Part 2: Residual Neural Fabrics

Suriya Singh [*]
IIIT Hyderabad

6. Related works

Recently, Bansal et al. [2] showed that having feature representation for each pixel (instead of upsampling and fusion of predictions) is helpful for semantic segmentation task. Their technique use hypercolumn features [9] from multiple scales of VGG model [24]. Our proposed forward connections can be interpreted as hypercolumn features from different layers of same scale. Bansal et al. [2] showed training from scratch by random sampling of pixels with pixel-wise MLP classifier for semantic segmentation task and achieved 48.7% on PASCAL VOC 2012 validation set. Compared to ours, their method is superior by 3.037%. The superior performance is probably due to pixel sampling to avoid class imbalance, use of very deep features, and use of more complex, non-linear pixel-wise classifier.

Fourure et al. [6] proposed GridNet, a fabric like network consisting of encoder (type B fabric units) and decoder (type B unit with all downsampling path replaced with upsampling path) phases. Another difference between fabric network and GridNet is that pre-activation bottleneck units [13] with dilated convolutions are used in GridNet. Their work shows training GridNet from scratch on CityScape dataset for 800 epochs using channel doubling across five scales, 3 downsampling and 3 upsampling streams (layers).

7. PyTorch *vs* Caffe Implementations

Config	**Input**	**Library**	**Params**	**GPU Mem**	**Train Time/Iter**	**Result**
			($\times 10^6$)	(in MiB)	(sec / 64 samples)	%
A – $8 \times 6 \times 128$	32×32	Caffe	18.022	4618	2.229	6.35
		PyTorch		**1979**	**0.534**	6.21
D – $8 \times 6 \times 128$	32×32	Caffe	22.449	6585	2.634	6.87
		PyTorch		**2219**	**0.585**	6.45
D – $8 \times 6 \times \{32, 64, ..., 1024\}$	256×256	Caffe	226.416	7993	41.453	—
		PyTorch		**2065**	**16.762**	—

Table 3: Comparisons of various parameters for Caffe and PyTorch implementations of Neural Fabrics. The experiments are performed on CIFAR10 dataset and PASCAL VOC 2012. Results are reported in terms of error rate for CIFAR10. The experiment protocol follows [23]. (Results for PASCAL VOC 2012 will be updated soon.) He init [11], bilinear interpolation for upsampling, and no forward connection is used and classifier is deployed at only one scale. Batch size of 64 for CIFAR10 and 1 for PASCAL VOC (with parameter update every 64 iterations). GPU used is NVIDIA Geforce GTX 1080 Ti with CUDA-8.0 and cuDNN-5.1. cuDNN backend is used wherever available. SGD solver is used for optimisation.

I start this part by reproducing results of previous experiments while using out-of-the-box Caffe library. We have previously observed that Caffe replicates layers' outputs when it is shared by multiple layers (downsample, upsample, forward, cropping operations, and forward connections). This results in upto 6× extra GPU usage. I re-implemented neural fabric's wrapper provided by Saxena and Verbeek [23] for PyTorch. For PyTorch implementation at every node the incoming activations are summed and applied BatchNorm and ReLU activations. The output of node operations are then used in subsequent fabric operations (downsample, upsample, and forward) sequentially and their outputs are saved at their respective target nodes.

[*] First Part done as Research Intern at Xerox Research Centre Europe
[*] Work continued independently at CVIT, IIITH

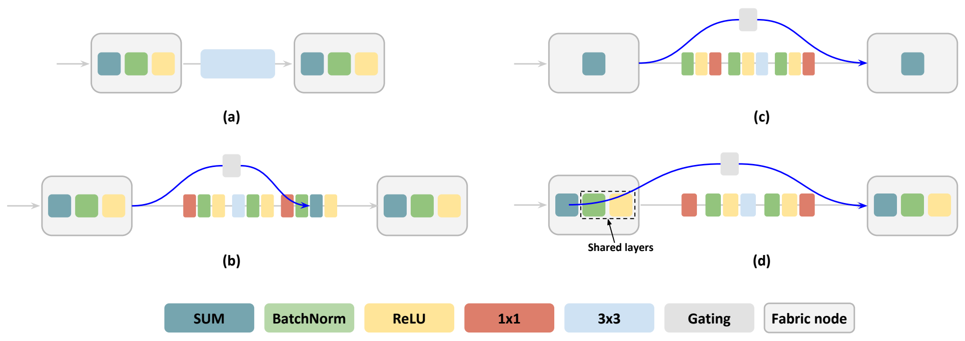


Figure 9: (a) A convolution unit in fabric networks [23]. (b) Post-activation and (c) Pre-activation bottleneck unit with skip connections. (d) Pre-activation bottleneck unit in which BatchNorm and ReLU units are shared by multiple connections. Skip connections are shown as blue arrows. Note that only one path per node has been shown for the sake of clarity.

The sequential fabric operations while being fast also saves memory since there is no data duplication. For Caffe implementation, a considerable amount of time and memory is spent for data duplication which is used in fabric operations (downsample, upsample, and forward). The memory and performance comparisons between Caffe and PyTorch implementations are provided in Table 3. PyTorch offers $3\times$ to $5\times$ speed improvement while saving $2.3\times$ to $3.8\times$ memory overhead. Subsequent experiments are done using PyTorch implementation.

8. Bottleneck Units in Neural Fabrics

Inspired by ResNets [12, 13] and Highway Networks [25], we explored using skip connections in fabric (part I). For this, we add a skip connection from 4^{th} previous layers into the fabric node during summation of other incoming connections and BatchNorm+ReLU is applied. However, we did not observe any improvement with this modification. As also pointed out by He et al. [13] (Table 2), BatchNorm after addition of skip connection results in worse performance. They explained, this is because BatchNorm alters the action maps that comes from skip connection which impedes information propagation and cause difficulty in training.

In this part, the convolution units have been replaced with bottleneck units [12, 13]. A bottleneck unit consists of a series of 3 convolutions (see Figure 9 (b) and (c)) viz., 1×1 conv which performs dimensionality reduction of incoming action maps by 4 times, 3×3 conv, and 1×1 conv which projects the feature dimension back to original dimensions. For downsampling path, 3×3 conv with stride of 2 are used and for upsampling path the incoming activation maps are bilinearly upsampled before applying bottleneck operations. The convolution layers (except for classifier) do not have bias to avoid redundancy as BatchNorm already has the bias term.

Consider a fabric network F of L layers and S scales with bottleneck units. The series of convolution, BatchNorm, and ReLU operations inside a bottleneck unit is denoted as $F(x, \theta)$. Let $B^d_{l,s}$ be an output of a bottle unit and $N_{l,s}$ of the fabric node, where $1 \leq l \leq L$, $1 \leq s \leq S$, and $d \in \{forward, up, down\}$.

$$N_{l,s} = z(B^{down}_{l-1,s-1} + B^{forward}_{l-1,s} + B^{up}_{l-1,s+1})$$
$$B^d_{l,s} = f(N_{l,s} + F(N_{l,s}, \theta^d_{l,s}))$$

For a post-activation bottleneck unit,

$$z(x) = ReLU(BatchNorm(x))$$
$$f(x) = ReLU(x)$$

whereas, for a pre-activation bottleneck unit,

$$z(x) = x$$
$$f(x) = x$$

For a convolution connection in regular fabric, the number of parameters is given by $C_{in} \cdot C_{out} \cdot 3^2$. Replacing this with a corresponding bottleneck unit would make parameters count (ignoring BatchNorm) to be $C_{in} \cdot \frac{C_{out}}{4} + \frac{C_{out}}{4} \cdot \frac{C_{out}}{4} \cdot 3^2 + \frac{C_{out}}{4} \cdot C_{out}$. When C_{in} and C_{out} are equal, this amount to reduction in parameters by a factor of $8\frac{8}{17}$.

Config	**Fabric Unit**	**Activations**	**Params** ($\times 10^6$)	**GPU Mem** (in MiB)	**Train Time/Iter** (sec / 64 samples)	**Result** %
A – 16 × 6 × 64	sparse conv	Post	2	NA	NA	18.89 [23]
A – 8 × 6 × 128	conv	Post	21.2	NA	NA	7.43 [23]
A – 9 × 6 × 128	Conv	Post	20.367	2171	0.548	5.99
A – $\{3 \times 3\} \times 6 \times 128$	Bottleneck	Post	0.756	1735	0.216	11.94
	Bottleneck	Pre	0.752	1239	0.198	12.77
A – $\{9 \times 3\} \times 6 \times 128$	Bottleneck	Post	2.473	4475	0.731	9.80
	Bottleneck	Pre	2.460	2571	0.658	11.89

Table 4: Comparisons between convolution, post-activation bottleneck and pre-activation bottleneck fabric units on CIFAR10. Each bottleneck module consists of 3 convolutional layers. Fabric configuration is of format $Layers \times Scales \times Channels$ in case of conv units and $\{Layers \times 3\} \times Scales \times Channels$ for bottleneck units.

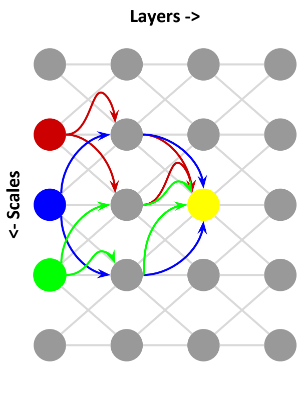
post-activations bottleneck [12]: In this variant, BatchNorm+ReLU is applied after every convolution operations. The skip connection is added before last ReLU operation within the block (see Figure 9 (b)). This model has 2.6× fewer parameters compared to sparse neural fabric reported in [23] while achieving 6.95% improvement (see Table 4).

pre-activations bottleneck [13]: In this variant, the BatchNorm+ReLU activations are applied before convolutions. The skip connection is added after the last convolution operation (see Figure 9 (c)). The authors of [13] explains that since no activation is applied on skip connection, the signal can pass through this path between any two units. Pre-activations have also shown to reduce overfitting.

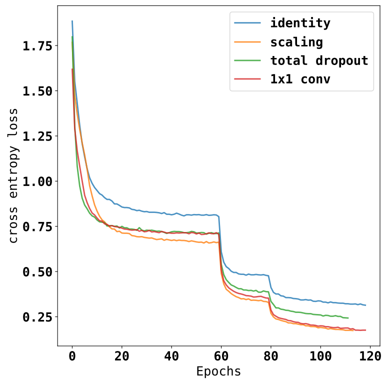
Performance comparisons can be seen in Table 4. The experiments show that post-activation bottleneck unit while being slower is slightly better (0.83%) than pre-activation unit. This is probably due to $f(x) = x$, multiple copies of same signal can freely reach target node via skip connections (see Figure 10a).

Gating in residual neural fabrics: At a fabric node, there are multiple incoming connections from other layers and/or scales. Naively replacing all convolution with bottleneck results in skip connections added multiple times which is equivalent to scalar multiplication to each skip connections (see Figure 10a). As noted by He et al. [13] (*eqn 8*), gradient tends to explode when this scalar is greater than 1 and vanish when it is lesser. To overcome this, three different gating mechanisms (inspired from [13, 25]) are investigated keeping in mind that only one copy of skip connection reaches at target node.

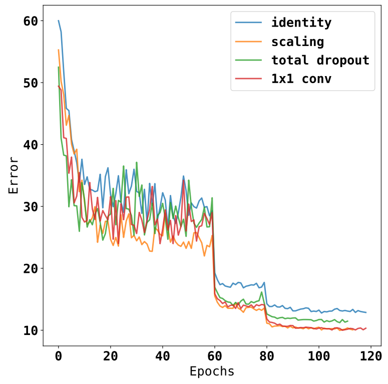
Constant Scaling: Each skip connection from a node is multiplied by a constant scalar. This scalar is fixed to reciprocal of number of outgoing skip connections. This causes summation of duplicate skip connections at target node to be close to single skip connection.



(a)



(b)



(c)

Figure 10: (a) A neural fabric with skip identity connections. Red, blue and green arrows show how multiple copies of same signals reach yellow node via different skip connection paths. (b) Training loss and (c) validation error *vs* epoch for different gating mechanisms with pre-activation bottleneck unit.

Config	**Activation**	**Gating**	**Params** $(\times 10^6)$	**GPU Mem** (in MiB)	**Train Time/Iter** (sec / 64 samples)	**Result** %
$A - \{3 \times 3\} \times 6 \times 128$	Post	None	0.756	1735	0.216	11.94
	Post	Scaling	0.756	1735	0.216	12.48
	Post	Total Dropout	0.756	1735	0.216	13.18
	Post	1×1 conv	1.455	1901	0.249	**10.44**
	Pre	None	0.752	1239	0.198	12.77
	Pre	Scaling	0.752	1239	0.198	**9.96**
	Pre	Total Dropout	0.752	1239	0.198	11.22
	Pre	1×1 conv	1.451	1435	0.230	**9.99**

Table 5: Comparisons between different gating mechanisms in residual fabric networks.

Total Dropout: In this variant, a skip connection is randomly set to zero. The probability of any skip connection remaining active is set to reciprocal of of number of outgoing skip connections. This causes summation of all skip connections at target node to be close to one instance of skip connection on average. This solution incurs no extra parameter. Note the difference that, for 'dropout shortcut' in [13], few random elements (of all skip connections) are set to zero. In this experiment, all elements (of a random skip connection) are set to zero. The inferior results could be because the branch carrying signals from the previous skip connections might get dropped and hence hinder the gradients to pass through this path.

$\mathbf{1 \times 1}$ **conv** : 1×1 convolution followed by BatchNorm is applied on skip connections (also known as 'conv shortcut' in [13]). The weights to each channel of each skip connection is learned independently. This is also crucial for channel doubling variant where number of filters changes across scales. This architecture allows the fabric to learn/embed various residual networks architecture (ResNets [12, 13], Highway Networks [25], GridNet [6], FRRN [22] etc.). By verifying the weights values of 1×1 conv, it could be visualized whether the skip connection between set of layers is needed or not.

Table 5 shows results of different gating mechanism when applied to post- and pre- activation bottleneck units. Pre-activation units have significant benefits when used with scaling or 1×1 conv gating. Figure 10b and 10c shows the training plots of these experiments for pre-activation bottleneck unit. It can be seen that gating mechanisms allow the model to converge to a better training loss. There are other possible ways for avoiding the redundancy of skip connections. One could allow skip connections for connections at same scale and not for across scale. This solution is chosen by Fourure et al. [6]. Another way is to allow for skip connections across scale only and not for within scale connections. However, both solutions would results in fragmented strands of networks with skip connections connected to other strands with convolution. Such solutions use specially designed variety of nodes and connections instead of allowing the network to learn the best suited architecture.

References

[1] V. Badrinarayanan, A. Handa, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. In *CoRR, abs/1505.07293*, 2015. 1, 3

[2] A. Bansal, X. Chen, B. Russell, A. Gupta, and D. Ramanan. Pixelnet: Representation of the pixels, by the pixels, and for the pixels. In *CoRR abs/1702.06506*, 2017. 12

[3] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015. 1, 2, 3, 4

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 1

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html. 5

[6] D. Fourure, R. Emonet, E. Fromont, D. Muselet, A. Trémeau, and C. Wolf. Residual conv-deconv grid network for semantic segmentation. In *BMVC*, 2017. 12, 15

[7] G. Ghiasi and C. C. Fowlkes. Laplacian pyramid reconstruction and refinement for semantic segmentation. In *ECCV*, 2016. 1

[8] B. Hariharan, P. Arbelaez, L. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *International Conference on Computer Vision (ICCV)*, 2011. 5

[9] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. In *CVPR*, 2015. 12

[10] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *CoRR, abs/1703.06870*, 2017. 6

[11] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 2, 5, 6, 12

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 2, 13, 14, 15

[13] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 12, 13, 14, 15

[14] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *CVPR*, 2017. 2, 16

[15] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *CoRR, abs/1611.09326*, 2016. 1, 2

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACMMM*, 2014. 6

[17] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *CoRR, abs/1412.6980*, 2014. 6

[18] P. Krähenbühl and V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *NIPS*, 2011. 1, 4

[19] G. Lin, A. Milan, C. Shen, and I. Reid. RefineNet: Multi-path refinement networks for high-resolution semantic segmentation. In *CVPR*, 2017. 1, 2

[20] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional models for semantic segmentation. In *CVPR*, 2015. 1, 3, 4, 5

[21] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015. 1, 3

[22] T. Pohlen, A. Hermans, M. Mathias, and B. Leibe. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *CoRR, abs/1611.08323*, 2016. 15

[23] S. Saxena and J. Verbeek. Convolutional neural fabrics. In *NIPS*, 2016. 1, 2, 3, 12, 13, 14

[24] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1, 2, 6, 12

[25] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *ICML*, 2015. 13, 14, 15

[26] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell. Understanding convolution for semantic segmentation. In *CoRR, abs/1702.08502*, 2017. 2, 4

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