Supplementary: Multi-Scale Grouped Prototypes for Interpretable Semantic Segmentation

6. Details on experimental set-up

In this section, we provide the details of the experimental set-up briefly described in Section 4.1 to support reproducing the results.

Firstly, across all training stages, the prototypes are the same size as the output feature maps of the non-concatenated ASPP in DeepLabv2 [9]: $D = 64$. For both multi-scale prototype training and the grouping mechanism we leverage augmentation techniques, such as random horizontal flipping, cropping, and scaling images by a factor between $[0.5, 1.5]$ for Cityscapes [14] and Pascal VOC [21], and $[0.5, 2]$ for ADE20K [100]. The batch size used in all the training stages is 10 and we select the Adam optimizer with a weight decay of $5e^{-4}$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ for our experiments. Moreover, as the batch size in our experiments is limited, we freeze the batch normalization parameters of the ResNet-101 backbone due to its impact on performance.

For the multi-scale prototype training stage, we set the weights of the loss terms to $\lambda_{L1} = 1e^{-4}$ and $\lambda_J = 0.25$ following ProtoSeg on both Cityscapes and Pascal VOC, and $\lambda_{L1} = 1e^{-5}$ for ADE20K due to the large number of classes. For the datasets: Cityscapes and Pascal VOC, we run first the warm-up step for 3000 batch iterations with a fixed learning rate of $2.5e^{-4}$. The joint training step is run for 30000 batch iterations with an initial learning rate of $2.5e^{-5}$ for ResNet-101 and $2.5e^{-4}$ for the prototype and ASPP layers. For this step, we leverage a learning rate scheduler following the polynomial learning rate policy with $power = 0.9$. Lastly, the fine-tuning step is done over 2000 batch iterations extended to 6000 for Pascal with a fixed learning rate of $1e^{-5}$. For ADE20K we leverage the same learning rate and learning rate scheduler as for the other two datasets but we double the number of batch iterations compared to Cityscapes on all steps: 6000 batches for warm-up, 60000 batches for joint training, and 4000 batches for fine-tuning.

The training of ProtoSeg on ADE20K follows the same experimental set-up as the multi-scale prototype training stage described above and as mentioned in Section 4.1: 12 prototypes per class were used to match ScaleProtoSeg.

During the training of the grouping mechanism, we set the weights of the loss terms to $\lambda_{L1} = 1e^{-3}$ for Cityscapes and Pascal VOC, and $\lambda_{L1} = 1e^{-4}$ for ADE20K. Moreover, we set $\lambda_{ent} = 0.05$ for Cityscapes and Pascal VOC, and $\lambda_{ent} = 0.25$ for ADE20K. We run the warm-up stage to train the group projections for 2000 batch iterations with a fixed learning rate of $2.5e^{-4}$ for all datasets. Then, we run the fine-tuning stage for 30000 batches with a learning rate of $2.5e^{-4}$ and the same optimizer and scheduling policy as in the prototype joint training phase for all datasets.

We evaluate ScaleProtoSeg on Pascal VOC for which we set the training image resolution to 321×321 and the testing one to 513×513. Moreover, we use multi-scale inputs [9] with scales $\{0.5, 0.75, 1\}$ during training. We also evaluate ScaleProtoSeg on Cityscapes, for this dataset, we do not use MSC input training, we set the training resolution to 513×513 and the testing one to 1024×2048. Lastly, we evaluate ScaleProtoSeg and ProtoSeg on ADE20K with no MSC input, a training resolution of 512×512, and at test time we resize the smallest size of the image to the training resolution and keep the aspect ratio. Lastly, the performance of DeepLabv2 on ADE20K was extracted from [36].

The interpretability evaluation of our method is done on the parts annotated validation sets from Cityscapes and PASCAL-Context [16, 58]. In particular, to align to the classes and images covered during training for Pascal, we tested on the overlap images between PASCAL VOC and PASCAL-Context and so used the semantic annotations from PASCAL VOC and the part annotations from [16, 58] for the 16 classes covered. This represents 925 validation images for Pascal. Moreover, the stability and consistency metrics require part keypoints in their formulation, for this purpose we leverage the standard algorithm from opencv with an 8-connectivity relation between non-zero elements to compute the centroids of every connected component in the part annotations, similar to Section 4.2. The prototype activations are computed using the images in their native size for both datasets and interpolated to their corresponding part annotation size. We leverage multiple binarization thresholds on the prototype activations as mentioned in Section 4.2 to improve the robustness of those metrics in terms of variance across runs and to avoid a strong dependency on a fixed hyperparameter contrary to the window size used in [37].

7. Transferability of ScaleProtoSeg

To demonstrate the transferability of the proposed ScaleProtoSeg method beyond the domain of natural images, we run experiments on a medical dataset. In particular, we use the EM segmentation challenge dataset from ISBI 2012 [4], containing 30 images of size 512×512 with 2 classes that are randomly split 2 times in 20 training and 10 validation images, similar to the experiment in ProtoSeg [70]. The architecture used as the backbone for this experiment is the original U-Net [67]. The applicability of ScaleProtoSeg to another segmentation backbone is straightforward as it simply requires stacking the ASPP at the network output before the classification layer, allowing the extraction of multi-scale feature maps. The proto-

type layer contains 10 and 12 prototypes per class for ProtoSeg and ScaleProtoSeg respectively, so 3 prototypes per scale and class for ScaleProtoSeg and similarly 3 groups per class. We run 3 experiments per method in the same set-up as in ProtoSeg with pruning and grouping for each method respectively. For the prototype training stage, we skip the warm-up step and run the joint training and fine-tuning step for 10000 batches iterations each with a batch size of 2. The learning rate for the joint step is fixed at $1e^{-4}$ and we use the same learning scheduler as in Section 6. The learning rate for the fine-tuning step is constant and fixed at $1e^{-5}$. The weights of the loss terms for both methods during the prototype training stage are set to $\lambda_{L1} = 1e^{-4}$ and $\lambda_J = 0.25$. In the grouping stage for ScaleProtoSeg, we also skip the warm-up step and run the joint training step for 10000 batch iterations with a learning of $5e^{-5}$ following the same scheduler as in Section 6. The weights of the loss terms during the grouping stage are set to $\lambda_{L1} = 1e^{-4}$ and $\lambda_{ent} = 0.25$. The optimizer and augmentation pipeline used across all stages and methods is the same as in Section 6. We report the mIoU in Table 5. Results show that *ScaleProtoSeg can transfer to another backbone and image modality*. Indeed, not only ScaleProtoSeg outperforms ProtoSeg in datasets with complex scenarios (CityScapes and ADE20K), but it also yields marginally better performance in the considered medical dataset despite the fact that the images do not present multi-scale features (similarly to the PASCAL VOC benchmark).

Method	EM Split 1	EM Split 2
ProtoSeg (U-Net + ASPP)	77.63 $\pm$0.29	78.92 $\pm$0.19
ScaleProtoSeg (U-Net + ASPP)	**77.97** $\pm$0.16	**79.29** $\pm$0.17

Table 5. IoU performance of ScaleProtoSeg compared against ProtoSeg. Results demonstrate the effective transferability of ScaleProtoSeg to another segmentation architecture (U-Net) and the medical domain (ISBI 2012 dataset) [4].

8. Extension to large dataset

In order to test our method in a use case close to real-world applications we extend our evaluation to a large benchmark: COCO-Stuff [7], containing 182 classes among 11 are not present in the dataset with 118k training and 5k validation samples. We leverage a similar setup to ADE20K for the model hyperparameters, except that in the multi-scale prototype training stage, we iterate for a total of 110k iterations: 6000 warm-up steps, 100k joint steps, and 4000 finetuning steps. Moreover, the iterations are done directly on a batch of size 10. We also use multi-scale inputs with scales $\{0.5, 0.75, 1\}$ and a training resolution of 321×321, while testing at the native resolution. The objective is to align to the set-up for the DeepLabv2 baseline provided at this repository: github.com/kazuto1011/deeplab-pytorch.

Method	COCO-Stuff mIoU
DeepLabv2	39.7
ScaleProtoSeg	34.50 $\pm$0.19

Table 6. ScaleProtoSeg mIoU performance on COCO-Stuff validation set. We report the results over 3 runs for our method.

The results for our method: ScaleProtoSeg are presented in Table 6, and showcase that our method despite lower performance stays competitive with DeepLabv2 on a larger dataset while providing interpretability. Moreover, **ProtoSeg trained in a similar setup has a performance of 32.97 mIoU without pruning for one run**, so ScaleProtoSeg slightly outperforms it.

9. Sparsity Regularization

In the grouping mechanism, it is also possible to control for the sparsity-performance trade-off via the thresholding of the grouping weights in the matrices $\mathbf{w}_{g,c}$ defined in Section 3.2. This is demonstrated in Table 7. The best results are obtained for threshold $\alpha = 0.05$ on all datasets. At this threshold, the groups are sparse with an average of 3.70, 3.39, and 4.08 active prototypes per group for Pascal, Cityscapes, and ADE20K, which leads to better interpretability performance for the sparsity metric compared to ProtoSeg. Moreover, we analyze the effect of the entropy regularization on the grouping mechanism as shown in Table 8. This regularization enables our method to use fewer prototypes in total and per group while providing similar performance, with up to 20 total prototypes dropped and 2.5 prototypes per group less for Cityscapes.

Dataset	Threshold	Prototypes	Group Avg	mIoU
Pascal	$\alpha = 0.$	133	4.00	72.11
	$\alpha = 0.05$	131	3.70	**72.26**
	$\alpha = 0.1$	**126**	**3.21**	72.12
Cityscapes	$\alpha = 0.$	114	3.68	69.20
	$\alpha = 0.05$	111	3.39	**69.22**
	$\alpha = 0.1$	**109**	**3.10**	69.03
ADE20K	$\alpha = 0.$	1168	4.53	34.03
	$\alpha = 0.05$	1132	4.08	**34.32**
	$\alpha = 0.1$	**1067**	**3.48**	34.16

Table 7. Ablation study of the effect of thresholding on the grouping mechanism with $\lambda_{\text{ent}} = 0.05$ on our ScaleProtoSeg best performing run.

10. Group Overlap

In this section, we extend the analysis on the effect of the entropy loss regularization on the grouping mechanism presented above, as entropy regularization also impacts the

Dataset	Regularization	Prototypes	Group Avg	mIoU
Pascal	$\lambda_{\text{ent}} = 0$	146	5.59	72.22
	$\lambda_{\text{ent}} = 0.05$	**131**	**3.70**	**72.26**
Cityscapes	$\lambda_{\text{ent}} = 0$	132	5.79	**69.22**
	$\lambda_{\text{ent}} = 0.05$	**111**	**3.39**	**69.22**
ADE20K	$\lambda_{\text{ent}} = 0$	1405	6.06	33.40
	$\lambda_{\text{ent}} = 0.25$	**1132**	**4.08**	**34.32**

Table 8. Ablation study on the effect of the entropy regularization on the grouping mechanism with $\alpha = 0.05$ on our ScaleProtoSeg best performing run.

overlap between group activations and supports identifying prototypical parts.

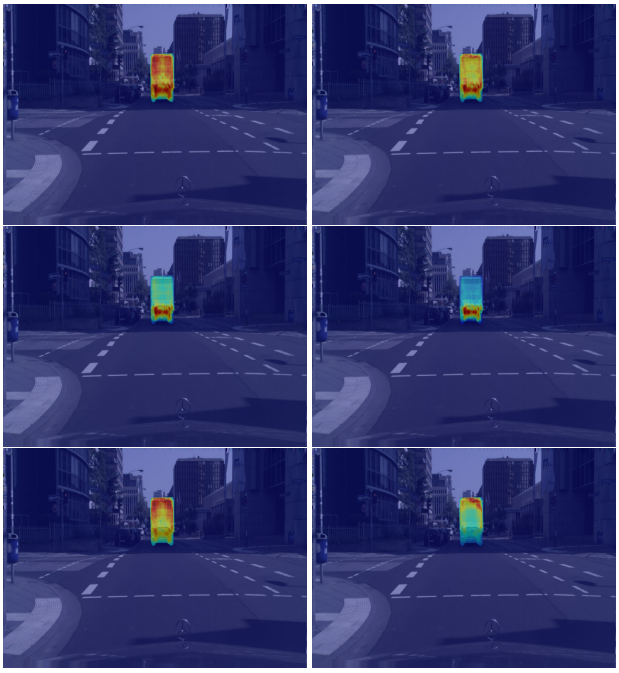
Dataset	Regularization	mIoU Groups
Pascal	$\lambda_{\text{ent}} = 0$	47.45
	$\lambda_{\text{ent}} = 0.05$	**28.29**
Cityscapes	$\lambda_{\text{ent}} = 0$	61.65
	$\lambda_{\text{ent}} = 0.05$	**43.48**
ADE20K	$\lambda_{\text{ent}} = 0$	50.25
	$\lambda_{\text{ent}} = 0.25$	**42.74**

Table 9. Analysis of the effect of the entropy regularization on the group activations overlap measured via mIoU with a threshold $\alpha = 0.05$ on our ScaleProtoSeg best performing run.

We observe that besides reducing the number of prototypes used in the groups as shown in Table 8, the entropy loss also encourages diversity in the activations of the groups assigned to the same class as shown in Figure 6 for Cityscapes. In Table 9, we present a quantitative analysis of this phenomenon. Firstly, on the validation sets of Pascal, Cityscapes, and ADE20K, we binarize all the group activations using the 95^{th} percentile. Then we compute on those validation sets the mIoU between the binarized group activation maps assigned to the same class, as a measure of overlap in the group activations. We observe that the entropy regularization on all three datasets decreases the overlap between groups of up to 19% for Pascal. A low overlap between group activations enables the model to focus on different prototypical parts of the object and avoid groups all focusing on the whole object. Those results can be explained by the increased sparsity of the grouping functions, which ultimately leads to more variation in prototype assignment between the groups.

11. Multi-scale prototype analysis

In the proposed ScaleProtoSeg method, each set of prototypes assigned to a scale $s \in S$ corresponds to a specific field of view (FOV) from the ASPP layer in [9]. We hypothesize that certain class-specific prototypes form pairs or groups of quasi-equivariant prototype activations between scales. Our definition of quasi-equivariance across scales



(a) Group activations **without** entropy regularization (b) Group activations **with** entropy regularization

Figure 6. Example of group activations with or without entropy regularization for the class *truck* on Cityscapes.

is as follows. For a class $c \in \mathcal{C}$, let us consider $\forall \mathbf{x} \in \mathbb{R}^{H \times W \times 3}$ and its downscaled version $\mathbf{x}' \in \mathbb{R}^{\frac{H}{2} \times \frac{W}{2} \times 3}$ (the factor of 2 here matched the atrous rate increase from scale 1 to 2). Let us select the candidate pair of quasi-equivariant prototypes $\mathbf{p}_{1,i} \in P_{1,c}$ and $\mathbf{p}_{2,j} \in P_{2,c}$ and compute their respective activation maps $\mathbf{A}_{1,i}$ from all $\mathbf{z}_1 \in f_1(\mathbf{x}')$ with $g_{\text{proto}}(\mathbf{z}_1, \mathbf{p}_{1,i})$, and $\mathbf{A}_{2,j}$ from all $\mathbf{z}_2 \in f_2(\mathbf{x})$ with $g_{\text{proto}}(\mathbf{z}_2, \mathbf{p}_{2,j})$. The pair of prototypes is considered as quasi-equivariant if:

$$f_{\text{upsample}}(\mathbf{A}_{1,i}) \sim \mathbf{A}_{2,j} \tag{10}$$

with f_{upsample} the upsampling function for $\mathbf{A}_{1,i}$ to the original size of $\mathbf{A}_{2,j}$. The similarity measure between activations is defined below.

In order to identify the sets of prototypes with quasi-equivariant activations for a specific class $c \in \mathcal{C}$ we first specify the increasing atrous rates ratio of each ASPP output feature maps with respect to the smallest one: $\{r_{1\to1}, r_{1\to2}, r_{1\to3}, r_{1\to4}\}$ in the case of $S = 4$. The objective is to compute for all training images $\mathbf{x} \in \mathbb{R}^{H \times W \times 3}$, downscaled versions of the original image $\mathbf{x}_{1\to s}$ such that $\mathbf{x}_{1\to s} \in \mathbb{R}^{\frac{H}{r_{1\to s}} \times \frac{W}{r_{1\to s}} \times 3}$. Then, for all distinct pairs of feature map scales such that $(s, s') \in S^2$ and $s < s'$, we select $\mathbf{x}_{1\to s}$ and $\mathbf{x}_{1\to s'}$ to compute $f_s(\mathbf{x}_{1\to s'})$ and $f_{s'}(\mathbf{x}_{1\to s})$. The objective is to align, through the downscaling of the image, the semantic parts covered by the FOV of each ASPP scale

s and s'. Then we compare all pairs of prototypes from the scales s and s'. For a specific pair $(\mathbf{p}_{s,i}, \mathbf{p}_{s',j})$ we compute from the feature maps $f_s(\mathbf{x}_{1\to s'})$ and $f_{s'}(\mathbf{x}_{1\to s})$, the activation maps $f_{\text{upsample}}(\mathbf{A}_{s,i})$ and $\mathbf{A}_{s',j}$, as described above. The similarity measure is then computed as follows, as we want to focus on the most activated parts for the objects assigned to c we threshold the activation maps to a percentile $p_{th} \in \{0.6, 0.7\}$ only considering the positions where the ground truth label $y_{\mathbf{z}} = c$. Then we derive the mIoU between the binarized activation maps across all the training set images, defining our similarity measure. Pairs of quasi-equivariant prototypes are identified when the mIoU is above a fixed threshold $\text{IoU}_{th} = 0.5$ on the training set. Lastly, pairs of quasi-equivariant prototypes are merged into groups if they overlap across scales.

In Figure 7, we present the results of the equivariance analysis on three datasets. Interestingly, we observe that for both Cityscapes and Pascal VOC there are more than 50% of classes with quasi-equivariant groups for $p_{th} = 0.6$, demonstrating that the network learns through its multi-scale prototypes similar activations for similar prototypical parts in the image at different scales. For ADE20K, we observe that only 12% of classes have quasi-equivariant groups. We argue that since ADE20K is characterized by many complex scenes, as we constrain the representation space to the learning of only 12 prototypes across scales, all the learned prototypes represent scale-specific contextual information. We suppose that if we were to increase the number of prototypes per scale, redundant information would start to appear across scales and more prototypes would be activated on similar prototypical parts at different scales, leading to an increase in the number of classes with quasi-equivariant groups. In Figure 8, 9, and 10 we show examples of pair of quasi-equivariant prototypes for $p_{th} = 0.6$.

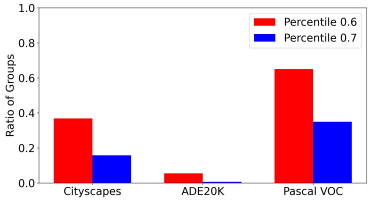
12. Sparsity and Computation Overhead

Dataset	Method	Avg Weight	Parameters	It/s
Pascal	ProtoSeg	0.527 ±0.004	12.1K ±0.3	**20.7** ±0.2
	ScaleProtoSeg	**0.063** ±0.001	**9.1K** ±0.0	15.6 ±0.5
Cityscapes	ProtoSeg	0.523 ±0.001	9.9K ±0.2	**52.5** ±0.2
	ScaleProtoSeg	**0.071** ±0.001	**7.6K** ±0.1	5.1 ±0.0
ADE20K	ProtoSeg	0.503 ±0.001	238.2K ±2.2	**51.9** ±2.0
	ScaleProtoSeg	**0.437** ±0.001	**135.4K** ±0.3	14.5 ±0.8

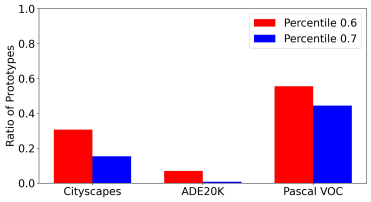
Table 10. Analysis of the sparsity of the final classification layer via the average absolute weight in $\mathbf{w}_{h_{\text{proto}}}$ for ProtoSeg and $\mathbf{w}_{h_{\text{group}}}$ for ScaleProtoSeg, and the computation overhead with respect to DeepLabv2 induced by the interpretable methods.

In the sparse grouping mechanism proposed in our method, we also enforce a strong sparsity regularization on the last layer h_{proto} which enables the model to constrain the negative effect of the prototypes not assigned to a class in the final decision process, as shown in Section 3.3. We see in Table 10 that our method presents a small average absolute weight on $\mathbf{w}_{h_{\text{group}}}$ and as a consequence a strong sparsity for both Pascal VOC and Cityscapes, especially compared to ProtoSeg. Moreover, despite the increased complexity of ADE20K, our method presents still a smaller average absolute weight than ProtoSeg.

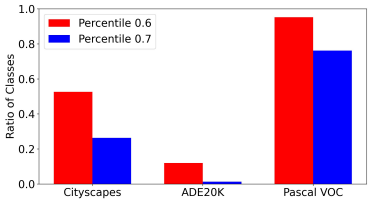
We also analyze in Table 10 the number of extra-parameters necessary in our method and ProtoSeg compared to the black-box baseline: DeepLabv2, after pruning at inference. We observe that, due to the limited number of active prototypes and the group projection, ScaleProtoSeg requires less computational overhead, especially for a large number of classes like in ADE20K, where our method uses 43% less extra parameters. Those results are computed on 3 local runs per method, and it is important to highlight that ProtoSeg on Pascal despite similar performance presents $\sim$ 25 more prototypes after pruning compared to [70], which also directly impacts the sparsity met-



(a) Ratio of quasi-equivariant groups identified.

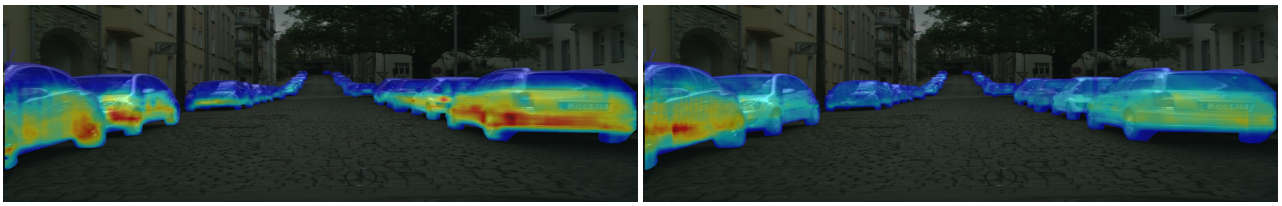


(b) Ratio of prototypes in the quasi-equivariant groups.

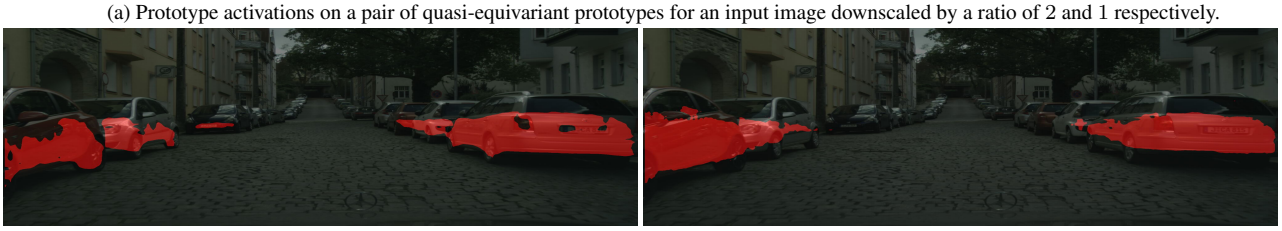


(c) Ratio of classes with quasi-equivariant groups.

Figure 7. Analysis of the presence of quasi-equivariant groups in all three datasets for $p_{th} \in \{0.6, 0.7\}$ on our best performing ScaleProtoSeg run. In **(a)**, the number of quasi-equivariant groups is compared to the total number of groups defined in the sparse grouping mechanism.



(a) Prototype activations on a pair of quasi-equivariant prototypes for an input image downscaled by a ratio of 2 and 1 respectively.



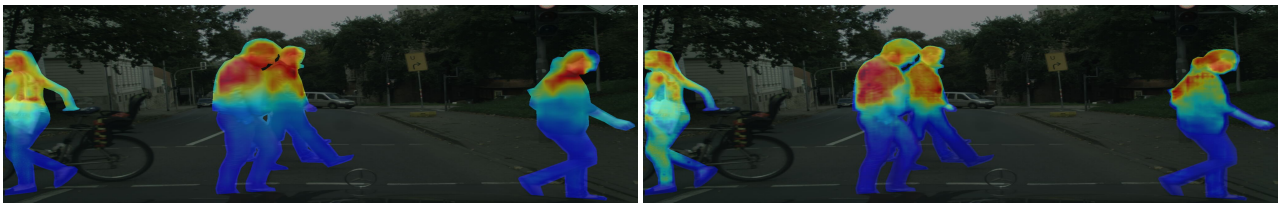
(b) Binarized prototype activations on a pair of quasi-equivariant prototypes for an input image downscaled by a ratio of 2 and 1 respectively.

Figure 8. Example of quasi-equivariant pair of prototypes for the class *car* on Cityscapes. Prototype on the left is from *Scale 1* and on the right is from *Scale 2*.

(a) Prototype activations on a pair of quasi-equivariant prototypes for an input image downscaled by a ratio of 3 and 1 respectively.

(b) Binarized prototype activations on a pair of quasi-equivariant prototypes for an input image downscaled by a ratio of 3 and 1 respectively.

Figure 9. Example of quasi-equivariant pair of prototypes for the class *rider* on Cityscapes. Prototype on the left is from *Scale 1* and on the right is from *Scale 3*.



(a) Prototype activations on a pair of quasi-equivariant prototypes for an input image downscaled by a ratio of 3 and 1 respectively.

(b) Binarized prototype activations on a pair of quasi-equivariant prototypes for an input image downscaled by a ratio of 3 and 1 respectively.

Figure 10. Example of quasi-equivariant pair of prototypes for the class *person* on Cityscapes. Prototype on the left is from *Scale 1* and on the right is from *Scale 3*.

ric in Table 6.

It is also important to consider the computational overhead during training. First, we showed that the added number of parameters compared to DeepLabv2 is minimal. However, some overhead complexity arises in ProtoSeg and ScaleProto from the KLD regularisation loss and the prototype projection. In particular, the impact on training time of the KLD loss is much more important in the case of ProtoSeg at a high number of prototypes compared to ScaleProtoSeg as in our method we parallelize the loss across scales. Moreover, at training time the fine-tuning of the classification layer and groups after the prototype projection of our method adds only a limited overhead to the training as we tune only a handful of parameters for a few epochs.

Finally as shown in Table 10 at inference our method is slower than ProtoSeg due to the grouping process but we believe that is likely due to a lack of parallelization in our code.

13. Group Representations and Assignments

In this section, we aim to present more representative examples of our results in terms of interpretability. First, we show in Figure 11 an example of group activations for the class *person* in Cityscapes, where groups 1 and 2 identify the head and feet respectively of a *person*, while group 3 identifies the main body of the *persons*. Interestingly, similarly to the class *car* in the main paper, group 2 is mainly activated by a prototype at scale 1 and as a consequence seems to be more present on *person* further away in the image as seen in **(b)**. This level of understanding of the scale effect on the model representation is a key contribution of our method compared to ProtoSeg in terms of interpretability.

Moreover, we show in Figure 12 and 13 more comparative results between our method and ProtoSeg, where again the limitations on the number of groups simplify the analysis of the decision process for our method as seen for the class *bicycle* and *chair*. In the analysis of the class *rider* and *aeroplane*, we can observe another advantage of our method as ProtoSeg presents a failure case of prototype pruning where only one prototype specific to the class *rider* or *aeroplane* is left after post-processing.

14. Nearest Image Patches to Prototypes

For the learned prototypes to be interpretable to the users, the semantic parts that they represent should be similar across images. To analyze whether our method is consistent across images, we extend the consistency metric from Section 4.2 with visualizations of the nearest images in the validation set from a few prototypes in Cityscapes and Pascal VOC. Those visualizations are shown in Figures [14 - 19], and showcase strong semantic correspondences between the prototypes and their closest patches.

15. Nearest Prototypes to Image Patches

To further analyze the semantic correspondences in the prototype layer of the model, we study the nearest prototypes to all image patches during the decision process. This is also directly linked to the consistency metric. To this purpose, in Figure [20 - 25] we visualize the three closest prototypes for random patches in an image. The visualizations again showcase good semantic matches for our proposed method.

16. Semantic meaning of groups

We provide a clearer understanding of the grouping mechanism by showing, in Figure 26, the activations of prototypes assigned to the same group for a given class. It can be observed that prototypes are grouped into semantically meaningful groups.

17. Failure case analysis

In this section, we analyze an image from PASCAL VOC where the IoU for a given class: cow, falls below a threshold set to $\epsilon_{IoU} = 0.2$, to describe how our method can be leveraged for interpretability analysis. For this failure, we observe in Figure 27 that the veal in the image is classified as a sheep. Indeed, we observe in Figure 28 that for the group activations of the class cow, only the first group is slightly activated essentially on the second cow behind, while the 3 sheep groups activate both the top and bottom of the animal. We observe that all the main prototypes for the cow groups in Figure [30 - 32] either activate a part not visible on the veal or seem to rather focus on the ”texture” of the animal which can be misleading in this case. For the class sheep, we observed in Figure [33 - 35] that the two most activated groups: 1 and 3, are driven by the same prototype which is a Scale 1 prototype focusing on texture. This seems reasonable knowing that the wool is a characteristic specific to this class and that a similar aspect can be seen for the veal. To avoid learning misleading ”texture” prototypes such as the one mentioned previously, the method presented in [6] can be leveraged to enforce forgetting such prototypes via human expert feedback.

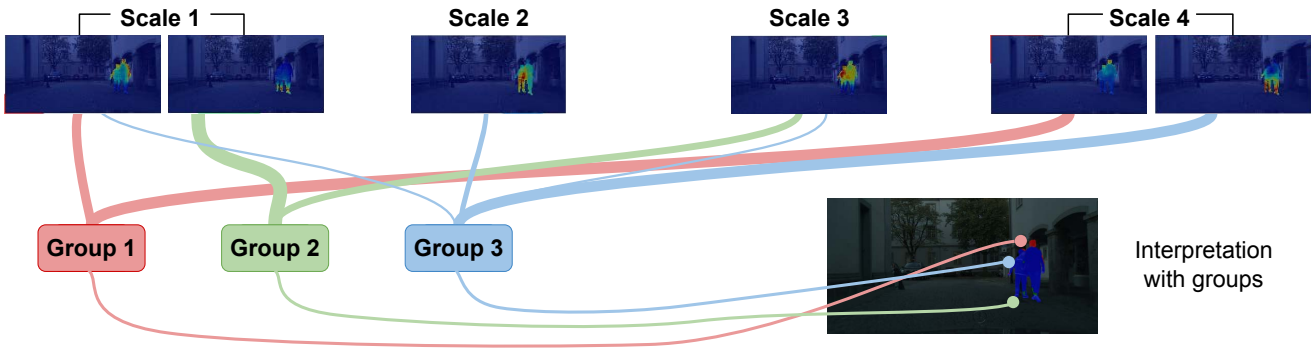


Figure 11. ScaleProtoSeg provides an interpretation of the resulting segmentation through groups of prototypes. For an example of the class *person* on Cityscapes, 2 prototypes per scale (whose activations are displayed **at the top** of the figure) are used by the model across the 3 learned groups shown **at the bottom right**. For this class, groups can correspond to the feet, the main body or the head of the person.

Figure 12. Model prototype, group assignments and prediction for the class *bicycle*, *rider*, and *chair* on Cityscapes, and ADE20K.

Figure 13. Model prototype, group assignments and prediction for the class *dinning table*, *aeroplane*, and *animal* on Pascal, and ADE20K.

Figure 14. The first row represents a prototype for the class *person* in Cityscapes marked by a red box and the activation of the prototype on its image. The second to fourth rows show the closest patches to the prototype and its activation on the images containing the patches.

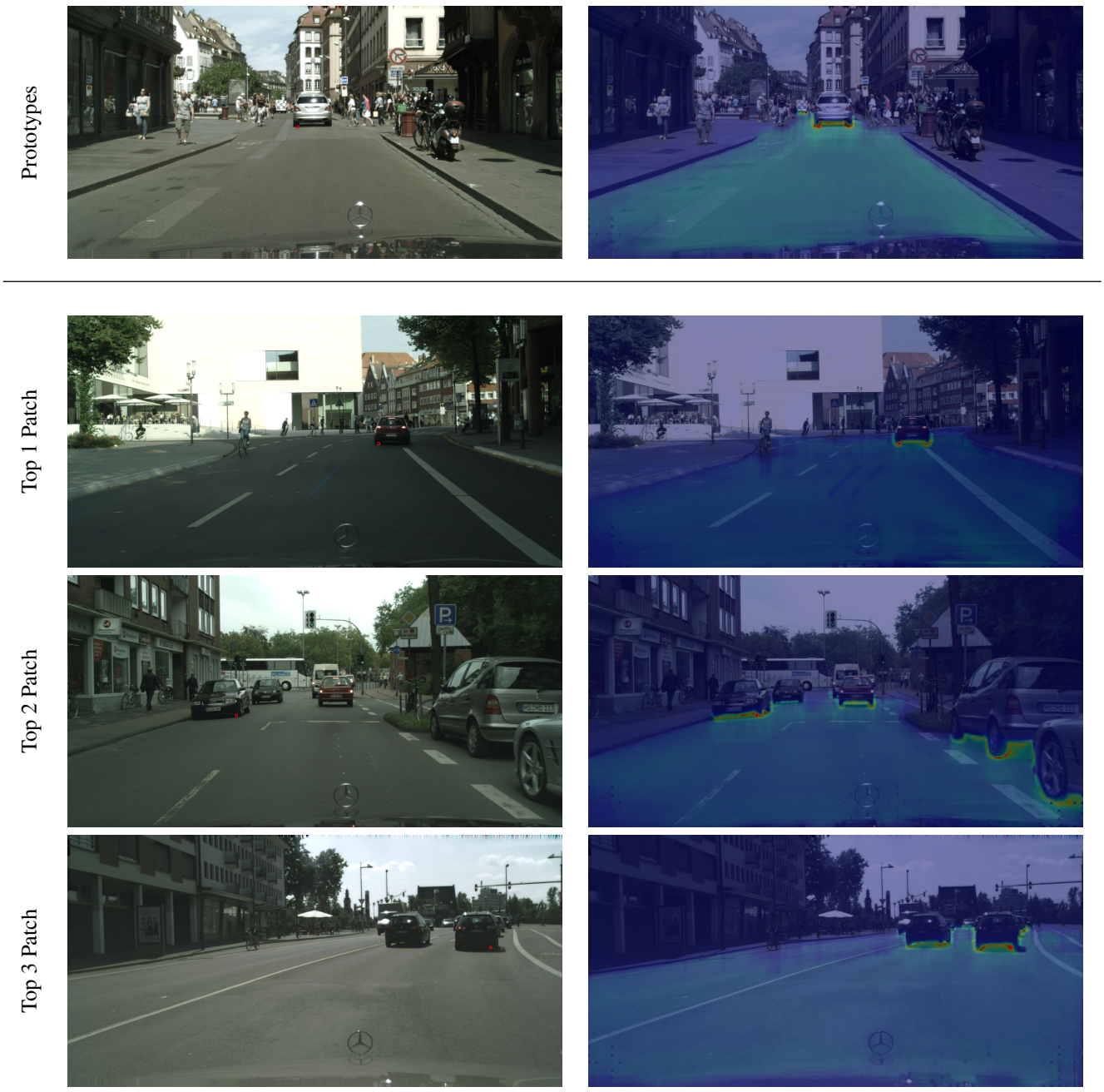


Figure 15. The first row represents a prototype for the class *car* in Cityscapes marked by a red box and the activation of the prototype on its image. The second to fourth rows show the closest patches to the prototype and its activation on the images containing the patches.

Figure 16. The first row represents a prototype for the class *bicycle* in Cityscapes marked by a red box and the activation of the prototype on its image. The second to fourth rows show the closest patches to the prototype and its activation on the images containing the patches.

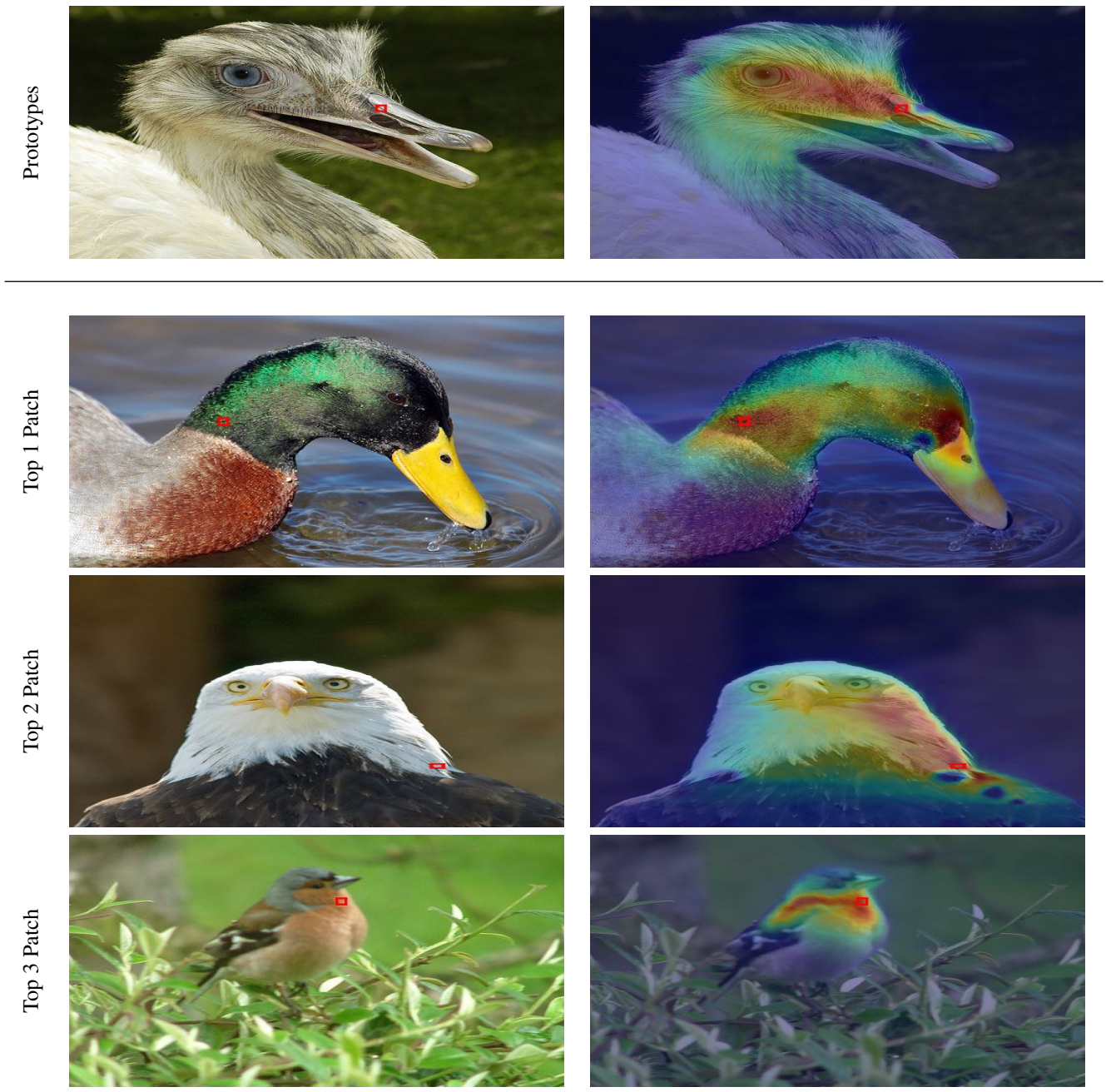


Figure 17. The first row represents a prototype for the class *bird* in PASCAL VOC marked by a red box and the activation of the prototype on its image. The second to fourth rows show the closest patches to the prototype and its activation on the images containing the patches.

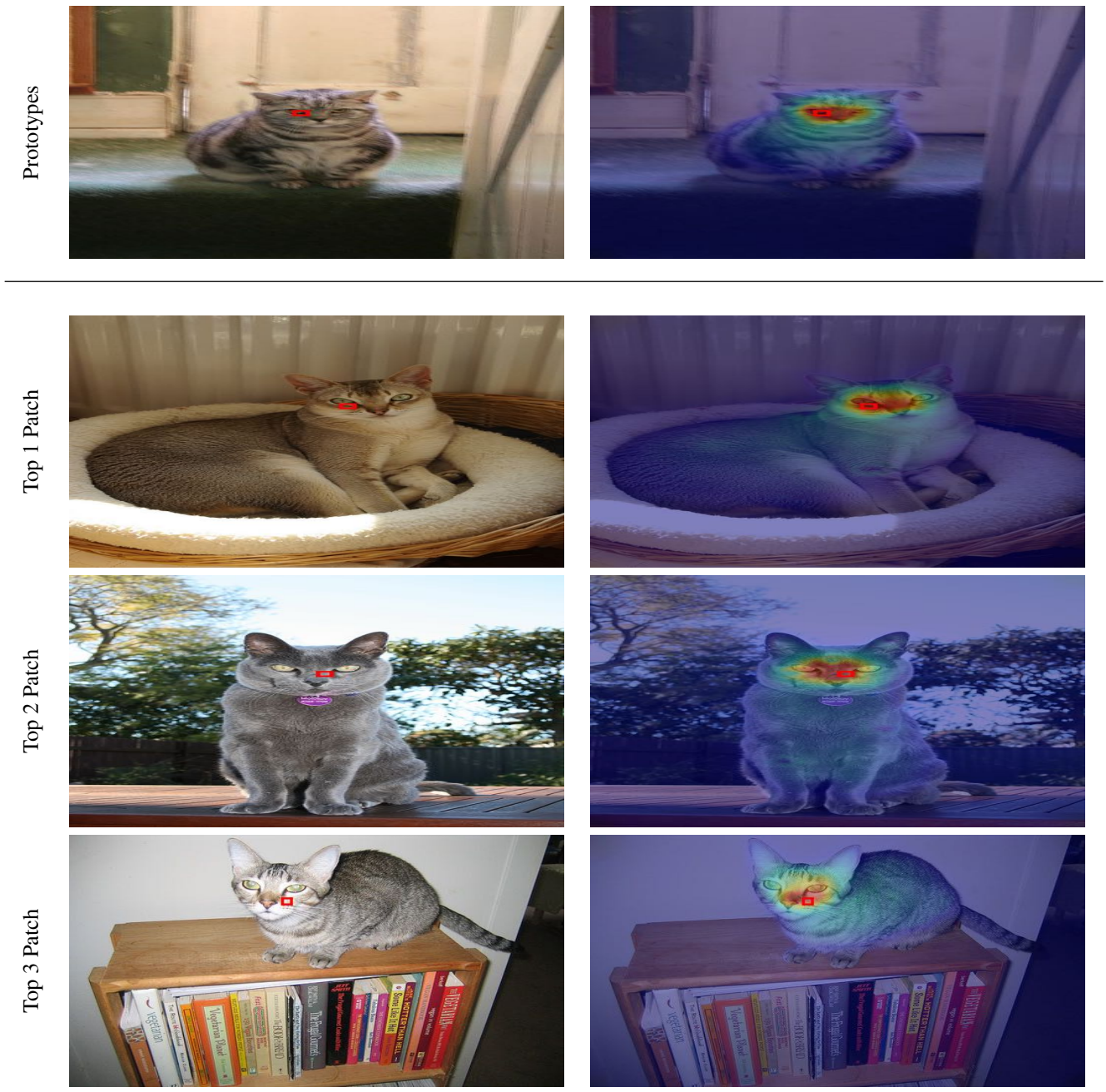


Figure 18. The first row represents a prototype for the class *cat* in PASCAL VOC marked by a red box and the activation of the prototype on its image. The second to fourth rows show the closest patches to the prototype and its activation on the images containing the patches.

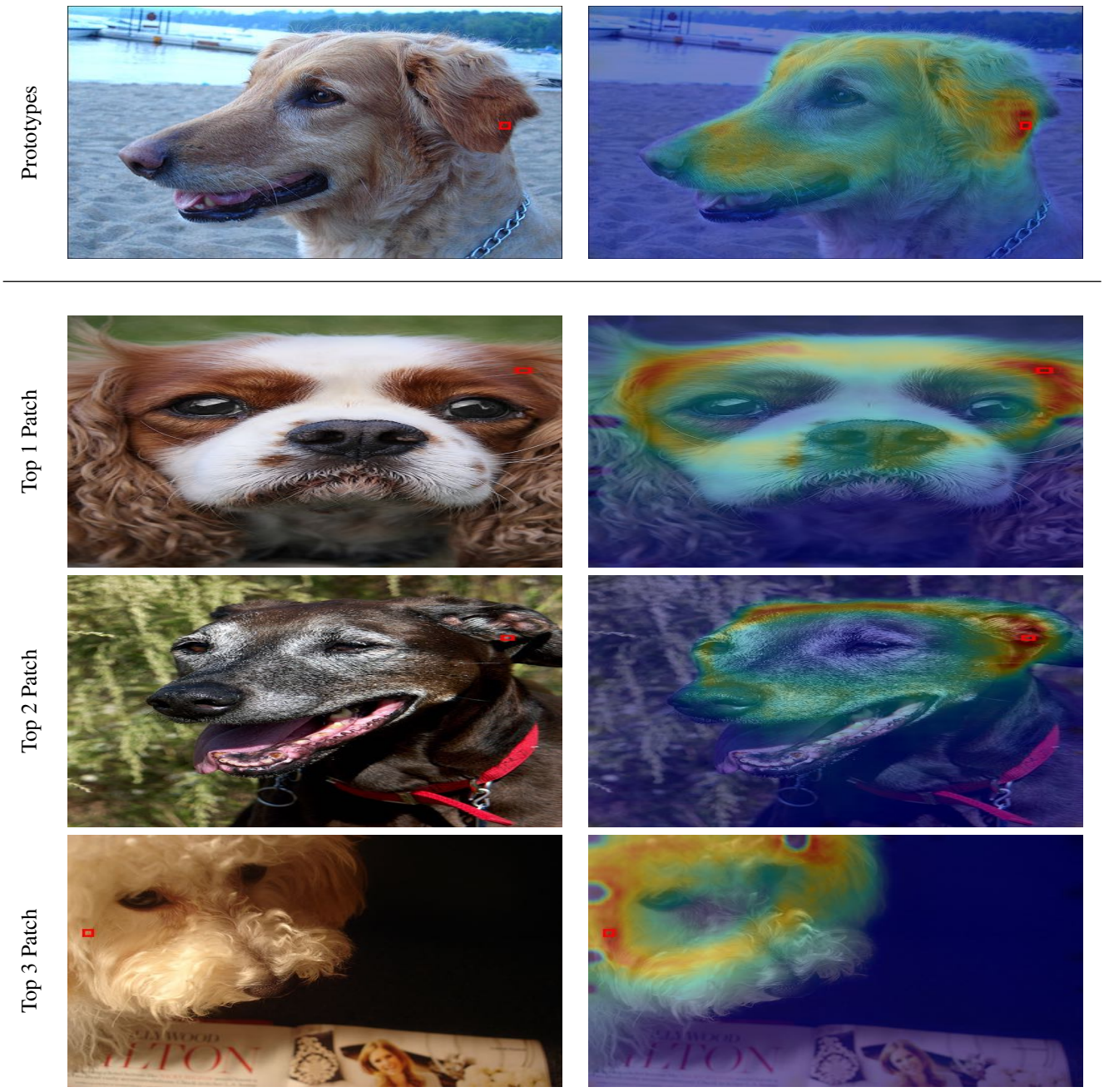


Figure 19. The first row represents a prototype for the class *dog* in PASCAL VOC marked by a red box and the activation of the prototype on its image. The second to fourth rows show the closest patches to the prototype and its activation on the images containing the patches.

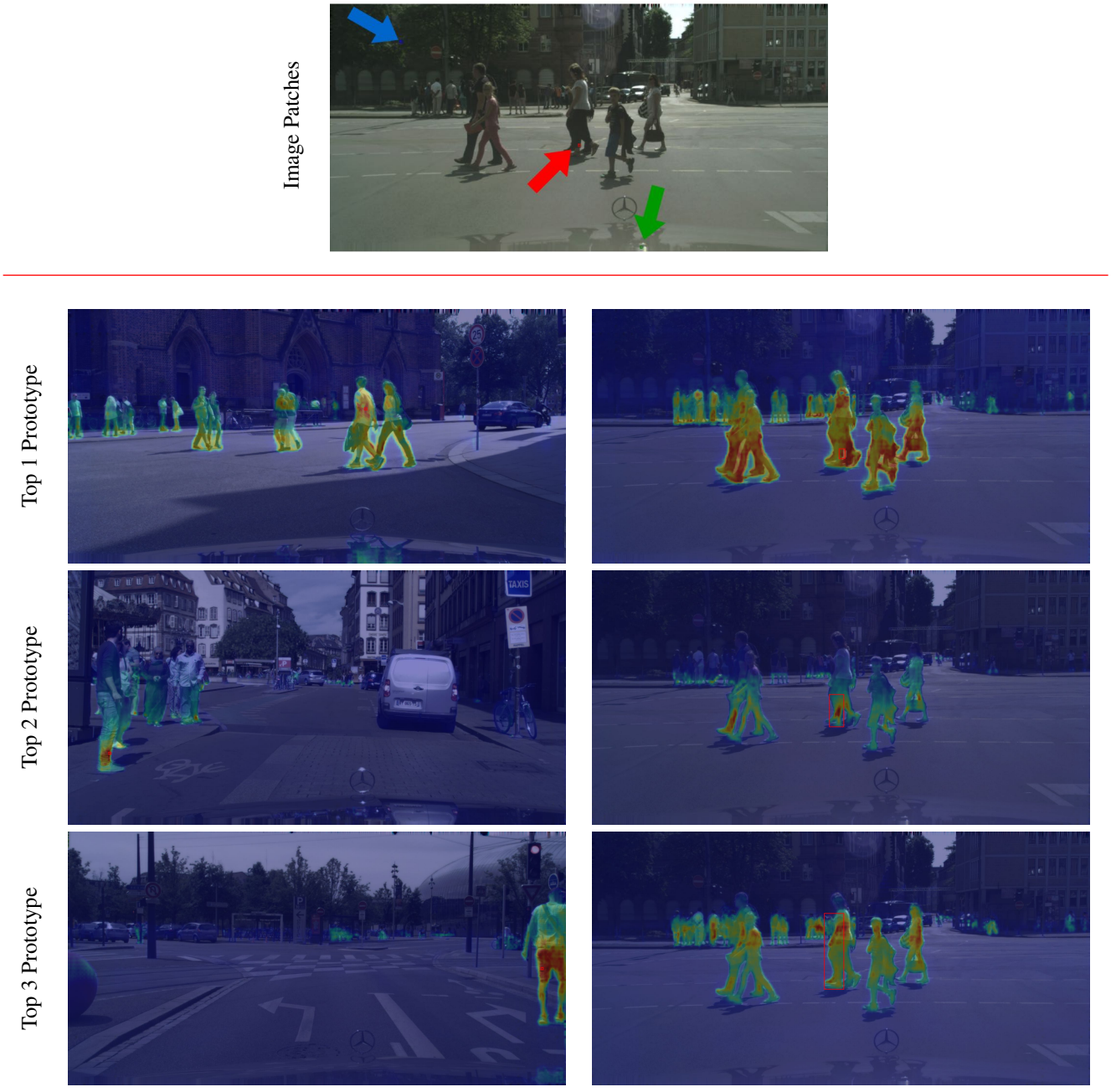


Figure 20. The first row represents a validation image from Cityscapes with three randomly selected patches red, green, blue. The second to fourth rows show the closest prototypes to the red patch via its activation on itself and on the image containing the red patch. In the second column, the bounding boxes correspond to the similarly activated area centered around the random patch.

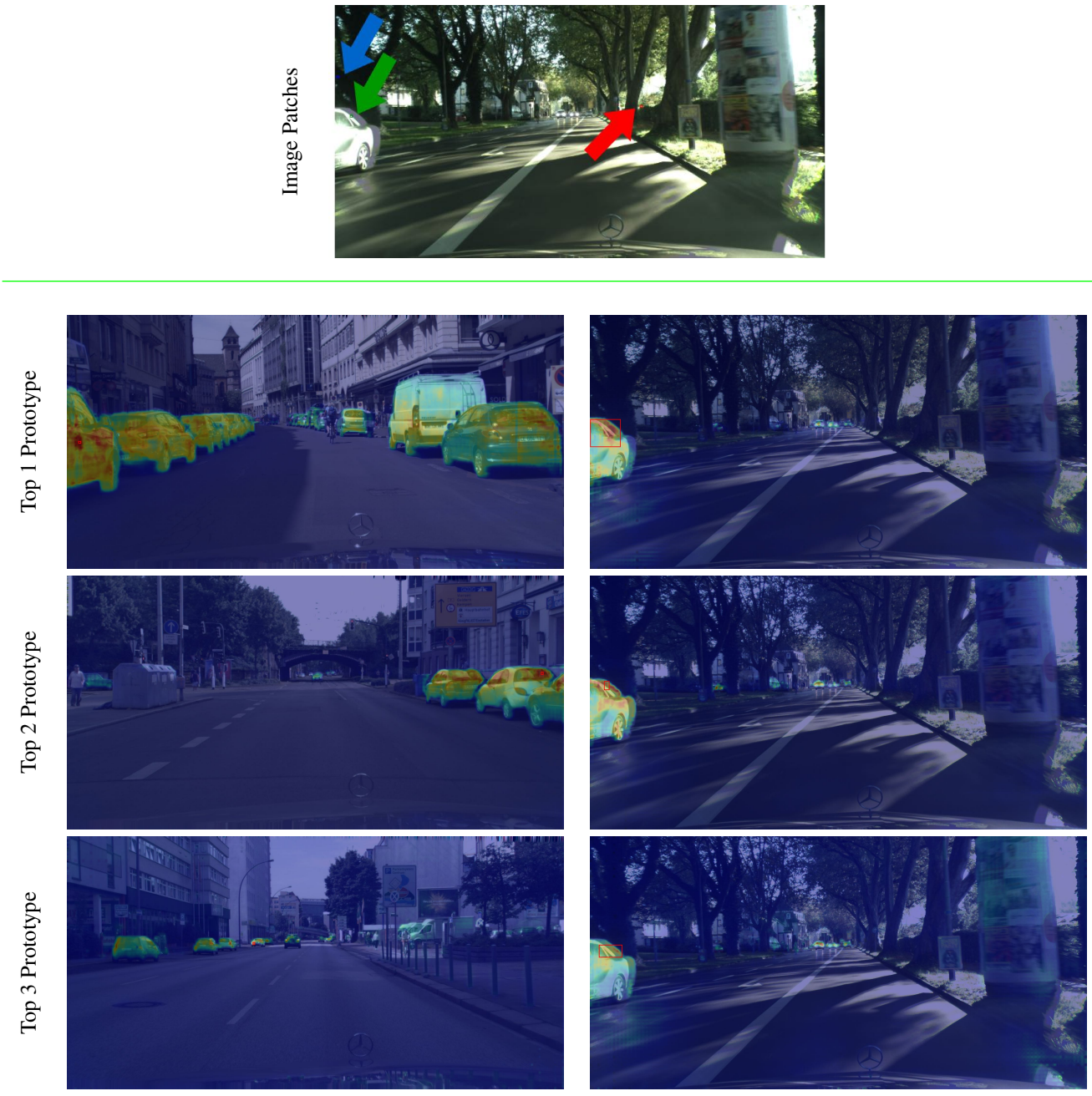


Figure 21. The first row represents a validation image from Cityscapes with three randomly selected patches red, green, blue. The second to fourth rows show the closest prototypes to the green patch via its activation on itself and on the image containing the green patch. In the second column, the bounding boxes correspond to the similarly activated area centered around the random patch.

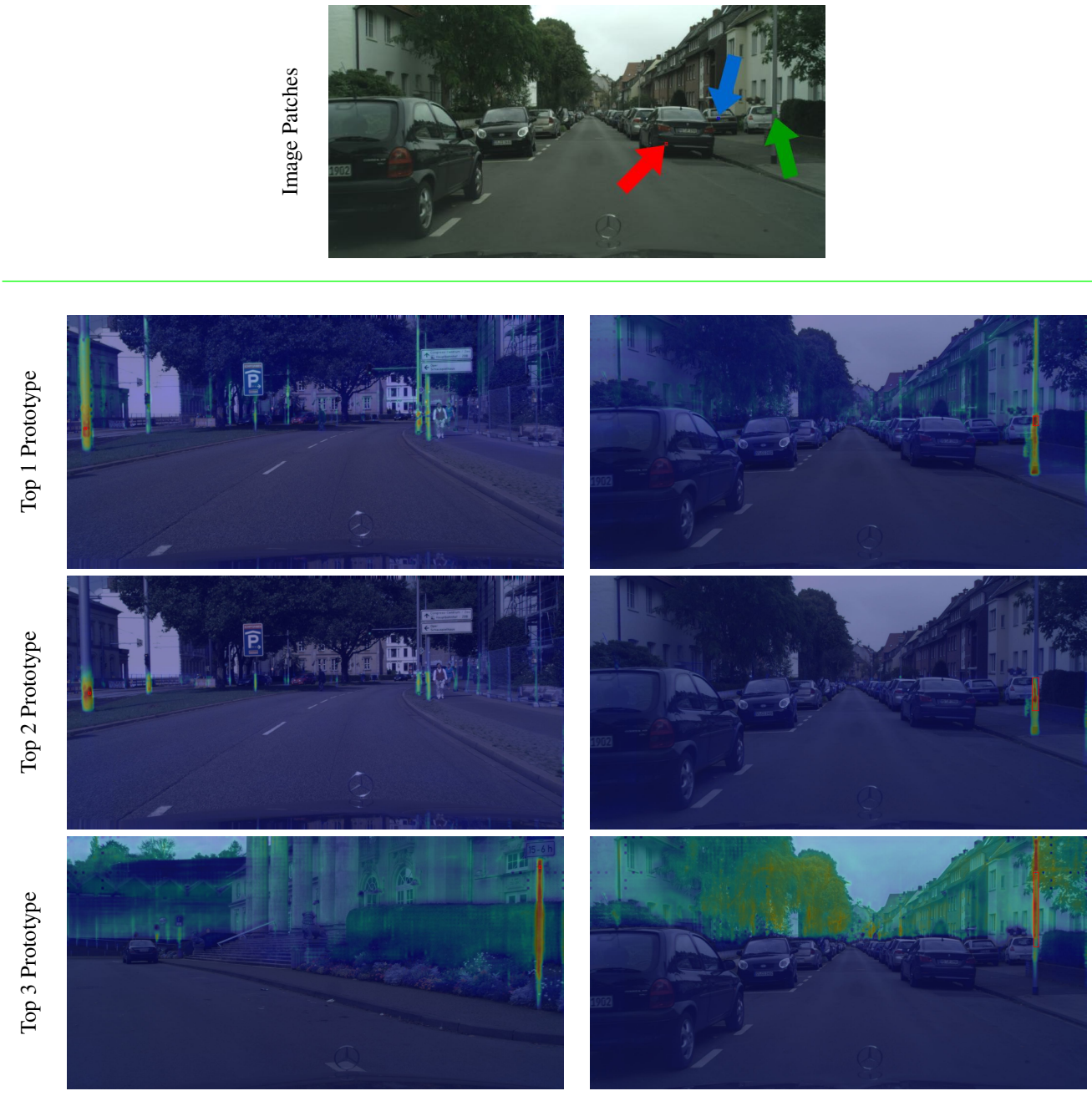


Figure 22. The first row represents a validation image from Cityscapes with three randomly selected patches red, green, blue. The second to fourth rows show the closest prototypes to the green patch via its activation on itself and on the image containing the green patch. In the second column, the bounding boxes correspond to the similarly activated area centered around the random patch.

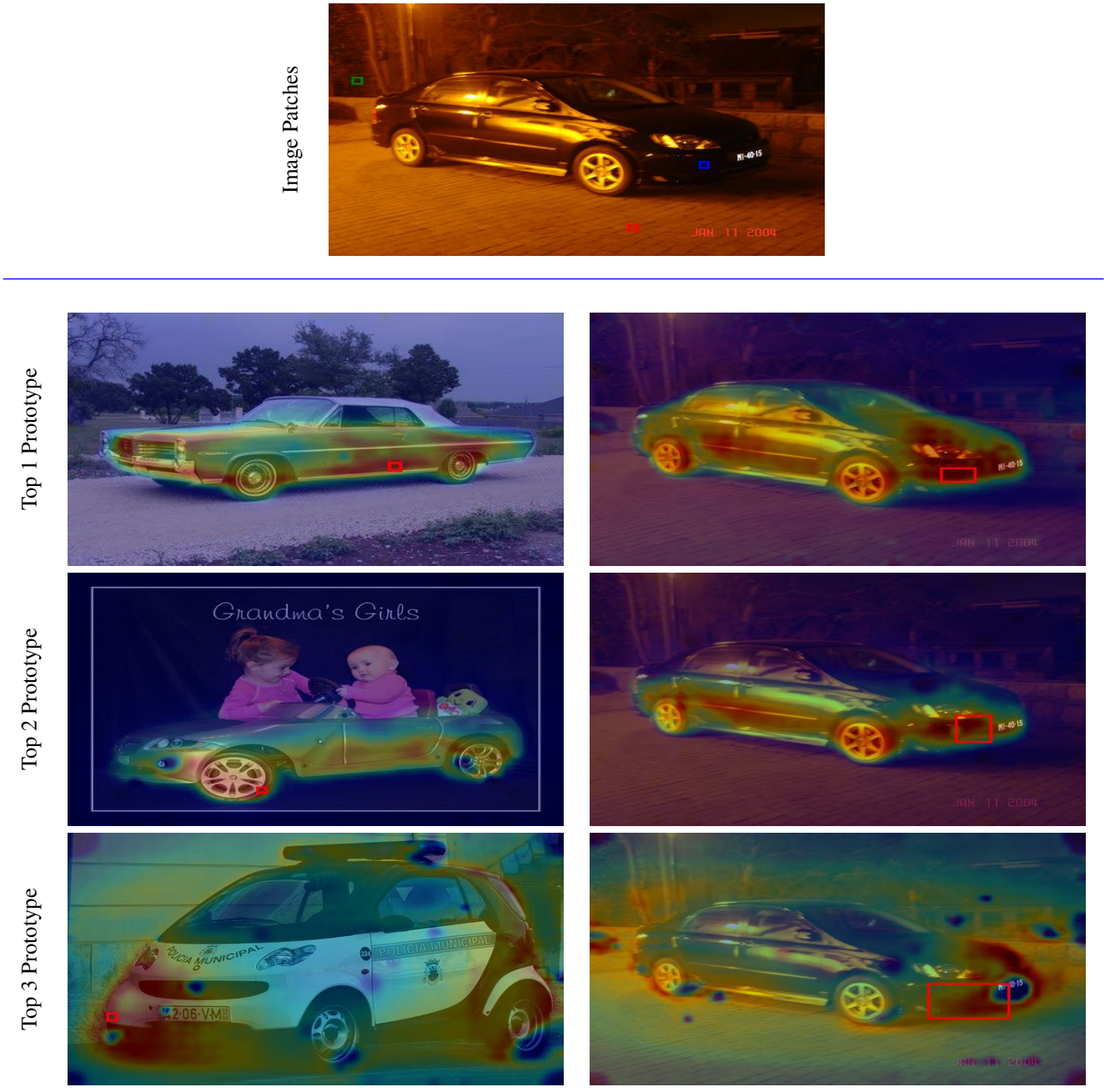


Figure 23. The first row represents a validation image from PASCAL VOC with three randomly selected patches red, green, blue. The second to fourth rows show the closest prototypes to the blue patch via its activation on itself and on the image containing the blue patch. In the second column, the bounding boxes correspond to the similarly activated area centered around the random patch.

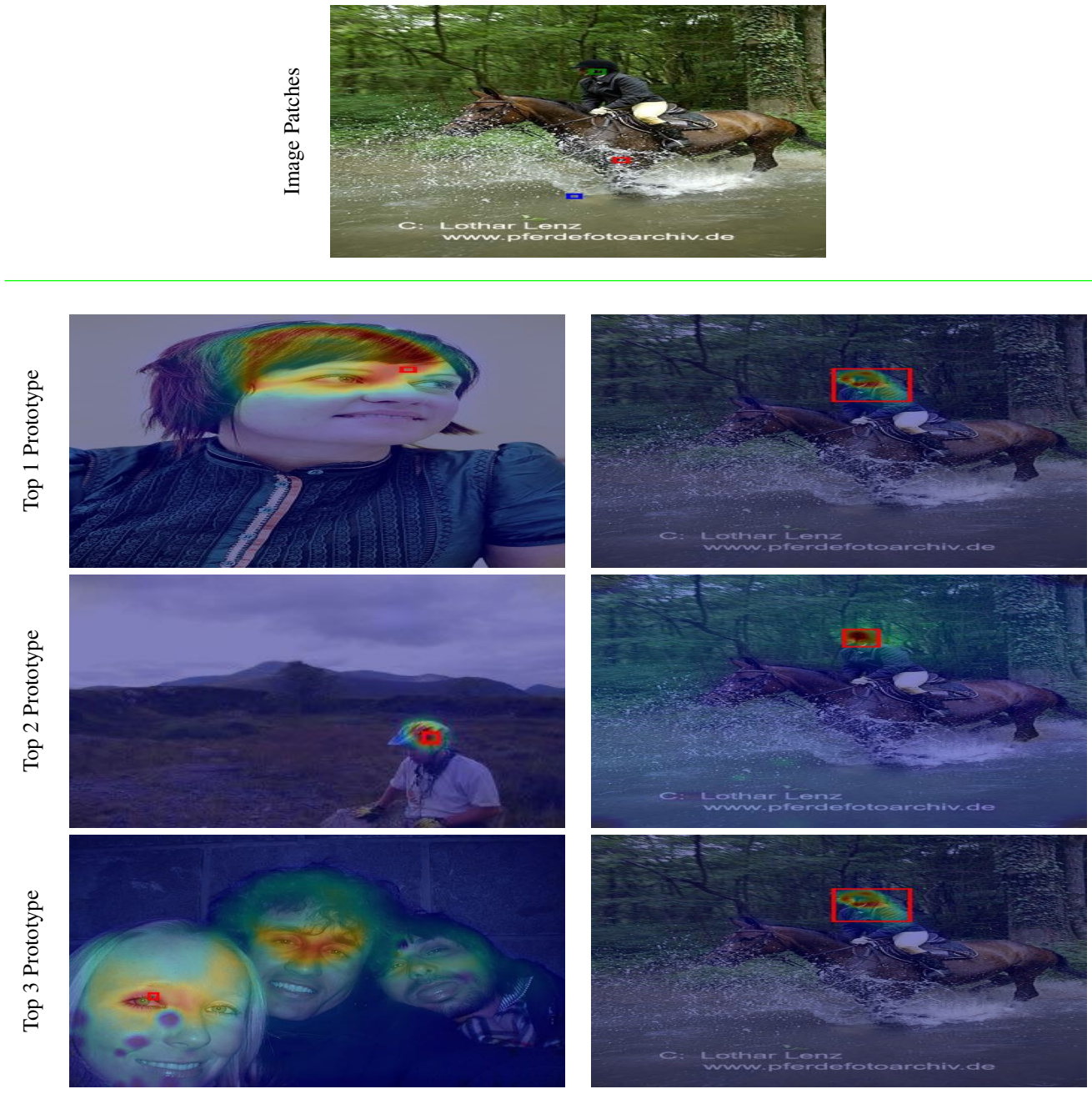


Figure 24. The first row represents a validation image from PASCAL VOC with three randomly selected patches red, green, blue. The second to fourth rows show the closest prototypes to the green patch via its activation on itself and on the image containing the green patch. In the second column, the bounding boxes correspond to the similarly activated area centered around the random patch.

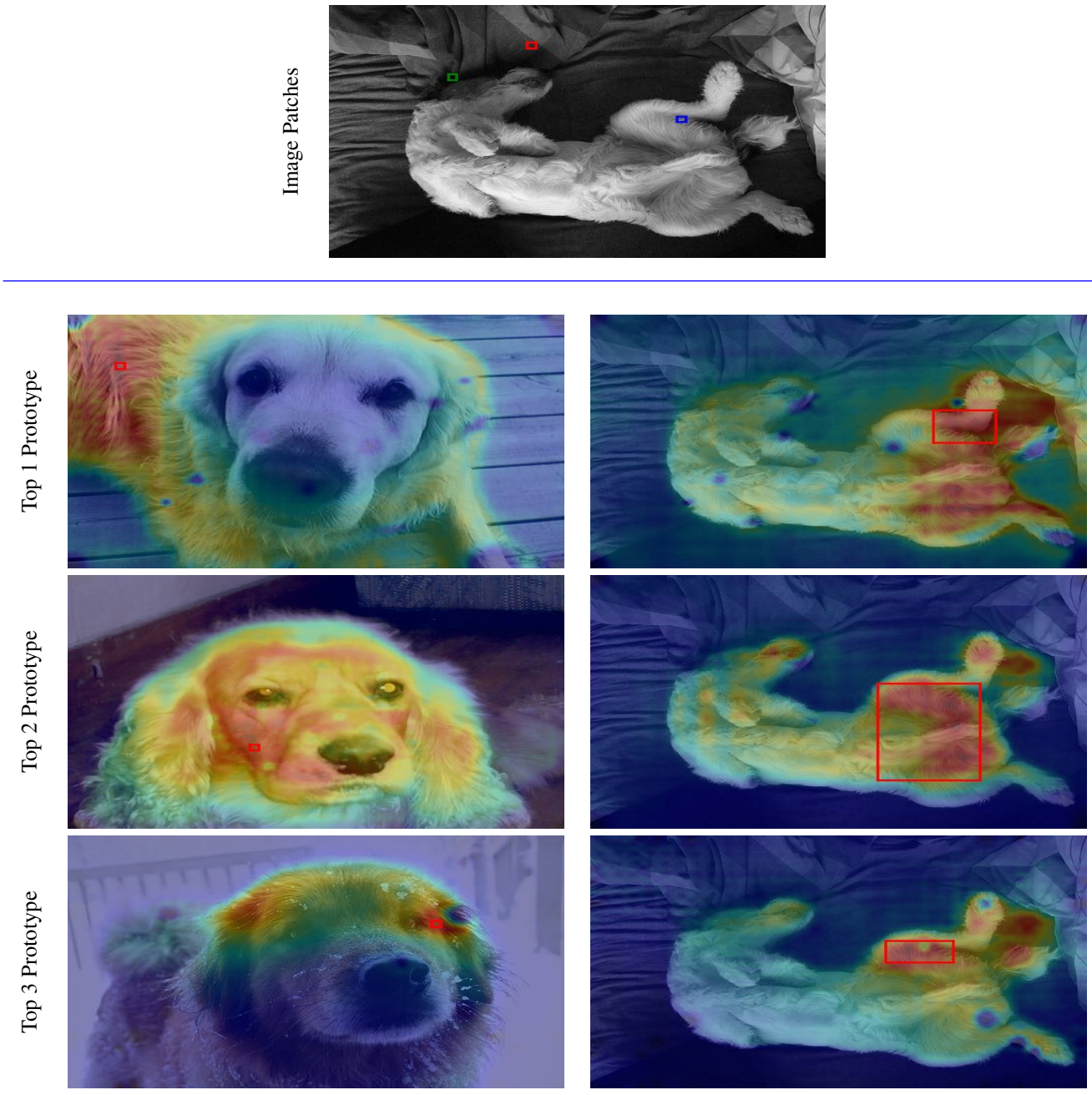


Figure 25. The first row represents a validation image from PASCAL VOC with three randomly selected patches red, green, blue. The second to fourth rows show the closest prototypes to the blue patch via its activation on itself and on the image containing the blue patch. In the second column, the bounding boxes correspond to the similarly activated area centered around the random patch.

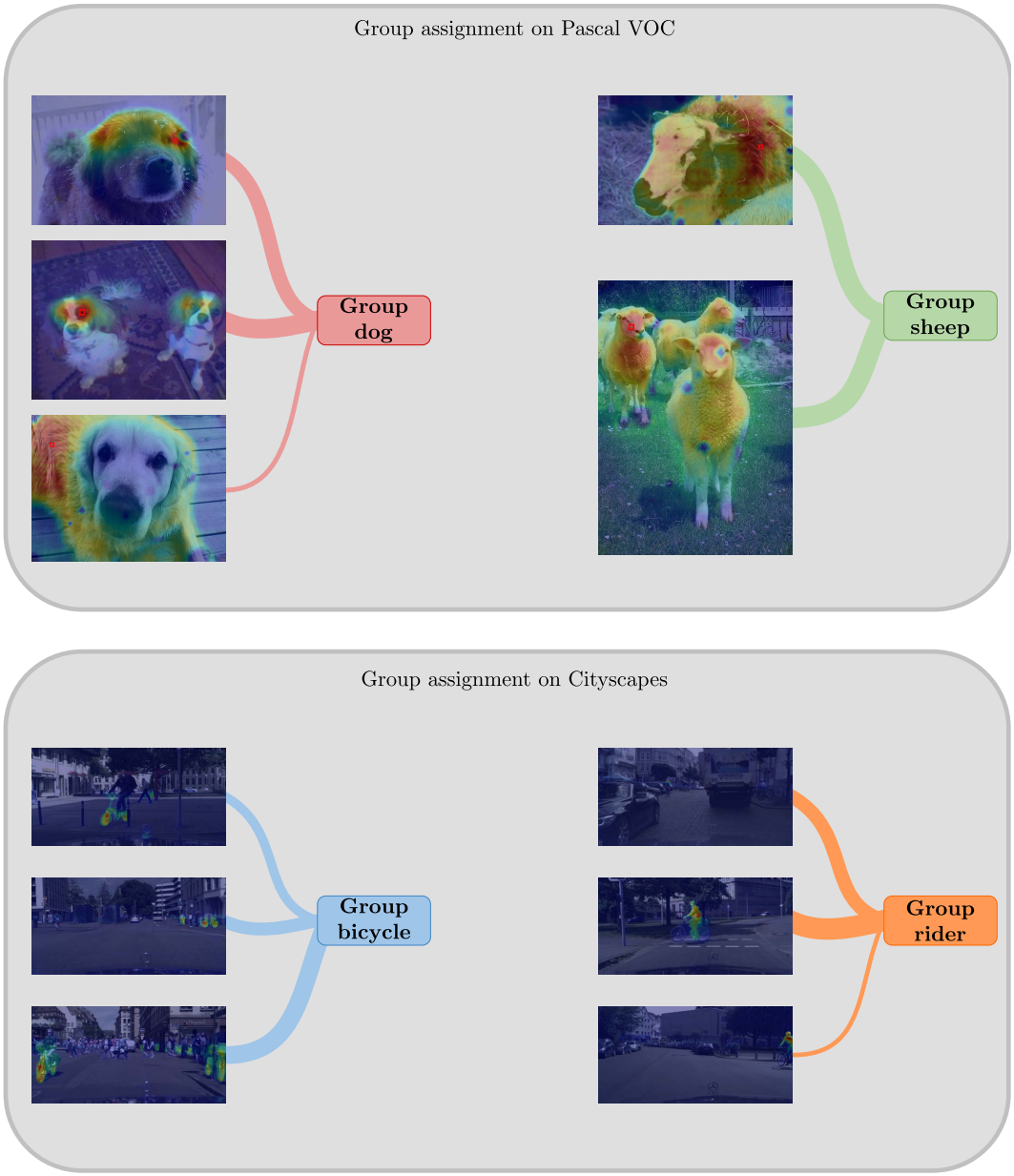


Figure 26. **Top:** for the classes *dog* and *sheep* on Pascal VOC, prototypes with the highest activation for the considered group represent the head. **Bottom**: for the class *bicycle* of Cityscapes, prototypes with the highest activation for the considered group represent the bicycle wheels, while for the class *rider* they mainly highlight the person's upper body.

Figure 27. PASCAL VOC image with its predictions for the class *cow* and in particular the most common error for this class is with the class *sheep*.

Figure 28. Group activations of the class *cow* for the image.

Figure 29. Group activations of the class *sheep* for the image.

Figure 30. Most activated prototypes represented by their training sample and their activation on the input image for Group 1 of class *cow*.

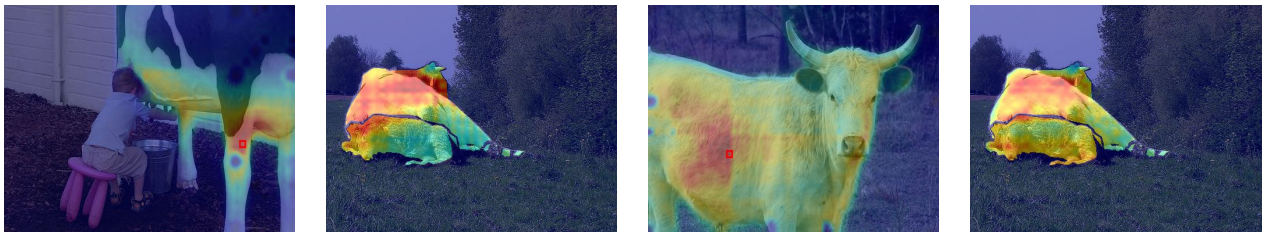


Figure 31. Most activated prototypes represented by their training sample and their activation on the input image for Group 2 of class *cow*.

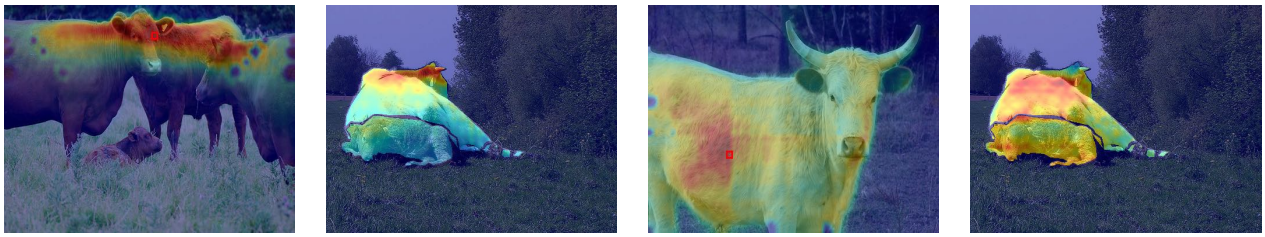


Figure 32. Most activated prototypes represented by their training sample and their activation on the input image for Group 3 of class *cow*.

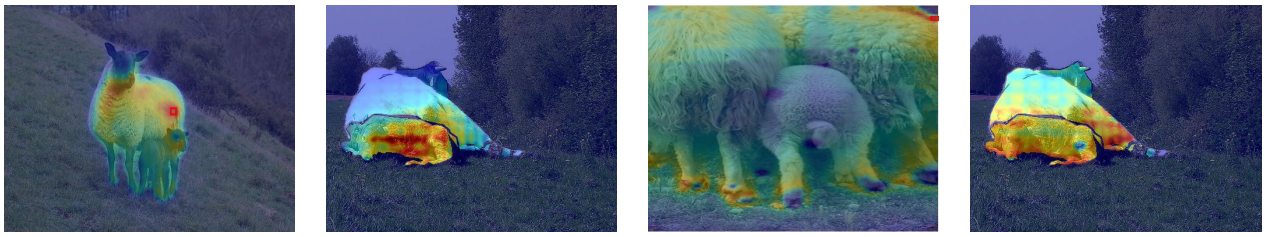


Figure 33. Most activated prototypes represented by their training sample and their activation on the input image for Group 1 of class *sheep*.

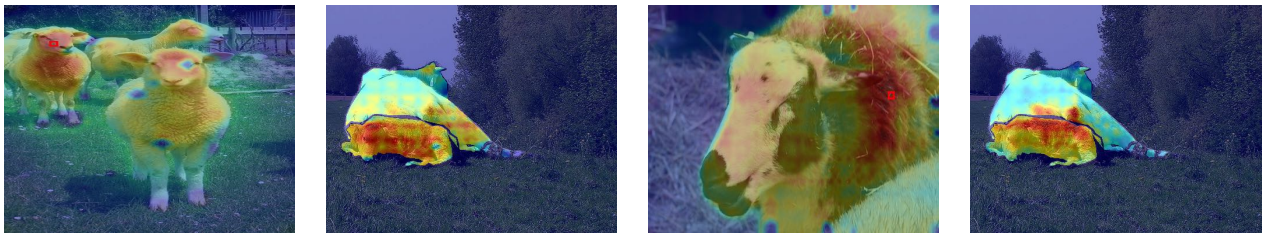


Figure 34. Most activated prototypes represented by their training sample and their activation on the input image for Group 2 of class *sheep*.

Figure 35. Most activated prototypes represented by their training sample and their activation on the input image for Group 3 of class *sheep*.