Supplementary Materials

Paper ID 797

May 14, 2017

1 Weight Hashing

In our Proposed Work, section 5 of the paper, we employed the weight hashing technique from [1] to map the 1000-dimensional vector output from the last layer of VGG to the 4097 dimensions of $\{w, b\}$. This mapping (1) reduces the variance of $\{w, b\}$ as was also noted by Huh *et al.* in [2], and (2) reduces the overfitting which would occur due to the massive number of extra parameters that a fully connected layer will introduce if used instead.

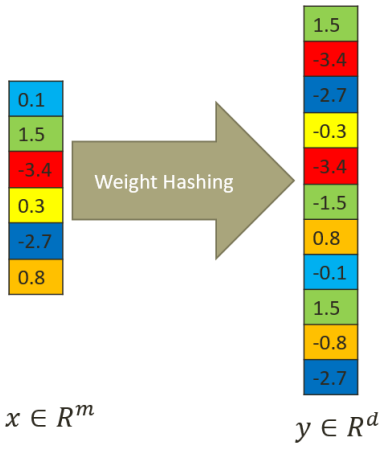

Figure 1: Illustration of weight hashing. In the figure, x is mapped to y by replicating coefficients of x in multiple random locations of y and randomly flipping the sign. The colors help indicate where the entries are copied from.

Weight hashing is explained as decompression in [1] and is performed as follows. Let $x \in \mathbb{R}^m$ and $\theta \in \mathbb{R}^d$, typically $d > m$, be the inputs and outputs of the layer respectively. Weight hashing works by replicating each coefficient of x in multiple locations of θ and randomly flipping the sign to reduce the covariance of copied coefficients. Illustrated in Figure 1. Specifically, the i^{th} coefficient of θ is

$$\theta(i) = x(p)\zeta(i), \tag{1}$$

$$p = \kappa(i), \tag{2}$$

where both $\kappa(i) \rightarrow \{1, \ldots, m\}$ and $\zeta(i) \rightarrow \{-1, +1\}$ are hashing functions determined randomly. While [1] perform the hashing implicitly to keep the memory footprint small, we implement it as a fully connected layer since we can keep both the hashing values and θ in memory. The weights are set as

$$W(i, j) = \zeta(i)\delta_j(\kappa(i)), \tag{3}$$

where $\delta_j(\cdot)$ is discreet Dirac delta function. The weights are set according to the random hashing functions before training and are kept fixed. This is both easier to implement and more computationally efficient than the original formulation and that used by [2]. The output of the inner product layer Wx is equal to θ from Equation 1.

2 Siamese Network for Dense Matching

In the paper, we used the adapted version of Siamese Neural Network for One-shot Image Recognition by Koch *et al.* [3] for one-shot image segmentation. Here we explain the implementation details. The method from [3] receives as input two images that are each passed through identical convolutional networks and produce a vector as the output for each of them. These vectors are then compared using a learned $L1$ similarity metric and the image is classified according to the label of its nearest neighbor in this metric space. In our case, we use an FCN that outputs a feature volume each for both query and support images. Then the feature for every pixel in the query image is compared to every pixel in the support using a learned $L1$ similarity metric. We implemented this cross similarity measurement between pixels as a python layer for Caffe. The binary label here is assigned to each pixel according to the label of the nearest pixel label in the support set. The whole structure is illustrated in Figure 2.

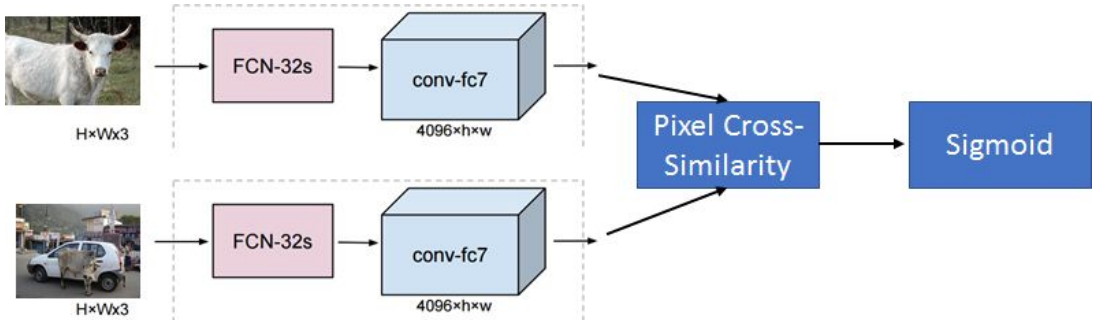

Figure 2: Siamese network architeture for dense matching.

During training, we use FCNs initialized on ImageNet [4] and at each iteration we sample a pair of images from the PASCAL-5^i training set. One of them is treated as the query image and the other becomes the support image. The gradient is computed according to the cross-entropy loss between the sigmoid of the similarity metric and the true binary label. Every batch contains a subset (50%) of the pixels of a query and a support image. Both the similarity metric and the FCN feature extraction are jointly as different parts of the same neural network.

3 Qualitative Results

We include some more qualitative results of our approach for One Shot Semantic Segmentation in Figure 3. We see that our method is capable of segmenting a variety of classes well and can distinguish an object from others in the scene given only a single support image.

We illustrate the effect of conditioning by segmenting the same query image with different support sets in Figure 4. We picked an unseen query image with two unseen classes, car and cow, and sample support image-mask pairs for each class. Figure 5 shows how increasing size of the support set helps improve the predicted mask. Note that in both Figures 5 and 4 yellow indicates the overlap between ground truth and the prediction.

References

[1] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick." in *ICML*, 2015, pp. 2285–2294.

[2] H. Noh, P. Hongsuck Seo, and B. Han, "Image question answering using convolutional neural network with dynamic parameter prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 30–38.

[3] G. Koch, "Siamese neural networks for one-shot image recognition," Ph.D. dissertation, University of Toronto, 2015.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

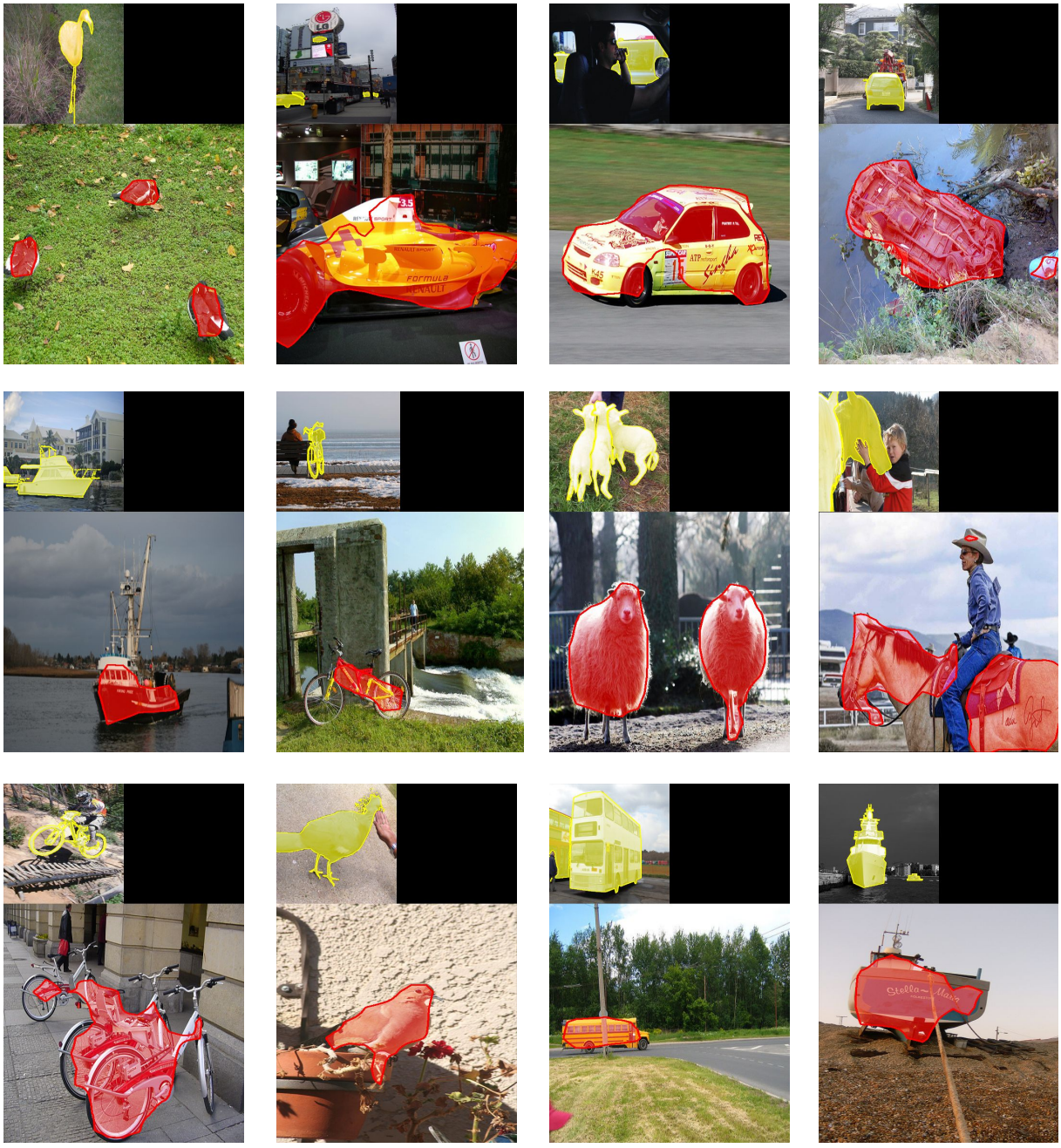

Figure 3: Qualitative results for 1-shot. Inside each tile, we have the support set at the top and the query image at the bottom. The support is overlaid with the ground truth in yellow and the query is overlaid with our predicted mask in red.

Figure 4: Illustration of conditioning effect. Given a fix query image, predicted mask changes by changing the support set. Ground-truth mask is shown in green. First row: support image-mask pairs are sampled from cow class. Second row: support image-mask pairs are sampled from car class. First column: only changing the support mask will will change the prediction.

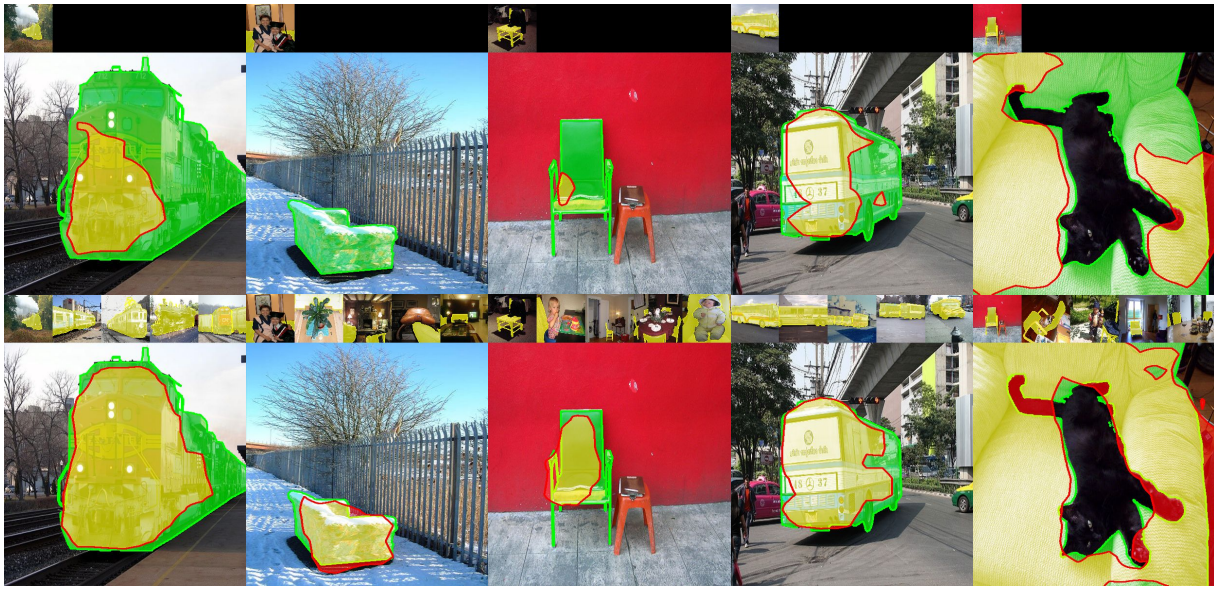

Figure 5: Effect of increasing the size of the support set. Results of 1-shot and 5-shot learning on the same query image are in the first and second rows respectively. Ground truth masks are shown in green and our prediction is in red. The overlap between ground truth and prediction appears yellow.