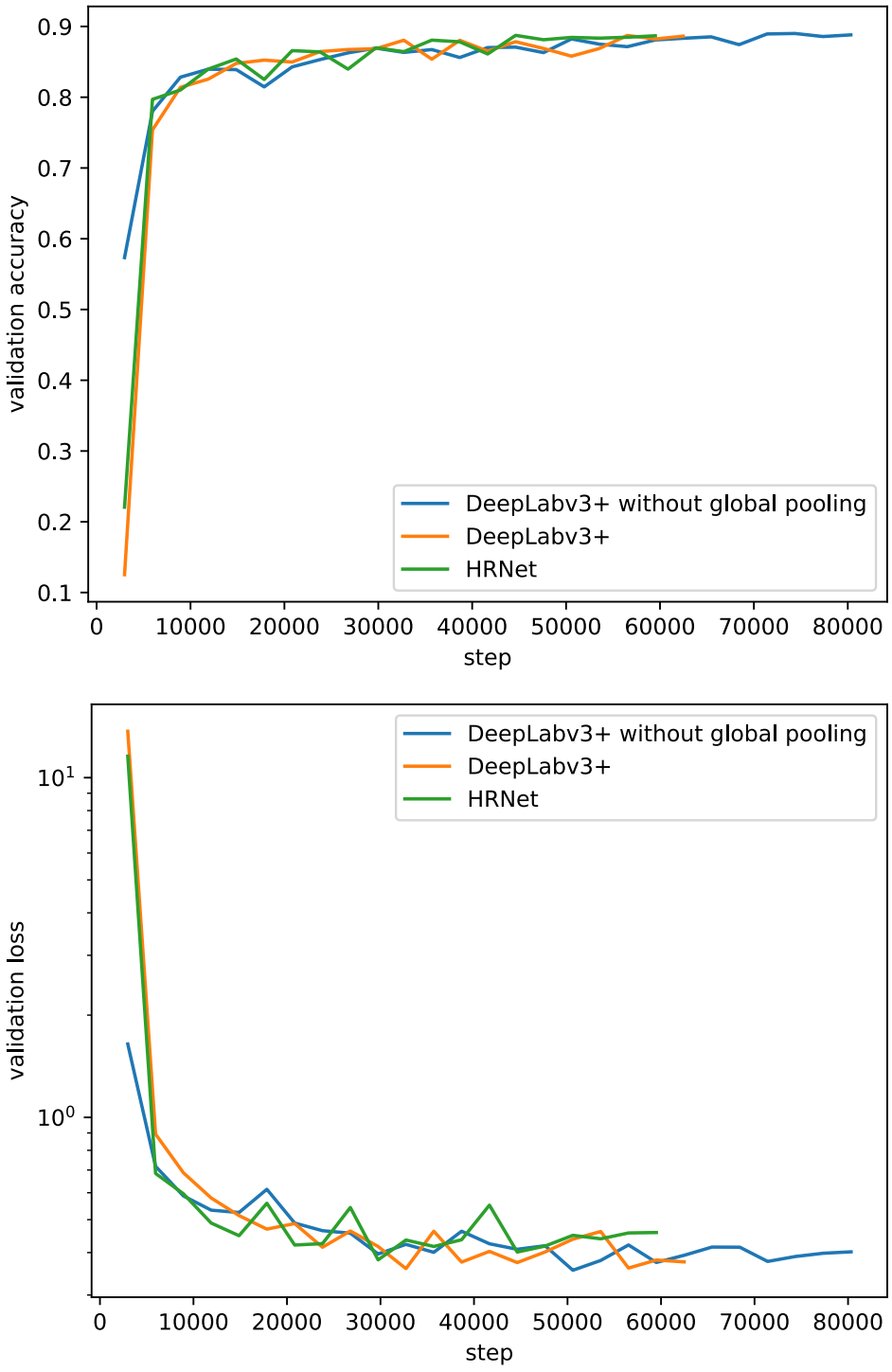

Figure 5.2: Accuracy and loss, computed on the validation dataset during the training of the various networks on the Cityscapes fine dataset.

"car". Although it can be argued whether classifying a "truck" as a "bus" or as a "car" are equally erroneous, it is definitely a greater error to classify it as a "building" which is more probable in case of *HRNet* (0.23) and DeepLabv3+ (0.17) than in case of the *modified DeepLabv3+* (0.10).

In addition, there are cases in which considerable differences can be observed. "Traffic light" and "traffic sign" objects are more accurately identified by *modified DeepLabv3+* with corresponding rates for correct classification of 0.50 and 0.61. In case of *DeepLabv3+* these rates are lower (0.33 and 0.48 respectively for "traffic light" and "traffic sign"), and also in case of *HRNet* these rate are similarly low (0.25 and 0.49). Interestingly, all methods missclassify "traffic light" objects to some extent either as "vegetation" (0.17, 0.19, 0.30 respectively for *DeepLabv3+*, *modified DeepLabv3+*, and *HRNet*) or as "building" (0.42, 0.24, 0.36). Another interesting difference between results that both *DeepLabv3+* and *HRNet* missclassify "train" objects as "buildings" (0.46 and 0.44 respectively), whereas the correct classification rate is low (0.02 and 0.14) which is probably due to the low frequency of "train" objects in the training samples. In contrast, the *modified DeepLabv3+* missclassifies "trains" as "buses" (0.54) with only identifies "trains" correctly in a small portion of the cases (0.14).

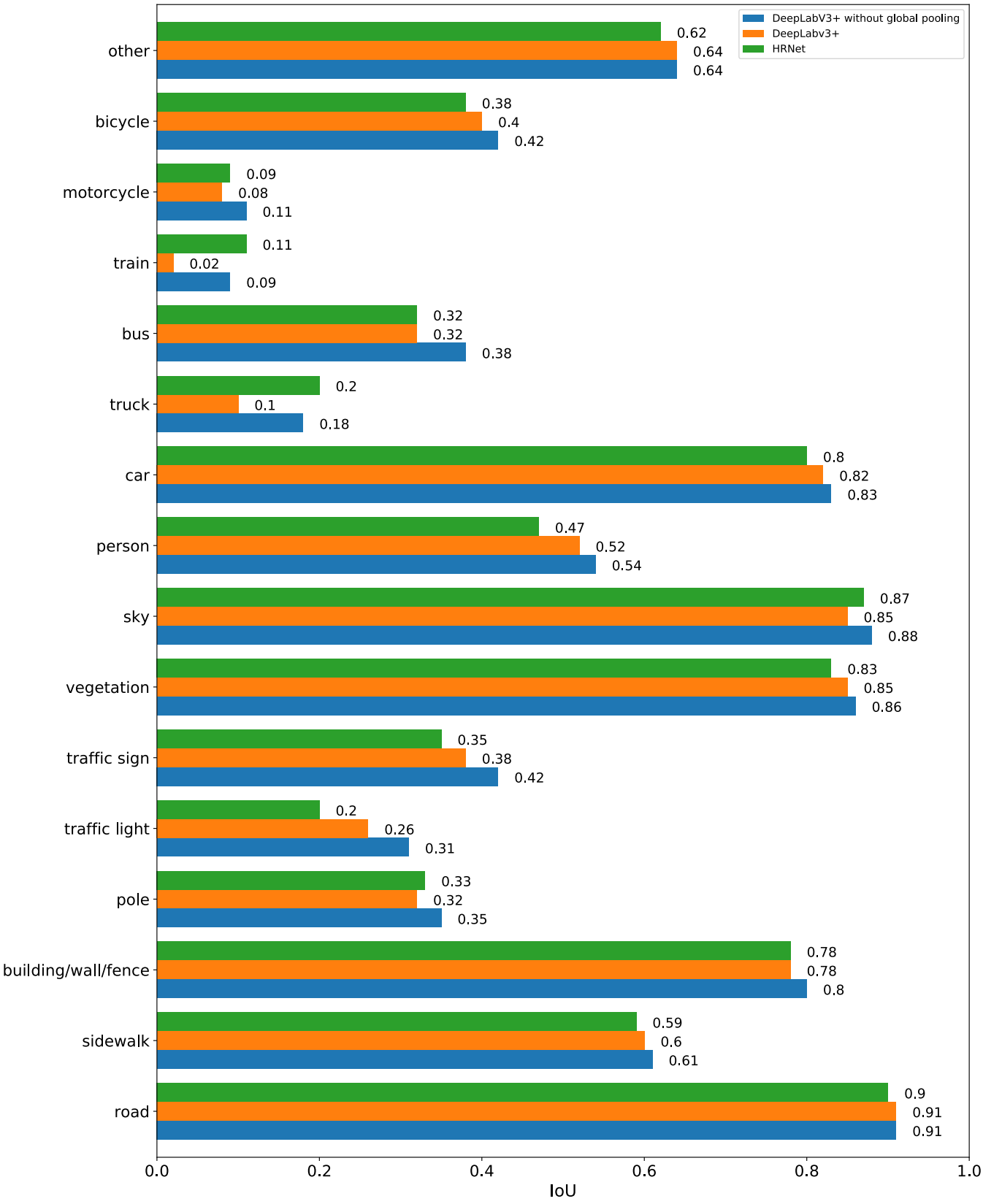

Figure 5.3: Intersection over union values, computed by evaluating the three different architectures trained (and evaluated) on Cityscapes fine.

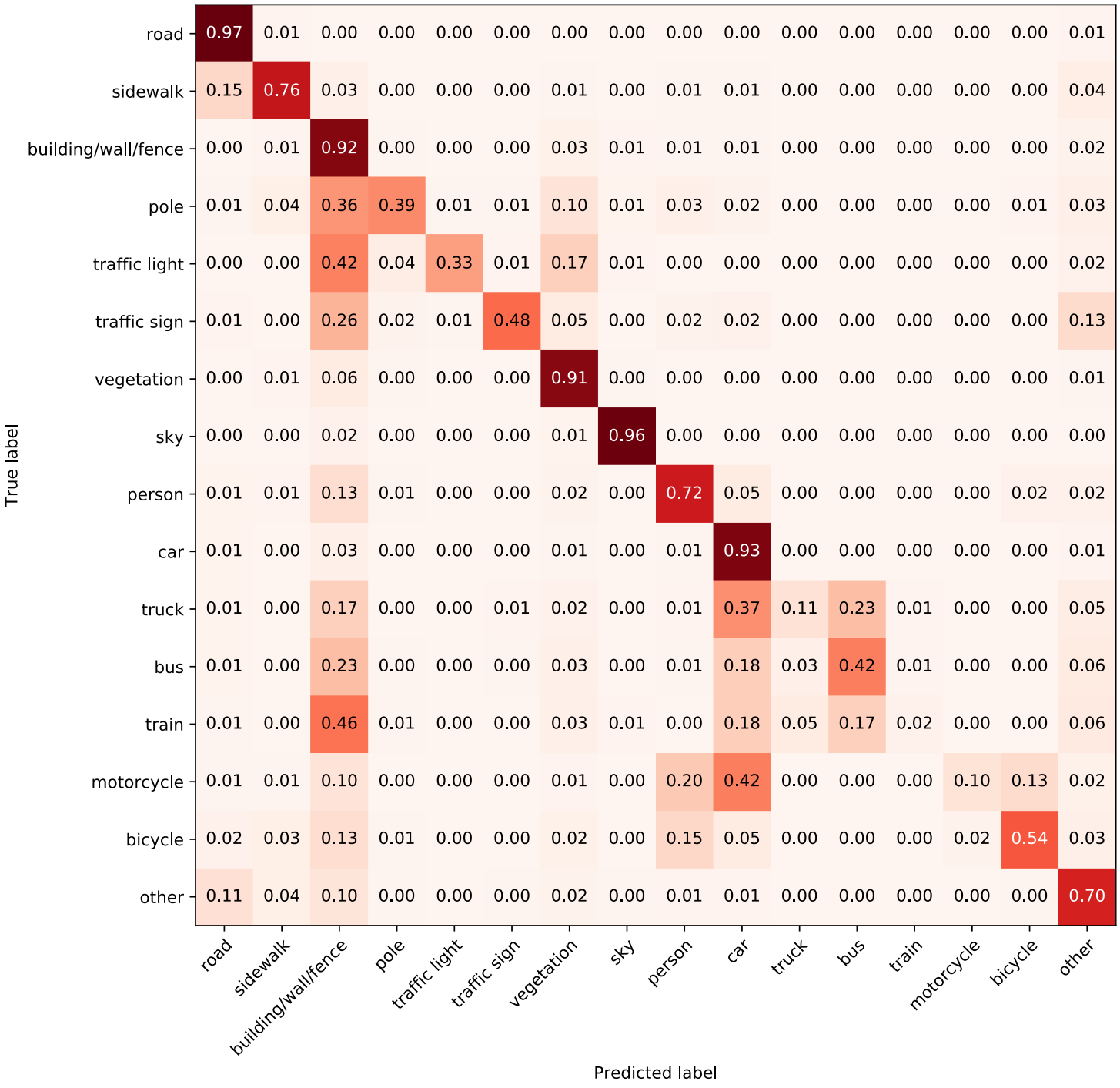

Figure 5.4: Confusion matrix of DeepLabv3+ trained on Cityscapes fine, evaluated on Cityscapes fine.

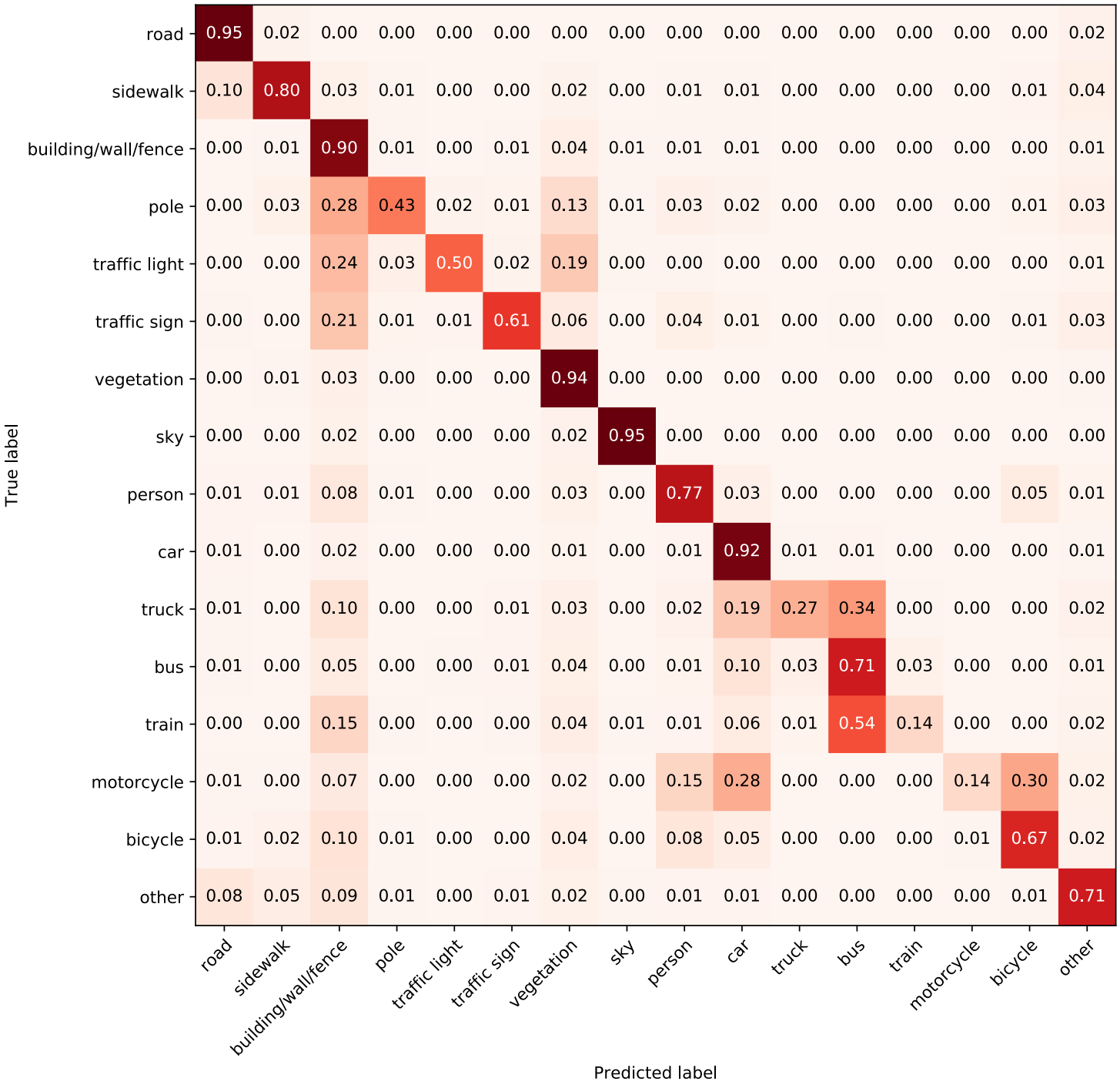

Figure 5.5: Confusion matrix of the modified DeepLabv3+ trained on Cityscapes fine, evaluated on Cityscapes fine.

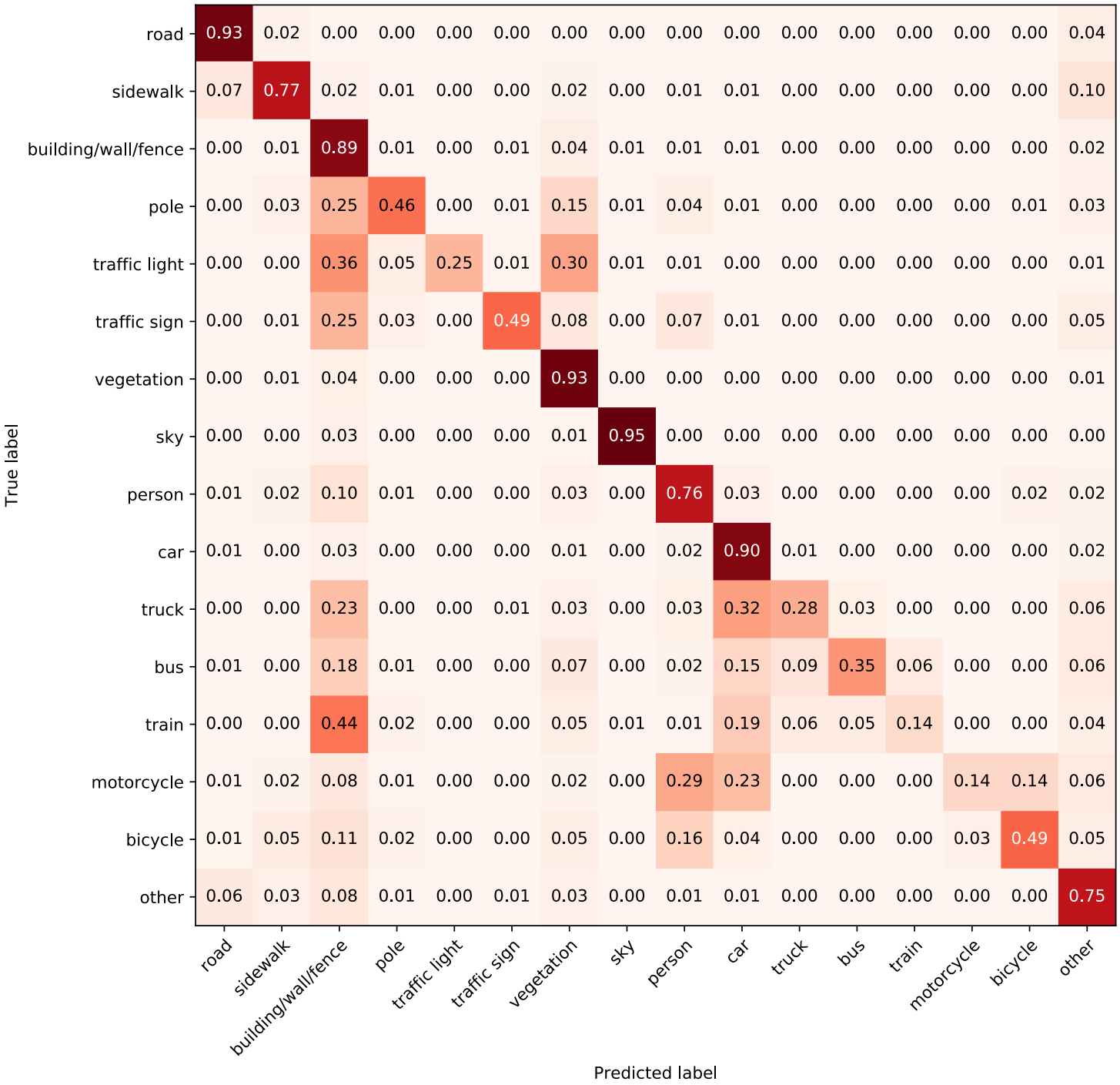

Figure 5.6: Confusion matrix of HRNet trained on Cityscapes fine, evaluated on Cityscapes fine.

5.3 Comparison of results using different training datasets

The *modified DeepLabv3+* is the best performing network, so this architecture was chosen to compare the results of training on the three different datasets: Cityscapes fine, coarse, and BDD. Figure 5.7 shows the learning curves of the trainings. The aim of this comparison is to investigate the difference between the classification models learned by *modified DeepLabv3+*. The evaluation is perfomed using the Cityscapes fine dataset, therefore it is expected that the model trained on the same dataset has the best performance. It is also expected that the model trained on Cityscapes coarse performs worse when evaluated on Cityscapes fine as it was trained on less refined training data. Lastly, the model trained on BDD can be considerably different than those that were trained on Cityscapes due to the differences between properties of the data sets.

Results indicate that the *modified DeepLabv3+* model trained on Cityscapes fine has better accuracy than the other two models which confirms expectations. After approximately 25.000 validation steps the model trained on Cityscapes fine surpasses an accuracy rare of 0.85, whereas the other two methods barely reach this even after 70.000 steps. In terms of accuracy, the model trained on BDD has a less steep learning curve than the model trained on Cityscapes coarse, i.e. the BDD model learns slower. In terms of loss, however, the model trained on BDD produced irregularities. As it is observable in Figure 5.7 there are rapid changes in the measured loss during the training on BDD which might be attributed to labeling incompatibilities between the BDD and Cityscapes data sets. Note that this noise was still present after several repeated measurements. This oscillation is also observable in the predictions as Figure 5.8 shows. Here, different colors represent different labels, for example pixels belonging to an object detected as a car is colored neon green, whereas pixels belonging to persons (i.e. pedestrians) are colored lime (although it is hard to see the difference on the small scaled version of the figures). The oscillation in that example can be observed (1) on the predicted mask for the pedestrian standing on the right hand side, (2) on the road, (3) on the traffic lamp and its pole, and (4) on the traffic sign on the top of the image.

The oscillations during the other two trainings are also visualized on figures 5.9 and 5.10. The oscillation during the training on Cityscapes fine (see figure5.9) can be observed on (1) the left part of the image: the building (colored red), and the pedestrians between the building and the car, and (2) on the right part of the image: the traffic sign above the pedestrian crossing. A slight oscillation is also visible during the training on the Cityscapes coarse dataset (see Figure 5.10): vehicles are well recognized in the horizontal centerline in one case (on the top) and not at all in the other two.

The result in Figure 5.11 shows the iou values of the *modified DeepLabv3+* trained on the three datasets evaluated on the Cityscapes fine dataset. Here, also training on Cityscapes fine shows the best results. Surprisingly, training on BDD gives better results than Cityscapes coarse in more than half of the cases, for example in case of "sky", "vegetation", "car", "person", "building", and "sidewalk" objects.

The confusion matrices of the model learned on Cityscapes coarse and BDD are shown in Figures 5.12 and 5.13. The main difference between the two models is that the Cityscapes coarse model either classifies an object correctly or it is classified as "other", whereas in case of BDD model, this latter option is rare and rather an object is misclassified as a different object. For example, "train" objects are either classified correctly (0.26), or as a "building" (0.11) or "other" (0.58) by the Cityscapes coarse model, while in case of the BDD model "train" objects are classified as "building" (0.56), "truck" (0.21), "bus" (0.09) or "other" (0.05). Interestingly, none of the "train", "motorcycle", "bicycle" objects

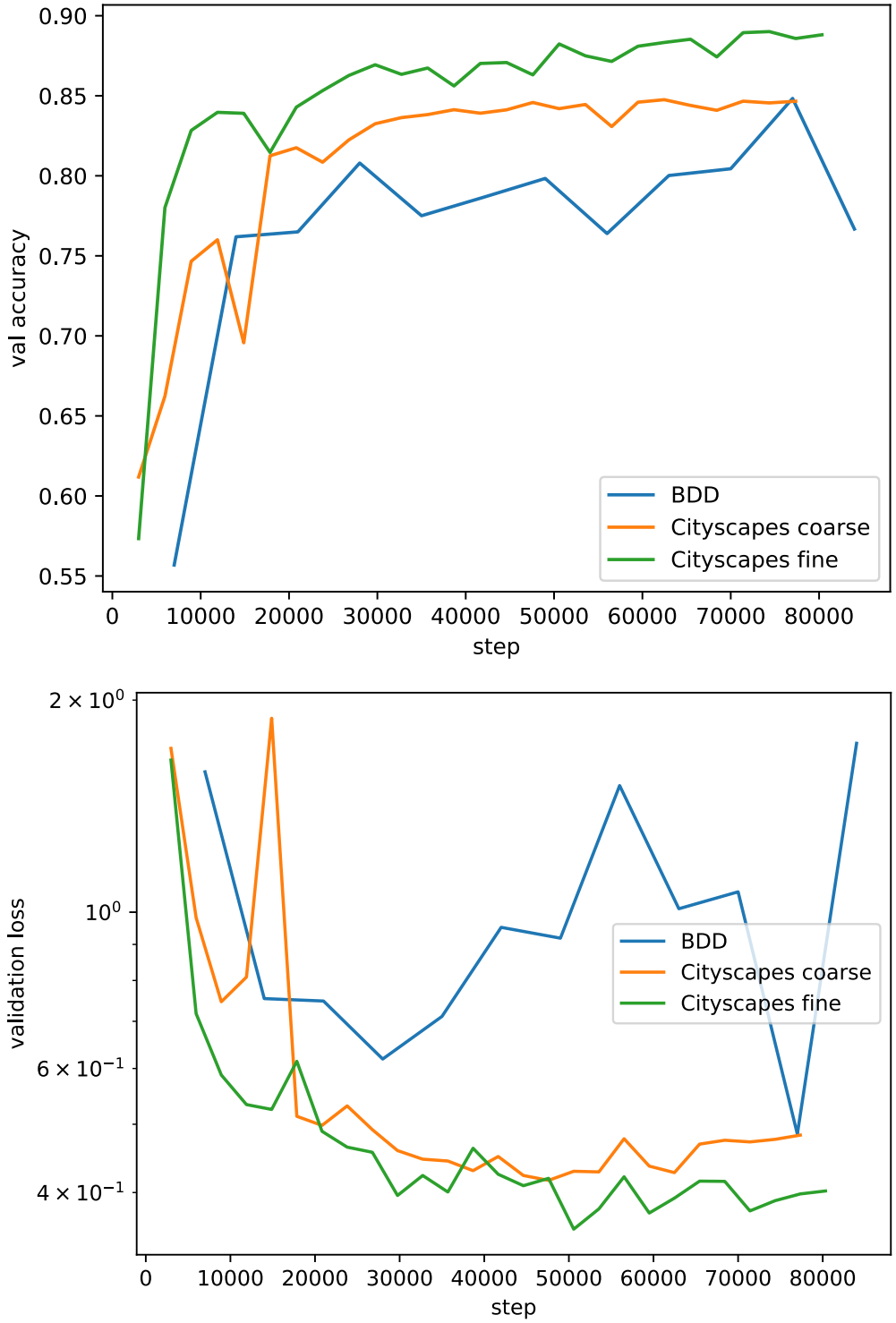

Figure 5.7: Accuracy and loss, computed on the corresponding validation datasets during the training of the modified DeepLabv3+ on the various datasets.

are classified correctly by the BDD model, which can be due to the lower frequency of occurrence of these objects in the BDD dataset with respect to the Cityscapes dataset. In addition, "traffic light" and "traffic sign" objects are also misclassified by the BDD model,

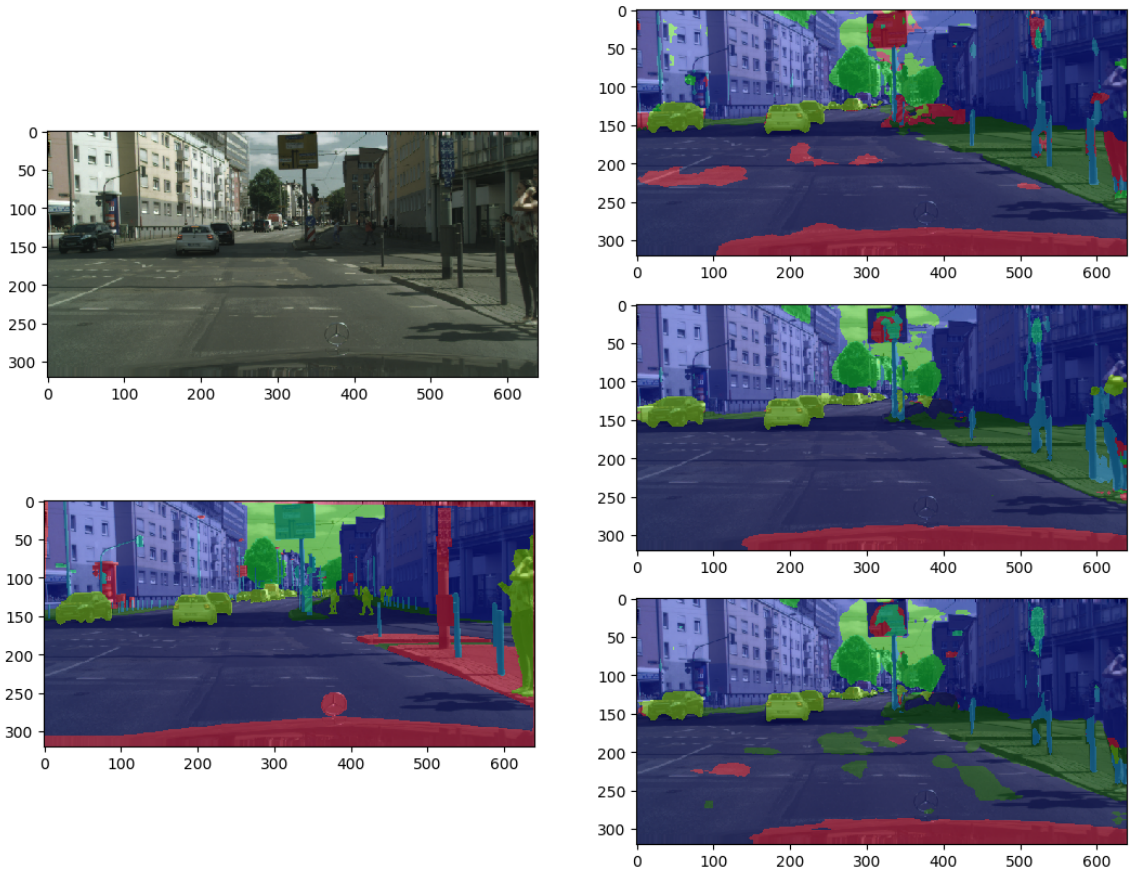

Figure 5.8: Input image (top left), expected segmentation mask (bottom left) and the predictions during the (27th, 28th and 29th) epochs of the training of the modified DeepLabv3+ on the BDD dataset.

more specifically "traffic light" is identified as "building" (0.45) or "vegetation" (0.40), and "traffic sign" objects are similarly misclassified as "building" (0.57) or "vegetation" (0.15).

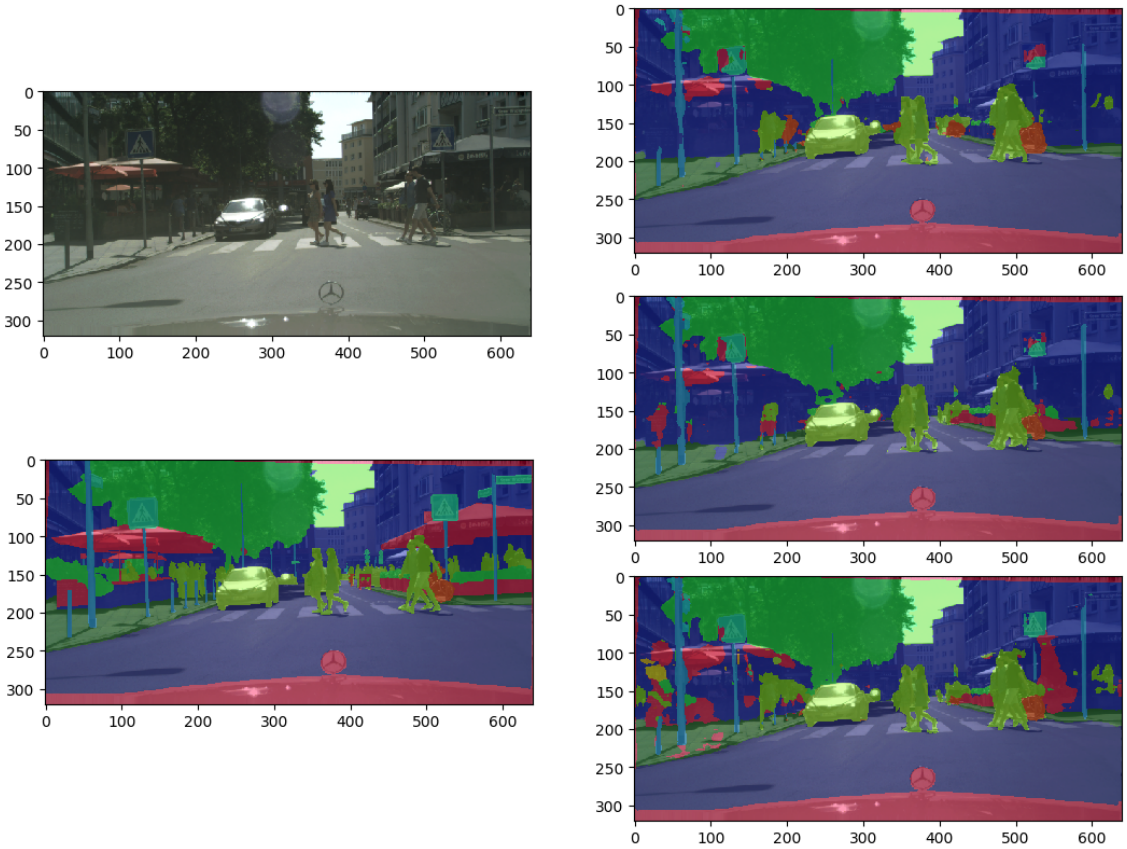

Figure 5.9: Input image (top left), expected segmentation mask (bottom left) and the oscillating predictions during the (24th, 25th and 26th) epochs of the training of the modified DeepLabv3+ on Cityscapes fine.

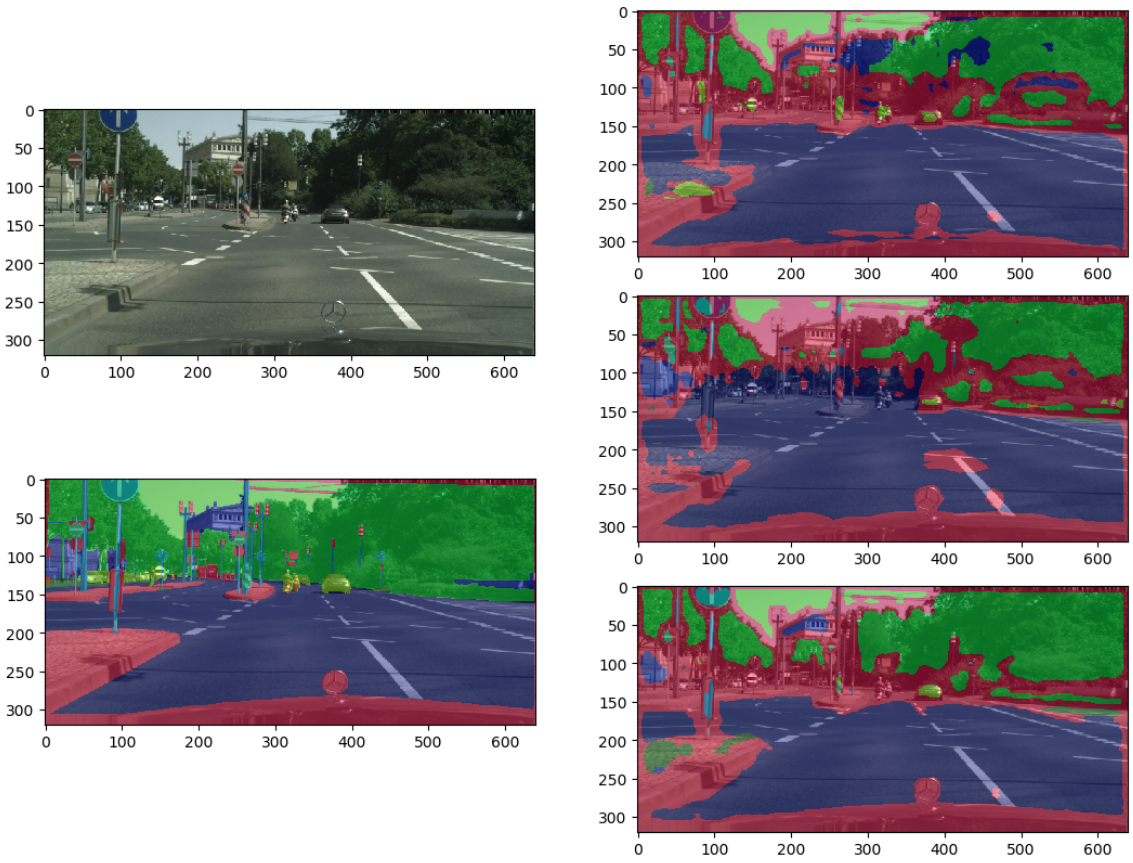

Figure 5.10: Input image (top left), expected segmentation mask (bottom left) and the predictions during the (3rd, 4th, and 5th) epochs of the training of the modified DeepLabv3+ on the Cityscapes coarse dataset.

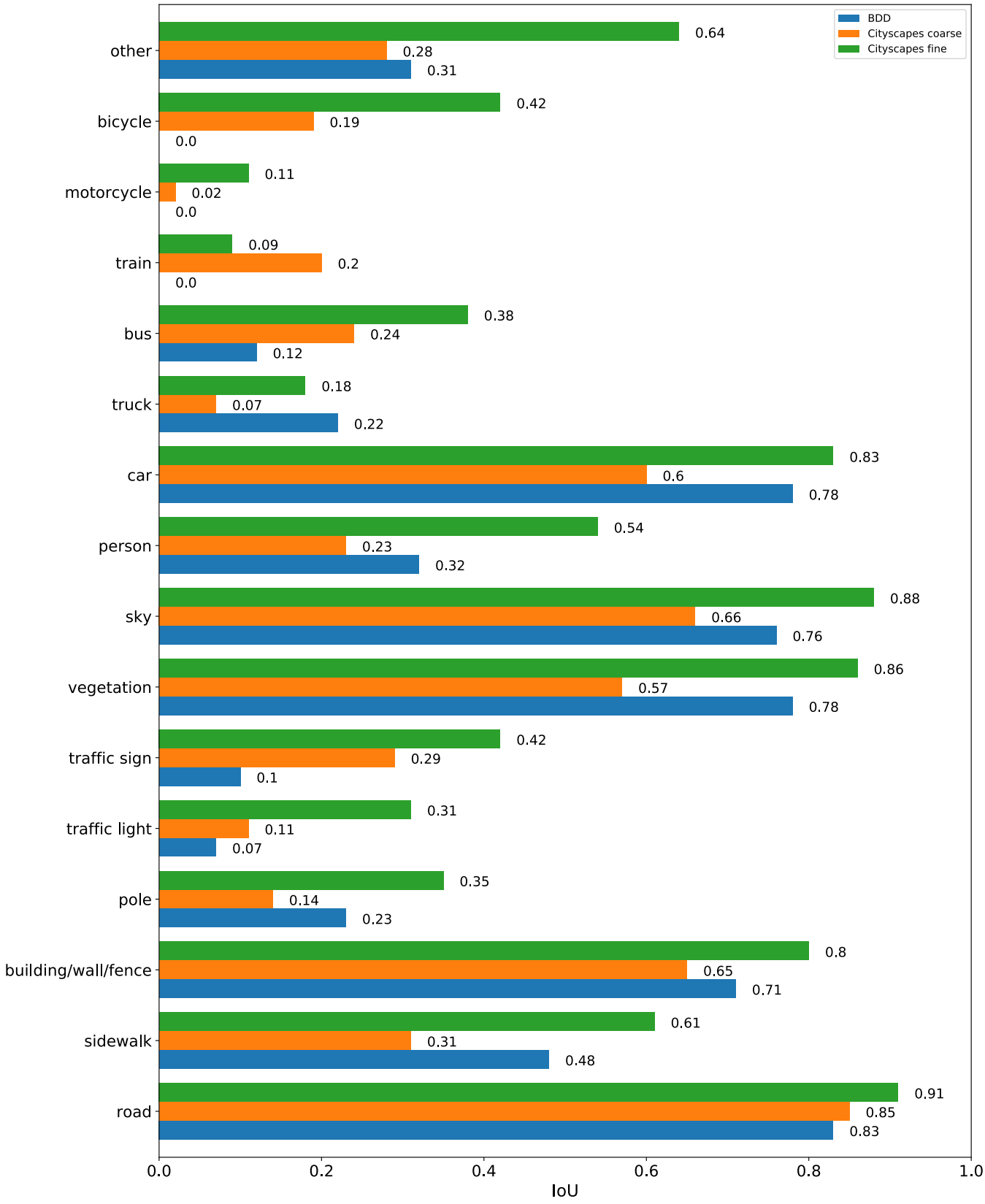

Figure 5.11: Intersection over union values, computed by evaluating the modified DeepLabv3+ trained on Cityscapes fine and BDD, evaluated on Cityscapes fine.

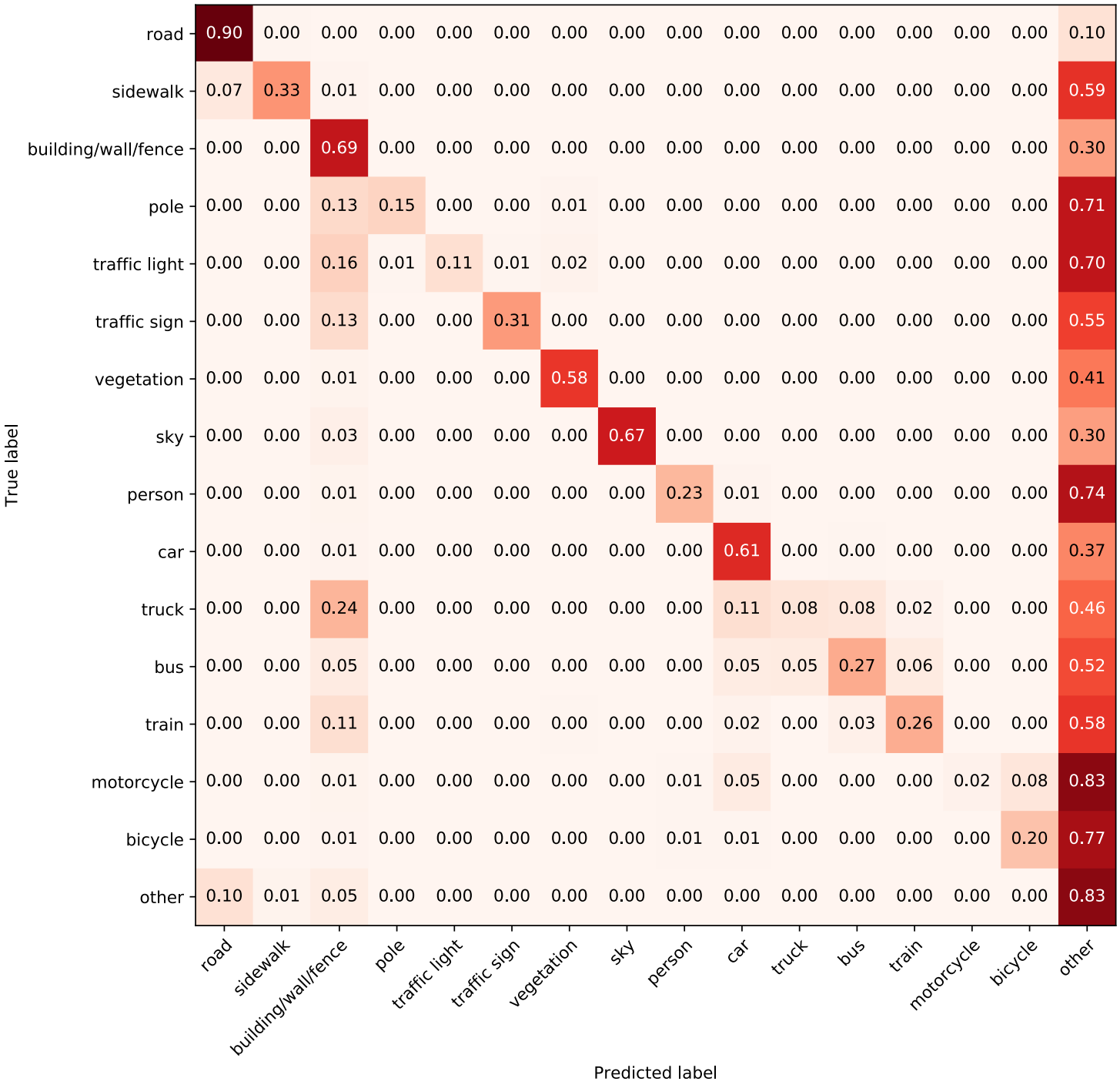

Figure 5.12: Confusion matrix of the modified DeepLabv3+ trained on Cityscapes coarse, evaluated on Cityscapes fine.

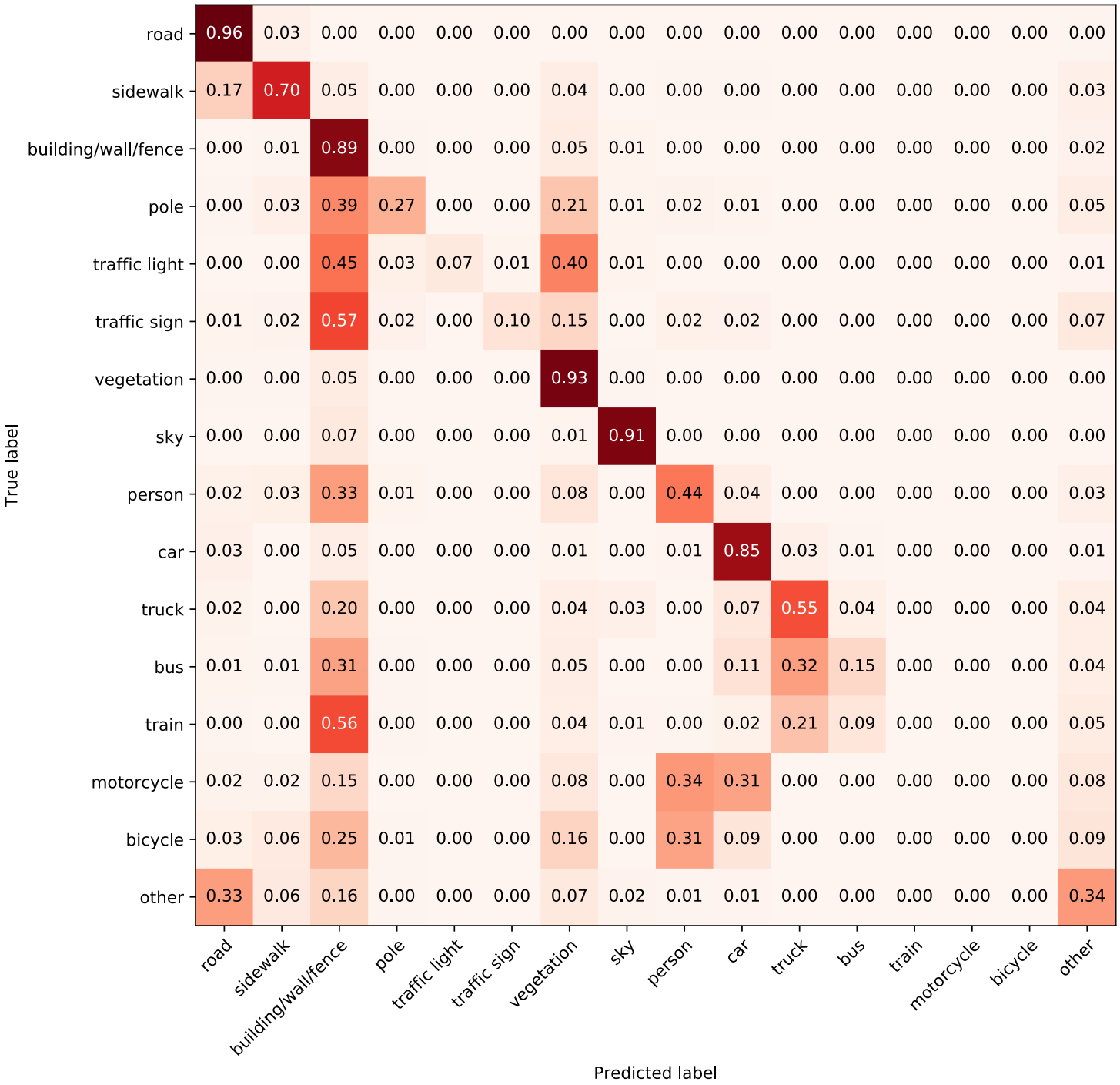

Figure 5.13: Confusion matrix of the modified DeepLabv3+ trained on BDD, evaluated on Cityscapes fine.

5.4 Comparison of results using BDD as evaluation dataset

In the previous sections the Cityscapes fine dataset was used for evaluation. In this comparison the BDD dataset is utilized as the evaluation dataset. The aim of this analysis is to investigate which of the two datasets can be considered more general Figure 5.14 shows the iou values of evaluating the *modified DeepLabv3+* trained on BDD and Cityscapes fine, evaluated on BDD.

It is expected that the model trained and evaluated on the BDD dataset performs better than the model that is trained on Cityscapes fine dataset. Results confirm these expectations, the BDD model outperforms the Cityscapes fine model in most categories. However both models perform poorly in classifying "bicycle", motorcycle", and "train" objects. In addition the recognition of "traffic sign" and "traffic light" objects is moderately successful even by the BDD model. Interestingly, "person" objects also have a low iou value in case of BDD. It was also a little bit surprising that the iou values of "road", "sidewalk", "building", "bus" are significantly worse in case of training on Cityscapes fine and evaluating on BDD than in case of training on BDD and evaluating on Cityscapes fine. It points in the direction that the BDD dataset is more general.

The confusion matrices of the model trained on Cityscapes fine and BDD are shown in Figures 5.12 and 5.13. These matrices indicate that certain objects, such as "car" (0.68, 0.92), "vegetation" (0.6, 0.92) and "sky" (0.83, 0.95 are recognized well by both networks. Interestingly, while these values are higher in case of learning on BDD, the values belonging to "person" (0.24, 0.69) and "traffic sign" (0.24, 0.45) are higher in case of training on Cityscapes fine. Classes "train", "motorcycle" and "bicycle" were not recognized by the network learned on BDD but they were misclassified as "car" (0.74, 0.68, 0.39) and a little as "person" (0.0, 0.16, 0.26). Both models often misclassified "pole" (0.32, 0.2), "traffic light" (0.33, 0.28) and "traffic sign" (0.24, 0.45) as "building" (0.32, 0.39, 0.42 and 0.24, 0.17, 0.19).

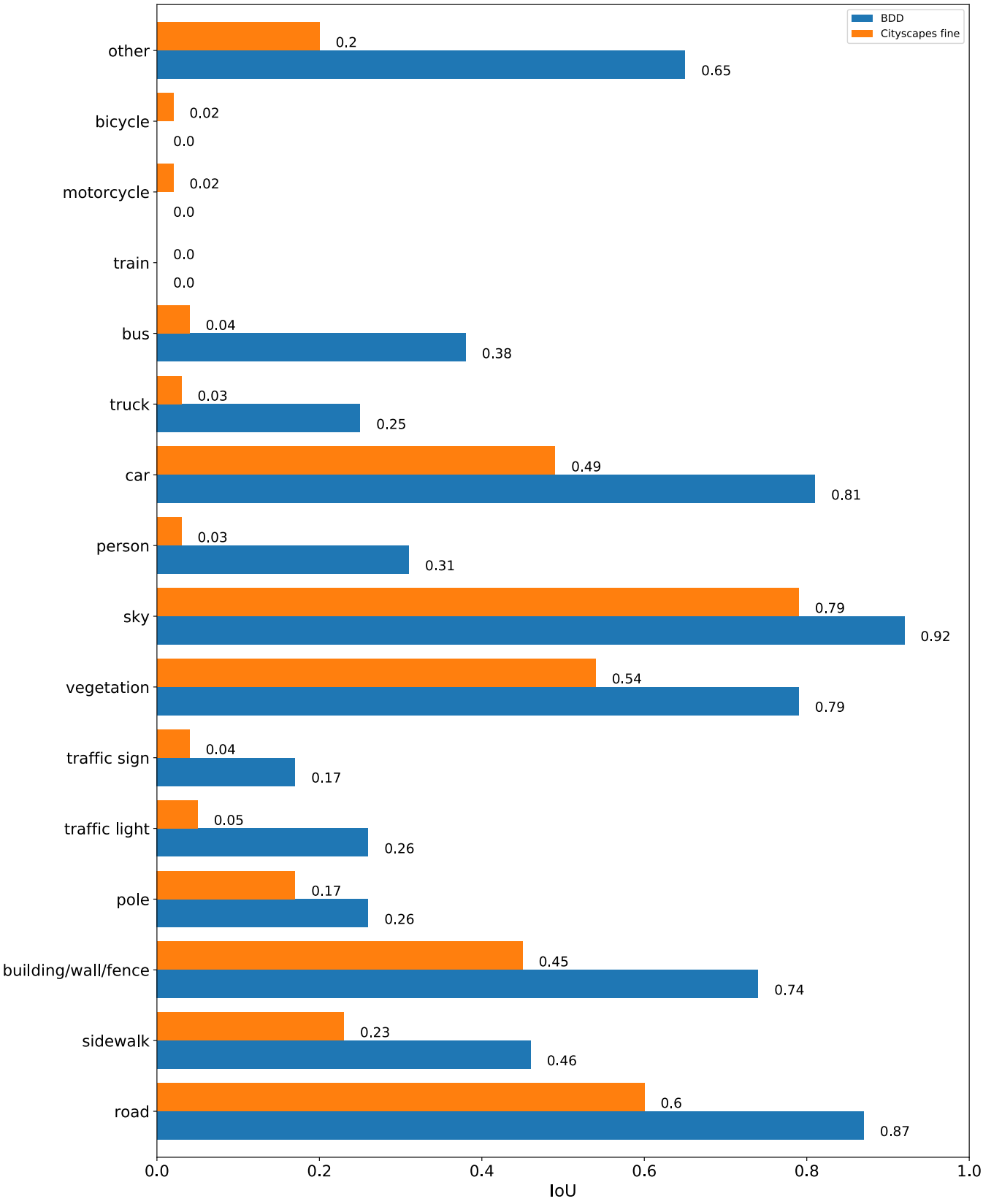

Figure 5.14: Intersection over union values, computed by evaluating the modified DeepLabv3+ trained on Cityscapes fine and BDD, evaluated on BDD.

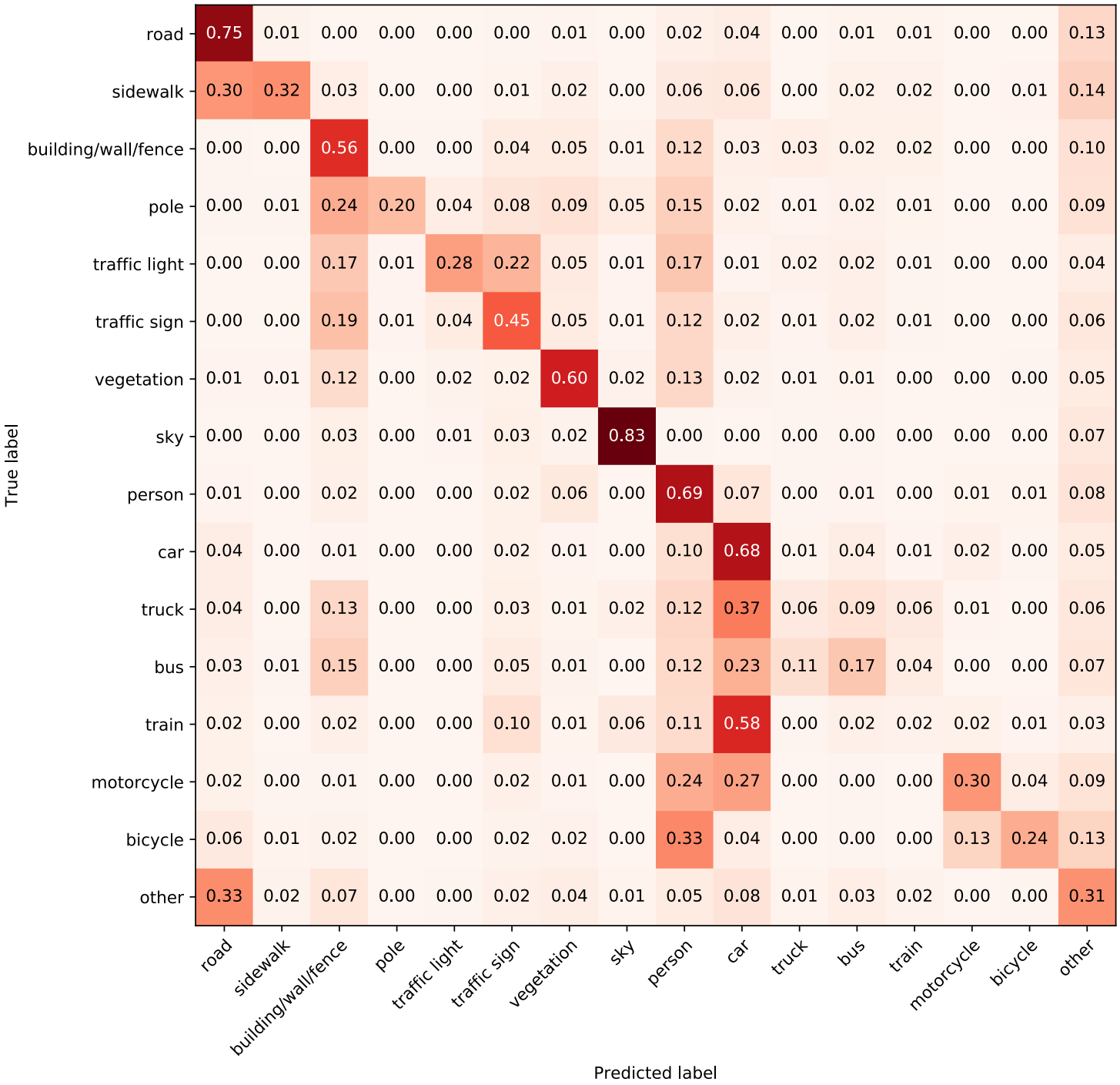

Figure 5.15: Confusion matrix of the modified DeepLabv3+ trained on Cityscapes fine, evaluated on BDD.

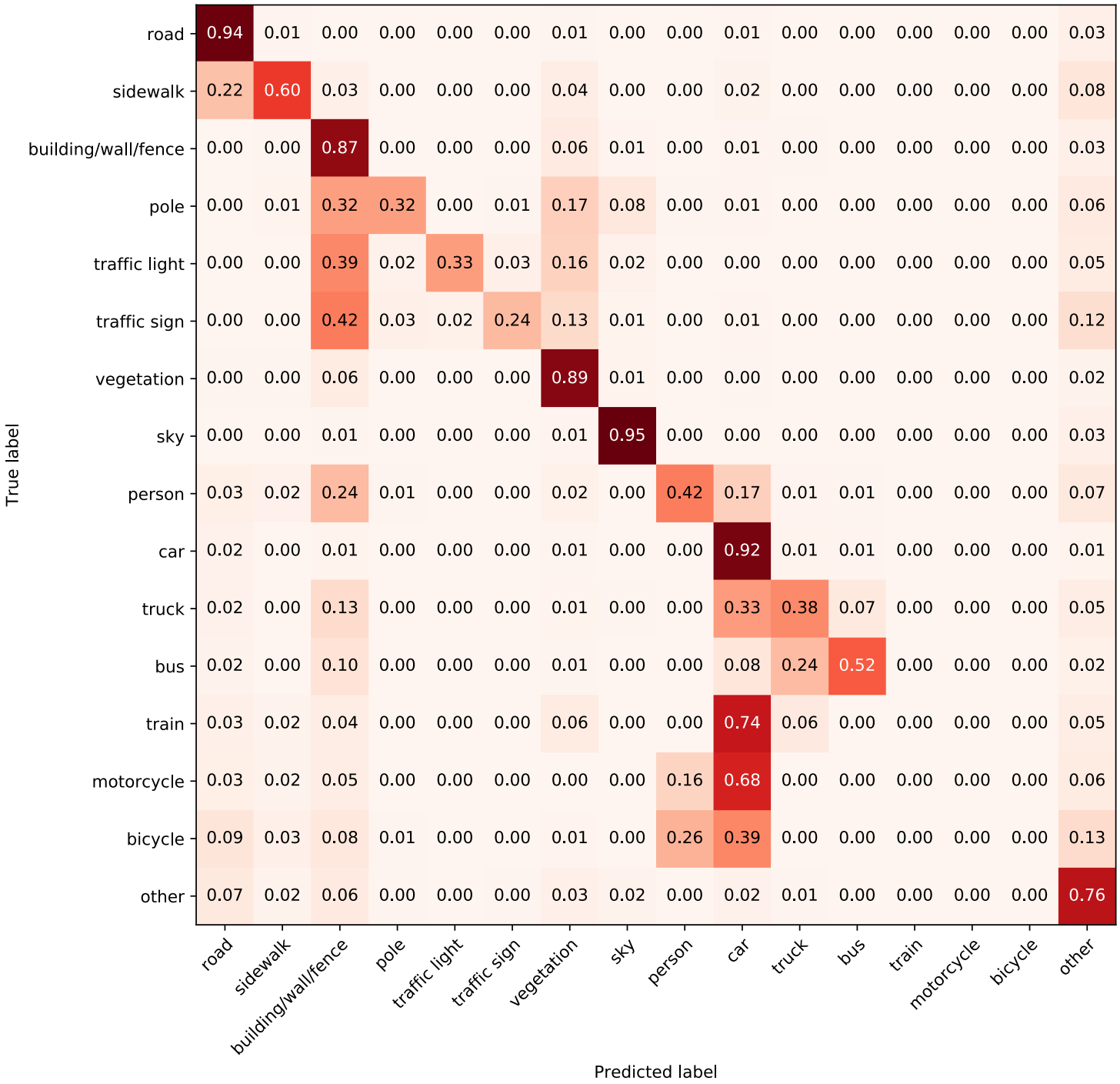

Figure 5.16: Confusion matrix of the modified DeepLabv3+ trained on BDD, evaluated on BDD.

Chapter 6

Conclusion

The purpose of my work was to examine state-of-the-art neural networks - that can be applied to semantic segmentation - on different datasets. For this reason, I first reviewed previous neural network based solutions. Then, two of them were selected that are publicly available and fairly accurate according to the Cityscapes benchmark: the *DeepLabv3+* and the *HRNet* networks. These two architectures were implemented using the Keras framework. Two public datasets were chosen for the training and evaluation of the networks: the Cityscapes, and the Berkeley DeepDrive databases, with the former having two variants the so-called coarse and fine. Using two datasets made it possible to investigate the effect of learning from different sources and also to analyze the differences between the databases.

My contribution was the modification of the *DeepLabv3+* network which involved the elimination of the global pooling layer which made it possible to utilize input images of arbitrary resolution at training and inference time. This modification produced better results in some cases.

After training, the networks were evaluated from various aspects, which included the comparison of the networks by using the same dataset and the comparison of the datasets using the same network. Based on the detailed results some general conclusions and suggestions for future work were formulated.

Some misclassifications do not seem to be as serious semantically as others. For example, classifying a riding "person" as "bicycle" is not as big a mistake as labeling it as a "wall". The weighting of such cases in the loss function may help the network to generalize better.

The various evaluations show that the accuracy of the models is highly dependent on the dataset. If such neural networks are to be used in the industry, then a uniformly accepted standard dataset must be developed to test their accuracy. Furthermore, the frequency of various object classes should also be balanced in such datasets, because insufficient occurrence of certain objects causes difficulties during training.

The training was performed using several sets of parameters, while the architecture remained as described in the original implementation. However, the various parameter sets did not produce considerably different results than the original settings, therefore the latter was used.

Evaluation on videos showed that the image segmentation system is very sensitive to a small perturbation of the input image. Using a recurrent neural network this oscillation on the output could be smoothened. Human vision can perform so well because it processes

not only a single image but a series of images. Using recurrency could also help to improve the accuracy of the network.

Further possible future improvements include the pretraining of the network, for example the classificator part of *HRNet*, which could improve the accuracy. In addition, it could be investigated whether pruning either of the networks would have beneficial effects on complexity, thus reducing training time and inference time.

Acknowledgements

I would like to thank the support of Dr. Gabor Hullam whose research has been supported by the ÚNKP-19-4 New National Excellence Program of the Ministry for Innovation and Technology (ÚNKP-19-4-BME-344).

This research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013).

Bibliography

[1] Márta Altrichter, Gábor Horváth, Béla Pataki, György Strausz, Gábor Takács, and József Valyon. *Az MLP tanítása, a hibavisszaterjesztéses algoritmus*. Panem Könyvkiadó Kft., 2006.

[2] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[3] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[4] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[5] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940, 2016.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[8] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[10] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[11] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[12] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[15] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

[16] Ke Sun, Yang Zhao, Borui Jiang, Tianheng Cheng, Bin Xiao, Dong Liu, Yadong Mu, Xinggang Wang, Wenyu Liu, and Jingdong Wang. High-resolution representations for labeling pixels and regions. *arXiv preprint arXiv:1904.04514*, 2019.

[17] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[18] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2018.

Appendix

A.1 Confusion matrices

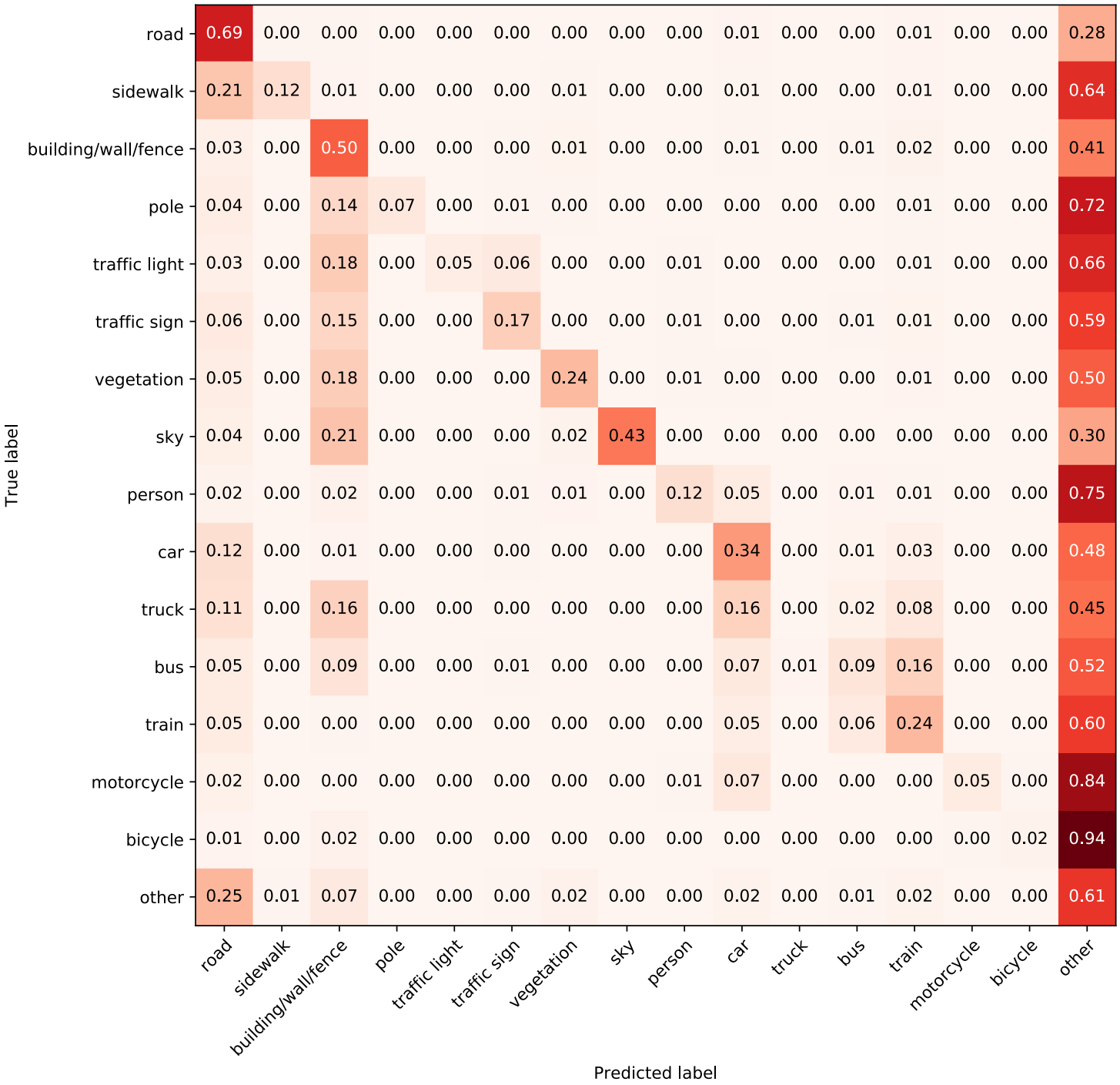

Figure A.1.1: Confusion matrix of the modified DeepLabv3+ trained on Cityscapes coarse, evaluated on BDD.

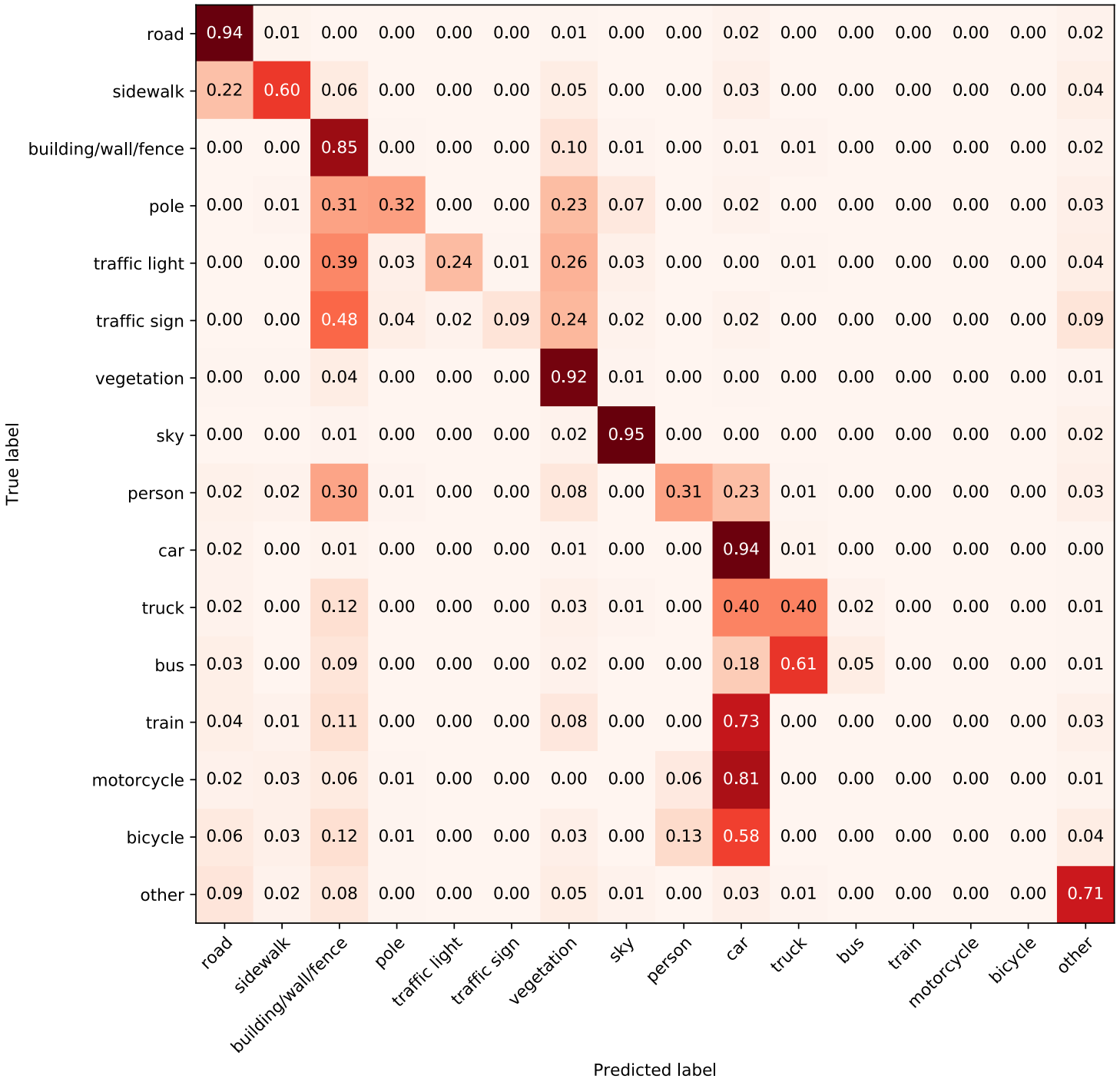

Figure A.1.2: Confusion matrix of DeepLabv3+ trained on BDD, evaluated on BDD.

Figure A.1.3: Confusion matrix of DeepLabv3+ trained on BDD, evaluated on Cityscapes fine.

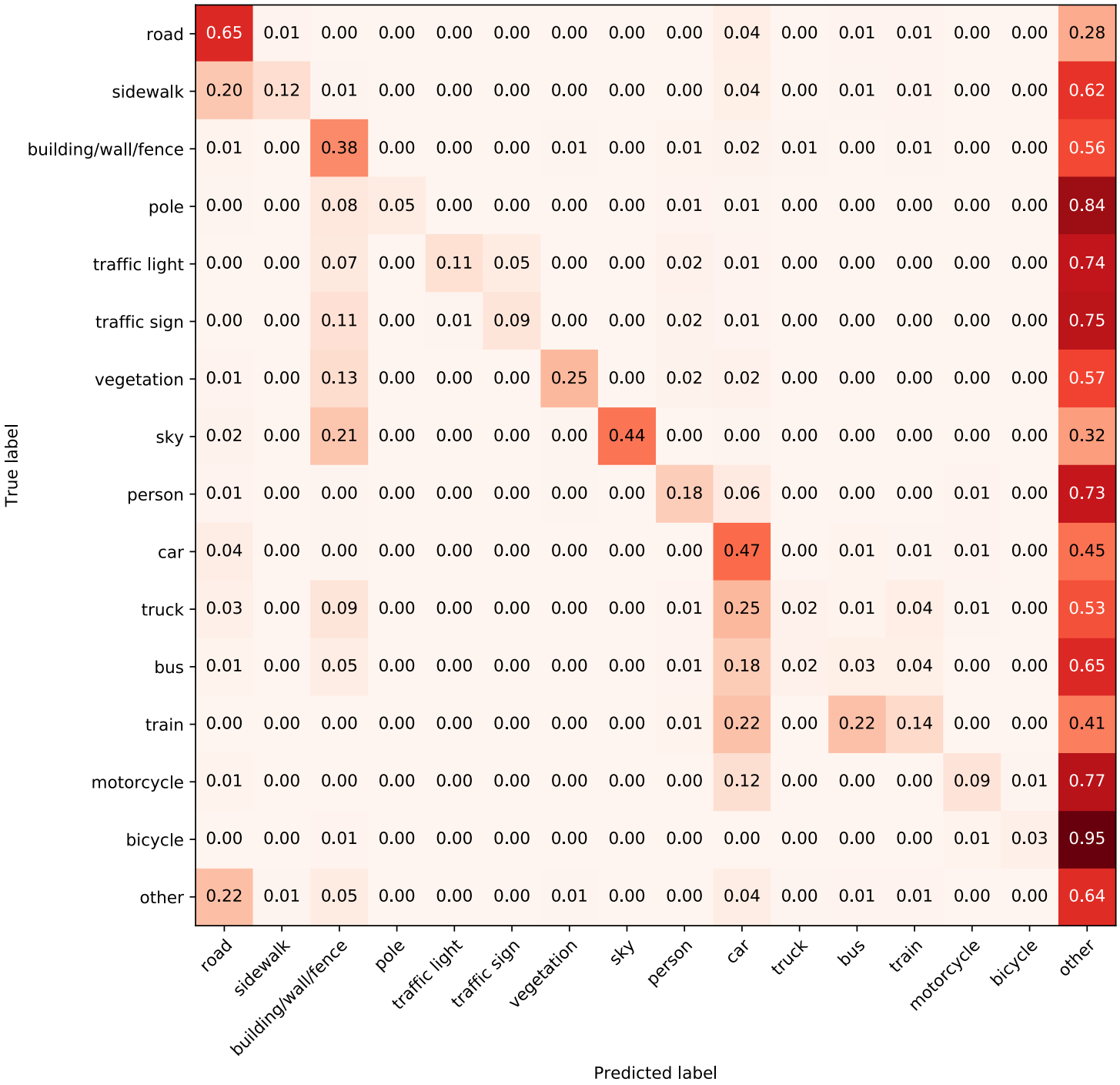

Figure A.1.4: Confusion matrix of DeepLabv3+ trained on Cityscapes coarse, evaluated on BDD.

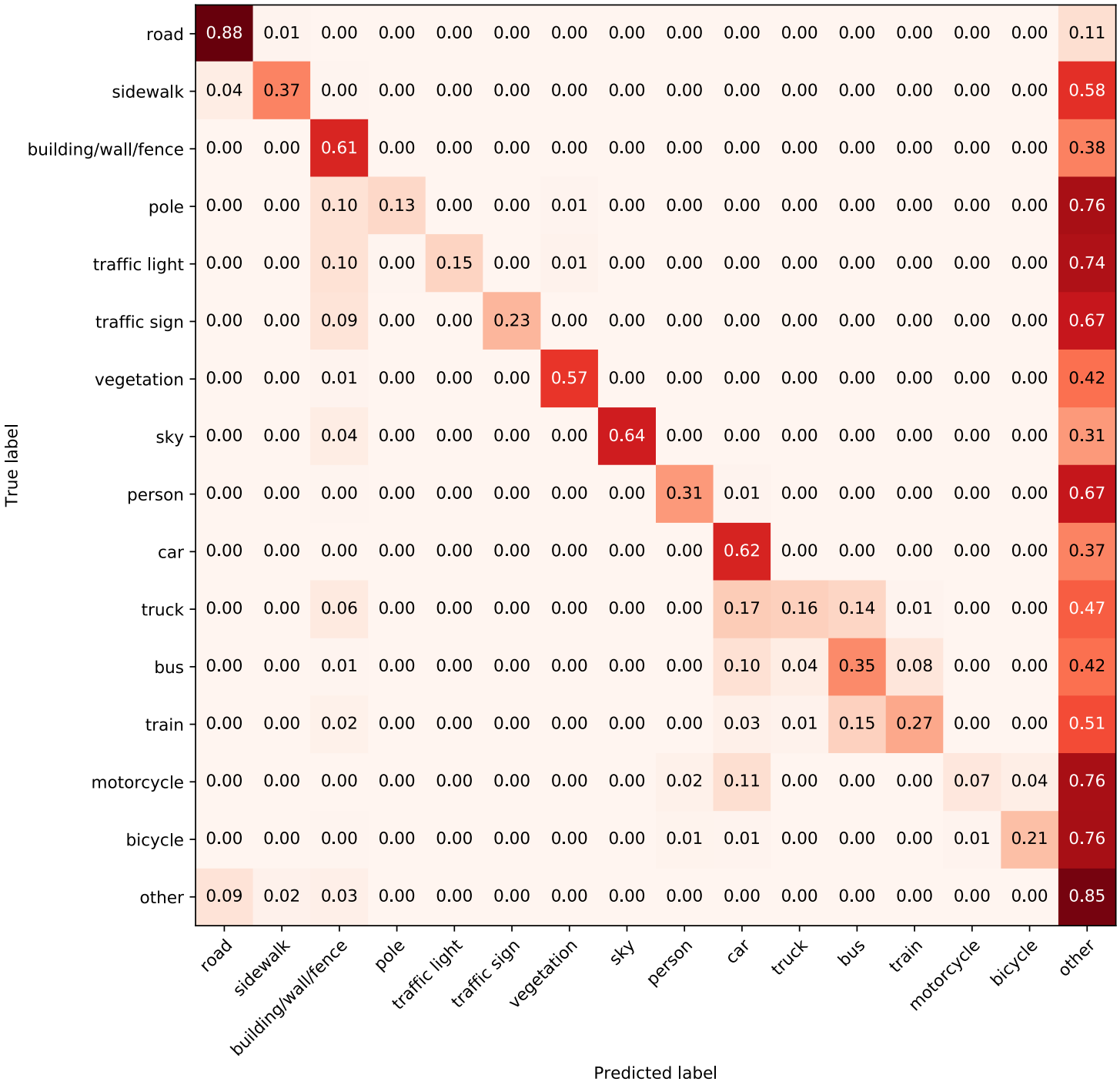

Figure A.1.5: Confusion matrix of DeepLabv3+ trained on Cityscapes coarse, evaluated on Cityscapes fine.

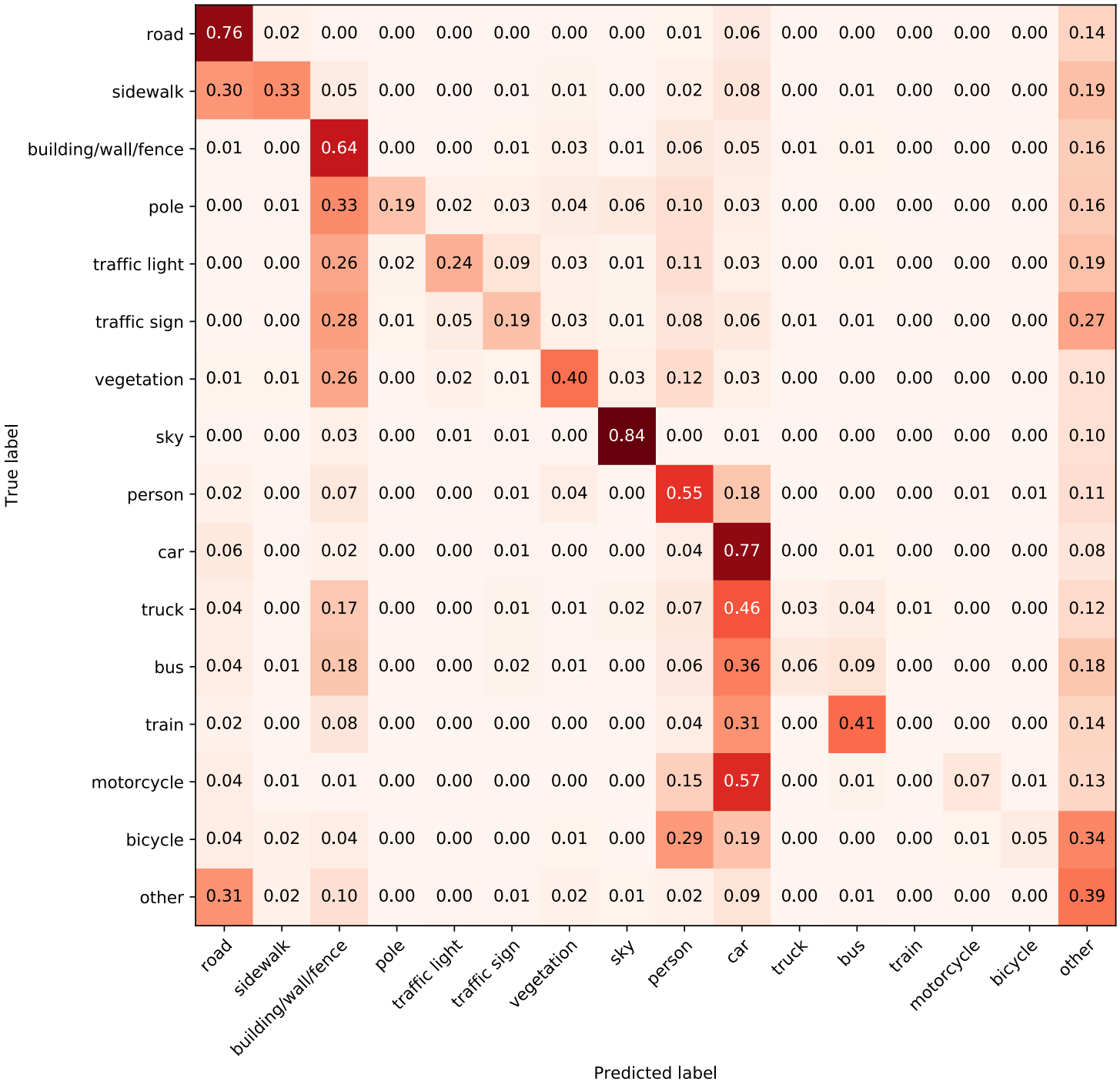

Figure A.1.6: Confusion matrix of DeepLabv3+ trained on Cityscapes fine, evaluated on BDD.

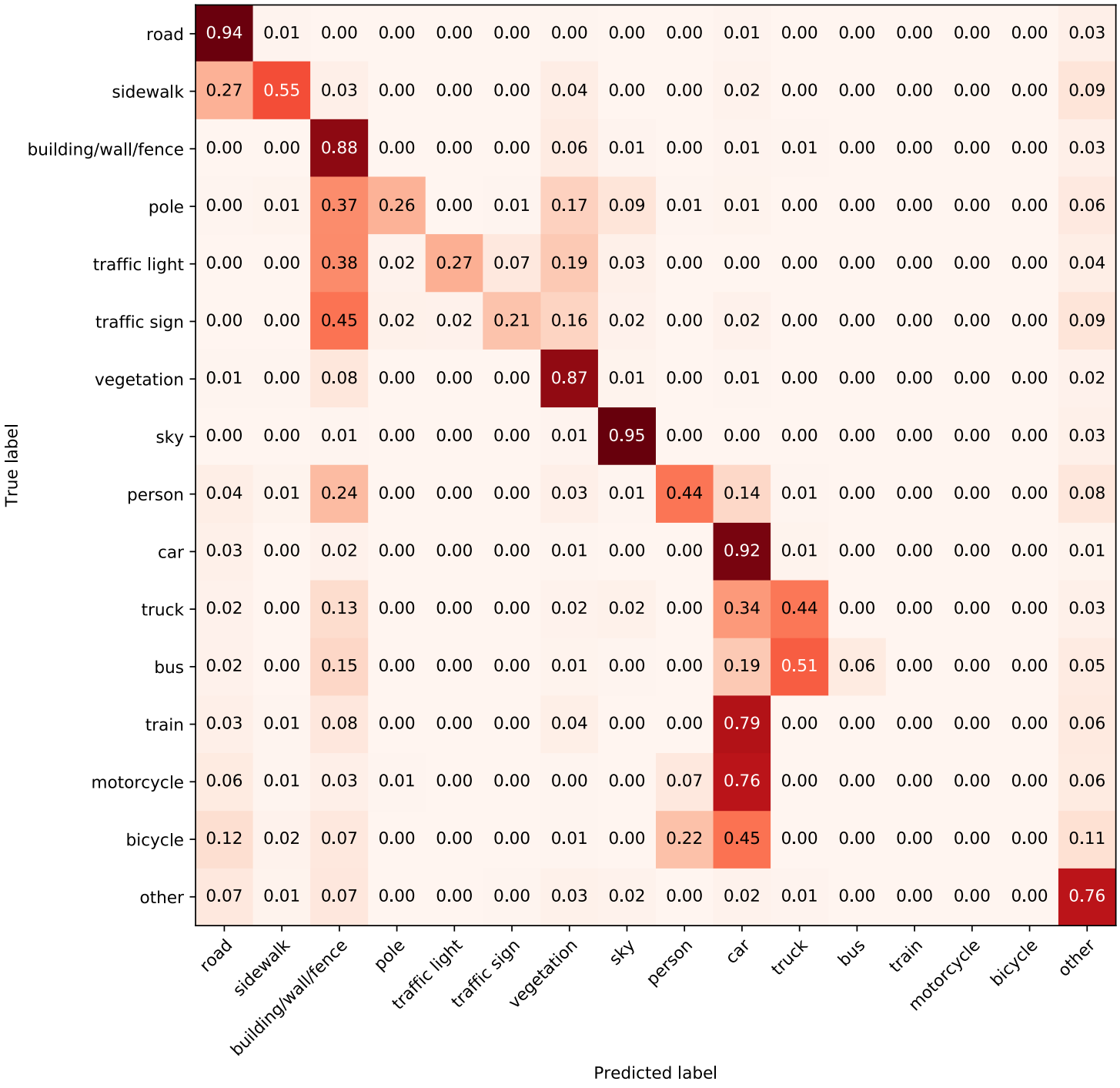

Figure A.1.7: Confusion matrix of HRNet trained on BDD, evaluated on BDD.

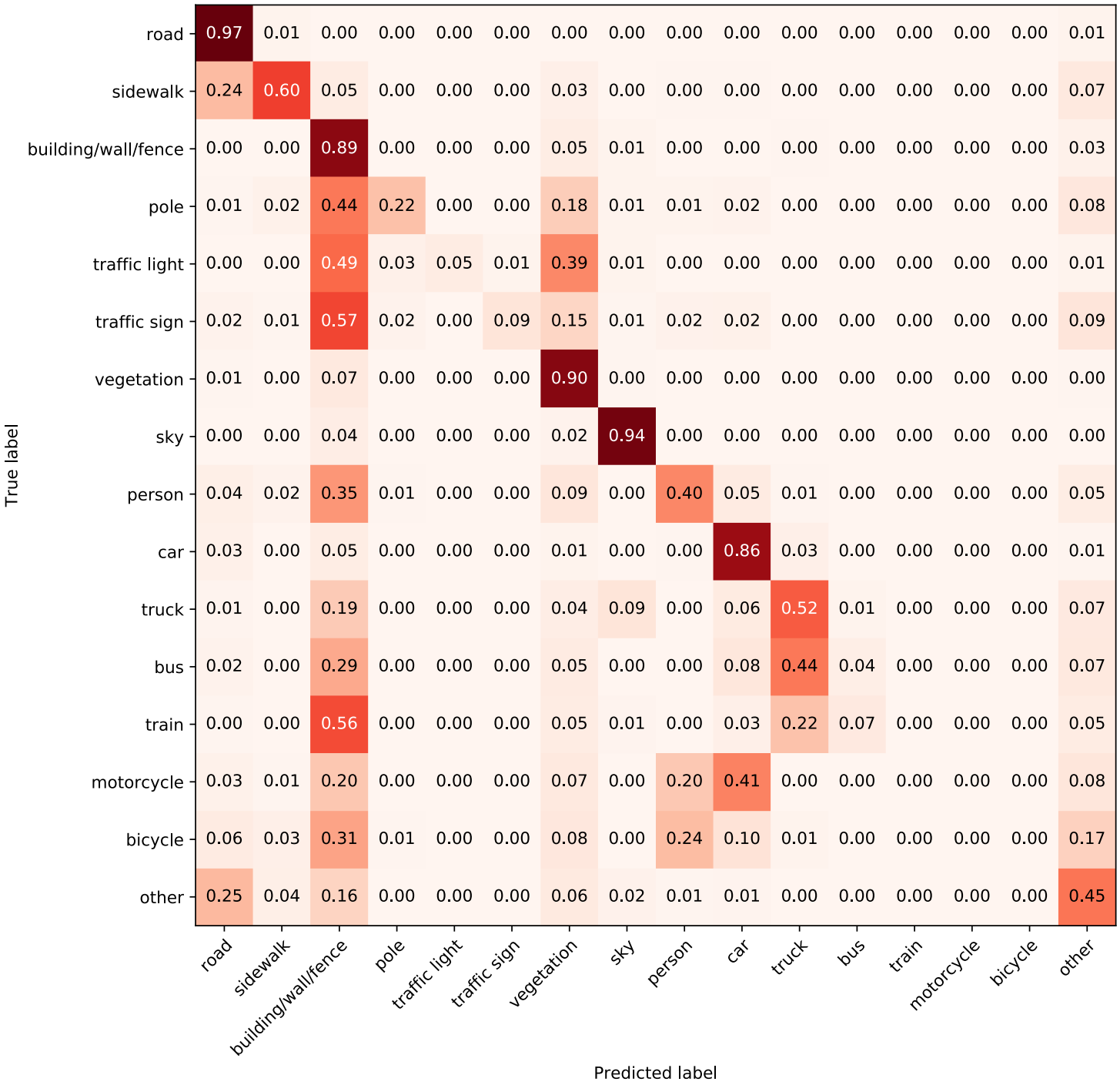

Figure A.1.8: Confusion matrix of HRNet trained on BDD, evaluated on Cityscapes fine.

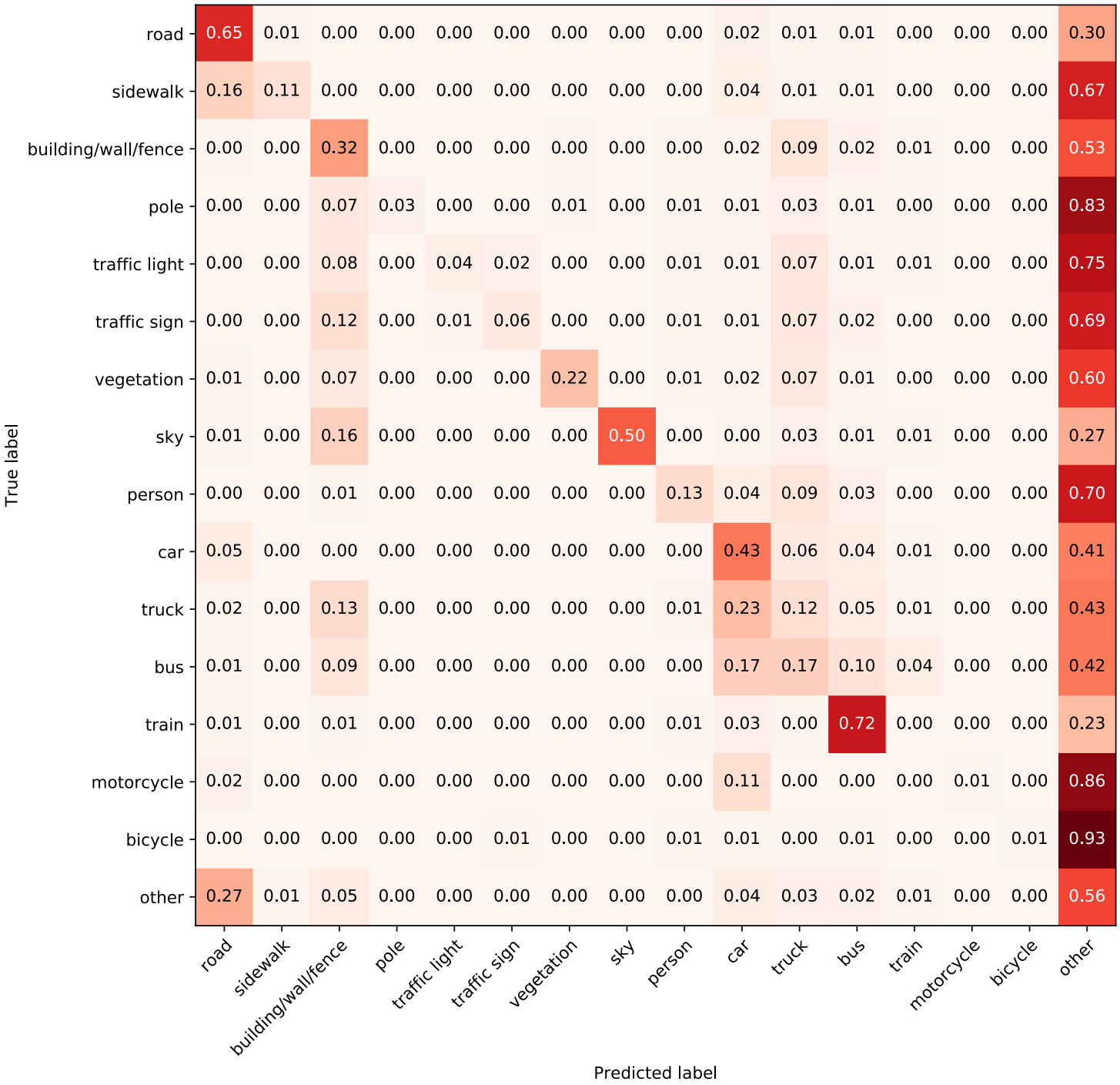

Figure A.1.9: Confusion matrix of HRNet trained on Cityscapes coarse, evaluated on BDD.

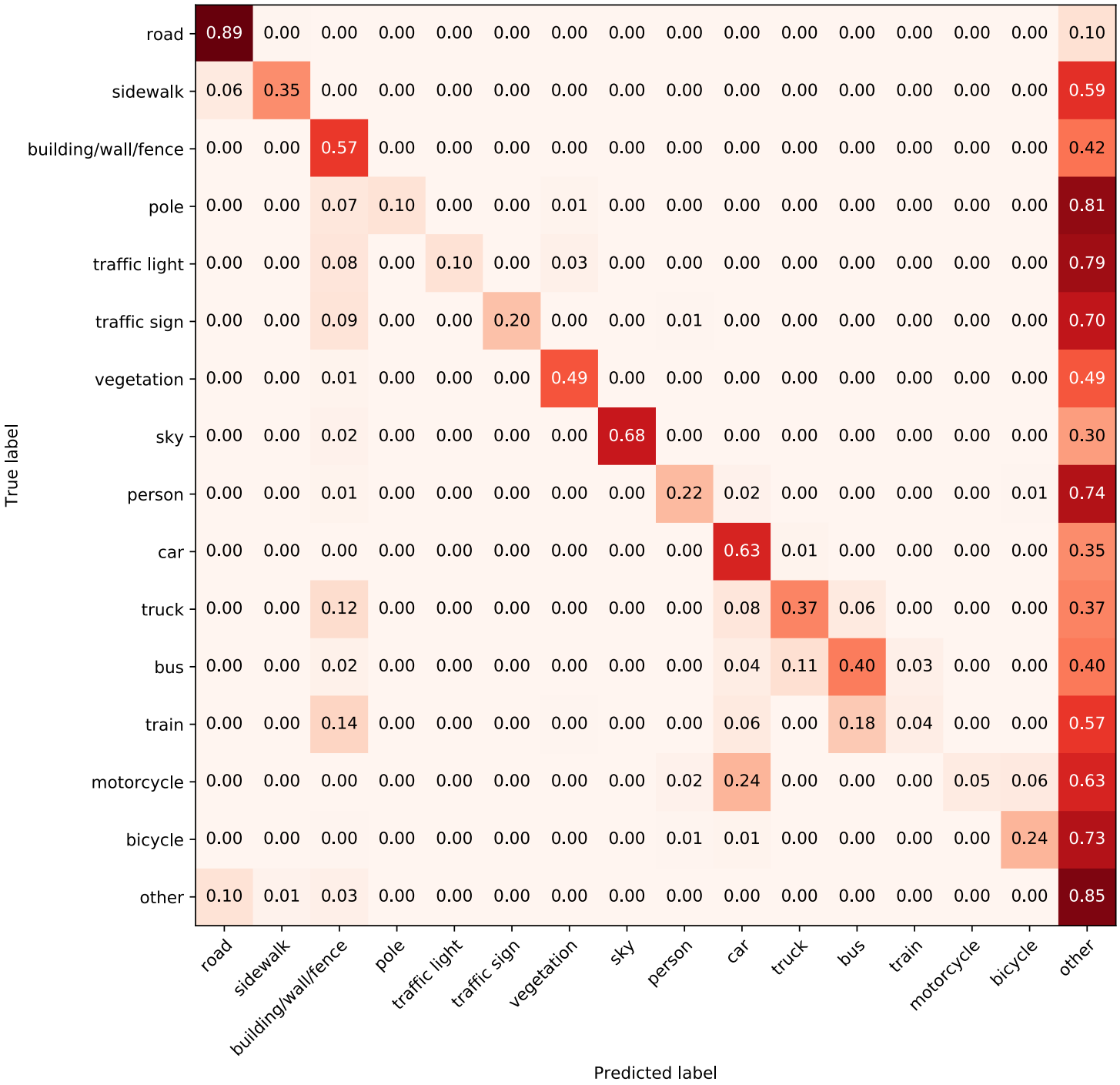

Figure A.1.10: Confusion matrix of HRNet trained on Cityscapes coarse, evaluated on Cityscapes fine.

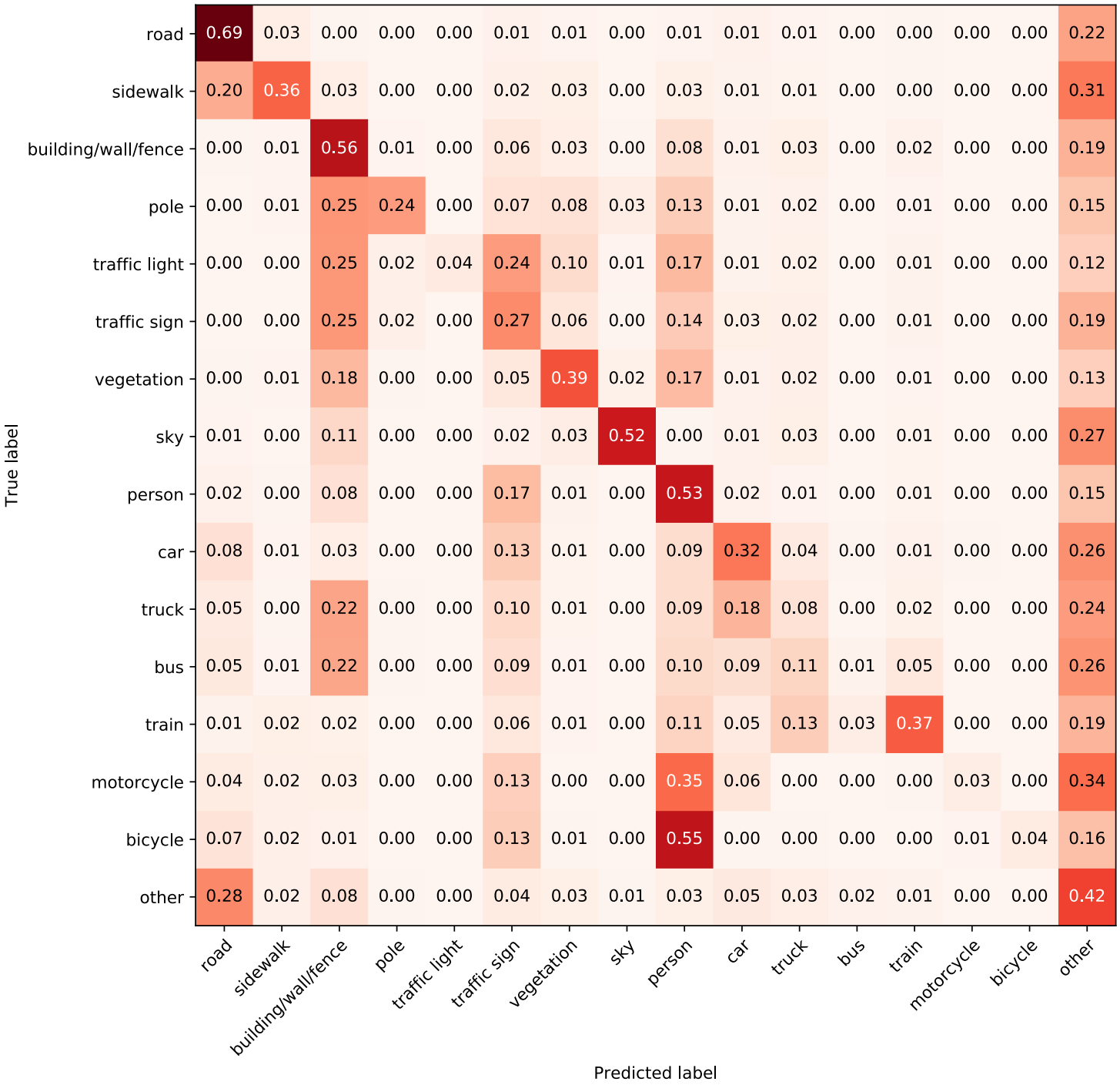

Figure A.1.11: Confusion matrix of HRNet trained on Cityscapes fine, evaluated on BDD.