Refining Semantic Segmentation with Superpixels using Transparent Initialization and Sparse Encoder

Zhiwei Xu[1,2]   Thalaiyasingam Ajanthan[1]   Richard Hartley[1]

[1]Australian National University and Australian Centre for Robotic Vision

[2]Data61, CSIRO, Australia

{zhiwei.xu,thalaiyasingam.ajanthan,richard.hartley}@anu.edu.au

Figure 1: *Edge comparison with the state-of-the-art, that is ResNeSt200 on ADE20K (top row) and ResNeSt269 on PASCAL Context (bottom row).* **Left: RGB, middle: state-of-the-art, right: ours.** *Ours has a better alignment with object edges than the state-of-the-art.*

Abstract

Although deep learning greatly improves the performance of semantic segmentation, its success mainly lies in object central areas without accurate edges. As superpixels are a popular and effective auxiliary to preserve object edges, in this paper, we jointly learn semantic segmentation with trainable superpixels. We achieve it with fully-connected layers with Transparent Initialization (TI) and efficient logit consistency using a sparse encoder. The proposed TI preserves the effects of learned parameters of pretrained networks. This avoids a significant increase of the loss of pretrained networks, which otherwise may be caused by inappropriate parameter initialization of the additional layers. Meanwhile, consistent pixel labels in each superpixel are guaranteed by logit consistency. The sparse encoder with sparse matrix operations substantially reduces both the memory requirement and the computational complexity. We demonstrated the superiority of TI over other parameter initialization methods and tested its numerical stability. The effectiveness of our proposal was validated on PASCAL VOC 2012, ADE20K, and PASCAL Context showing enhanced semantic segmentation edges. With quantitative evaluations on segmentation edges using performance ratio and F-measure, our method outperforms the state-of-the-art.

1. Introduction

Semantic segmentation is an essential task in computer vision which requires mapping image pixels of interesting objects to predefined semantic labels. Applications include autonomous driving [14], object identification [36, 38], image editing [22], and scene analysis [21]. Recent developments of semantic segmentation are greatly promoted by deep learning on several large-scale datasets [35, 32], resulting in effective networks [34, 55, 58, 8, 7, 9, 53, 54]. Semantic segmentation obtained by these methods, however, is not substantially aligned with object edges.

This problem can be alleviated by using qualified edge-preserving methods [28, 58, 8] or independently learning edges for semantic segmentation [25, 17, 7]. Nevertheless, those edges are usually incomplete and oversegmented. To solve this problem, denseCRF [28] based methods aggregate object features over a large and dense range using bilateral filters with high-dimensional lattice computations. This, however, could be more efficient and desirable to learn on locally oversegmented areas, such as superpixels that aggregate similar pixels into a higher-order clique [1, 30].

Existing superpixel segmentation approaches are mainly

categorized into traditional methods, such as SLIC [1], LSC [30], Crisp [24], BASS [47], and those using neural networks, such as SSN [25], affinity loss [46], and superpixels by FCN [52]. Although those traditional methods have qualified performance on superpixel segmentation, they cannot be easily embedded into neural networks for end-to-end learning due to nondifferentiability or large computational complexity. In contrast, [25] provides an end-to-end learning for superpixel semantic segmentation, but the pixel labels in each superpixel are not always consistent. Similarly, [52] uses a fully convolutional network to learn superpixels for stereo matching, which is the current state-of-the-art in superpixel learning. However, the problem of inconsistent pixel labels in each superpixel will also occur when [52] is applied to semantic segmentation.

To deal with the problems of edge loss and inconsistent superpixel labelling, we jointly train these networks with additional fully-connected layers using transparent initialization and logit consistency, resulting in enhanced object edges in Fig. 1. Specifically, transparent initialization warm starts the training by recovering the pretrained network output from its input at initialization, followed by a gradual improvement in learning superpixels. Simultaneously, logit consistency with sparse encoder enables efficient logit averaging to guarantee consistent pixel labels in each superpixel. Furthermore, we used the popular performance ratio [27] and F-measure [42, 15] to evaluate the quality of semantic segmentation edges. Our code will be available upon publication. The main contributions of this work are:

- We jointly learn the state-of-the-art semantic segmentation network and superpixel network to enhance the labelling performance with sharp object edges. The improvement is vivid as demonstrated by evaluating on PASCAL VOC 2012, ADE20K and PASCAL Context.

- Transparent initialization is proposed for learning fused features while adding fully-connected layers to pretrained networks. This helps to join and fine-tune state-of-the-art networks without interrupting the effect of pretrained network parameters. The transparent initialization may also be adapted to fine-tune multiple and deeper pretrained networks for other tasks.

- Logit consistency, implemented by a sparse encoder with sparse matrix operations, ensures consistent semantics for all pixels in each superpixel. This makes it feasible and efficient to index pixels by superpixels, because of greatly reduced computational complexity.

2. Related Work

Semantic Segmentation. Semantic segmentation can be traced back to early techniques [45] based on classifiers, such as random decision forests [39], SVMs [49], and graphic models with MRFs and CRFs [26, 28, 3]. In contrast, modern state-of-the-art methods rely on advanced exploitation of deep CNN classifiers, such as ResNet [20], DenseNet [23], and VGG16 [41]. Fully Convolutional Network (FCN) [34] and related methods are typical architectures that use rich image features from classifiers usually pretrained on ImageNet [12]. SegNet [5] uses a U-Net structure for an encoder-decoder module to compensate for low resolution by using multiple upsampled feature maps. In addition, several multiscale contextual fusion methods [7, 31, 9] have been proposed to aggregate pyramid feature maps for fine-grained segmentation. Typically, DeepLabV3+ [9] combines spatial pyramid pooling and encoder-decoder modules to refine the segmentation along with object boundaries. Recently, attention-based networks [29, 56, 11, 16] improve object labelling confidence by aggregating features of a single pixel with those from other positions. Zhang *et al.* [54] introduce a split-attention block in ResNet (ResNeSt), which enables multiscale scores with softmax for attention across feature-map groups. This achieves the new state-of-the-art performance in semantic segmentation and image classification. Due to the limited capability of network architectures, however, a huge improvement on mean Intersection over Union (mIoU) is hard to achieve; see the minor improvement in [57].

Superpixel Segmentation. Superpixel segmentation has been well studied for years with a comprehensive survey in [43]. In contrast to classical methods that initialize superpixel regions with seeds and cluster pixel sets using distance measurement [48], boundary pixel exchange [6], etc., the widely-used SLIC based methods [1, 33, 30, 2] employ (weighted) K-means clustering on pixel feature vectors to group neighbouring pixels. In deep learning, SSN [25] first proposes an end-to-end learning framework for superpixels with differentiable SLIC for semantic segmentation and optical flow. By comparison, [52] replaces the soft K-means manner in SSN with a simple fully convolutional network and applies it to stereo matching with downsampling and upsampling modules. While SSN results in superpixel-level semantic segmentation, the pixel labelling is not aligned with the superpixels. For instance, inconsistent labels exist in each superpixel, which reduces the effects of superpixels on semantic assignments.

Superpixel Semantic Segmentation. Some works use superpixels to optimize graph relations [19, 50] or downsample images as a pooling alternative to max or average pooling [17, 37, 40]. These methods usually use fixed superpixels obtained by traditional methods mentioned above ("Superpixel Segmentation") and lose the exact alignment of segmentation edges with superpixel contours after upsampling. This, however, can be easily achieved by our logit consistency module. Moreover, fixed superpixels are unsuitable for end-to-end learning since traditional meth-
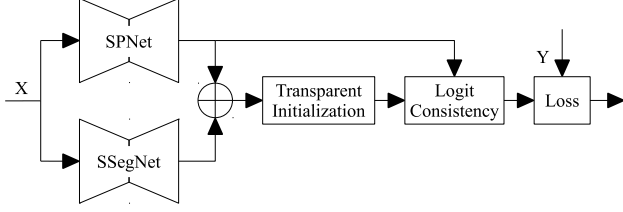
Figure 2: *Flowchart.* **X:** *input image,* **Y:** *semantic segmentation ground truth,* **SPNet:** *superpixel network,* **SSegNet:** *semantic segmentation network,* $\oplus$ *: concatenatation,* **Transparent Initialization:** *identity mapping by fully-connected layers,* **Logit Consistency:** *consistent pixel labels in each superpixel.*
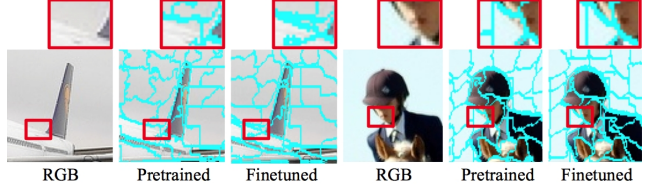

Figure 3: *Pretrained superpixles on BSDS500 [52] vs. fine-tuned superpixels on PASCAL VOC 2012 using our joint learning. Fine-tuned superpixels recover accurate object edges to alleviate the domain gap between datasets. Best view by zoom-in.*

ods, such as SLIC [1], are computationally expensive due to CPU execution and inflexible for fine-tuning superpixels around object edges, especially for small objects. Instead, superpixels learned by CNN alleviate this problem.

3. Methodology

The flowchart of our method is shown in Fig. 2. Subsequently, we discuss each of these modules in detail.

3.1. Network Architectures

We use the state-of-the-art DeepLabV3+ [9] and ResNeSt [54] for semantic segmentation and the state-of-the-art superpixels with FCN [52] for superpixel contours.

DeepLabV3+ achieves high and robust performance with atrous spatial pyramid pooling for multiscale feature maps and encoder-decoder modules for deep features with different output strides. It is widely used as a network backbone for semantic segmentation due to these well-studied modules evaluated by empirical experiments. ResNeSt [54] replaces ResNet with multiscale scores using softmax-based feature map attention and achieves the new state-of-the-art. The loss function for semantic segmentation is the standard cross-entropy loss of predicted logits and ground truth labelling. Readers can refer [9, 54] for more details.

Superpixel with FCN [52] is the current state-of-the-art superpixel network. The core idea is to construct a distance-based loss function with aggregations of neighbouring pixel and superpixel properties and locations. This is similar to the SLIC method [1], where the property vectors can be CIELAB colors or one-hot semantic encoding. The loss function for superpixel network with FCN [52] is in Eq. (1).

Given an image with N_p pixels and N_s superpixels, we denote the subsets of pixels as $\mathcal{P} = \{\mathcal{P}_0, ..., \mathcal{P}_{N_s-1}\}$, where $\mathcal{P}_i$ is a subset of pixels belonging to superpixel i. With pixel property $\mathbf{f} \in \mathbb{R}^{N_p \times K}$ (K features for each pixel) and probability map $\mathbf{q} \in \mathbb{R}^{N_s \times N_p}$, the loss function is

$$\mathcal{L}(\mathbf{f}, \mathbf{q}) = \sum_{p \in \mathcal{P}} E\left(\mathbf{f}(p), \mathbf{f}'(p)\right) + \frac{m}{D}||\mathbf{c}(p) - \mathbf{c}'(p)||_2 \quad (1)$$

with

$$\mathbf{u}_s = \frac{\sum_{p \in \mathcal{P}_s} \mathbf{f}(p) q_s(p)}{\sum_{p \in \mathcal{P}_s} q_s(p)}, \quad \mathbf{l}_s = \frac{\sum_{p \in \mathcal{P}_s} \mathbf{c}(p) q_s(p)}{\sum_{p \in \mathcal{P}_s} q_s(p)}, \quad (2a)$$

$$\mathbf{f}'(p) = \sum_{s \in \mathcal{N}_p} \mathbf{u}_s q_s(p), \quad \mathbf{c}'(p) = \sum_{s \in \mathcal{N}_p} \mathbf{l}_s q_s(p), \quad (2b)$$

where m is a weight balancing the effects of property and coordinates on loss, D is a superpixel sampling interval in proportion to superpixel size, $\mathbf{c}(i) = [x_i, y_i]^T$ for all $i \in \{1, ..., N_p\}$ are pixel coordinates, $E(\cdot, \cdot)$ is a distance measure function involving l_2 norm or cross-entropy, $q_s(p) \in \mathbf{q}$ is the probability of pixel p belonging to superpixel s, and $\mathcal{N}_p$ is a set of superpixels surrounding pixel p.

Here, $\mathbf{u}_s$ and $\mathbf{l}_s$ are superpixel-level property vector and central coordinates aggregated from involved pixels respectively, and $\mathbf{f}'$ and $\mathbf{c}'$ are pixel-level property and coordinates aggregated from surrounding superpixels. This updates between pixels and superpixels until the loss converges.

3.2. Learning with Transparent Initialization

Although some semantic segmentation datasets, such as PASCAL VOC [13] and Berkeley benchmark [35], have no accurate edges for supervised edge learning, it can be compensated for by a superpixel network on edge-specified datasets, such as SBDS500 [4]. Due to the domain gap of datasets, however, pretrained superpixel models are more desirable than learning from scratch for fast loss convergence. Superpixel maps pretrained on BSDS500, however, are not always suitable for semantic segmentation datasets. Hence, fine-tuning by joint learning is necessary to improve the quality of superpixel contours, compared with using the pretrained network, as shown in Fig. 3.

Therefore, in our proposal, linear layers or convolutional layers with 1×1 kernels are added to fuse the outputs of superpixel and semantic segmentation networks. Either or both can be pretrained. Importantly, selecting an appropriate initialization on these additional layers is important to avoid overriding the effect of learned network parameters. It is straightforward to cast the layer operations as an identity mapping between input and output at the early training stage. Net2Net [10] achieves this by using identity matrices

to initialize linear layers. This method, however, is inefficient in learning since the identity matrices will result in highly sparse gradients. In addition, it cannot handle activation functions with negative values.

In contrast, we introduce *transparent initialization* with non-zero values for dense gradients, while identically mapping the layer input to output and preserving the effect of the learned parameters of the pretrained networks.

Affine Layers. A linear layer without activation (such as a convolution or fully-connected layer) can be written as $\mathbf{y} = \mathbf{x}\mathbf{A} + \mathbf{b}$ with layer weights $\mathbf{A}$, bias $\mathbf{b}$, and input $\mathbf{x}$, by matrix multiplication with $\tilde{\mathbf{y}} = [\mathbf{y}, \mathbf{1}]$:

$$\tilde{\mathbf{y}} = \tilde{\mathbf{x}}M = [\mathbf{x}, \mathbf{1}] \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{b} & \mathbf{1} \end{bmatrix}. \qquad (3)$$

For simplicity the mapping Eq. (3) will be denoted as $\mathbf{y} = \mathbf{x}\,\mathtt{A}$ where $\mathtt{A}$ denotes the affine transformation, and we place the functions on the right. Note the difference between $\mathtt{A}$, an affine transform and $\mathbf{A}$, a matrix. If an activation function σ is included, then the mapping is $\mathbf{x} \mapsto \mathbf{x}\,\mathtt{A}\,\sigma$.

The right-inverse of an affine transformation Eq. (3) is represented by the matrix

$$M^R = \begin{bmatrix} \mathbf{A}^R & \mathbf{0} \\ -\mathbf{b}\mathbf{A}^R & \mathbf{1} \end{bmatrix}, \qquad (4)$$

satisfying $MM^R = I$, the identity map. Here, $\mathbf{A}^R$ is the right-inverse of matrix $\mathbf{A}$, which exists if $\mathbf{A}$ has dimension $m \times n$ and rank m, in which case $\mathbf{A}^R = \mathbf{A}^\top(\mathbf{A}\mathbf{A}^\top)^{-1}$.

Transparent Initialization. The idea behind transparent layer initialization is to construct a sequence of $k \geq 2$ affine layers denoted $\mathtt{A}_1, \mathtt{A}_2, \ldots, \mathtt{A}_k$ such that $\mathtt{A}_k$ is a right-inverse of the product $\mathtt{A}_1 \ldots \mathtt{A}_{k-1}$, satisfying $\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_k = \mathtt{I}$, the identity transformation.

A necessary condition for this right inverse to exist is that $\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_{k-1}$ should be of full rank. Since a matrix with random entries will almost surely have full rank, this leads to the following condition.

Theorem 1 *Let a sequence of affine transformations $\mathtt{A}_i : \mathbb{R}^{m_{i-1}} \to \mathbb{R}^{m_i}$, for $i = 1, \ldots, k-1$ be chosen at random. Then $\mathtt{A}_1 \ldots \mathtt{A}_{k-1}$ almost surely has a right inverse if and only if $m_i \geq m_0$ for $i = 1, \ldots, k-1$.*

Therefore, the strategy for selecting a set of parameters for a sequence of affine layers may be described as follows. For the sequence of layers to represent an identity transform, we need that the input and output space have the same dimension, namely $m_0 = m_k$. Then

1. Select intermediate dimensions $m_1, \ldots, m_{k-1}$ such that $m_i \geq m_0$ for all $i = 1, \ldots, k-1$.

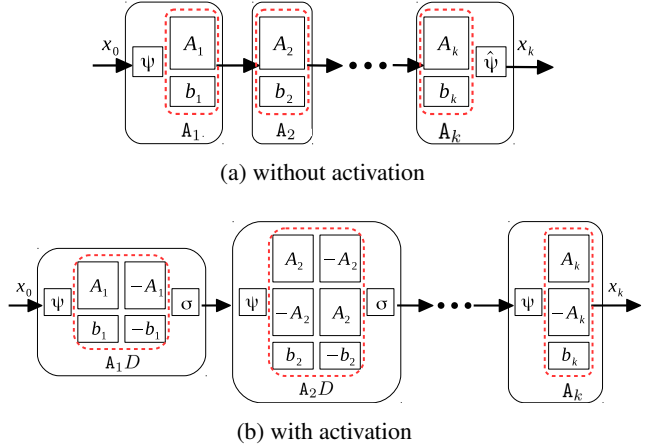(a) without activation

(b) with activation

Figure 4: *Transparent initialization.* $\psi : \mathbf{x} \mapsto [\mathbf{x}, \mathbf{1}]$; $\hat{\psi} : [\mathbf{x}, \mathbf{1}] \mapsto \mathbf{x}$; σ: *activation function. The three modules, from left to right, in (b) are corresponding to Eqs.* (8)-(10).

2. Define random affine transforms $\mathtt{A}_i$ by selecting the entries of matrices $\mathbf{A}_i$ and vectors $\mathbf{b}_i$ randomly, using a suitable random number generator, for instance by a zero-mean normal (Gaussian) distribution.

3. Compute the composition $\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_{k-1}$ by matrix multiplication, and take its right-inverse.

4. Set $\mathtt{A}_k$ to equal $(\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_{k-1})^R$.

The resulting composite mapping $\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_k$ is the identity affine transformation.

With Activation. Usually, each affine mapping $\mathtt{A}$ will be followed by an activation function σ. For now, we assume that this is $\mathrm{ReLU}(\cdot)$. Our approach including activations is to apply the activation to both $\mathbf{x}$ and $-\mathbf{x}$ and then sum them. Consider

$$\mathbf{x} \overset{D}{\mapsto} [\mathbf{x}, -\mathbf{x}] \overset{\sigma}{\mapsto} [\mathbf{x}^+, \mathbf{x}^-] \overset{S}{\mapsto} \mathbf{x}^+ - \mathbf{x}^- = \mathbf{x}, \quad (5)$$

where mappings D (duplicate) and S (subtract) are the mappings as shown above, and $\mathbf{x}^+$ and $\mathbf{x}^-$ are the positive and negative components of $\mathbf{x}$. This shows that $\mathbf{x}D\sigma S = \mathbf{x}$, and so $D\sigma S$ is the identity mapping. Note also that both D and S are affine (in fact linear) transformations.

Applying this to a sequence of affine transformations $\mathtt{A}_i$ such that $\mathbf{x}\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_k = \mathbf{x}$ gives

$$\mathbf{x}\,(\mathtt{A}_1 D\sigma S)\,(\mathtt{A}_2 D\sigma S)\,\ldots\,(\mathtt{A}_{k-1} D\sigma S)\,\mathtt{A}_k = \mathbf{x}. \quad (6)$$

Bracketing this differently gives

$$\mathbf{x}(\mathtt{A}_1 D\sigma)\,(S\mathtt{A}_2 D\sigma)\,\ldots\,(S\mathtt{A}_{k-1} D\sigma)\,(S\mathtt{A}_k) = \mathbf{x}, \quad (7)$$

where now each bracket is an affine mapping followed by the activation σ (except the last). This may then be implemented as a sequence of affine layers (convolution or fully-connected) with activations. Thus, $\mathtt{A}_1 D$ is the mapping

$$\mathbf{x} \mapsto \mathbf{x}\begin{bmatrix} \mathbf{A}_1 & -\mathbf{A}_1 \end{bmatrix} + [\mathbf{b}_1, -\mathbf{b}_1]. \quad (8)$$

4

		pixel index						
		0	1	2	3	4	$\cdots$	N_p-1
superpixel index	0	0	1	1	0	0	$\cdots$	0
	1	0	0	0	1	1	$\cdots$	0
	2	1	0	0	0	0	$\cdots$	1
	$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	N_s-1	0	0	0	0	0	$\cdots$	0
	sum	1	1	1	1	1	$\cdots$	1

Table 1: *Example of sparse property for indexing N_p pixels by N_s superpixels. Each superpixel contains only a few pixels ("1" in each row) for logit consistency. Hence, an efficient encoding is achieved by a sparse $N_s \times N_p$ matrix with N_p non-zero elements.*

The mapping $S\mathtt{A}_i D$ is the affine transformation

$$[\mathbf{x}^+, \mathbf{x}^-] \mapsto [\mathbf{x}^+, \mathbf{x}^-] \begin{bmatrix} \mathbf{A}_i & -\mathbf{A}_i \\ -\mathbf{A}_i & \mathbf{A}_i \end{bmatrix} + [\mathbf{b}_i, -\mathbf{b}_i] \quad (9)$$

and $S\mathtt{A}_k$ is the mapping

$$[\mathbf{x}^+, \mathbf{x}^-] \mapsto [\mathbf{x}^+, \mathbf{x}^-] \begin{bmatrix} \mathbf{A}_k \\ -\mathbf{A}_k \end{bmatrix} + \mathbf{b}_k . \quad (10)$$

The structure of the network may be represented as in Fig. 4. Observe that the outputs of intermediate layers have twice the dimension of the output of the affine mappings $\mathtt{A}_i$.

It is important to note that the structured form of the affine mappings in Eqs. (8)-(9) are for initialization only. There is no sharing parameters (such as $\mathtt{A}_i$ and $-\mathtt{A}_i$) and layers are free to implement any affine transform during training. In fact, $\mathbf{x}$ in any activation functions satisfying

$$\sigma(\mathbf{x}) - \sigma(-\mathbf{x}) = c\mathbf{x}, \quad (11)$$

where c is a non-zero constant, can be recovered, such as SoftReLU($\cdot$) defined by $\sigma(\mathbf{x}) = \log(1 + e^{\mathbf{x}})$ and LeakyReLU($\cdot$) [51]:

$$\mathbf{x} = \frac{1}{1+\delta} \left[\sigma(\mathbf{x}), \sigma(-\mathbf{x})\right] \begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \end{bmatrix}, \quad (12)$$

where $\delta > 0$ is the slope of the negative part of LeakyReLU($\cdot$). The substitution of Eq. (12) to Eqs. (8)-(9) contributes to the identity mapping in our transparent initialization. One can derive it in a similar way to ReLU($\cdot$). More details are given in the supplementary material.

3.3. Logit Consistency with Sparse Encoder

In addition to the notations of pixel number N_p, superpixel number N_s, and subsets of pixels $\mathcal{P} = \{\mathcal{P}_0, ..., \mathcal{P}_{N_s-1}\}$ in Sec. 3.1, the label set is defined as $\mathcal{L} = \{0, ..., N_l - 1\}$ given N_l labels. For logit x_s^l of $\mathcal{P}_s$ for superpixel s at label l, the logit consistency follows

$$x_s^l(p) \leftarrow \frac{1}{|\mathcal{P}_s|} \sum_{p \in \mathcal{P}_s} x_s^l(p), \ \forall p \in \mathcal{P}_s, \quad (13)$$

which guarantees all pixels in $\mathcal{P}_s$ having the same logit at each label so as to be assigned with the same label.

Nevertheless, considering the high complexity of indexing $x_s^l(p)$, a dense matrix operation requires a large GPU memory, that is $N_l N_s N_p$, especially for the backpropagation in CNN learning. This makes it infeasible for training due to the limited GPU memory in our experiments.

Hence, we adopted a sparse encoder, including sparse encoding and decoding, with sparse matrix operations for the consistency. Let us set matrix for indexing pixels by superpixels as $M(s,p)$, logit matrix as $M(l,p)$, where $s \in \mathcal{S}, p \in \mathcal{P}$, and $l \in \mathcal{L}$, sparse encoding and decoding are

$$\text{Encoding: } M(s,l) = \frac{\text{SMM}(M(s,p), M^T(l,p))}{\text{SADD}_{p \in \mathcal{P}_s}(M(s,p))}, \quad (14a)$$

$$\text{Decoding: } M(l,p) \leftarrow \text{SMM}(M^T(s,l), M(s,p)), \quad (14b)$$

where $M(s,p) \in \mathcal{B}^{N_s N_p}$ is $\{0, 1\}$ binary, shown in Table 1, SMM($\cdot$) is sparse matrix multiplication, and SADD($\cdot$) is sparse addition. This converts Eq. (13) from dense operations to sparse with reduced complexity from $N_l N_s N_p$ to $N_l N_p$. Otherwise, it is infeasible to jointly train the networks due to the limited GPU memory in our experiments.

4. Experiments

We first evaluated the properties of our transparent initialization including its effectiveness of data recovery and numerical stability. Then, we demonstrated its effect on jointly learning pretrained networks of semantic segmentation and superpixels together with a sparse encoder for logit consistency. Our code in PyTorch will be released on GitHub upon publication.

4.1. Properties of Transparent Initialization

Effectiveness. Our transparent initialization aims at identically mapping the output of linear layer(s) to the input at the early stage when fine-tuning pretrained network(s). It retains the effect of learned parameters of pretrained model(s). Off-the-shelf parameter initialization methods, such as random (uniform) and Xavier [18] initialization, lead to random values of the output at the early training stage, which cannot generate effective features by the pretrained models. Also, compared with the identical initialization with identity matrices for deeper networks in Net2Net [10], our transparent initialization has a high initialization rate with much more non-zero parameters for dense gradients in the backpropagation.

We evaluated these methods on 3 fully-connected layers. The (in_channels, out_channels) for each layer is (42, 64)→(64, 64)→(64, 42). Since Net2Net only supports square linear layer, *i.e.*, in_channels equals out_channels, to increase network depth, all layers have 42 in_channels and

Manner	Init. Rate↑	Recovery Rate↑		Non-square Filter Supported
		w/o Activation	w Activation	
Random	98.2	0.0	0.0	✓
Xavier [18]	98.2	0.0	0.0	✓
Net2Net [10]	2.3	100.0	50.0	✗
Ours	**99.9**	**100.0**	**100.0**	✓

Table 2: *Our transparent initialization has high initialization and recovery rates on 3 fully-connected (FC) layers, and supports non-square filters. "init. rate": percentage of non-zero (absolute value $> \epsilon$) parameters; "recovery rate": percentage of outputs with the same (difference $< \epsilon$) values as inputs; "activation": ReLU($\cdot$); "non-square filter": a FC layer with different in_channels and out_channels. Inputs are in [-10, 10] and ϵ=1e-4.*

	[-1, 1]	[-10, 10]	[-1e2, 1e2]	[-1e3, 1e3]
Max Error	∼6.8e-6	∼6.6e-5	∼6.5e-4	∼6.6e-3

Table 3: *Stability of transparent initialization with numerical orders 1, 10, 1e2, 1e3. Layer parameters are consistent with Table 2.*

out_channels. Input data is normally distributed with size (4, 42, 512, 512) as (batch, in_channels, height, width).

In Table 2, random and Xavier initialization have ∼98% initialization rate but cannot recover the output from its input, leading to 0% recovery rate. Net2Net [10] has only ∼2% initialization rate and 50% recovery rate with ReLU($\cdot$) for non-negative values only. In contrast, our transparent initialization has a high initialization rate and 100% recovery rate by Eqs. (8)-(10) with ReLU($\cdot$).

Numerical Stability. Since the effect of our transparent initialization is distributed across layers and Eq. (4) is achieved by a pseudo-inverse matrix for a rank-deficient matrix, hidden layers have round-off errors, and thus, the matrix multiplication of layer parameters is not strictly identical. We therefore tested the numerical stability of our transparent initialization on 4 numerical orders in Table 3. Clearly, the max error between input and output is in proportion to the magnitude of input values. For a pretrained model, its output is usually stable in a numerical range, such as probability in [0, 1]. One can easily enforce a numerical regularization if the model output is out of range.

4.2. Implementation Setup

Datasets. We evaluated our proposal on 3 popular semantic segmentation datasets, ADE20K, PASCAL VOC 2012, and PASCAL Context. *ADE20K* [59, 60] has 150 semantic categories for indoor and outdoor objects. It contains 20,210 samples for training and 2,000 samples for validation. *PASCAL Context* [36] has additional annotations for PASCAL VOC 2010 and provides annotations for the whole scene with 400+ classes. We selected the given 59 categories for semantic segmentation with 4,996 training samples and 5,104 validation samples. *PASCAL VOC 2012* [13] and Berkeley benchmark [35] were used as a combined ver-

sion for 21 classes segmentation. This dataset has 1,449 images from PASCAL VOC 2012 val set for validation and 10,582 images for training. Meanwhile, *MS-COCO* [32] was used to pretrain the semantic segmentation network, *i.e.*, DeepLabV3+ in our case, for PASCAL VOC 2012. It has 92,516 images for training and 3,899 images for validation, while 20 object classes from the primary 80 classes were selected in accordance with PASCAL VOC 2012.

For superpixel networks, we used the state-of-the-art *superpixel network with FCN* from [52] that was pretrained on Berkeley Segmentation Data Set and Benchmarks 500 (BSDS500) [4] containing 500 images with handcrafted ground truth edges.

Learning Details. For the ablation study on *PASCAL VOC 2012*, we trained DeepLabV3+ from scratch on the combined dataset above with crop size 512^2. We set Learning Rate (LR) as 0.007 for ResNet and 0.07 for ASPP and decoder. To train on MS-COCO, LR was set to 0.01 for ResNet and 0.1 for the others. LR for superpixel network pretrained on BSDS500 was the same as ResNet which was pretrained on ImageNet and is accessible in PyTorch model zoo. We used SGD [44] with momentum 0.9, weight decay 5e-4, and "poly" scheduler for 60 epochs.

For the joint training, we set LR as 1e-6 for pretrained DeepLabV3+ and superpixel with FCN. LR for the transparent initialization module is 1e-7. These LRs were fixed in the joint training for 20 epochs.

For *ADE20K* and *PASCAL Context*, we fine-tuned the state-of-the-art pretrained ResNeSt101 and superpixel with FCN with crop size 480^2. LR for TI module is 1e-9, and 1e-6 for the others. We fine-tuned for 100 epochs by SGD with momentum 0.9 and weight decay 5e-4. Batch size was decreased from 16 to 8 due to the limited GPU memory.

Metrics. For semantic segmentation, we used mean Intersection over Union (mIoU) and pixel accuracy [34]. For segmentation edges, we used performance ratio of true edges to false edges [27] and F-measure, $2RP/(R + P)$ with edge recall rate R and precision P [42, 15].

4.3. Ablation Study

The ablation study was on PASCAL VOC 2012. We first reproduced DeepLabV3+ with ResNet101, resulting in 78.85% mIoU comparable to 78.43% [1]. Applying superpixel network over it by logit consistency increased the mIoU by 0.37%. We then adopted ResNet152 for a qualified baseline with 77.78% on the full size for evaluations.

Applying superpixel contours increased the mIoU from 77.78% to 78.15%, based on which fine-tuning by our transparent initialization further increased it by 0.79% (1.16% to DeepLabV3+). Pretrained on MS-COCO for PASCAL VOC 2012, our proposal has 0.61% increase of mIoU. Note

---

[1] https://github.com/jfzhang95/pytorch-deeplab-xception.git

Manner	Backbone	SP	TI	MS-COCO	mIoU (512^2/full)
DeepLabV3+ [9]	ResNet101	-	-	-	78.85 / 76.47
Ours	ResNet101	✓	-	-	**79.22 / 76.98**
DeepLabV3+ [9]	ResNet152	-	-	-	79.32 / 77.78
Ours	ResNet152	✓	-	-	79.94 / 78.15
	ResNet152	✓	✓	-	**80.46 / 78.94**
DeepLabV3+ [9]	ResNet152	-	-	✓	82.62 / 80.76
Ours	ResNet152	✓	✓	✓	**83.39 / 81.37**

Table 4: *Ablation study:* **single-scale** *evaluation on PAS-CAL VOC 2012. Ours used DeepLabV3+ with ResNet101 and ResNet152 and superpixel net with a 3-layer TI module. "SP": superpixel with logit consistency; "TI": transparent initialization. "mIoU" is on 512^2 and full image size.*

Manner	pixAcc.	mIoU
PSPNet [55]	81.39	43.29
EncNet [53]	81.69	44.65
ResNeSt50 [54]	81.17	45.12
ResNeSt101 [54]	82.07	46.91
Ours[3]	**82.37**	**47.42**

Table 5: **Multiscale** *evaluation on ADE20K. Ours used ResNeSt101 and superpixel with a 2-layer TI module.*

Manner	pixAcc.	mIoU
FCN (ResNet50) [34]	73.40	41.00
EncNet [53]	80.70	54.10
ResNeSt50 [54]	80.41	53.19
ResNeSt101 [54]	81.91	56.49
Ours[3]	**82.43**	**57.32**

Table 6: **Multiscale** *evaluation on PASCAL Context. Ours used ResNeSt101 and superpixel with a 3-layer TI module.*

that most edges in the ground truth were neglected in the evaluation, marked white in Fig. 5(d).

Again, our goal is to preserve sharp edges aligned with object contours by superpixels. Fig. 5 vividly shows the enhanced object edges, especially objects that are highly contrastive to the background, such as birds and human heads.

4.4. Evaluations

Semantic Segmentation. For ADE20K and PASCAL Context, we used the most recent state-of-the-art semantic segmentation network ResNeSt [54] [2]. Its state-of-the-art performance on ADE20K using **ResNeSt200** is 82.45% pixel accuracy and 48.36% mIoU, and 83.06% pixel accuracy and 58.92% mIoU on PASCAL Context using **ResNeSt269**. Since we have only 4 P100 (16 GB) GPUs for our experiments while [54] has 64 V100 (16 GB) GPUs, we chose ResNeSt101 instead of ResNeSt200 or ResNeSt269 as our baseline network. Note that it is possible to enhance semantic segmentation edges on those state-of-the-art networks given sufficient GPU memory.

In Table 5, our method improves the pixel accuracy by

---

[3]Since [54] has 64 GPUs for state-of-the-art ResNeSt training while only 4 GPUs are accessible for ours, we used ResNeSt101 as baseline.

[2]https://github.com/zhanghang1989/PyTorch-Encoding

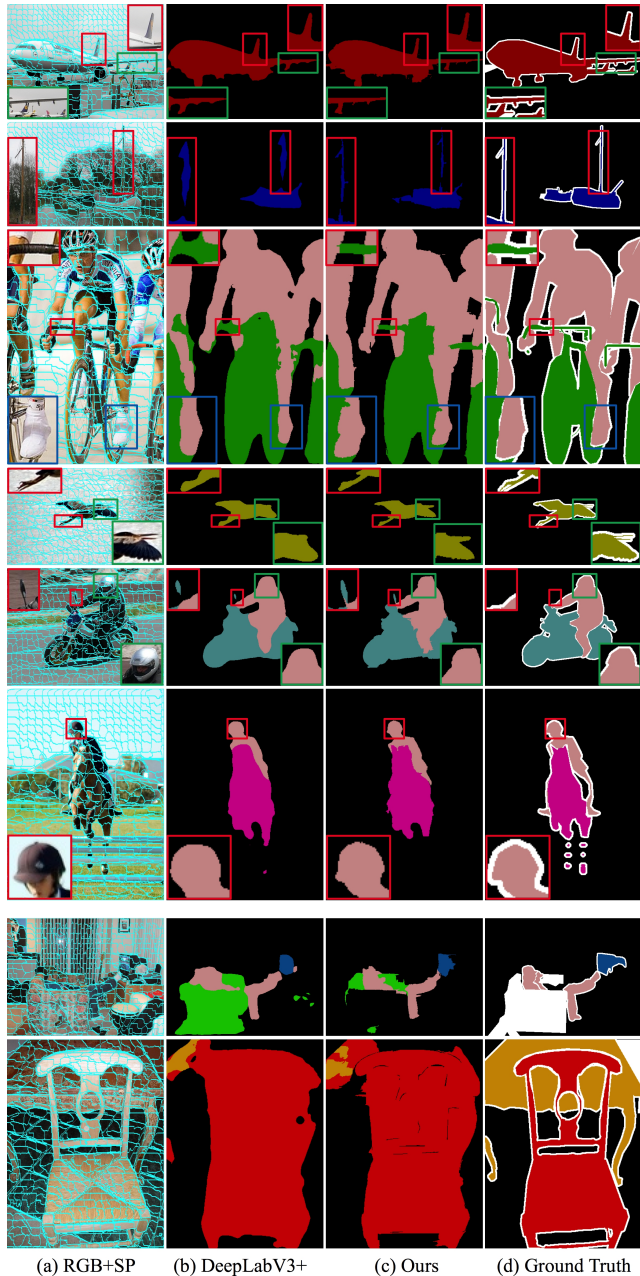(a) RGB+SP   (b) DeepLabV3+   (c) Ours   (d) Ground Truth

Figure 5: **Single-scale** *evaluation on PASCAL VOC 2012. First 6 rows are successful cases; last 2 rows are failed cases. SP maps are single-scale. Best view by zoom-in.*

0.39% and mIoU by 0.61% on ADE20K over the baseline. In Table 6, the pixel accuracy is improved by 0.71% and the mIoU by 0.83% over the baseline.

More importantly, visualizations in Fig. 6 for ADE20K and Fig. 7 for PASCAL Context vividly show the enhanced object edge details. This is the core of our refinement on semantic segmentation with superpixel constraints. For a fair comparison with [54], multiscale evaluations with multiscale superpixel maps were used while the superpixel maps in Figs. 6-7 are single-scale merely for demonstration.

Figure 6: *Multiscale evaluation on ADE20K. SP maps are single-scale for demonstration. Best view by zoom-in.*

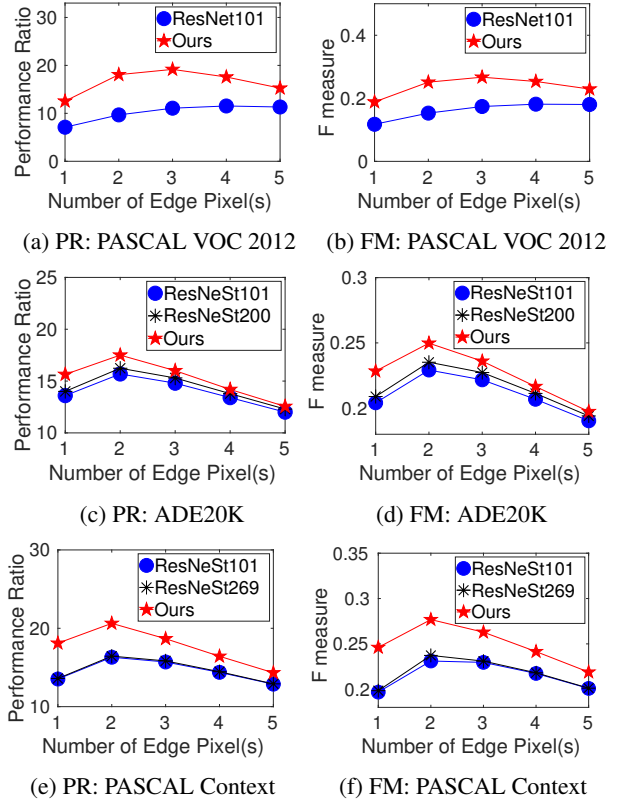Figure 7: *Multiscale evaluation on PASCAL Context. SPs are single-scale for demonstration. Best view by zoom-in.*

Figure 8: *Evaluation on segmentation edges using Performance Ratio (PR) and F-measure (FM). Edges in ground truth are extended to {1,2,3,4,5} pixels. Ours outperform the segmentation edges of the most recent state-of-the-art.*

Semantic Segmentation Edges. Since our refinement of semantic segmentation mainly lies in object edge areas, we used the popular Performance Ratio (PR) [27] and F-measure (FM) [42, 15] to evaluate the enhanced segmentation edges. In Fig. 8, ours outperform the baselines, *i.e.*, ResNet101 on PASCAL VOC 2012 and ResNeSt101 on the others, as well as the most recent state-of-the-art [54], *i.e.*, ResNeSt200 on ADE20K and ResNeSt269 on PASCAL Context, with higher PR and FM. Illustrations of the state-of-the-art edges compared with ours are shown in Fig. 1.

5. Conclusion

With transparent initialization and sparse encoder introduced in our paper, the joint learning of the state-of-the-art networks for semantic segmentation and superpixels preserves object edges. The proposed transparent initialization used to fine-tune pretrained models retains the effect of learned parameters through identically mapping the network output to its input at the early learning stage. It is more robust and effective than other parameter initialization methods, such as Xavier and Net2Net. Moreover, the sparse encoder enables the feasibility of efficient matrix multiplications with largely reduced computational complexity. Evaluations on PASCAL VOC 2012, ADE20K, and PAS-

CAL Context datasets validate the effectiveness of our proposal with enhanced object edges. Meanwhile, the quality of our semantic segmentation edges, evaluated by performance ratio and F-measure, is higher than other methods Additionally, transparent initialization is not limited to the joint learning for semantic segmentation but can also be used to initialize additional fully-connected layers for other tasks such as deep knowledge transfer.

Acknowledgement

Appendix

This provides additional details about transparent layers, giving more details of the initialization and structure of the layers. In particular, it shows how the approach may be used for any activation functions, not only forms of ReLU that were discussed in the main paper. This also speculates about the application of these ideas to form transparent convolutional layers. We now describe the ideas at a slower pace than is possible in the main paper, sometimes using different notation.

A. Introduction for Warmup

A common technique is to extend an existing network by the addition of further affine layers (classification layers) before the loss layer. These layers may then be trained separately or the whole extended network can be trained. We refer to the original (unextended) network as the base-network, and the network extended by some additional layers as the extended network. We assume that the extension layers consist of fully-connected layers, including offset, followed by an activation layer, such as a ReLU. These extension layers are inserted before the loss-layer.

Assuming that the base layer is trained to attain a low value of the loss, we wish to insert the extension layers into the network without increasing the loss of the network, so that additional training may continue to decrease the value of the loss-function. However, if the extension layers are initialized with random parameters, it will result in an increase in the loss of the network, so that the extension layers, or the whole network needs to be trained from scratch.

Simple transparent layers. This problem has also been addressed in previous work of Goodfellow [10]. One very simple method to implement a transparent layer is to let the matrix of layer parameters be the identity matrix. This has the disadvantage, however, that all the parameters are either 0 or 1, which fails to introduce sufficient randomness into

the system, which would perhaps be desirable. In addition, if the function f is represented by the identity mapping, then many of the parameters are equivalent, perhaps also an undesirable feature. We wish to allow the parameters of the affine transformation to be chosen randomly so as to mix things up a bit.

It is also not clear to do in the case where $f : \mathbb{R}^m \to \mathbb{R}^n$, where $m \neq n$, for then there is no such thing as an identity matrix. We wish to allow the possibility that the output and input of the extension layers are not equal.

B. Affine Layers

We are interested in layers in a Neural Network (NN) that implement an affine transformation on the input, that is, a linear transformation followed by an offset. For the present, we ignore any non-linear activatation that might be applied to the output of the layer.

The most obvious example of such a layer is a fully-connected layer, and that will be our main focus. However, the same idea could be applied to handle convolution layers. We raise this possibility and make a few comments on this later. Any development of this idea is left to further work.

An affine layer, in the following discussion, is seen as composed of a linear transform, followed by an offset and then a non-linear activation. The linear transform, followed by offset performs an affine transformation on the data, which will be written as[3]

$$f(\mathbf{x}) = \mathbf{x}\mathbf{A} + \mathbf{b} \tag{15}$$

where $f : \mathbb{R}^m \to \mathbb{R}^n$. In this note, the convention is that vectors, such as $\mathbf{x}$ are row vectors, which is contrary to a convention common in computer vision literature that vectors represent column vectors. The present notation is common in computer graphics literature, however.

This mapping satisfies the condition

$$f(\lambda \mathbf{x}) + f((1 - \lambda)\mathbf{x}) = f(\mathbf{x}) \tag{16}$$

and it may be represented by matrix multiplication on the homogenized vector $\tilde{x}$, according to

$$\tilde{x} \mapsto \tilde{x}\tilde{\mathbf{A}} \tag{17}$$

where the matrix $\tilde{\mathbf{A}}$ is given by

$$\tilde{\mathbf{A}} = \left[\begin{array}{cc} \mathbf{A} & \mathbf{0}^\top \\ \mathbf{b} & 1 \end{array} \right] . \tag{18}$$

---

[3]The notation in this exposition differs slightly, mainly in choice of fonts, from the main paper. Here, we deonte a matrix by $\mathbf{A}$, an affine transformation by A and the matrix that expresses an affine transformation in terms of homogeneous coordinates by $\tilde{\mathbf{A}}$. Homogeneous quantities in general (such as homogeneous vectors are denoted with a tilde).

For future reference, the matrix that performs the inverse affine transformation is equal to

$$\tilde{\mathbf{A}}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0}^\top \\ -\mathbf{b}\mathbf{A}^{-1} & 1 \end{bmatrix} . \qquad (19)$$

This works if $\mathbf{A}$ is a square matrix and has an inverse. More generally, in the case where $\mathbf{A}$ is of dimension $m \times n$, with $m < n$, it is possible that $\mathbf{A}$ has a right-inverse, which is a matrix, denoted by $\mathbf{A}^R$ such that $\mathbf{A}\mathbf{A}^R = \mathbf{I}$. In this case, a right inverse for $\tilde{\mathbf{A}}$ is given by

$$\tilde{\mathbf{A}}^R = \begin{bmatrix} \mathbf{A}^R & \mathbf{0}^\top \\ -\mathbf{b}\mathbf{A}^R & 1 \end{bmatrix} . \qquad (20)$$

where $\mathbf{A}^R$ is the right-inverse of matrix $\mathbf{A}$.

An affine transform maps $\mathbb{R}^m$ onto an affine subspace of $\mathbb{R}^n$, and the *rank* of the affine transform may be defined as the dimension of the affine space that is the image of this transform. Such a transformation will be said to be of *full rank* if its rank is equal to m, the dimension of input space.

From here on, we shall use the symbol A to represent an affine transformation, writing $\mathbf{x}$A. The symbol A represents the transformation itself, not the matrix (in this case $\tilde{\mathbf{A}}$) that implements it. **Caution:** to reemphasize this, the symbols $\mathbf{A}$ and A do not represent the same thing. In particular $\mathbf{A}$ is a matrix and A is an affine transform, which is equivalent to matrix multiplication by a matrix $\mathbf{A}$ and offset by a vector $\mathbf{b}$, so that $\mathbf{x}$A is short-hand notation for $\mathbf{x}\mathbf{A} + \mathbf{b}$.

Interlude: right inverse of a matrix. Let $\mathbf{A}$ be an $m \times n$ matrix with $m \le n$. An $n \times m$ matrix $\mathbf{A}^R$ is the right-inverse of $\mathbf{A}$ if $\mathbf{A}\mathbf{A}^R = \mathbf{I}_{m \times m}$. Such a matrix exists if and only if $\mathbf{A}$ has rank m equal to its row-dimension. This cannot happen if $m > n$, where a right-inverse cannot exist.

In this case, $\mathbf{A}^R$ is given by the formula

$$\mathbf{A}^R = \mathbf{A}^\top (\mathbf{A}\mathbf{A}^\top)^{-1} , \qquad (21)$$

where the condition $\mathbf{A}\mathbf{A}^R = \mathbf{I}$ is easily verified. This formula relies on the fact that $\mathbf{A}\mathbf{A}^\top$ has an inverse, which is ensured because the rank of $\mathbf{A}$ is m.

An alternative procedure is to take the Singular Value Decomposition (SVD) $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$, where $\mathbf{V}^\top$ has dimension $m \times n$ and $\mathbf{V}^\top\mathbf{V} = \mathbf{I}_{m \times m}$. The matrix $\mathbf{D}$ (of dimension $m \times m$) is non-singular (since $\mathbf{A}$ has rank m). Then, the right-inverse is given by

$$\mathbf{A}^R = \mathbf{V}\mathbf{D}^{-1}\mathbf{U}^\top \qquad (22)$$

as is easily verified.

C. Sequences of Layers without Activation

For the time being, we consider the unrealistic case that the activation layer is missing from the affine layers. In this case, each affine layer performs an affine transformation. A sequence of transformations $\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_k$ may be applied to an input $\mathbf{x}$, giving

$$\mathbf{x} \mapsto \mathbf{x}\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_k . \qquad (23)$$

If we choose $\mathtt{A}_k$ to be a right-inverse of $\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_{k-1}$ then we have

$$\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_k = \mathtt{A}_1 \ldots \mathtt{A}_{k-1}(\mathtt{A}_1 \ldots \mathtt{A}_{k-1})^R = I , \qquad (24)$$

the identity mapping.

The condition for this right inverse to exist is that $\mathtt{A}_1\mathtt{A}_2 \ldots \mathtt{A}_{n-1}$ should be of full rank. A necessary condition for this to happen is as follow. Let $\mathtt{A}_i$ represent an affine transformation $\mathtt{A}_i : \mathbb{R}^{m_{i-1}} \to \mathbb{R}^{m_i}$. The first and last spaces in this sequence are $\mathbb{R}^{m_0}$ and $\mathbb{R}^{m_k}$, so $m_0 = m$ and $m_k = n$. Then $\mathtt{A}_1 \ldots \mathtt{A}_{k-1}$ has a right inverse only if $m_i \ge m_0$ for all $i = 1, \ldots, k-1$. In other words, a necessary condition for the right inverse to exist is that the dimensions of all the intermediate spaces $\mathbb{R}^{m_1}, \ldots, \mathbb{R}^{m_{k-1}}$ should be at least equal to m_0. It is not hard to see that *generically*, this is also a sufficient condition, meaning that it is true for almost all sequences of transformations.[4]

Without being too formal here, we can state that this condition will hold *generically* if all the transformations are chosen at random.

Theorem 2 *Let a sequence of affine transformations $\mathtt{A}_i : \mathbb{R}^{m_{i-1}} \to \mathbb{R}^{m_i}$, for $i = 1, \ldots, k-1$ be chosen at random. Then $\mathtt{A}_1 \ldots \mathtt{A}_{k-1}$ almost surely has a right inverse if and only if $m_i \ge m_0$ for $i = 1, \ldots, k-1$.*

For practical purposes it is safe to proceed as if this result holds always (and not just almost always), since the probability that $\mathtt{A}_1 \ldots \mathtt{A}_{k-1}$ does not have an inverse is vanishingly small (in fact zero).

Therefore, the strategy for selecting a set of parameters for a sequence of affine layers may be described. For the sequence of layers to represent an identity transform, we need that the input and output space have the same dimension, namely $m_0 = m_k$. Then

1. Select intermediate dimensions $m_1, \ldots, m_{k-1}$ such that $m_i \ge m_0$ for all $i = 1, \ldots, k-1$.

2. Define random affine transforms $\mathtt{A}_i$ by selecting the entries of matrices $\mathbf{A}_i$ and vectors $\mathbf{b}_i$ randomly, using a suitable random number generator, for instance by a zero-mean normal (Gaussian) distribution.

---

[4]The terms *almost all, almost always, etc.* are intended in the standard mathematical sense, meaning that the set of cases for which the relevant condition fails to be true has measure or probability zero. For instance the set of $m \times n$, with $m \le n$ that do not have full rank is a set of measure 0 in the set of all such matrices, so one can state that almost all such matrices have full rank.

As another example, a matrix chosen at random (which implies that numbers are chosen from some probability distribution such as a normal distribution) will be of full rank with probability 1, hence almost always.

3. Compute the composition $\mathtt{A}_1\mathtt{A}_2\ldots\mathtt{A}_{k-1}$ by matrix multiplication, and take its right-inverse.

4. Set $\mathtt{A}_k$ to equal $(\mathtt{A}_1\mathtt{A}_2\ldots\mathtt{A}_{k-1})^R$.

The resulting product $\mathtt{A}_1\mathtt{A}_2\ldots\mathtt{A}_k$ is the identity affine transformation.

D. Gaussian Random Matrices

Since we are using matrices initialized from a normal distribution, we give some properties of such matrices. The purpose is to ensure that the concatenation of affine transformations does not cause explosion of the entries of the product $\mathtt{A}_1\mathtt{A}_2\ldots\mathtt{A}_{k-1}$. If this is not done properly, then this product will have very large entries and so will any value of $x_0\mathtt{A}_1\mathtt{A}_2\ldots\mathtt{A}_{k-1}$.

The solution is to make the matrices $\mathbf{A}_i$ appearing as the linear-transform part of $\mathbf{A}_i$ as close to orthogonal as possible. As an additional advantage, the entries of $\mathtt{A}_k = (\mathtt{A}_1\ldots\mathtt{A}_{k-1})^R$ will be approximately of the same order as the entries of each of the other $\mathtt{A}_i$. It will turn out that the entries of each $\mathtt{A}_i$ should be chosen from a Gaussian distribution with variance $1/m$ where m is its row-dimension.

We are assuming here that the dimensions of the matrix are large. In addition, if the matrix is not square, by saying it is orthogonal we mean that both its rows (and similarly it columns) are orthogonal vectors, all of the same length (a different length for the row and column vectors, of course). One may mistakenly assume that by randomly selecting each entry of a matrix randomly one obtains a random matrix in some vague sense. In reality, one obtains an (nearly) orthogonal matrix.

To see this, we first see that two random vectors from distribution $\mathcal{D}$ are almost orthogonal. Let X and Y be two i.i.d random variables in $\mathcal{N}(0,\sigma^2)$, we have

$$E[XY] = E[X]E[Y] = 0 \qquad (25)$$

and

$$E[(XY)^2] = E[X^2]E[Y^2] . \qquad (26)$$

Then,

$$
\begin{aligned}
\mathrm{var}(XY) &= E[(XY)^2] - E^2[XY] \\
&= E[X^2]\,E[Y^2] \\
&= \left(E[X^2] - E^2[X]\right)\left(E[Y^2] - E^2[Y]\right) \\
&= \mathrm{var}(X)\,\mathrm{var}(Y) .
\end{aligned} \qquad (27)
$$

Now, given a column vector $\mathbf{v} \in \mathbb{R}^m$ with entries chosen from distribution $\mathcal{D}$, we have

$$\mathrm{var}\left(\sum_{i=1}^m X_i Y_i\right) = m\,\mathrm{var}\left(X_i Y_i\right) = m\,\mathrm{var}^2(X) = m\sigma^4, \qquad (28)$$

where X_i and Y_i are independent entries in two of such column vectors respectively. Then, the *expected squared length* of such a vector $\mathbf{v}$ is $mE[X^2] = m\sigma^2$. If we choose $\sigma^2 = 1/m$, $\mathbf{v}$ has the expected squared length equal to 1. This satisfies the attribute of an orthogonal matrix, $\mathbf{v}^T\mathbf{v} = \mathbf{1}$. Then, Eq. (27) follows

$$\mathrm{var}\left(\sum_{i=1}^m X_i Y_i\right) = m\sigma^4 = \frac{1}{m} . \qquad (29)$$

Hence, two m-length vectors randomly chosen from $\mathcal{N}(0, 1/m)$ will have expected squared length 1 and expected inner-product 0 (by Eq. (25)) with variance of the inner product as $1/m$.

On the other hand, the *variance of the square length* of the vector is

$$\mathrm{var}\left(\sum_{i=1}^m X_i^2\right) = m\,\mathrm{var}(X^2) = \frac{2}{m} , \qquad (30)$$

where $\mathrm{var}(X^2) = 2\sigma^4$ is obtained by $E(X^4) - E^2(X)$ with $E^2(X^2) = \sigma^4$ and

$$
\begin{aligned}
&E(X^4) \\
&= \frac{\int X^4 \exp\left(-\frac{X^2}{2\sigma^2}\right) dX}{\int \exp\left(-\frac{X^2}{2\sigma^2}\right) dX} \\
&= \frac{-\sigma^2 X^3 \exp\left(-\frac{X^2}{2\sigma^2}\right)\Big|_{-\infty}^{+\infty} + \sigma^2 \int \exp\left(-\frac{X^2}{2\sigma^2}\right) dX^3}{\int \exp\left(-\frac{X^2}{2\sigma^2}\right) dX} \\
&= \frac{3\sigma^2 \int X^2 \exp\left(-\frac{X^2}{2\sigma^2}\right) dX}{\int \exp\left(-\frac{X^2}{2\sigma^2}\right) dX} \\
&= \frac{3\sigma^2(-\sigma^2)\left(X \exp\left(-\frac{X^2}{2\sigma^2}\right)\Big|_{-\infty}^{+\infty} - \int \exp\left(-\frac{X^2}{2\sigma^2}\right) dX\right)}{\int \exp\left(-\frac{X^2}{2\sigma^2}\right) dX} \\
&= 3\sigma^4 .
\end{aligned}
$$

$$(31)$$

To this end, it shows

Theorem 3 *If entries of an $m \times n$ matrix are chosen from a zero-mean distribution with variance $\sigma^2 = 1/m$, then the column vectors have expected squared length 1 with variance $2/m$ and the expected inner product of each two columns is 0 with variance $1/m$.*

As m increases, the matrix approximates more and more an orthogonal matrix. Corresponding to Fig. 4 in the main paper, weights A_i and bias b_i can be initialized by

1. Choose the dimension of i-th layer filter as $m_{i-1} \times m_i$, for $i = 1, ..., k-1$, where $m_i \geq m_0 = m_k$.

2. Define a random affine transformation A_i by selecting its weight matrix $\mathbf{A}_i$ from a $\mathcal{N}(0, 1/m_i)$ Gaussian distribution and its bias vector $\mathbf{b}_i$ from a $\mathcal{N}(0, 1)$ Gaussian distribution.

3. Initialize A_k with the right inverse of $A_1 A_2 ... A_{k-1}$.

Again, **note** that A_i is an affine transformation (also used as a layer) where $\mathbf{A}_i$ is the weight matrix of A_i. Then, the above initialization will lead to an identity mapping as $A_1 A_2 ... A_k = \mathbf{1}$.

E. Getting Past the Activation Layer

We concentrate first on the case where each affine transform is followed by a ReLU activation. To do this, we will need to double the size of the intermediate layers, as will be seen next. We assume that affine transforms A_i are given, let $\mathbf{x} = \mathbf{x}_0$ be the input of A_1 and define $\mathbf{x}_i = \mathbf{x}_{i-1} A_i$ for $i = 1, \ldots, k$, the output of the i-th affine transform.

In the course of the following discussion, we shall be defining new affine transforms A_i' and layers represented by $A_i' \sigma$, namely an affine transform followed by an activation σ. Once more let $\mathbf{x}_0'$ be the input of the first layer, $\mathbf{x}_0' = \mathbf{x}_0$, and define $\mathbf{x}_i' = \mathbf{x}_{i-1}' A_i' \sigma$, the result of the affine transformation followed by the activation. Quantities without primes (A_i and $\mathbf{x}_i'$) belong to the original sequence of affine transforms, whereas those with primes (A_i' and $\mathbf{x}_i'$) belong to the sequence, with activation, being constructed).

The first layer will be modified as follows. Suppose that A_1 is represented in matrix form as

$$\mathbf{x}_0 A_1 = (\mathbf{x}_0, 1) \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{b}_1 \end{bmatrix} = \mathbf{x}_0 \mathbf{A}_1 + \mathbf{b}_1 . \quad (32)$$

This is replaced by

$$\begin{aligned} \mathbf{x}_0 A_1' &= (\mathbf{x}_0, 1) \begin{bmatrix} \mathbf{A}_1 & -\mathbf{A}_1 \\ \mathbf{b}_1 & -\mathbf{b}_1 \end{bmatrix} \\ &= \mathbf{x}_0 \left[\mathbf{A}_1 \mid -\mathbf{A}_1 \right] + (\mathbf{b}_1, -\mathbf{b}_1) \quad (33) \\ &= (\mathbf{x}_0 \mathbf{A}_1 + \mathbf{b}_1, -\mathbf{x}_0 \mathbf{A}_1 - \mathbf{b}_1) \\ &= (\mathbf{x}_0 A_1, -\mathbf{x}_0 A_1) \end{aligned}$$

In other words, the $m_0 \times m_1$ matrix $\mathbf{A}$ is replaced by the $m_0 \times 2m_1$ matrix $\mathbf{A}_1' = \left[\mathbf{A}_1 \mid -\mathbf{A}_1 \right]$, and $\mathbf{b}_1$ is replaced by the $2m_1$ dimensional vector $\mathbf{b}_1' = (\mathbf{b}_1, -\mathbf{b}_1)$.[5]

---

[5]**Important note:** The affine transformation A' is defined

$$\begin{bmatrix} \mathbf{A}_1 & -\mathbf{A}_1 \\ \mathbf{b}_1 & -\mathbf{b}_1 \end{bmatrix} .$$

This will output a vector of the form $(\mathbf{x}_0 A_1, -\mathbf{x}_0 A_1)$ with two parts that are negatives of each other. It defines the **initial** form of the affine transform only. During training, all entries of the above matrix are free to vary (and will) independently, and its output will not maintain this symmetric form. This applies to all the affine transforms A_i' that will be defined.

Now, when this is passed through an activation layer, represented by the activation function σ (ReLU), the result is

$$\mathbf{x}_0 A_1' \sigma = (\mathbf{x}_0 A_1 \sigma , (-\mathbf{x}_0 A_1)\sigma) = \mathbf{x}_1' . \quad (34)$$

This is the output of our modified affine layer with activation. The point to note here is the identity $\mathbf{v}\sigma - (-\mathbf{v})\sigma = \mathbf{v}$. This gives

$$(\mathbf{x}_0 A_1)\sigma - (-\mathbf{x}_0 A_1)\sigma = \mathbf{x}_0 A_1 . \quad (35)$$

Thus, by subtracting the two halves of $\mathbf{x}_0 A_1' \sigma$, one arrives back at the simple affine transform $\mathbf{x}_0 A_1$.

We simplify the notation as follows. Let $\mathbf{x}_i^+$ and $\mathbf{x}_i^-$ be defined as

$$\mathbf{x}_i^+ = \mathbf{x}_i \sigma \quad (36a)$$
$$\mathbf{x}_i^- = (-\mathbf{x}_i)\sigma \quad (36b)$$

which are the positive and negative parts of $\mathbf{x}_i$ respectively, satisfying $\mathbf{x}_i = \mathbf{x}_i^+ - \mathbf{x}_i^-$. Then, we see that

$$\mathbf{x}_1' = (\mathbf{x}_1^+, \mathbf{x}_1^-) . \quad (37)$$

Now, the first thing to do at the beginning of the next layer is to subtract the two parts of the previous output, illustrated by

$$\mathbf{x}_1' \begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \end{bmatrix} = (\mathbf{x}_1^+, \mathbf{x}_1^-) \begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \end{bmatrix} = \mathbf{x}_1 . \quad (38)$$

This is followed by the same trick of separating the positive and negative parts as before. The affine part (without activation) of the second layer is then

$$\begin{aligned} \mathbf{x}_1' A_2' &= \mathbf{x}_1' \begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \end{bmatrix} [\mathbf{A}_2 \mid -\mathbf{A}_2] + (\mathbf{b}_2, -\mathbf{b}_2) \\ &= \mathbf{x}_1' \begin{bmatrix} \mathbf{A}_2 & -\mathbf{A}_2 \\ -\mathbf{A}_2 & \mathbf{A}_2 \end{bmatrix} + (\mathbf{b}_2, -\mathbf{b}_2) \quad (39) \\ &= \mathbf{x}_1' \mathbf{A}_2' + \mathbf{b}_2' , \end{aligned}$$

where $\mathbf{A}_2'$, $\mathbf{b}_2'$ and A_2' are defined by this equation. Noting Eq. (38), we arrive at

$$\mathbf{x}_1' A_2' = (\mathbf{x}_1 \mathbf{A}_2, -\mathbf{x}_1 \mathbf{A}_2) . \quad (40)$$

and so

$$\mathbf{x}_1' A_2' \sigma = \mathbf{x}_2' = (\mathbf{x}_1 \mathbf{A}_2 \sigma, (-\mathbf{x}_1 \mathbf{A}_2)\sigma) = (\mathbf{x}_2^+, \mathbf{x}_2^-) . \quad (41)$$

Combining this with Eq. (37) gives

$$\mathbf{x}_0 A_1' \sigma A_2' \sigma = (\mathbf{x}_2^+, \mathbf{x}_2^-) . \quad (42)$$

Continuing to define layers in this way, according to Eq. (44), for layers up to layer $k - 1$, gives that

$$\mathbf{x}_0 A_1' \sigma A_2' \sigma \ldots A_{k-1}' \sigma = (\mathbf{x}_{k-1}^+, \mathbf{x}_{k-1}^-) = \mathbf{x}_{k-1}' . \quad (43)$$

Finally, we define the last layer k by

$$
\begin{aligned}
\mathbf{x}'_{k-1}\mathtt{A}'_k = (\mathbf{x}^+_{k-1}, \mathbf{x}^-_{k-1}) \begin{bmatrix} \mathbf{A}_k \\ -\mathbf{A}_k \end{bmatrix} + \mathbf{b}_k \\
= (\mathbf{x}^+_{k-1} - \mathbf{x}^-_{k-1})\mathbf{A}_k + \mathbf{b}_k \\
= \mathbf{x}_{k-1}\mathbf{A}_k + \mathbf{b}_k \\
= \mathbf{x}_{k-1}\mathtt{A}_k \\
= \mathbf{x}_k \ .
\end{aligned} \tag{44}
$$

Putting those defined $\mathtt{A}_i$ all together gives

$$
\mathbf{x}_0 \, \mathtt{A}'_1 \sigma \, \mathtt{A}_2 \sigma \ldots \mathtt{A}'_k = \mathbf{x}_k = \mathbf{x}_0 \ , \tag{45}
$$

where the final step is because the sequence of affine transforms $\mathtt{A}_1 \ldots \mathtt{A}_k$ is chosen to be the identity map, so $\mathbf{x}_k = \mathbf{x}_0 \mathtt{A}_1 \ldots \mathtt{A}_k = \mathbf{x}_0$.

F. Exploration of Layer Initialization Effects

In Table 2 in the main paper, we compared the effectiveness of our transparent initialization with others, that is random, Xavier [18], and Net2Net [10]. Transparent initialization can 100% recover the input data from the layers output with a high initialization rate and a small around-off error shown in Table 3 and supports non-square filters, by saying filters we mean 1^2 kernel-size convolutional layers or fully-connected layers. In contrast, random and Xavier initialization are ineffective to recover the input data from the output while Net2Net is only effective for non-negative data and is infeasible for non-square filters. Despite these quantitative experiments to analyse the attributes of transparent initialization, we directly compare their effects on joint learning pretrained networks of semantic segmentation and superpixels in addition to the ablation study in the main paper.

In Fig. 9, for the task of semantic segmentation, the cross-entropy loss function highly relies on the maximum values of logits, the network outputs, along the label dimension. In Fig. 9(a)-9(c), input data of the add-on FC layers contains positive and non-positive values while in Fig. 9(d) input data is non-positive by subtracting the maximum value along the label dimension. This is to explore the data recovering ability for different numerical space (that is non-negative and non-positive in our case).

The final mIoUs via random and Xavier initialization are nearly 0% as they totally interrupt the learned parameters, leading to a high loss. To recover **non-negative** data, in Fig. 9(c), Net2Net and ours have similar loss decreasing tendency. However, as the epoch increases, the loss by ours becomes lower than Net2Net because the gradients by transparent initialization are dense for parameter updates compared with the sparse ones by Net2Net. Furthermore, in Fig. 9(d), Net2Net is unable to recover **negative** values,
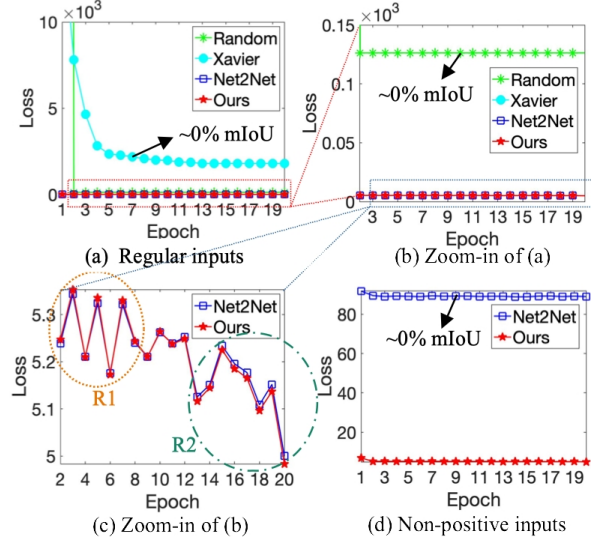

Figure 9: *Training loss with different layer initialization methods. This is an extended ablation study to Tables 2 and 4 in the main paper. "Regular": values containing positive, negative, and zero values. Note that in our semantic segmentation task, a low loss is determined by a high (mostly positive) logit from the NN along the label dimension. The joint learning is for 21-label semantic segmentation using pretrained DeepLabV3+ [9] and superpixel with FCN [52] networks with add-on 3 Fully-Connected (FC) layers. Here, in (a) and (b), random and Xavier initialization on these add-on FC layers lead to a high loss, and thus, decreasing the mIoU from $\sim$80% to $\sim$0%. It is obvious that they cannot work in our case, since they are unable to recover the pretrained results as shown in Table 2 in the main paper. On the other hand, Xavier initialization may have a worse local minima than random initialization due to the interrupted output values. In contrast, for regular data containing positive, negative, and zero values, both Net2Net and our transparent initialization have similar good effectiveness. Because, as mentioned before, negative logit values (hardly to be the highest) may not have significant effects on the joint learning as in our case positive logits always have high softmax values. So, the negative values of the input data can even be zero-out by ReLU or Net2Net initialization. For non-positive inputs in (d), however, Net2Net is unable to recover negative values, leading to $\sim$0% mIoU while ours has the same low loss as it is in (c). Additionally, although Net2Net and ours in (c) have similar losses, both achieving $\sim$83.3% mIoU, we note that the loss of ours starts to be less than Net2Net, shown in R2. In R1, ours has a high loss due to the effects of dense gradients that change network parameters more dramatically than Net2Net. This is expected as the transparent initialization should have a strong learning ability than Net2Net. Overall, ours outperforms random and Xavier initialization, both regular and non-positive data, and Net2Net for non-positive data. For regular data (depends on task), ours tends to have a smaller loss than Net2Net due to its strong learning ability with dense gradients. More details are in Sec. F.*
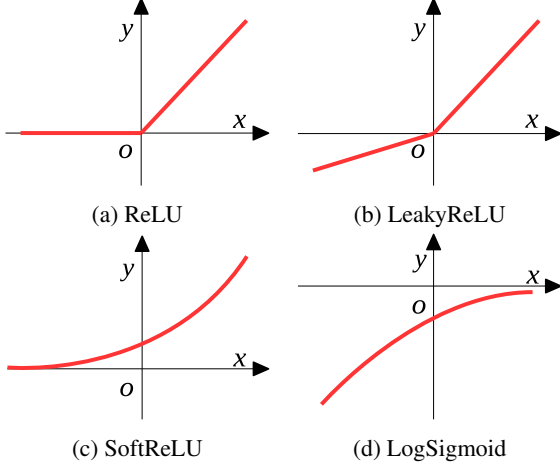
(a) ReLU   (b) LeakyReLU

(c) SoftReLU   (d) LogSigmoid

Figure 10: *Examples of non-linear active functions for transparent initialization.*

as shown in Table 2 in the main paper, resulting in a high loss. In Fig. 9(c), the mIoU by Net2Net and ours are similar, both nearly 83.3%. Corresponding to Fig. 9(d), however, the mIoU by Net2Net is nearly 0% while ours is still nearly 83.3% since it is invariant to numerical space.

G. Activation Functions

Again, note that the notation of activation function $\sigma(x)$ has the same meaning as $x\sigma$ for a simplicity of sequence layers. For the proposed transparent initialization, any non-linear activation functions satisfying Eq. (46) are feasible to recover the input data from its output.

$$\sigma(x) - \sigma(-x) = cx \, , \tag{46}$$

where c is a non-zero constant. This can be expressed by $xD\sigma S$ with D (duplicate) and S (subtract) defined in the main paper. In Fig. 10, we give 4 examples with corresponding definitions as follows:

$$\text{ReLU:} \quad y(x) = \begin{cases} x & \text{if } x \geq 0 \, , \\ 0 & \text{otherwise} \, , \end{cases} \tag{47a}$$

$$\text{LeakyReLU:} \quad y(x) = \begin{cases} x & \text{if } x \geq 0 \, , \\ \delta x & \text{otherwise} \, , \end{cases} \tag{47b}$$

$$\text{SoftReLU:} \quad y(x) = \log\left(1 + e^x\right) \, , \tag{47c}$$

$$\text{LogSigmoid:} \quad y(x) = \log\left(\frac{1}{1 + e^{-x}}\right) \, . \tag{47d}$$

It is easy to verify Eq. (46) for those activation functions. Given the activation function as LogSigmoid by $\sigma(x) = \log(1/(1 + e^{-x}))$, for instance, it follows

$$\begin{aligned} &\log\left(\frac{1}{1 + e^{-x}}\right) - \log\left(\frac{1}{1 + e^{x}}\right) \\ &= \log\left(\frac{1 + e^x}{1 + e^{-x}}\right) = \log(e^x) = x \, . \end{aligned} \tag{48}$$

(a) For $y = (x_0\sigma, x_1\sigma)\,S$
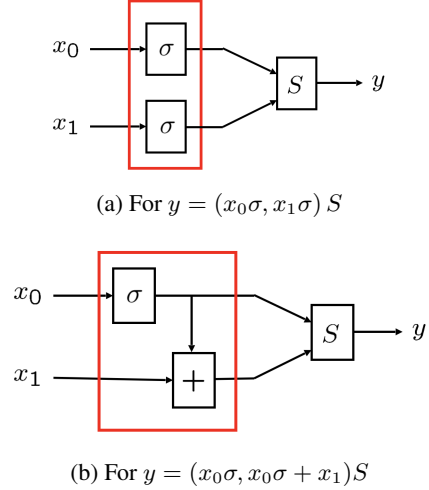

(b) For $y = (x_0\sigma, x_0\sigma + x_1)S$

Figure 11: *Two different forms of layer activation (the part in the red box) top be used as see-through activations for transparent layers. Here, σ represents any function, for instance, a commonly used non-linear function such as sigmoid, ReLU or hyperbolic tangent, and the block marked S carries out some operation on the two inputs. Thus, (a) implements $y = (x_0\sigma, x_1\sigma)\,S$ and (b) represents $y = (x_0\sigma, x_0\sigma + x_1)S$. At initialization, $x_0 = -x_1 = x$ and $(a, b)S = a - b$. In this case, the output is $y = x$ in both case. During the training of the network, x_0 and x_1 will diverge, and S will also learn to carry out a different operation, so networks will behave differently. However, using (b) to implement the affine layer activation will provide a transparent initialization, whatever function σ is chosen.*

Meanwhile, with c given in Eq. (46), $xD\sigma S$ should be $xD\sigma S/c$, where $c = 1/\delta$ is for LeakyReLU and $c = 1$ for the others. To be more general, this property holds for any functions of the form of $f(x) = cx + s(x)$, where $s(x)$ is an even function.

For other activations σ such as the sigmoid function, or arctangent, or hyperbolic tangent, it does not work directly. The trick is to observe that for the ReLU function, $(-x)\sigma = x\sigma - x$. This allows us to rewrite the sequence of operations

$$x \xmapsto{\sigma} (x\sigma, (-x)\sigma) = (x_1, x_2) \mapsto x_1 - x_2 = x \, ,$$

where σ is the ReLU function, as

$$x \xmapsto{\sigma} (x\sigma, x\sigma - x) = (x_1, x_2) \mapsto x_1 - x_2 = x \, .$$

But now this also holds for any activation σ, ReLU or not. This requires a slight change to the architecture of the transparent affine layer. Instead of outputting $(x\sigma, (-x)\sigma)$, it must output $(x\sigma, x\sigma - x)$. This is shown in Fig. 11.

H. Extension to Convolutional Layers

A common practice in experimenting with modifications to neural network architectures is to add additional convolutional layers. If a base network is already trained to achieve

minimal loss, then the addition of extra convolutional layers can perturb the loss, and require the network to retrain again to achieve a low loss. The loss after addition of the new layers may not be as low as the loss of the base network. However, by initializing the extra add-on layers to implement an identity transformation will ensure that the loss achieved (on the training set) by the modified network cannot be worse than that of the base network.

This section gives some ideas on the addition of transparent convolutional layers with pseudo-random initialization that could be used for this purpose. We have not explored this topic in any great depth, leaving it rather to be the subject of future work, and is somewhat speculative at present.

A convolutional layer as usually implemented is an example of an affine layer, since without any non-linear activation it implements a linear transformation on the input and then adds bias. However, unlike fully-connected layers, which are essentially a matrix multiplication, for which an inverse (or right-inverse) can be easily found, convolution is not conveniently represented by matrix multiplication and is less easily inverted.

It is clear that convolution, being linear, can be represented as matrix multiplication on a vectorized form of the input, but this will be a sparse matrix multiplication, and not easily inverted, at least by convolutional layers. By analogy with the method implemented for fully-connected layers, we require that a final convolution will invert the effect of a sequence of previous pseudo-random convolutions.

Not all convolutions are exactly invertible. In fact (ignoring edge effects) a convolution will be invertible by another convolution if and only if its Fourier transform is everywhere non-zero. However, with this caveat, it is possible to find inverse convolutions.

Rather than carrying out a string of convolutions, followed by a single convolution to undo the previous ones, a better strategy may be to add convolutions in pairs such as a high-pass and low-pass filter that cancel each other out. Exploring this topic will be the subject of further work.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. *TPAMI*, 34(11):2274–2282, 2012. 1, 2, 3

[2] R. Achanta and S. Susstrunk. Superpixels and polygons using simple non-iterative clustering. *CVPR*, 2017. 2

[3] T. Ajanthan, A. Desmaison, R. Rudy, M. Salzmann, P.H.S. Torr, and M.P. Kumar. Efficient linear programming for dense CRFs. *CVPR*, 2017. 2

[4] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *TPAMI*, 33(5):898–916, 2010. 3, 6

[5] V. Badrinarayanan, A. Kendall, and R. Cipolla. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *TPAMI*, 39(12):2481–2495, 2017. 2

[6] M.V. Bergh, X. Boix, G. Roig, and L.V. Gool. SEEDS: Superpixels extracted via energydriven sampling. *IJCV*, 2015. 2

[7] L.C. Chen, J.T. Barron, G. Papandreou, K Murphy, and A.L. Yuille. Semantic image segmentation with task-specific edge detection using CNNs and a discriminatively trained domain transform. *CVPR*, 2016. 1, 2

[8] 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. *TPAMI*, 40(4):834–848, 2017. 1

[9] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *ECCV*, 2018. 1, 2, 3, 7, 13

[10] T. Chen, I. Goodfellow, and J. Shlens. Net2Net: Accelerating learning via knowledge transfer. *ICLR*, 2016. 3, 5, 6, 9, 13

[11] Y. Chen, Y. Kalantidis, J. Li, S. Yan, and J. Feng. A2-nets: Double attention networks. *NeurIPS*, 2018. 2

[12] J. Deng, W. Dong, R. Socher, L.J. Li, K. Li, and F. Li. ImageNet: A large-scale hierarchical image database. *CVPR*, 2009. 2

[13] M. Everingham, S.M.A. Eslami, L.V. Gool, C.K.I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge a retrospective. *IJCV*, 2014. 3, 6

[14] D. Feng, C.H. Schutz, L. Rosenbaum, H. Hertlein, C. Glaser, F. Tim, W. Wiesbeck, and K. Dietmayer. Deep multi-model object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. *IEEE Transactions on Intelligent Transportation systems*, pages 1–20, 2020. 1

[15] P.A. Flach and M. Kull. Precision-recall-gain curves: Pr analysis done right. *NeurIPS*, 2015. 2, 6, 8

[16] J. Fu, J. Liu, H. Tian, Y. Li, Y. Bao, Z. Fang, and H. Lu. Dual attention network for scene segmentation. *CVPR*, 2019. 2

[17] R. Gadde, V. Jampani, M. Kiefel, D. Kappler, and P.V. Gehler. Superpixel convolutional networks using bilateral inceptions. *ECCV*, 2016. 1, 2

[18] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *AISTATS*, 2010. 5, 6, 13

[19] S. Gould, J. Zhao, X. He, and Y. Zhang. Superpixel graph label transfer with learned distance metric. *ECCV*, 2014. 2

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, 2016. 2

[21] M. Hofmarcher, T. Unterthiner, J.A. Medina, G. Klambauer, S. Hochreiter, and B. Nessler. Visual scene understanding for autonomous driving using semantic segmentation. *Explainable AI. LNCS*, 11700:285–296, 2019. 1

[22] S. Hong, X. Yan, T. Huang, and H. Lee. Learning hierarchical semantic image manipulation through structured representations. *NeurIPS*, 2018. 1

[23] G. Huang, Z. Liu, L.V.D. Maaten, and K.Q. Weinberger. Densely connected convolutional networks. *CVPR*, 2017. 2

[24] P. Isola, D. Zoran, D. Krishnan, and E.H. Adelson. Crisp boundary detection using pointwise mutual information. *ECCV*, 2014. 2

[25] V. Jampani, D. Sun, M.Y. Liu, M.H. Yang, and J. Kautz. Superpixel sampling networks. *ECCV*, 2018. 1, 2

[26] M. Jordan. Learning in graphical models. *MIT Press*, 1998. 2

[27] P.A. Khaire and Dr.N.V. Thakur. A fuzzy set approach for edge detection. *International Journal of Image Processing*, 6(6):403–412, 2012. 2, 6, 8

[28] P. Krahenbuhl and V. Koltun. Efficient inference in fully connected CRFs with Gaussian edge potentials. *NeurIPS*, 2011. 1, 2

[29] X. Li, Z. Zhong, J. Wu, Y. Yang, Z. Lin, and H. Liu. Expectation-maximization attention networks for semantic segmentation. *ICCV*, 2019. 2

[30] Z. Li and J. Chen. Superpixel segmentation using linear spectral clustering. *CVPR*, 2015. 1, 2

[31] G. Lin, A. Milan, C. Shen, and I. Reid. RefineNet: Multi-path refinement networks for high resolution semantic segmentation. *CVPR*, 2017. 2

[32] T.Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C.L. Zitnick, and P. Dollar. Microsoft COCO: Common objects in context. *ECCV*, 2014. 1, 6

[33] Y.J. Liu, C.C. Yu, M. Yu, and Y. He. Manifold SLIC: A fast method to compute content-sensitive superpixels. *CVPR*, 2015. 2

[34] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CVPR*, 2015. 1, 2, 6, 7

[35] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. *ICCV*, 2001. 1, 3, 6

[36] R. Mottaghi, X. Chen, X. Liu, N.G. Cho, S.W. Lee, S. Fidler, R. Urtasun, and A. Yuille. The role of context for object detection and semantic segmentation in the wild. *CVPR*, 2014. 1, 6

[37] H. Park, J. Jeong, Y. Yoo, and N. Kwak. Superpixel-based semantic segmentation rrained by statistical process control. *BMVC*, 2017. 2

[38] N.O. Salscheider. Simultaneous object detectin and semantic segmentation. *International conference on pattern recognition applications and methods (ICPRAM)*, 2019. 1

[39] F. Schroff, A. Criminisi, and A. Zisserman. Object class segmentation using random forests. *BMVC*, 2008. 2

[40] M. Schuurmans, M. Berman, and M.B. Blaschko. Efficient semantic image segmentation with superpixel pooling. *arXiv preprint arXiv:1806.02705*, 2018. 2

[41] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015. 2

[42] D. Stutz, A. Hermans, and B. Leibe. Superpixels: An evaluation of the state-of-the-art. *Computer Vision and Image Understanding*, 166:1–27, 2018. 2, 6, 8

[43] D. Stutz, A. Hermans, and B. Leibe. Superpixels: An evaluation of the state-of-the-art. *CVIU*, 2018. 2

[44] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. *ICML*, 2013. 6

[45] M. Thoma. A survey of semantic segmentation. *arXiv:1602.06541*, 2016. 2

[46] W.C. Tu, M.Y. Liu, V. Jampani, D. Sun, S.Y. Chien, M.H. Yang, and J. Kautz. Learning superpixels with segmentation-aware affinity loss. *CVPR*, 2018. 2

[47] R. Uziel, M. Ronen, and O. Freifeld. Bayesian adaptive superpixel segmentation. *ICCV*, 2019. 2

[48] P. Wang, G. Zeng, R. Gan, J. Wang, and H. Zha. Structure-sensitive superpixels via geodesic distance. *IJCV*, 2013. 2

[49] X. Wang, T. Wang, and J. Bu. Color image segmentation using pixel wise support vector machine classification. *pattern recognition*, 2011. 2

[50] F.Z. Xing, E. Cambria, W.B. Huang, and Y. Xu. Weakly supervised semantic segmentation with superpixel embedding. *ICIP*, 2016. 2

[51] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv:1505.00853*, 2015. 5

[52] F. Yang, Q. Sun, H. Jin, and Z. Zhou. Superpixel segmentation with fully convolutional networks. *CVPR*, 2020. 2, 3, 6, 13

[53] H. Zhang, K. Dana, J. Ping, Z. Zhang X. Wang, A. Tyagi, and A. Agrawal. Context encoding for semantic segmentation. *CVPR*, 2018. 1, 7

[54] H. Zhang, C. Wu, Z. Zhang, Y. Zhu, Z. Zhang, H. Lin, Y. Sun, T. He, J. Mueller, R. Manmatha, M. Li, and A. Smola. ResNeSt: Split-attention networks. *arXiv:2004.08955*, 2020. 1, 2, 3, 7, 8

[55] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. *CVPR*, 2017. 1, 7

[56] H. Zhao, Y. Zhang, S. Liu, J. Shi, C.C. Loy, D. Lin, and J. Jia. PSANet: Point-wise spatial attention network for scene parsing. *ECCV*, 2018. 2

[57] S. Zhao, Y. Wang, Z. Yang, and D. Cai. Region mutual information loss for semantic segmentation. *NeurIPS*, 2019. 2

[58] Shuai Zheng, S. Jayasumana, B.R. Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P.H.S. Torr. Conditional random fields as recurrent neural networks. *ICCV*, 2015. 1

[59] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. *CVPR*, 2017. 6

[60] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba. Semantic understanding of scenes through ade20k dataset. *IJCV*, 2016. 6