Universal Semi-Supervised Semantic Segmentation
Supplementary Material

Tarun Kalluri[1] Girish Varma[1] Manmohan Chandraker[2] C V Jawahar[1]

[1]Center for Visual Information Technology, IIIT Hyderabad
[2]University of California, San Diego

tarun.05.kalluri@gmail.com

A. Training Details

We give details of the parameters used for training the universal segmentation models in various settings. We use the openly available PyTorch implementation of dilated residual network [4], with encoders designed using ResNet-18 (*drn-d-22*), ResNet-50 (*drn-d-54*) as well as ResNet-101 (*drn-d-105*) architectures. The decoder consists of a 1x1 convolution layer followed by a bilinear upsampling layer. We train every model on 2 Nvidia GeForce GTX 1080 GPUs for 200 epochs. During training, we use a crop size of 512x512 for Cityscapes and IDD datasets, 360x480 for the Camvid dataset and 480x640 for the SUN-RGB dataset. Validation mIoUs are reported on the standard resolutions from the dataset. We employ SGD learning algorithm with an initial learning rate of 0.001 and a momentum of 0.9, along with a poly learning rate schedule with a power of 0.9 [1]. We use a batch size of 10, and take the embedding dimension to be 128. The default values for α and β are taken to be 1.

B. Label Embeddings

B.1. Calculating the label embeddings

In this section, we describe the method used to obtain the vector representations for the labels. For each dataset separately, we train an end-to-end segmentation network from scratch using only the limited training data available in that dataset. We use this trained segmentation network to calculate the encoder outputs of the training data at each pixel. Typically, the size of the output dimension of the encoder at each pixel (512 for a ResNet encoder) is not equal to the dimension of the label embeddings (d=128, in our case). So we first apply a dimensionality reduction technique like PCA to reduce the dimension of the outputs to match the dimension of the label embeddings d, and then calculate the class wise centroids to obtain the label embeddings.

Method	N=50			N=100		
	CS	CVD	Avg.	CS	CVD	Avg.
Ours[$\theta = 0.5$]	33.28	48.7	40.99	33.51	49.49	41.50
Ours[Word2Vec]	33.48	53.19	43.34	36.18	52.72	44.45
Ours[K=1]	34.01	53.23	43.62	41.03	54.62	47.83
Ours[K=3]	35.23	52.38	**43.81**	41.82	54.96	**48.39**
Ours[K=5]	33.76	52.77	43.27	40.08	55.02	47.55
Direct SER	21.36	35.24	28.30	23.7	30.67	27.19

Table 1: Extension of Table 3 from the original paper. θ is the update factor during training, and the default value is 1. K is the number of embeddings per label. *Ours[Word2Vec]* uses word vectors as label embeddings. The model gives best performance for K=3, $\theta = 1$ while using prototype embeddings.

B.2. Updating the label embeddings

In our original experiments, we fixed the pretrained label embeddings over the phase of training the universal model. Here, we present a method to jointly train the segmentation model as well as the label embeddings. We initialize the embeddings with the values computed from the pretrained networks, and make use of the following exponentially weighted average rule to update the centroids at the t^{th} time step.

$$\mathrm{c}_t^{(k)} = \theta \cdot \mathrm{c}_{t-1}^{(k)} + (1-\theta) \cdot \mu_L(\mathcal{F}_t(x_u^{(k)})). \quad (1)$$

In Eq (1), $\mathrm{c}_{t-1}^{(k)}$ denotes the centroids at the $(t-1)^{\text{th}}$ time step, $\mathcal{F}_t$ is the state of the encoder module at the t^{th} time step and μ_L calculates the class wise centroids. θ is the update factor, where a value of $\theta = 1$ implies that the centroids are not updated from their initial state, and a value of $\theta = 0$ means that the centroids are calculated afresh at each update. We make an update to the centroids after every mini-batch of the original training data.

From Table 1, experiments with $\theta = 0.5$ suggests that jointly training the network as well as updating the label embeddings reduces the performance compared to having fixed label embeddings. We believe that this is primarily

due to having insufficient training data for jointly updating embeddings as well as the network weights, although this merits a deeper investigation.

B.3. Multiple Label Embeddings

Many labels in a segmentation dataset often appear in more than one visual form or modalities. For example, *road* class can appear as dry road, wet road, shady road etc., or a class labeled as *building* can come in different structures and sizes. To better capture the multiple modalities involved in the visual information of the label, we propose using multiple embeddings for each label instead of a single mean centroid. This is analogous to polysemy in vocabulary, where many words can have multiple meanings and can occur in different contexts, and context specific word vector representations are used to capture this behavior. To calculate the multiple label embeddings, we perform K-means clustering of the pixel level encoder feature representations calculated from networks pretrained on the limited supervised data, and calculate similarity scores with all the multiple label embeddings.

Table 1 shows that using K=3 embeddings per label gives an advantage over using 1 embedding per label, so apparently some amount of over segmentation helps. However, further increasing K to 5 hurts the performance, as not all the labels benefit from having multiple modalities per label. So, an interesting future direction can be to examine optimum number of embeddings per label.

Particularly, from Table 2, it is evident that classes like *Road*, *Building*, *Person* etc. benefit largely from having multiple embeddings per label.

B.4. Choice of label embeddings

In our work, we chose the pixel level class prototypes to be the label embeddings. We believe that this helps in better capturing visual information from the images compared to other approaches like Word2Vec [2]. To this end, we provide results of our approach replacing the prototype label embeddings with word vectors of the labels, by using the publicly available 128 dimensional word vectors for the labels from the Cityscapes and CamVid datasets.

From Table 1, having class prototypes as label embeddings, which are computed from the labeled data, performs better than using Word2vec based embeddings, which capture semantics of the word meaning rather than the visual appearance of the label. The performance improvement is more evident in case of N=100, which demonstrates that in presence of sufficient labeled data, class prototypes are better suited as label embeddings than word vector representations. Similar observations have been made in [3] as well.

C. Direct Softmax Entropy Regularization

Entropy regularization is used to enhance the confidence of predictions made on unlabeled samples. In the case of deep neural networks, applying this directly to the softmax scores will make the predictions confident by simply increasing the weights of the last layer. So, we follow an approach where we calculate similarity between normalized label prototypes and encoder embeddings through our entropy module. *Direct SER* result from Table 1 further demonstrates the fact that applying SER (softmax entropy regularization) directly to our network shows inferior performance compared to the proposed entropy module based approach.

D. Qualitative examples

From the qualitative examples presented in Figure 1 for Cityscapes and Figure 2 for IDD, we can observe that the entropy module is particularly useful in better segmentation of finer objects and foreground classes.

References

[1] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018. 1

[2] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. 2

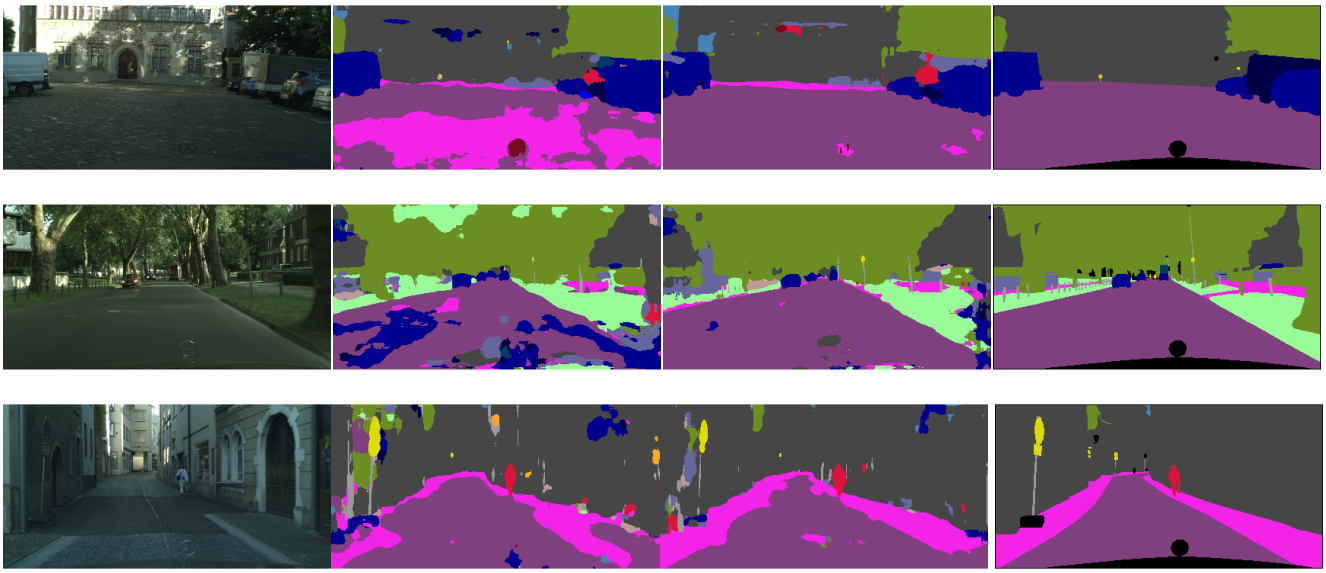
[3] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017. 2

[4] F. Yu, V. Koltun, and T. A. Funkhouser. Dilated residual networks. In *CVPR*, volume 2, page 3, 2017. 1

Method	Road	SideWalk	Building	Wall	Fence	Pole	Traffic light	Traffic Sign	Vegetation	Train	Sky	Person	Rider	Car	Truck	Bus	Train	MotorCycle	Bicycle	mIoU
K=1	92.18	51.29	80.07	00.00	**24.01**	**33.73**	**26.16**	**38.71**	82.30	36.39	**81.61**	54.38	20.48	**81.71**	02.37	**22.79**	03.85	01.31	46.23	41.03
K=3	**93.10**	**56.82**	80.48	00.03	17.84	32.49	24.07	33.51	**82.52**	**38.52**	80.12	53.22	15.35	81.34	**07.79**	20.79	04.18	**22.57**	**49.90**	**41.82**
K=5	89.58	48.50	**81.21**	**14.55**	07.89	27.77	22.72	33.95	84.36	34.33	80.52	**54.39**	**22.51**	81.52	02.41	09.43	**07.28**	10.40	48.18	40.08

Method	Sky	Buil.	Pole	Road	Pave.	Tree	Sign	Fence	Car	Ped.	Bicy.	mIoU
K=1	86.3	77.23	**17.13**	84.99	53.35	70.57	31.99	32.45	72.94	36.61	**37.22**	54.61
K=3	**87.67**	**78.51**	16.37	84.84	53.18	**73.33**	**34.02**	27.71	74.42	**40.36**	34.24	54.96
K=5	86.07	76.39	15.25	**87.87**	**63.6**	70.95	32.6	**34.6**	**76.63**	30.42	30.9	**55.02**

Table 2: Class-wise IoU values for the 19 classes in Cityscapes dataset and 11 classes in the CamVid dataset for different K, for N=100 on Resnet-18.



(a) Original Image (b) Without Entropy Module (c) With Entropy Module (d) Ground Truth Segmentation

Figure 1: Qualitative examples from the cityscapes dataset with and without the proposed entropy regularization module, when trained on a universal model on CS+CamVid.