Doubly Deformable Aggregation of Covariance Matrices for Few-shot Segmentation

Zhitong Xiong [1], Haopeng Li [2], and Xiao Xiang Zhu [1,3]

[1] Data Science in Earth Observation, Technical University of Munich (TUM)
[2] School of Computing and Information Systems, University of Melbourne
[3] Remote Sensing Technology Institute (IMF), German Aerospace Center (DLR)

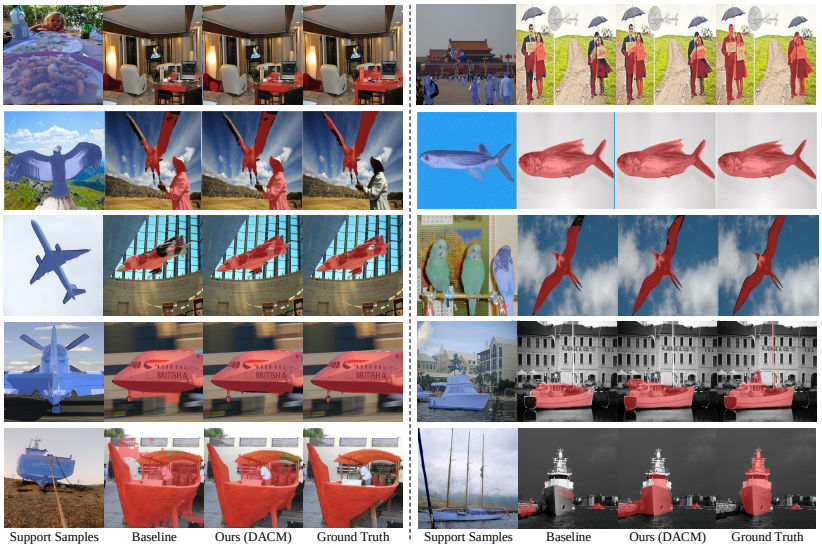

Fig. 1. Visualization of the few-shot segmentation results on PASCAL-5^i, COCO-20^i [2], and FSS-1000 [1] datasets. From left to right: support samples, results of the baseline method, results of the DACM, and ground truth labels. It can be clearly seen that our method can output better results than the baseline method.

1 Few-shot Segmentation Visualization

Three datasets for few-shot segmentation are exploited to evaluate the proposed method, including the PASCAL-5^i [4], COCO-20^i [2], and FSS-1000 [1]. As we can see in Fig. 1, there exist large variations between the support images and the query images in scale, pose and appearance.

Although only 50 epochs are used for training the proposed DACM method, the performance of our model is clearly better than that of the baseline method for some difficult test samples. Compared with HSNet [3], qualitative results in

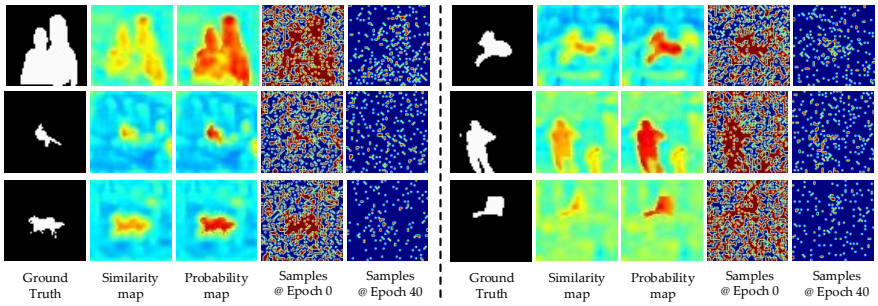

Fig. 2. Visualization of some qualitative examples of the proposed hard example-aware sampling strategy for training GP models of the DACM model.

Fig. 1 show that DACM can obtain more complete segmentation maps owing to larger receptive fields of the Transformer architecture. In Fig. 2, we also visualize more examples of sampled locations during the training process of our DACM method.

2 Covariance Matrices Analysis

Different from HSNet [3], we use the learned covariance kernel functions to measure the similarity between the support features and the query features instead of a fixed cosine similarity. The fixed cosine similarity is computed as follows:

$$\boldsymbol{C}_1(\boldsymbol{x}_q, \boldsymbol{z}_s) = \text{ReLU}\left(\frac{\boldsymbol{x}_q^{\text{T}} \boldsymbol{z}_s}{||\boldsymbol{x}_q|| ||\boldsymbol{z}_s||}\right), \tag{1}$$

where $\boldsymbol{x}_q$ is the query feature, and $\boldsymbol{z}_s$ is the masked support feature.

Theoretically, the cosine similarity used in [3] can be viewed as a special case of our method, *i.e.*, using a fixed linear kernel. For a more effective similarity measurement, we target at learning the kernel functions $k(\cdot, \cdot)$ on different datasets using Gaussian process. The covariance matrices can be computed as follows:

$$\boldsymbol{C}_2(\boldsymbol{x}_q, \boldsymbol{z}_s) = \text{ReLU}\left(k\left(\frac{\boldsymbol{x}_q}{||\boldsymbol{x}_q||}, \frac{\boldsymbol{z}_s}{||\boldsymbol{z}_s||}\right)\right). \tag{2}$$

To better understand the learned kernel functions in GP, we have visualized the covariance matrices in Fig. 3. For the cosine similarity map $\boldsymbol{C}_1 \in \mathbb{R}^{H_q \times W_q \times H_s \times W_s}$, we reshape it into $\boldsymbol{C}_1 \in \mathbb{R}^{H_q \times W_q \times H_s W_s}$ and sum it up along the third dimension. Then we can get 2D similarity maps of the query image. The 2D similarity maps are shown in Fig. 3. For the covariance matrices $\boldsymbol{C}_2 \in \mathbb{R}^{H_q \times W_q \times H_s \times W_s}$, we conduct the similar computation as the cosine similarity map $\boldsymbol{C}_1$.

From Fig. 3 we can see that covariance matrices learned by the DACM model are more reasonable than the simple cosine similarity map. Then, the learned similarity maps can be processed by the proposed DDT model to further enhance the performance of few-shot segmentation.

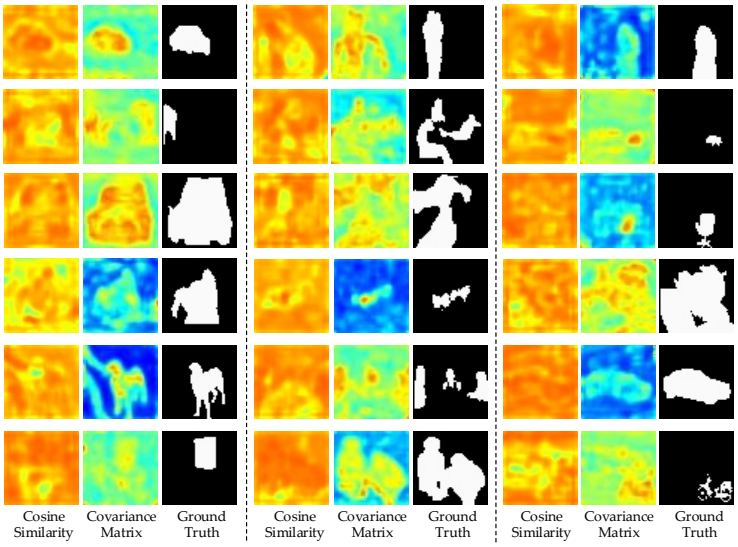

Fig. 3. Visualization of the learned variance matrices. For a clear visualization, we pool the matrices into 2D similarity map along the support dimension. It can be seen that similarity maps of DACM are more consistent with the ground truth label.

3 Limitations and Failure Cases

In this subsection, we will visualize and analyze some failure cases of the proposed DACM method. In Fig. 4, we can see some failure cases of the learned covariance matrices. It can be seen that, the similarity between the support and the background of query samples is clearly low. However, in some cases, the similarity between the support and foreground objects of query samples is incorrectly high. From the visualization results, we can find that although the global representation learning of Transformer is beneficial in most cases, there still exist some situations where smaller receptive fields work better.

In Fig. 5, we also visualize some segmentation failures. As for the top-left image, the man holding the bottle is misclassified as a foreground object. Take the top-right result as an example, DACM incorrectly classify the reflection of the boat as a foreground object. Note that the baseline method obtains better mIoU results on these visualized samples than DACM. In the future work, we believe that studying how to effectively combine global and local representations can further enhance the performance of the few-shot segmentation task.

4 Details of Gaussian Process

Gaussian process (GP) is a non-linear and non-parametric Bayesian model for regression and classification [5]. By exploiting the correlation between data sam-

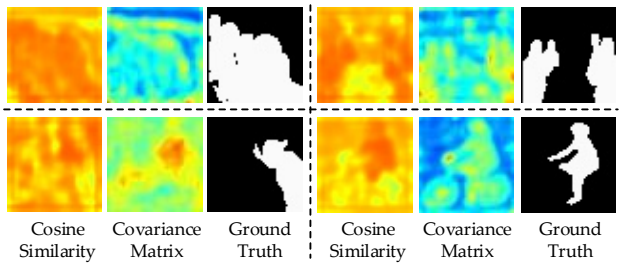

Fig. 4. Visualization of some failure cases of the learned covariance matrices. Compared with the cosine similarity, it can be seen that the learned covariance matrices fail to separate the foreground objects and the background.

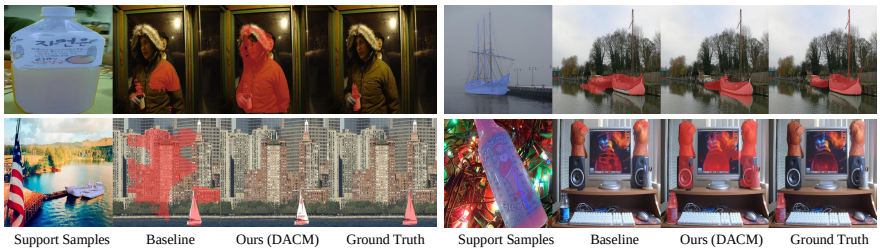

Fig. 5. Visualization of some failure cases of 1-shot segmentation results.

ples, Gaussian process performs probabilistic non-linear prediction that is described by Gaussian distribution with estimated mean and covariance. We explain the details of general GP as follows. Formally, the data set consists of N samples of dimension D, *i.e.*, $\{X = \{x_i\}_{i=1}^N, \boldsymbol{y} = \{y_i\}_{i=1}^N\}$, where $x_i \in \mathbb{R}^D$ is a data point and y_i is the corresponding label. The GP regression model assumes that the outputs y_i can be regarded as certain deterministic latent function $f(x_i)$ with zero-mean Gaussian noise ε, *i.e.*, $y_i = f(x_i) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. GP sets a zero-mean prior on f, with covariance $k(x_i, x_j)$. The covariance function (kernel) k reflects the smoothness of f. Commonly-used kernels can be found in Section 5. The hyper-parameters in the kernels are optimized by maximizing the marginal likelihood of the training data, which is given as follows,

$$p(\boldsymbol{y}|X) = \int p(\boldsymbol{y}|\boldsymbol{f}, X)p(\boldsymbol{f}|X)\mathrm{d}\boldsymbol{f}, \tag{3}$$

where $\boldsymbol{f} = [f(x_1), f(x_2), \cdots, f(x_N)]^{\mathrm{T}}$. Here, the term marginal likelihood refers to the marginalization over the function values $\boldsymbol{f}$. Setting the prior of the Gaussian process model $p(\boldsymbol{f}|X)$ to be a Gaussian distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{K})$, the marginal log-likelihood $\log p(\boldsymbol{y}|X)$ can be expressed by,

$$\log p(\boldsymbol{y}|X) = -\frac{1}{2}\log|\boldsymbol{K}| - \frac{1}{2}\boldsymbol{y}^{\mathrm{T}}(\boldsymbol{K} + \sigma^2\boldsymbol{I}_N)^{-1}\boldsymbol{y} - \frac{N}{2}\log 2\pi, \tag{4}$$

where $\boldsymbol{y} = [y_1, y_2, \cdots, y_N]^{\mathrm{T}}$, $\boldsymbol{K}_{ij} = k(x_i, x_j)$, and $\boldsymbol{I}_N$ is the identity matrix of size N. For example, the Automatic Relevance Determination Squared Exponential (ARD SE) kernel function is defined as

$$k(x_i, x_j) = \sigma_0^2 \exp\left\{-\frac{1}{2}\sum_{d=1}^{D}\frac{((x_i)_d - (x_j)_d)^2}{l_d^2}\right\}, \tag{5}$$

where $\sigma^2, \sigma_0^2, \{l_d\}_{d=1}^{D}$ are hyper-parameters. By maximizing Eq. 4, the hyper-parameters in Eq. 5 for computing the covariance matrices can be optimized.

After the hyper-parameters are optimized, the predictive distribution for the test case x^* can be calculated in a closed-form as follows:

$$\mu_{y^*} = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{K} + \sigma^2\boldsymbol{I}_N)^{-1}\boldsymbol{y}, \tag{6}$$

$$\sigma_{y^*}^2 = k^* - \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{K} + \sigma^2\boldsymbol{I}_N)^{-1}\boldsymbol{k} + \sigma^2, \tag{7}$$

where $\boldsymbol{k} = [k(x_1, x^*), k(x_2, x^*), \cdots, k(x_N, x^*)]^{\mathrm{T}}$ and $k^* = k(x^*, x^*)$.

Algorithm 1: Pseudocode for implementation of the Gaussian process

```
# Gaussian process implementation details
class ExactGPModel(models.ExactGP):
    def init(self, kernel='rbf'):
        self.mean_module = means.ConstantMean() # using constant mean
        # Different Kernels
        if kernel == 'linear':
            # Linear kernel is equivalent to cosine similarity.
            self.linear_kernel = kernels.LinearKernel()
        if kernel == 'rbf':
            # RBF kernel with automatic relevance determination.
            self.rbf_kernel = kernels.RBFKernel(ARD)
        if kernel == 'additive':
            # Additive kernel by summing over multiple kernels.
            self.covar_module = kernels.LinearKernel() +
             kernels.RBFKernel(ARD)
    def forward(self, x):
        # Mean function computation
        mean_x = self.mean_module(x)
        # Covariance function computation
        covar_x = self.covar_module(x)
        return distributions.MultivariateNormal(mean_x, covar_x)
```

5 Gaussian Process Implementation

In this section, we present the Pytorch-style implementation of the GP model. The most critical part of the GP model is the choice of covariance kernel func-

tions. In algorithm 1, we show three types of kernel functions, including the linear kernel, the radial basis function (RBF) kernel and the additive mixed kernel.

For two vectors $\mathbf{x_1}, \mathbf{x_2}$ in the feature map, the linear kernel, the RBF kernel, and the additive mixed kernel, with hyperparameters $\{v, \Theta\}$, are defined as follows:

$$\begin{aligned} k_{\text{Linear}}\ (\mathbf{x_1}, \mathbf{x_2}) &= v\mathbf{x_1}^{\top}\mathbf{x_2}, \\ k_{\text{RBF}}\left(\mathbf{x_1}, \mathbf{x_2}\right) &= \exp\left(-\frac{1}{2}\left(\mathbf{x_1}-\mathbf{x_2}\right)^{\top}\Theta^{-2}\left(\mathbf{x_1}-\mathbf{x_2}\right)\right), \\ k_{\text{Additive}}\ (\mathbf{x_1}, \mathbf{x_2}) &= k_{\text{Linear}}\ (\mathbf{x_1}, \mathbf{x_2}) + k_{\text{RBF}}\left(\mathbf{x_1}, \mathbf{x_2}\right). \end{aligned} \tag{8}$$

Although the RBF kernel is used in this work, here we show that the proposed framework can be easily extended by using different kernel types or their combinations.

References

1. Li, X., Wei, T., Chen, Y.P., Tai, Y.W., Tang, C.K.: Fss-1000: A 1000-class dataset for few-shot segmentation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2869–2878 (2020)
2. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: European conference on computer vision. pp. 740–755. Springer (2014)
3. Min, J., Kang, D., Cho, M.: Hypercorrelation squeeze for few-shot segmentation. arXiv preprint arXiv:2104.01538 (2021)
4. Shaban, A., Bansal, S., Liu, Z., Essa, I., Boots, B.: One-shot learning for semantic segmentation. arXiv preprint arXiv:1709.03410 (2017)
5. Williams, C.K., Rasmussen, C.E.: Gaussian processes for regression (1996)