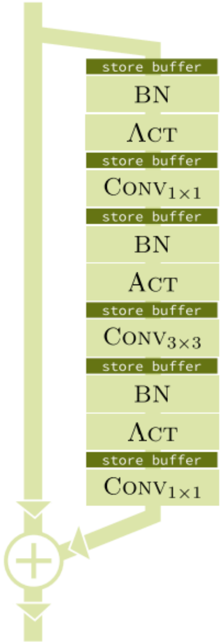
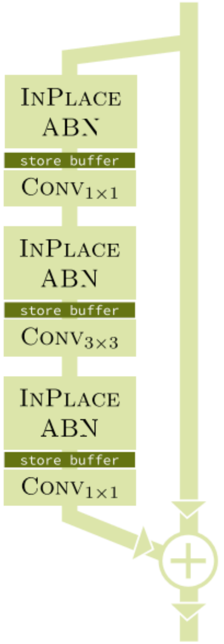


In-Place Activated BatchNorm for Memory-Optimized Training of DNNs



Samuel Rota Bulò, Lorenzo Porzi, Peter Kontschieder
Mapillary Research

Paper: https://arxiv.org/abs/1712.02616

Code: https://github.com/mapillary/inplace_abn

CSC2548, 2018 Winter

Harris Chan

Jan 31, 2018

Overview

- Motivation for Efficient Memory management
- Related Works
  - Reducing precision
  - Checkpointing
  - Reversible Networks [9] (Gomez et al., 2017)
- In-Place Activated Batch Normalization
  - Review: Batch Normalization
  - In-place Activated Batch Normalization
- Experiments
- Future Directions

Overview

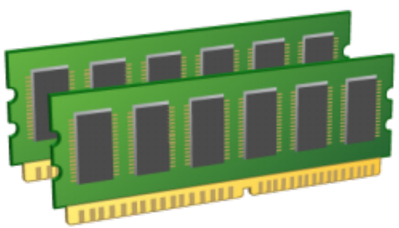
- Motivation for Efficient Memory management
- Related Works
  - Reducing precision
  - Checkpointing
  - Reversible Networks [9] (Gomez et al., 2017)
- In-Place Activated Batch Normalization
  - Review: Batch Normalization
  - In-place Activated Batch Normalization
- Experiments
- Future Directions

Why Reduce Memory Usage?

- Modern computer vision recognition models use deep neural networks to extract features
- Depth/width of networks ~ GPU memory requirements
  - Semantic segmentation: may even only do just a single crop per GPU during training due to suboptimal memory management
- More efficient memory usage during training lets you:
  - Train larger models
  - Use bigger batch size / image resolutions
- This paper focuses on **increasing memory efficiency of the training process** of deep network architectures at the expense of **small additional computation time**

Approaches to Reducing Memory

Increasing
Computation
Time



Reduce Memory by…

Reducing
Precision
(& Accuracy)

Overview

- Motivation for Efficient Memory management
- Related Works
  - Reducing precision
  - Checkpointing
  - Reversible Networks [9] (Gomez et al., 2017)
- In-Place Activated Batch Normalization
  - Review: Batch Normalization
  - In-place Activated Batch Normalization
- Experiments
- Future Directions

Related Works:
Reducing Precision

Work	Weight	Activation	Gradients
BinaryConnect (M. Courbariaux et al. 2015)	Binary	Full Precision	Full Precision
Binarized neural networks (I. Hubara et al. 2016)	Binary	Binary	Full Precision
Quantized neural networks (I. Hubara et al)	Quantized 2,4,6 bits	Quantized 2,4,6 bits	Full Precision
Mixed precision training (P. Micikevicius et al. 2017)	Half Precision (fwd/bw) & Full Precision (master weights)	Half Precision	Half Precision

Related Works:
Reducing Precision

- **Idea:** During training, lower the precision (up to binary) of the weights / activations / gradients

Strength	Weakness
Reduce memory requirement and size of the model	**Often decrease in accuracy** performance (newer work attempts to address this)
Less power: efficient forward pass	
Faster: 1-bit XNOR-count vs. 32-bit floating point multiply	

Related Works:
Computation Time

- **Checkpointing:** trade off memory with computation time
- **Idea:** During backpropagation, store a subset of activations ("checkpoints") and recompute the remaining activations as needed
- Depending on the architecture, we can use different strategies to figure out which subsets of activations to store

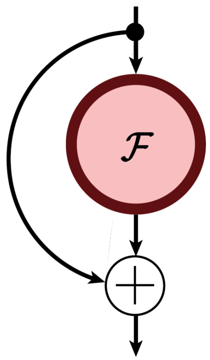
Related Works: Computation Time

- Let L be the number of identical feed-forward layers:

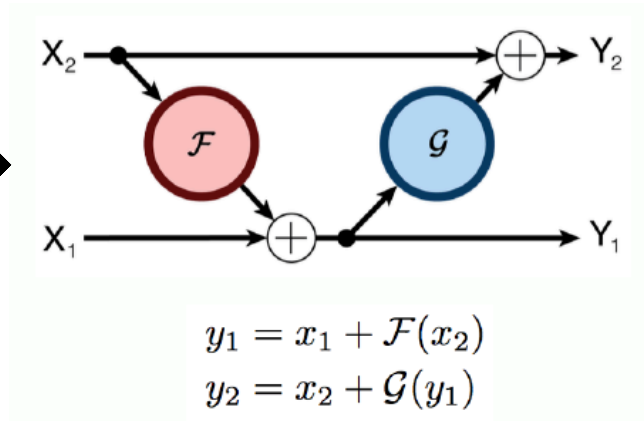
Work	Spatial Complexity	Computation Complexity
Naive	$O(L)$	$O(L)$
Checkpointing (Martens and Sutskever, 2012)	$O(\sqrt{L})$	$O(L)$
Recursive Checkpointing (T. Chen et al., 2016)	$O(\log L)$	$O(L \log L)$
Reversible Networks (Gomez et al., 2017)	$O(1)$	$O(L)$

Table adapted from Gomez et al., 2017. "The Reversible Residual Network: Backpropagation Without Storing Activations". [ArXiv Link]

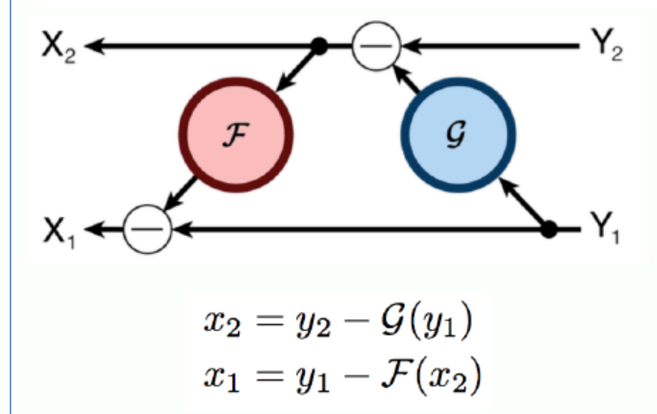
Related Works: Computation Time Reversible ResNet (Gomez et al., 2017)



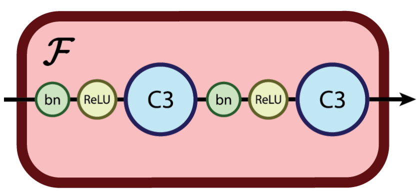
Residual Block



RevNet (Forward)



RevNet (Backward)



Basic Residual Function

Idea: Reversible Residual module allows the current layer's activation to be reconstructed exactly from the next layer's. No need to store any activations for backpropagation!

Gomez et al., 2017. "The Reversible Residual Network: Backpropagation Without Storing Activations". ArXiv Link

Related Works: Computation Time Reversible ResNet (Gomez et al., 2017)

Advantage

- No noticeable loss in performance
- Gains in network depth: ~600 vs ~100
- 4x increase in batch size (128 vs 32)

Disadvantage

- Runtime cost: 1.5x of normal training (sometimes less in practice)
- Restrict reversible blocks to have a stride of 1 to not discard information (i.e. no bottleneck layer)

Table 3: Classification error on CIFAR

Architecture	CIFAR-10 [15]		CIFAR-100 [15]	
	ResNet	RevNet	ResNet	RevNet
32 (38)	**7.14%**	7.24%	29.95%	**28.96%**
110	**5.74%**	5.76%	26.44%	**25.40%**
164	5.24%	**5.17%**	**23.37%**	23.69%

Table 4: Top-1 classification error on ImageNet (single crop)

ResNet-101	RevNet-104
23.01%	23.10%

Gomez et al., 2017. "The Reversible Residual Network: Backpropagation Without Storing Activations". ArXiv Link

Overview

- Motivation for Efficient Memory management
- Related Works
  - Reducing precision
  - Checkpointing
  - Reversible Networks [9] (Gomez et al., 2017)
- In-Place Activated Batch Normalization
  - Review: Batch Normalization
  - In-place Activated Batch Normalization
- Experiments
- Future Directions

Review: Batch Normalization (BN)

- Apply BN on current features (x_i) across the mini-batch
- Helps reduce internal covariate shift & accelerate training process
- Less sensitive to initialization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1 \ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

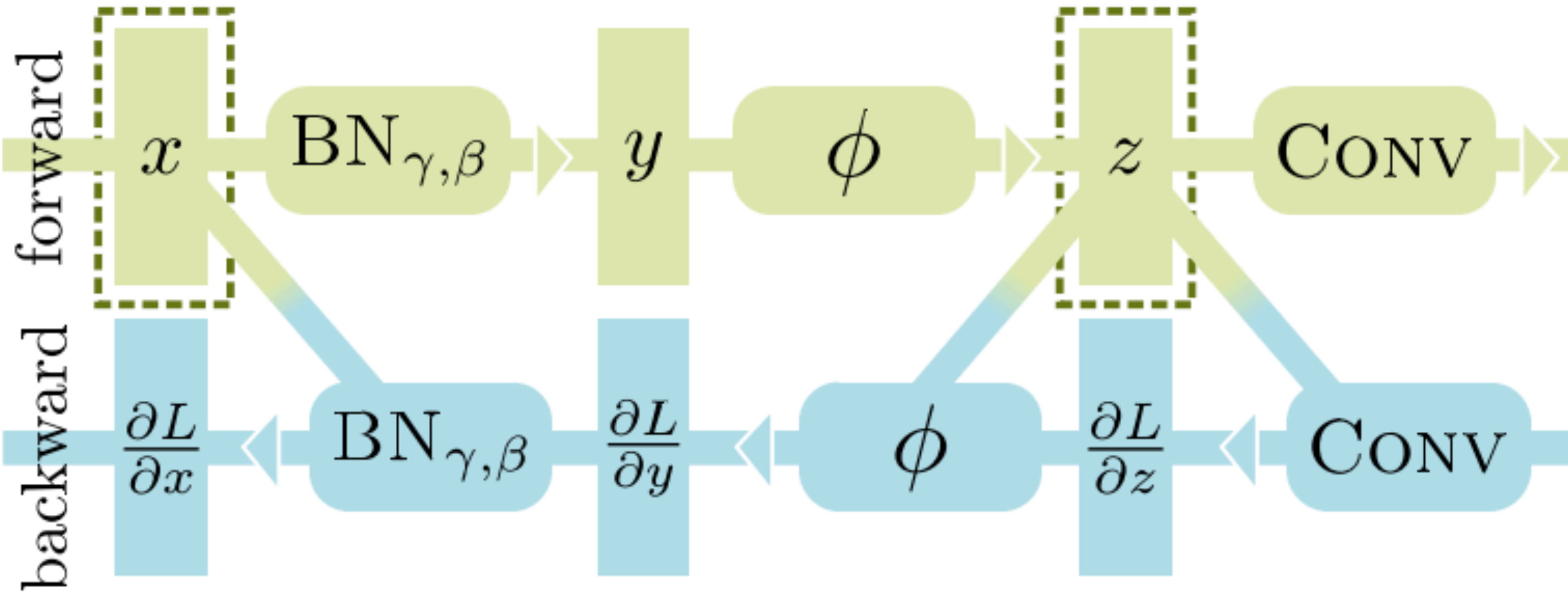
Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Credit: Ioffe & Szegedy, 2015. [ArXiv link]

Memory Optimization Strategies

- Let’s compare the various strategies for BN+Act:
    1. Standard
    2. Checkpointing (baseline)
    3. Checkpointing (proposed)
    4. In-Place Activated Batch Normalization I
    5. In-Place Activated Batch Normalization II

1: Standard BN Implementation



(a) Standard building block (memory-inefficient)

Gradients for Batch Normalization

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

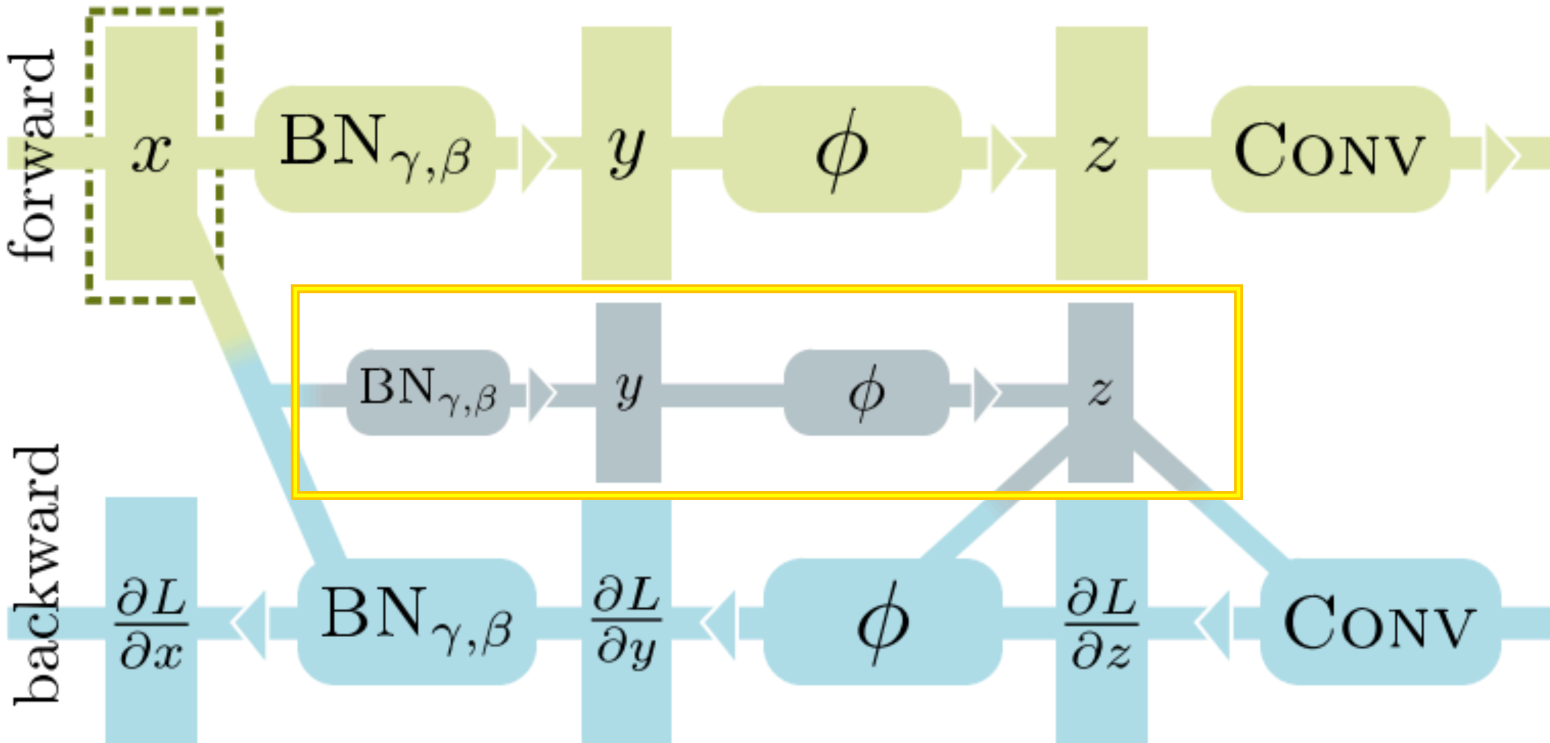
$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\boxed{\sigma_{\mathcal{B}}^2} + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \boxed{\mu_{\mathcal{B}}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \boxed{\widehat{x}_i}$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

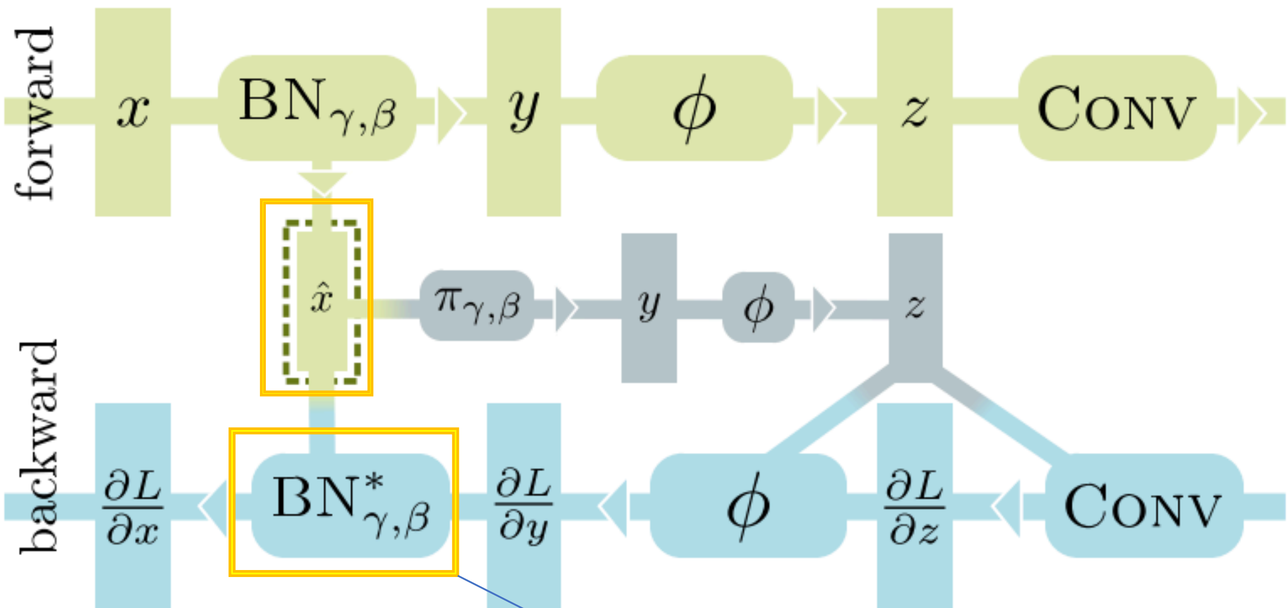
Credit: Ioffe & Szegedy, 2015. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. [ArXiv link]

2: Checkpointing (baseline)



(b) Checkpointing [4, 21]

3: Checkpointing (Proposed)



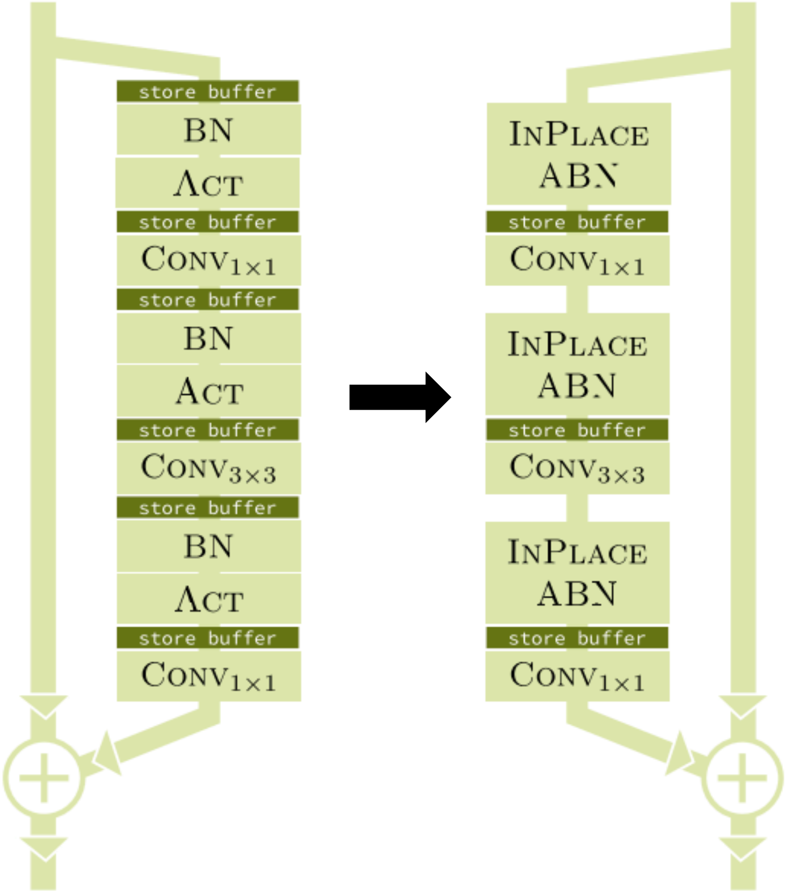
(c) Checkpointing (proposed version)

$$\frac{\partial L}{\partial x_i} = \left\{ \frac{\partial L}{\partial y_i} - \frac{1}{m}\frac{\partial L}{\partial \gamma}\boxed{\hat{x}_i} - \frac{1}{m}\frac{\partial L}{\partial \beta} \right\} \frac{\gamma}{\sqrt{\boxed{\sigma_{\mathcal{B}}^2} + \epsilon}}$$

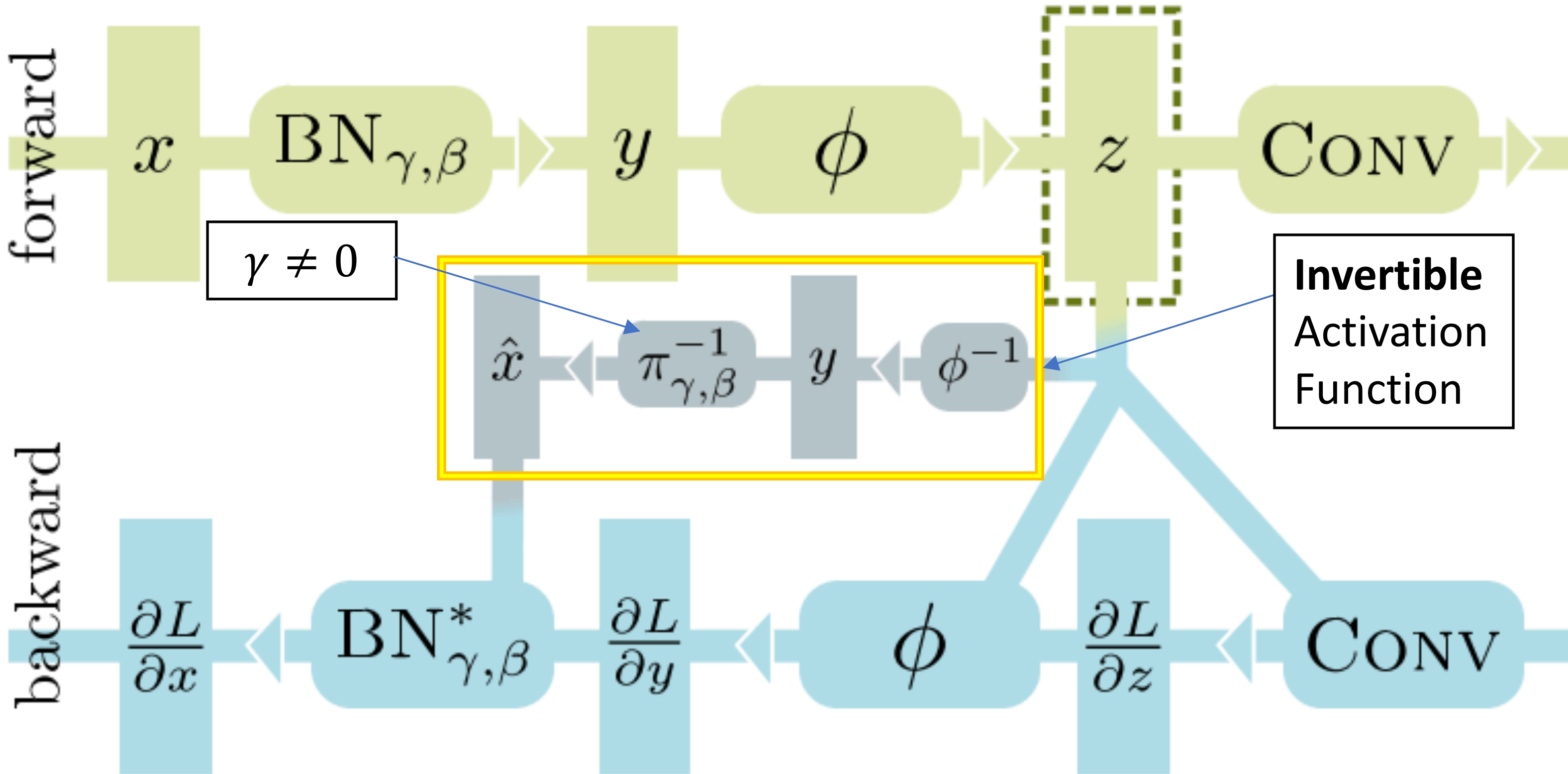
$$\frac{\partial L}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i}\boxed{\hat{x}_i}, \quad \frac{\partial L}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i}$$

In-Place ABN

- Fuse batch norm and activation layer to enable in-place computation, using only a single memory buffer to store results.
- Encapsulation makes it easy to implement and deploy
- Implemented INPLACE ABN-I layer in PyTorch as a new module



4: In-Place ABN I (Proposed)



(d) In-Place Activated Batch Normalization I (proposed method)

Leaky ReLU is Invertible

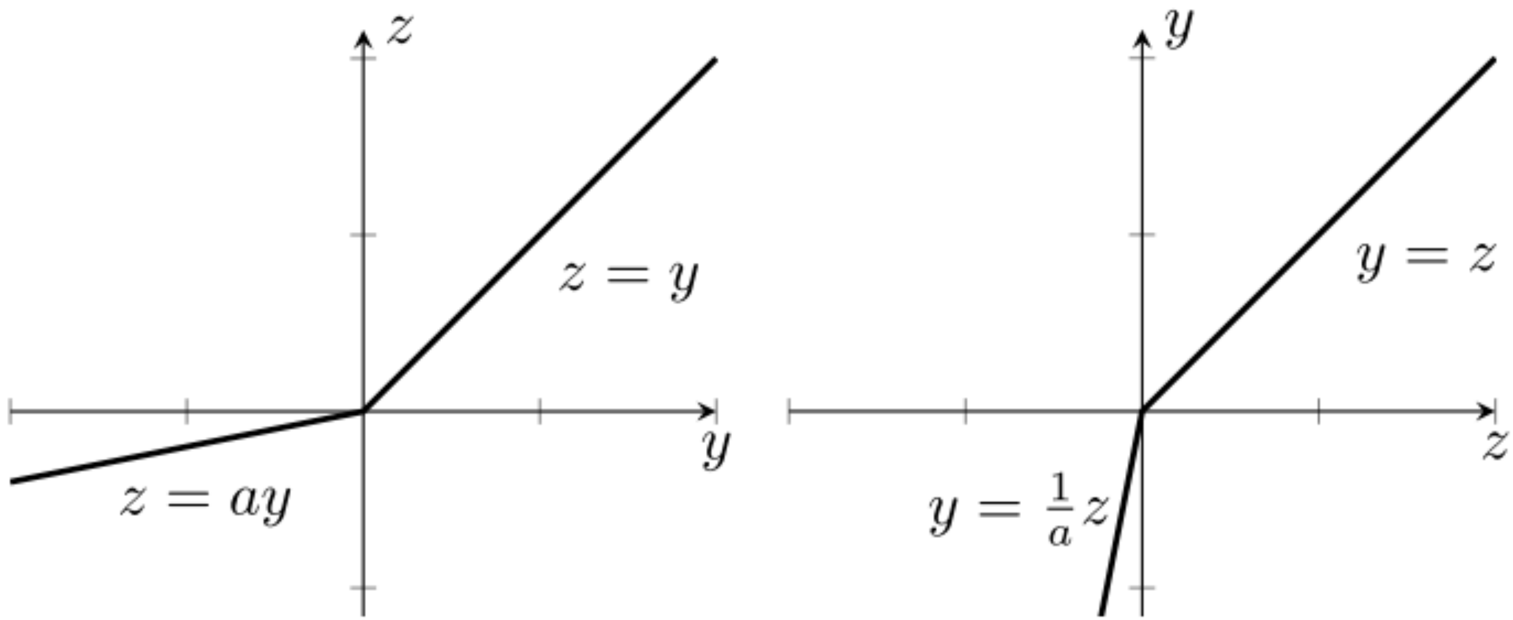
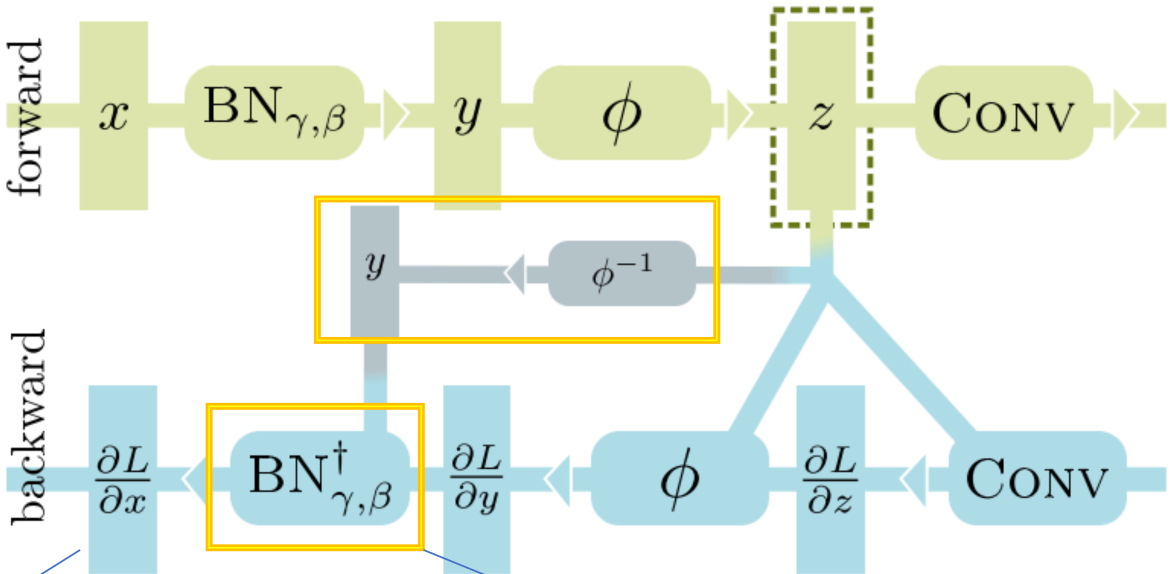


Figure 3. LEAKY RELU with slope a (left) and its inverse (right).

5: In-Place ABN II (Proposed)



(e) In-Place Activated Batch Normalization II (proposed method)

$$\frac{\partial L}{\partial x_i} = \left[\frac{\partial L}{\partial y_i} - \frac{1}{\gamma m}\frac{\partial L}{\partial \gamma}\boxed{y_i} - \frac{1}{m}\left(\frac{\partial L}{\partial \beta} + \frac{\beta}{\gamma}\frac{\partial L}{\partial \gamma}\right)\right]\frac{\gamma}{\sqrt{\boxed{\sigma_{\mathcal{B}}^2} + \epsilon}}$$

$$\frac{\partial L}{\partial \gamma} = \frac{1}{\gamma}\left[\sum_{j=1}^{m}\frac{\partial L}{\partial y_j}\boxed{y_j} - \beta\frac{\partial L}{\partial \beta}\right]$$

Strategies Comparisons

Strategy	Store	Computation Overhead
Standard	$\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\sigma}_{\mathcal{B}}, \boldsymbol{\mu}_{\mathcal{B}}$	-
Checkpointing	$\boldsymbol{x}, \boldsymbol{\sigma}_{\mathcal{B}}, \boldsymbol{\mu}_{\mathcal{B}}$	$BN_{\gamma,\beta}, \phi$
Checkpointing (proposed)	$\boldsymbol{x}, \boldsymbol{\sigma}_{\mathcal{B}}$	$\pi_{\gamma,\beta}, \phi$
In-Place ABN I (proposed)	$\boldsymbol{z}, \boldsymbol{\sigma}_{\mathcal{B}}$	$\phi^{-1}, \pi_{\gamma,\beta}^{-1}$
In-Place ABN II (proposed)	$\boldsymbol{z}, \boldsymbol{\sigma}_{\mathcal{B}}$	ϕ^{-1}

In-Place ABN (Proposed)

Algorithm 1 InPlace-ABN Forward

Require: x, γ, β

$y, \sigma_{\mathcal{B}} \leftarrow \mathrm{BN}_{\gamma,\beta}(x)$

$z \leftarrow \phi(y)$

save for backward $z, \sigma_{\mathcal{B}}$

return z

Algorithm 2 InPlace-ABN Backward

Require: $\frac{\partial L}{\partial z}, \gamma, \beta$

$z, \sigma_{\mathcal{B}} \leftarrow$ saved tensors during forward

$\frac{\partial L}{\partial y} \leftarrow \phi_{\text{backward}}(z, \frac{\partial L}{\partial z})$

$y \leftarrow \phi^{-1}(z)$

if InPlace-ABN I (see Fig. 2(d)) **then**

$\quad \hat{x} \leftarrow \boxed{\pi^{-1}_{\gamma,\beta}(y)}$

$\quad \frac{\partial L}{\partial x}, \frac{\partial L}{\partial \gamma}, \frac{\partial L}{\partial \beta} \leftarrow \mathrm{BN}^{*}_{\gamma,\beta}(\boxed{\hat{x}, \frac{\partial L}{\partial y}, \sigma_{\mathcal{B}}})$

else if InPlace-ABN II (see Fig. 2(e)) **then**

$\quad \frac{\partial L}{\partial x}, \frac{\partial L}{\partial \gamma}, \frac{\partial L}{\partial \beta} \leftarrow \mathrm{BN}^{\dagger}_{\gamma,\beta}(\boxed{y, \frac{\partial L}{\partial y}, \sigma_{\mathcal{B}}})$

return $\frac{\partial L}{\partial x}, \frac{\partial L}{\partial \gamma}, \frac{\partial L}{\partial \beta}$

In-Place ABN (Proposed)

Strength	**Weakness**
Reduce memory requirement by **half** compared to standard; same savings as check pointing	Requires **invertible activation function**
Empirically **faster than naïve checkpointing**	…but still slower than standard (memory hungry) implementation.
Encapsulating BN & Activation together makes it easy **to implement and deploy (plug & play)**	

Overview

- Motivation for Efficient Memory management
- Related Works
  - Reducing precision
  - Checkpointing
  - Reversible Networks [9] (Gomez et al., 2017)
- In-Place Activated Batch Normalization
  - Review: Batch Normalization
  - In-place Activated Batch Normalization
- Experiments
- Future Directions

Experiments: Overview

- 3 Major types:
  - Performance on: (1) **Image Classification**, (2) **Semantic Segmentation**
  - (3) **Timing Analysis** compared to standard / checkpointing
- Experiment Setup:
  - NVIDIA Titan Xp (12 GB RAM/GPU)
  - PyTorch
  - Leaky ReLU activation

Experiments: Image Classification

	ResNeXt-101/ResNeXt-152	WideResNet-38
Dataset	ImageNet-1k	ImageNet-1k
Description	Bottleneck residual units are replaced with a multi-branch version = “cardinality” of 64	More feature channels but shallower
Data Augmentation	Scale smallest side = 256 pixels then randomly crop 224 × 224, per-channel mean and variance normalization	(Same as ResNeXt-101/152)
Optimizer	• SGD with Nesterov Updates • Initial learning rate=0.1 • weight decay=10^{-4} • momentum=0.9 • 90 Epoch, reduce by factor of 10 per 30 epoch	• (Same as ResNeXt) • 90 Epoch, linearly decreasing from 0.1 to 10^{-6}

Experiments: Leaky ReLU impact

Network	activation		224^2 center		224^2 10-crops		320^2 center	
	training	validation	top-1	top-5	top-1	top-5	top-1	top-5
ResNeXt-101	ReLU	ReLU	77.74	93.86	79.21	94.67	79.17	94.67
ResNeXt-101	ReLU	Leaky ReLU	76.88	93.42	78.74	94.46	78.37	94.25
ResNeXt-101	Leaky ReLU	Leaky ReLU	77.04	93.50	78.72	94.47	77.92	94.28
ResNeXt-101	Leaky ReLU	ReLU	76.81	93.53	78.46	94.38	77.84	94.20

Table 1. Imagenet validation set results using ResNeXt-101 and ReLU/Leaky ReLU exchanged activation functions during training and validation.

- Using Leaky ReLU performs slightly worse than with ReLU
- Within ~1% , except for 320^2 center crop—authours argued it was due to non-deterministic training behaviour
- Weaknesses
  - Showing an average + standard deviation can be more convincing of the improvements.

Experiments: Exploiting Memory Saving

	Network	batch size	224^2 center		224^2 10-crops		320^2 center	
			top-1	top-5	top-1	top-5	top-1	top-5
Baseline	ResNeXt-101, STD-BN	256	77.04	93.50	78.72	94.47	77.92	94.28
1) Larger Batch Size	ResNeXt-101, INPLACE-ABN	512	78.08	93.79	79.52	94.66	79.38	94.67
2) Deeper Network	ResNeXt-152, INPLACE-ABN	256	78.28	94.04	79.73	94.82	79.56	94.67
3) Larger Network	WideResNet-38, INPLACE-ABN	256	79.72	94.78	81.03	95.43	80.69	95.27
4) Sync BN	ResNeXt-101, INPLACE-ABN^{sync}	256	77.70	93.78	79.18	94.60	78.98	94.56

Table 2. Imagenet validation set results using different architectures and training batch sizes.

- Performance increase for 1-3
- Similar performance with larger batch size vs deeper model (1 vs 2)
- Synchronized INPLACE-ABN did not increase the performance that much
  - Notes on synchronized BN: http://hangzh.com/PyTorch-Encoding/notes/syncbn.html

Experiments: Semantic Segmentation

- **Semantic Segmentation**: Assign categorical labels to each pixel in an image
- **Datasets**
  - CityScapes
  - COCO-Stuff
  - Mapillary Vistas

Figure Credit: https://www.cityscapes-dataset.com/examples/

Experiments: Semantic Segmentation

- Architecture contains 2 parts that are jointly fine-tuned on segmentation data:
  - **Body:** Classification models pre-trained on ImageNet
  - **Head**: Segmentation specific architectures
- Authours used **DeepLabV3*** as the head
  - Cascaded atrous (dilated) convolutions for capturing contextual info
  - Crop-level features encoding global context
- Maximize GPU Usage by:
  - (**FIXED CROP**) fixing the training crop size and therefore pushing the amount of crops per minibatch to the limit
  - (**FIXED BATCH**) fixing the number of crops per minibatch and maximizing the training crop resolutions

*L. Chen, G. Papandreou, F. Schroff, and H. Adam. "Rethinking atrous convolution for semantic image segmentation." ArXiv Link

Experiments: Semantic Segmentation

BatchNorm	ResNeXt-101				WideResNet-38			
	Cityscapes		COCO-Stuff		Cityscapes		COCO-Stuff	
Std-BN + Leaky ReLU	16×512^2	74.42	16×480^2	20.30	20×512^2	75.82	20×496^2	22.44
InPlace-ABN, fixed crop	28×512^2 **[+75%]**	75.80	24×480^2 **[+50%]**	22.63	28×512^2 **[+40%]**	77.75	28×496^2 **[+40%]**	22.96
InPlace-ABN, fixed batch	16×672^2 **[+72%]**	77.04	16×600^2 **[+56%]**	23.35	20×640^2 **[+56%]**	**78.31**	20×576^2 **[+35%]**	24.10
InPlace-ABN^{sync}, fixed batch	16×672^2 **[+72%]**	**77.58**	16×600^2 **[+56%]**	**24.91**	20×640^2 **[+56%]**	78.06	20×576^2 **[+35%]**	**25.11**

Table 3. Validation data results (single scale test) for semantic segmentation experiments on Cityscapes and COCO-Stuff, using ResNeXt-101 and WideResNet-38 network bodies and different batch normalization settings (see text). All result numbers in [%].

- More training data **(FIXED CROP)** helps a little bit
- Higher input resolution **(FIXED BATCH)** helps even more than adding more crops

- No qualitative result: probably visually similar to DeepLabV3

Experiments: Semantic Segmentation Fine-Tuned on CityScapes and Mapillary Vistas

	ResNeXt-152		WideResNet-38	
Cityscapes				
INPLACE-ABN^{sync}	12×680^2	78.49	–	
INPLACE-ABN	–		16×712^2	78.45
INPLACE-ABN^{sync}	–		16×712^2	79.02
INPLACE-ABN^{sync}	–		12×872^2	79.16
INPLACE-ABN^{sync} + CLASS-UNIFORM SAMPLING	–		12×872^2	**79.40**
Mapillary Vistas				
INPLACE-ABN^{sync} + CLASS-UNIFORM SAMPLING	–		12×776^2	**53.12**
LSUN 2017 winner [35] (based on PSPNet)	ResNet-101			
PSPNet + auxiliary loss	16×713^2		49.76	
+ Hybrid dilated convolutions [29]	16×713^2		50.28	
+ Inverse frequency label reweighting	16×713^2		51.50	
+ Cityscapes pretraining	16×713^2		51.59	

Table 4. Validation data results (single scale test, no horizontal flipping) for semantic segmentation experiments on Cityscapes and Vistas, using ResNeXt-152 and WideResNet-38 bodies with different settings for #crops per minibatch and crop sizes. All results in [%].

- Combination of INPLACE-ABN sync with larger crop sizes improves by ≈ 0.9% over the best performing setting in Table 3
- **Class- Uniform sampling**: Class-uniformly sampled from eligible image candidates, making sure to take training crops from areas containing the class of interest.

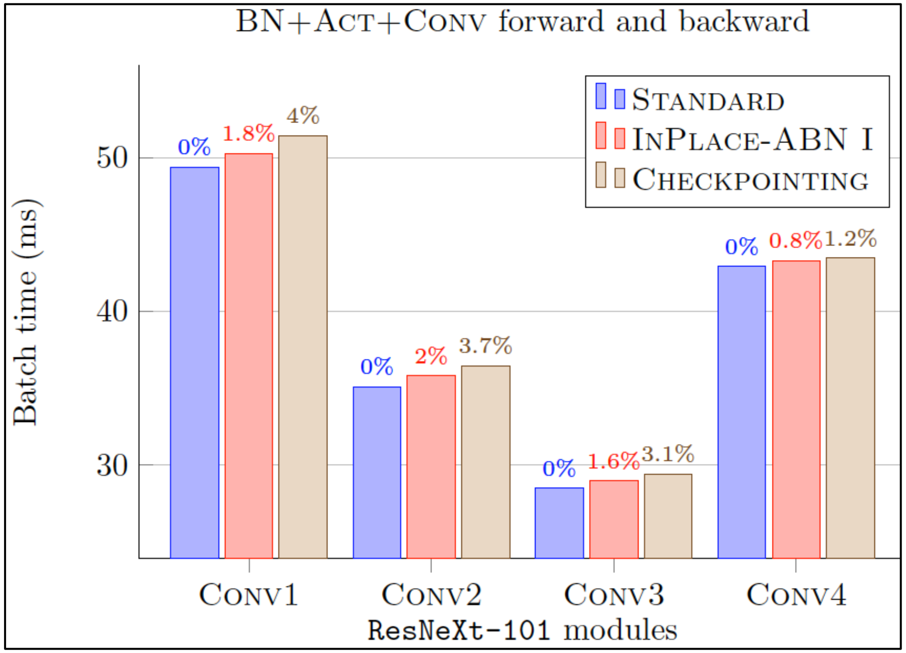
Experiments: Semantic Segmentation

- Currently state of the art for CityScapes for IoU class and iIoU (instance) Class
  - **iIoU:** Weighting the contribution of each pixel by the ratio of the class' average instance size to the size of the respective ground truth instance.

name	fine	coarse	16-bit	depth	video	sub	IoU class	iIoU class	IoU category	iIoU category	Runtime [s]	code
Mapillary Research: In-Place Activated BatchNorm	yes	yes	no	no	no	no	82.0	65.9	91.2	81.7	n/a	yes
SR-AIC	yes	yes	no	no	no	no	81.9	60.7	91.3	79.6	n/a	no
iFLYTEK-CV	yes	yes	no	no	no	no	81.4	60.9	91.0	79.5	n/a	no
DeepMotion	yes	no	no	no	no	no	81.4	58.6	90.7	78.1	n/a	no
DeepLabv3	yes	yes	no	no	no	no	81.3	62.1	91.6	81.7	n/a	no

Experiments: Timing Analyses

- They isolated a single BN+ACT+CONV block & evaluate the computational times required for a forward and backward pass
- **Result:** Narrowed the gap between **standard** vs **checkpointing** by **half**
- Ensured fair comparison by re-implementing checkpointing in PyTorch



Overview

- Motivation for Efficient Memory management
- Related Works
  - Reducing precision
  - Checkpointing
  - Reversible Networks [9] (Gomez et al., 2017)
- In-Place Activated Batch Normalization
  - Review: Batch Normalization
  - In-place Activated Batch Normalization
- Experiments
- Future Directions

Future Directions:

- Apply INPLACE-ABN in other…
  - **Architectures**: DenseNet, Squeeze-Excitation Networks, Deformable Convolutional Networks
  - **Problem Domains:** Object detection, instance-specific segmentation, 3D data learning
- Combine INPLACE-ABN with other memory reduction techniques, ex: Mixed precision training
- Apply same InPlace idea on ’newer’ Batch Norm, ex: Batch Renormalization*

*S. Ioffe. “Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models.” ArXiv Link

Links and References

- INPLACE-ABN Paper: https://arxiv.org/pdf/1712.02616.pdf
- Official Github code **(PyTorch)**: https://github.com/mapillary/inplace_abn
- CityScapes Dataset: https://www.cityscapes-dataset.com/benchmarks/#scene-labeling-task
- Reduced Precision:
  - BinaryConnect: https://arxiv.org/abs/1511.00363
  - Binarized Networks: https://arxiv.org/abs/1602.02830
  - Mixed Precision Training: https://arxiv.org/abs/1710.03740
- Trade off with Computation Time
  - Checkpointing: https://www.cs.utoronto.ca/~jmartens/docs/HF_book_chapter.pdf
  - Recursive Checkpointing: https://arxiv.org/abs/1604.06174
  - Reversible Networks: https://arxiv.org/abs/1707.04585