# Reversible Fixup Networks for Memory-Efficient Training

**Vithursan Thangarasa[1,2,*], Chuan-Yung Tsai[3,†], Graham W. Taylor[1,2,4], Urs Köster[5]**
[1]School of Engineering, University of Guelph, Canada
[2]Vector Institute for Artificial Intelligence, Toronto, Canada
[3]South Park Commons, [4]Canada CIFAR AI Chair
[5]Cerebras Systems, Los Altos, California
[1]{vthangar, gwtaylor}@uoguelph.ca, [3]cytsai@gmail.com, [5]urs@cerebras.net

## Abstract

Deep residual networks (ResNets) have been used successfully for many computer vision tasks, but are difficult to scale to high resolution images or 3D volumetric datasets. Activation memory requirements, which scale with network depth and mini-batch size, quickly become prohibitive. Recently, invertible neural networks (INNs) have been successfully applied to classification problems to reduce memory requirements during neural network training, enabling constant memory complexity in depth. This is accomplished by removing the need to store the input activations computed in the forward pass and instead reconstruct them on-the-fly during the backward pass. But existing approaches require additive coupling layers, which perform channel-wise splitting and add additional parameters to the standard ResNet. In addition, they require a large enough mini-batch size to perform effective Batch Normalization. We propose RevUp, an invertible ResNet architecture with Fixup initialization that is memory efficient, achieves high generalization, removes mini-batch dependence and does not require modifications via coupling layers as seen in previous methods. We show that RevUp achieves competitive test accuracy against comparable baselines on CIFAR-10 and ILSVRC (ImageNet).

## 1 Introduction

In recent years, convolutional neural networks (CNNs) have dominated the field of computer vision due to their success in achieving state-of-the-art performance on a wide range of computer vision tasks. For example, top-ranked architectures used in the recent LSUN, MS COCO segmentation/detection, and ILSVRC challenges are predominantly based on deep ResNet [11, 12] or ResNeXt [24] substructures. Hence, memory becomes a significant hardware bottleneck when *training* such large architectures, often by orders of magnitude. This is because they require retaining the computed activations of all intermediate layers, which are needed to compute gradients during backpropagation. The problem becomes even more apparent when needing to train residual networks with mini-batches of high resolution images or 3D volumetric data which can directly limit the size of the model or the size of the mini-batch used for training.

Recent work on invertible neural networks (INNs) have shown that reversible functions enable us to retrieve its input from its output, thus significantly reducing activation memory footprint when training neural networks with backpropagation. This makes INNs very memory-efficient because intermediate activations do not have to be stored in memory to perform backpropagation and yet still

[*]Work done during an internship at Cerebras Systems in Summer 2019.
[†]Work done while at Cerebras Systems.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

achieve competitive test accuracy to standard ResNets on image classification tasks [9]. Interestingly, the i-RevNet [14] showed that we can achieve good test performance on image classification without loss of information in the hidden layers. Most of the work on INNs heavily relies upon the transformations based on coupling layers introduced in NICE [6] and improved with RealNVP [7]. However, the i-ResNet [3] showed that we can enforce the invertibility of ResNet by ensuring that the Lipschitz constant of each residual block is less than unity and using the fixed-point theorem for computing the inverse. There has also been a rapidly growing interest in INNs in the context of normalizing flow-based methods [16, 22] for generating large high quality images [15] and solving inverse problems in natural science [1]. However, in our work we are mainly interested in the memory-saving property of INNs to achieve constant memory reduction when training residual network architectures for classification.

One limitation of INNs is that their design restricts the use of some popular normalization techniques for neural networks, such as Batch Normalization (BN) [13]. This is because ResNets with BN can have singular values well above one [3]. This makes residual blocks with BN layers non-invertible and would require modifying the ResNet structure using coupling layers [9]. Glow [15] introduced ActNorm, an invertible normalization layer that was mainly shown to work well in practice for flow-based generative models [15] and has not been evaluated to the same extent for discriminative models. The recent Fixup initialization method [27] demonstrates that we can train deep ResNet models without normalization[3]. Moreover, Fixup allows reducing the batch size to one, lowering memory requirements by an order of batch size. Fixup is the only known method which allows us to maintain invertibility without using coupling layers, remove dependence on the mini-batch size and yet still achieve a high test accuracy when performing classification with ResNet architectures.

In this work, we focus on increasing the memory efficiency of the training process of ResNet architectures in order to further leverage their performance on tasks such as image classification. We combine the memory-saving property of INNs and favorable generalization properties from training without normalization using Fixup to develop a new ResNet-type architecture we refer to as RevUp. Our architecture builds upon i-ResNet, where we replace several components of the i-ResNet to make it architecturally as close as possible to a vanilla ResNet while remaining invertible. In addition, we also remove all normalization layers in place of Fixup. We are the first to investigate Fixup in the context of invertible architectures for highly memory-efficient training. We demonstrate that with RevUp, residual networks with different depths can be trained using the reconstructed activations from a single fixed-point iteration, while enforcing invertibility by bounding the Lipschitz constant to be less than unity using a single power iteration.RevUp achieves competitive test accuracy on CIFAR-10 [17] and ImageNet [23] while being far more memory efficient.

## 2 RevUp Networks

We introduce unnormalized invertible residual networks via Fixup initialization (RevUp), a variant of ResNets which are invertible and do not require Batch Norm to generalize well. Moreover, we achieve memory efficient training by exploiting the invertibility in RevUp, which allows us to reconstruct each layer's activations by computing the subsequent layer's activations, thus avoiding the need to store intermediate activations in memory. Although the i-ResNet was demonstrated on image classification, it has several noticeable differences in comparison to vanilla ResNets (visualized in Figure 1), which our RevUp architecture aims to resolve. We make no modifications to the ResNet + Fixup residual block besides moving the ReLU after the summation into the residual. Also, RevUp performs strided downsampling instead of squeeze operations [3, 7, 15], therefore we checkpoint to store the input activations only when downsampling.

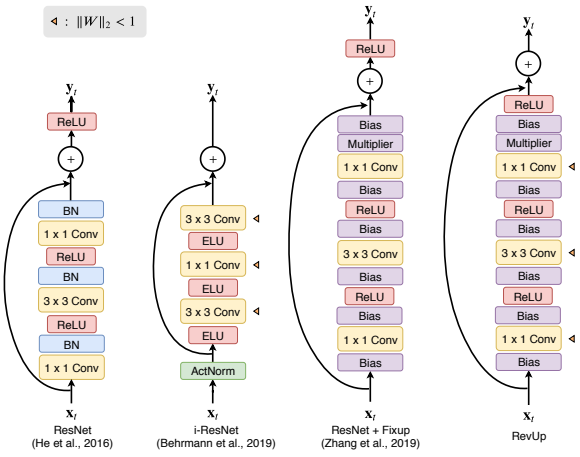



Figure 1: **From left:** The bottleneck structure of vanilla ResNet, i-ResNet, Fixup and our proposed RevUp.

[3]Refer to Appendix C for relevant work on INNs and training residual networks without normalization.

### 2.1 Invertible Residual Blocks

In standard ResNet architectures, given the activations $\mathbf{x}_t$ and the residual block $g_{\theta_t}$, the output $\mathbf{y}_t$ of the residual function can be represented with Eq. 1 and the inverse of this residual function can be computed by solving Eq. 2 for $\mathbf{x}_t$.

$$\mathbf{y}_t = \mathbf{x}_t + g_{\theta_t}(\mathbf{x}_t), \tag{1}$$

$$\mathbf{x}_t = \mathbf{y}_t - g_{\theta_t}(\mathbf{x}_t). \tag{2}$$

Assume we have a ResNet $\mathcal{F}_\theta$ containing $T$ residual blocks where $\mathcal{F}_{\theta_t} = I + g_{\theta_t}\ \forall\, t \in \{1, \ldots, T\}$ and $I$ is the identity function. Now, for $\mathcal{F}_\theta$ to be invertible such that $\mathcal{F}_\theta(\mathbf{x}) = \mathcal{F}_{\theta_1} \circ \cdots \circ \mathcal{F}_{\theta_T}(\mathbf{x})$, with each $\mathcal{F}_{\theta_t}$ having an inverse, we need to ensure that the Lipschitz constant of each residual block $\mathrm{Lip}(g_{\theta_t})$ is less than unity,

$$\mathrm{Lip}(g_{\theta_t}) < 1\ \forall\, t \in \{1, \ldots, T\}. \tag{3}$$

We can then perform fixed-point iteration using Algorithm 1 to compute Eq. 2 by initializing $\mathbf{x}^0$ to be the output $\mathbf{y}$ from Eq. 1. Then, from the Banach fixed-point theorem, $\mathbf{x}^n$ satisfies the a-priori estimate of the error bound for all future iterates $\mathbf{x}^n$,

$$\|\mathbf{x}^* - \mathbf{x}^n\|_2 \leq \frac{\mathrm{Lip}(g_{\theta_t})^n}{1 - \mathrm{Lip}(g_{\theta_t})} \|\mathbf{x}^1 - \mathbf{x}^0\|_2 \tag{4}$$

Therefore, fixed point iterates given by $\mathbf{x}^n$ converge exponentially to $\mathbf{x}^*$ as $n \to \infty$, where smaller Lipschitz constants will result in faster convergence to $\mathbf{x}^*$. This effectively allows us to compute the inversion $\mathcal{F}_\theta^{-1}(\mathbf{y}) = \mathcal{F}_{\theta_T}^{-1} \circ \cdots \circ \mathcal{F}_{\theta_1}^{-1}(\mathbf{y})$ and not store intermediate activations of each layer. Although Algorithm 1 shows several iterations and Eq. 4 presents an error bound for $N > 1$ iterations, in practice we only perform a single fixed point iteration during training. In Section 3.1, we analyze the invertibility and show that $N = 1$ is sufficient for training RevUp.

**Algorithm 1** Fixed-point for $\mathcal{F}_\theta^{-1}(\mathbf{y})$.

**Input:** $\mathbf{y}_t$ (residual block output), $g_{\theta_t}$ (contractive residual), $N$ (fixed point iterations)
**Output:** $\mathbf{x}^*$ (reconstructed input)
Initialize $\mathbf{x}^0 \leftarrow \mathbf{y}$
**for** $n = 0$ **to** $N$ **do**
  $\mathbf{x}^{n+1} \leftarrow \mathbf{y}_t - g_{\theta_t}(\mathbf{x}^n)$
**end for**
$\mathbf{x}^* \leftarrow \mathbf{x}^{n+1}$

### 2.2 Satisfying the Lipschitz Condition

We perform spectral normalization on each convolutional layer $\{W_m\}_{m=1}^M$ for $M$ layers in each residual block $g_{\theta_t}\ \forall\, t \in \{1, \ldots, T\}$, so that the spectral norm (denoted by $\|\cdot\|_2$) of each layer $i$ is bounded $\|W_m\|_2 < 1$. The basic block of the ResNet, $g_{\theta_t}$ is composed of $W_2\phi(W_1)$ and bottleneck $g_{\theta_t} = W_3\phi(W_2\phi(W_1))$, where $\phi$ are contractive ReLU non-linearities. Thus, we enforce $\mathrm{Lip}(g_{\theta_t}) < 1$ as follows,

$$\mathrm{Lip}(g_{\theta_t}) < 1,\ \text{if}\ \|W_m\|_2 < 1,\ \forall\, m \in \{1, \ldots, M\}. \tag{5}$$

We perform power-iteration using Algorithm 2 proposed in [10] to estimate the weight matrix's largest singular value (SV), $\sigma_{\max}$, using $W_m$ and $W_m^{\mathrm{T}}$, and with only $P = 1$ power iteration. The estimate is normalized using a scaling coefficient $c < 1$ because the estimate $\sigma_i$ from the power-iteration method is an under-estimate, thus $\sigma_m < \|W_m\|_2$. Therefore, if $\frac{c}{\sigma_m} < 1$, then $W_m$ is normalized to $c \cdot \frac{W_m}{\sigma_m}$. Although we can compute the inverse faster for small values of $c$, it can hurt test accuracy. Empirically though, even with a high $c$ value we can compute a reasonably accurate inverse with a single fixed point and not harm the test accuracy by much.

**Algorithm 2** Power-iteration method.

**Input:** $W_m$ (weight matrix), $P$ (power iterations)
**Output:** $\sigma_{\max}$ (largest SV est.)
Randomly initialize $\mathbf{x}_0$
**for** $p = 0$ **to** $P$ **do**
  $\mathbf{x}_{p+1} \leftarrow W_m^{\mathrm{T}} W_m \mathbf{x}_p$
**end for**
$\sigma_{\max} \leftarrow \frac{\|W_m x_P\|_2}{\|x_P\|_2}$

### 2.3 Complexity Analysis

We analyze the space and time complexity of our method and extend the table described in [9] for consistency. Similar to [9], we assume that we have a feed-forward neural network consisting of $L$ layers and the time and storage cost of forward- or back-propagation through a single layer is one.

**Space Complexity:** The space complexity of RevUp is effectively $\mathcal{O}(1)$ because on the forward pass we free the memory of $\mathbf{x}_t$ after computing $\mathcal{F}(\mathbf{x}_t) = \mathbf{y}_t$, and on the backward pass we compute the inverse $\mathcal{F}^{-1}(\mathbf{y}_t) = \mathbf{x}_t$ using Algorithm 1. Once the input is restored, the gradients for the weights and the inputs can be recomputed as normal using the PyTorch `autograd` solver. RevUp improves on RevNet's memory complexity because we remove all data-dependent normalization layers and employ Fixup initialization which removes mini-batch $B$ as a dependency for training the model to convergence. The ResNet + Fixup architecture already improves space complexity from $\mathcal{O}(B * L)$ (standard ResNet + Backpropagation) to $\mathcal{O}(L)$. Now, RevUp improves on that even further by reducing the complexity from $\mathcal{O}(L)$ to $\mathcal{O}(1)$.

**Computational Complexity:** The computational cost of naïve backpropagation is $3L$ for computing forward, backward and gradients through the network. Therefore, the computational cost of RevUp with memory savings enabled is $7L$ due to the additional $4L$ of compute because: 1) **Forward:** Algorithm 2 performs 1 power iteration which consists of a transposed convolution followed by a standard convolution (i.e., computational cost of $2L$). 2) **Backward:** Algorithm 1 performs 1 fixed point iteration to obtain the reconstructed input activations which is equivalent to an additional forward pass for each layer and then a *re-forward* through the residual block to compute the gradients (i.e., computational cost of $2L$). This results in a time complexity of $\mathcal{O}(L)$, similar to RevNet, but with the advantage of not having to modify the existing architecture with coupling layers. Table 1 summarizes the space and time complexity of different memory saving techniques for training residual networks.

Table 1: Comparison of space and computational complexity for training ResNets (containing $L$ residual layers and $B$ elements in a mini-batch) between various memory-saving techniques.

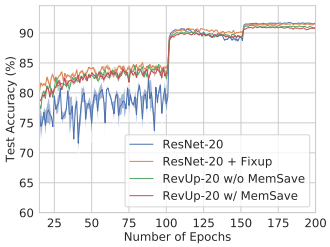
| Architecture | Space Complexity | Computational Complexity |
|---|---|---|
| ResNet + Naive Backpropagation | $\mathcal{O}(B * L)$ | $\mathcal{O}(L)$ |
| ResNet + Checkpointing [20] | $\mathcal{O}(B * \sqrt{L})$ | $\mathcal{O}(L)$ |
| ResNet + Recursive Checkpointing [4] | $\mathcal{O}(B * \log L)$ | $\mathcal{O}(L \log L)$ |
| RevNet (additive coupling) [9] | $\mathcal{O}(B)$ | $\mathcal{O}(L)$ |
| ResNet + Fixup [27] | $\mathcal{O}(L)$ | $\mathcal{O}(L)$ |
| RevUp (Ours) | $\mathcal{O}(1)$ | $\mathcal{O}(L)$ |

### 2.4 Implementation Details

We implemented custom `forward` and `backward` functions which wrap any `torch.nn.Sequential` container with modules for each residual block implemented as a `torch.nn.Module`. Here, each memory saving residual block is implemented using the `torch.autograd` package, thus allowing the `forward` and `backward` functions to work effectively with PyTorch's automatic differentiation system. We achieve memory savings by not storing the input activations $\mathbf{x}_t$ during the forward pass, except for the strided downsampling blocks, and only save the output activations $\mathbf{y}_t$ with `ctx.save_for_backward(`$\mathbf{y}_t$`)`. Then in `backward`, given $\mathbf{y}_t$, we use Algorithm 1 where, $N = 1$, to reconstruct the input activations $\mathbf{x}_t$ when they are needed for computing the gradients.

## 3 Results

We experimented with RevUp on CIFAR-10 and ILSVRC (ImageNet) where we compare to relevant baselines including the vanilla ResNet [11] and ResNet + Fixup [27]. We also compare to RevUp without memory savings (RevUp w/o MemSave) where the residual blocks are invertible, but the actual activations were used for training. In all RevUp experiments, we perform $P = 1$ power iteration and use a $c$ value of 0.9. The input activations of the residual block were recomputed with $N = 1$ fixed-point iteration in the backward pass. In our experiments, we are mainly interested in comparing the test accuracy of our RevUp with memory saving (RevUp w/ MemSave) architecture to the ResNet + Fixup networks. Although we do not train with additional regularization as demonstrated in [27], we believe techniques such as Cutout [5], Mixup [26] or CutMix [25] can help improve test accuracy and be on par with standard ResNets. In Table 2, we compare the test performance of RevUp architectures to their respective baselines. Refer to Appendix A for details on the experimental setups.



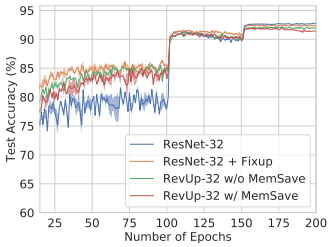




Figure 2: Test accuracy curves on CIFAR-10 for residual networks with (left) 20 layers and (right) 32 layers. The shaded regions correspond to the standard error of the mean (SEM) across 5 trials.

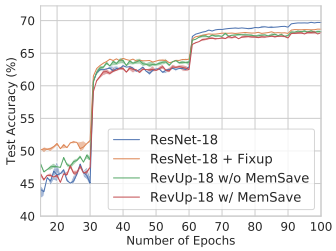



Figure 3: Top-1 validation curves on ImageNet. The shaded regions represent the SEM of 5 trials.

### 3.1 CIFAR-10

In Figure 2, we compare the training curves of residual networks trained with the actual activations and the reconstructed activations (RevUp w/ MemSave) where, the final test accuracy of the MemSave model remains close to the baseline while being very memory efficient (quantified below). RevUp-20 w/ MemSave achieves a test accuracy of $91.24 \pm 0.05\%$, which is only a $0.34\%$ and $0.49\%$ decrease from ResNet-20 + Fixup and ResNet-20, respectively. The RevUp-32 w/ MemSave attains $92.01 \pm 0.12\%$, which is $0.47\%$ and $0.62\%$ decrease from ResNet-32 + Fixup and ResNet-32, respectively.

**Invertibility Analysis:** We verify the invertibility of RevUp by inverting each residual block using Algorithm 1 with $N = 1$ to 100 iterations to ensure convergence to the exact solution. For the purposes of verifying invertibility, we replace the $3\times3$ convolutional stage at the first layer with injective padding [3, 14] which pads 13 channels of zeros to the input to match the 16 input channels in the first residual block. Then, we compute the $\ell_2$ reconstruction error on each image in the mini-batch of size 128 from the CIFAR-10 test set and plot the average reconstruction error in Figure 5 for the various fixed point iterations.

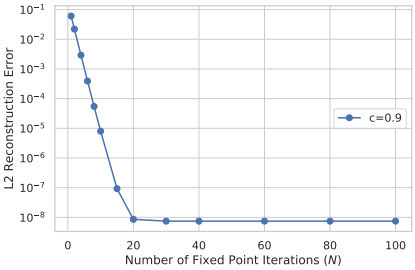



Figure 5: The average $\ell_2$ reconstruction error on the CIFAR-10 test set for a RevUp-20 model at different fixed point iterations for $c = 0.9$.

**Reconstructed Activations:** We observe that the RevUp-20 model can be inverted with minimal error even at a single fixed-point iteration for a high normalization scaling coefficient of $c = 0.9$ (see Figure 4). Upon closer inspection, the reconstructed inputs with 1 fixed point iteration have a noticeable shift in the blue colours which turn teal or turquoise. However, after 40 fixed point iterations, the reconstructed inputs are numerically as exact as the actual activations (see Figure 5) and visually indistinguishable from the original inputs (refer to Figure 4).

(a) Original Inputs

(b) $N = 1$

(c) $N = 40$

Figure 4: **Left:** We visualize the original input images that are randomly selected from the CIFAR-10 test set. We then depict the corresponding reconstructed inputs from an RevUp-20 at **(middle)** $N = 1$ and **(right)** $N = 40$ fixed point iterations for a $c$ value of 0.9 (see Algorithm 1).

### 3.2 ImageNet

We compare the training curves of 18 layer residual networks trained with actual and reconstructed activations (see Figure 3). RevUp-18 w/ MemSave attains a top-1 validation accuracy of 68.51 $\pm$ 0.03% while being very memory efficient and only a 0.27% decrease from ResNet-18 + Fixup.

Table 2: Test accuracy (%) on CIFAR-10 and the top-1 validation accuracy (%) on ImageNet with single center crop. The results are averaged over 5 trials $\pm$ the SEM.

| Dataset | # of Layers | Architecture | | | |
|---|---|---|---|---|---|
| | | ResNet | ResNet + Fixup | RevUp w/o MS | RevUp w/ MS |
| CIFAR-10 | 20 | 91.73 $\pm$ 0.06 | 91.58 $\pm$ 0.05 | 91.31 $\pm$ 0.05 | 91.24 $\pm$ 0.05 |
| | 32 | 92.63 $\pm$ 0.11 | 92.48 $\pm$ 0.06 | 92.13 $\pm$ 0.11 | 92.01 $\pm$ 0.12 |
| | 56 | 93.13 $\pm$ 0.11 | 92.85 $\pm$ 0.08 | 92.25 $\pm$ 0.07 | 92.08 $\pm$ 0.11 |
| | 18 | 95.17 $\pm$ 0.04 | 94.17 $\pm$ 0.03 | 93.90 $\pm$ 0.03 | 93.85 $\pm$ 0.05 |
| ImageNet | 18 | 69.74 $\pm$ 0.03 | 68.78 $\pm$ 0.02 | 68.67 $\pm$ 0.04 | 68.51 $\pm$ 0.03 |

**Memory Consumption**

We plot the memory allocated cumulatively (measured in GB) after each residual block $t$ during forward- and back-propagation on a single NVIDIA V100 GPU for the ImageNet experiments above (see left plot in Figure 6). RevUp-18 has a peak memory usage of 3.2 GB while vanilla ResNet-18 and ResNet-18 + Fixup have peak memory usages of 7.6 GB and 5.6 GB, respectively. In addition, we perform a memory test on RevUp-50 with a 4096 $\times$ 2160 image at batch size $B = 1$, to demonstrate the memory saving abilities (see right plot in Figure 6). For ResNet-50 + Fixup, we extrapolate the memory usage on a 4K image from the measured usage on a 2K image. The memory usage for vanilla ResNet-50 was excluded in this plot because training it would not converge at $B = 1$. In all cases, the memory consumption for RevUp was $\mathcal{O}(1)$, whereas the rest are $\mathcal{O}(L)$ (see Appendix B for more details).

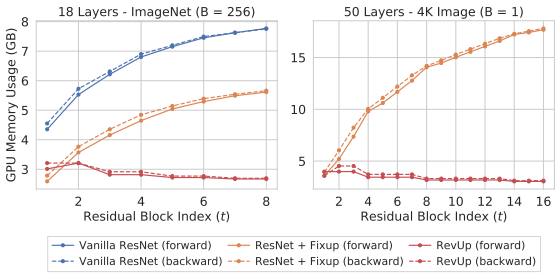



Figure 6: The GPU memory usage of **(left)** 18 layer residual networks for a batch of 256 ImageNet samples and **(right)** 50 layer residual networks for a full 4096 $\times$ 2160 image.

## 4 Conclusion and Future Work

We introduce RevUp, an invertible residual network architecture that does not use data-dependent normalization and enables high memory efficiency during training as the activations of all non-downsampling residual blocks are not stored in memory. We enforce invertibility in the residual blocks by performing a single power iteration to ensure each residual block is a contraction by bounding the Lipschitz constant of each layer to be less than unity. Then, we use a single fixed point iteration to reconstruct the activations on-demand during backpropagation. We demonstrate that we can replace all normalization layers with Fixup to retain invertibility, remove batch dependence and still achieve competitive test accuracy on popular image classification benchmarks. In the future, we are interested in applying RevUp for semantic segmentation on high resolution images. Given that RevUp is not batch dependent, we have the ability to train with batch size of 1 or even small batches of high resolution images without having to perform any downsampling or patch cropping. We hope our work will create new investigations with RevUp architectures for memory intensive training on real world datasets containing high-res images (e.g., medical imaging, satellite or aerial imagery and perception for autonomous vehicles).

## References


[1] Lynton Ardizzone, Jakob Kruse, Carsten Rother, and Ullrich Köthe. Analyzing inverse problems with invertible neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

[2] David Balduzzi, Marcus Frean, Lennox Leary, J P Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 342–350, 2017.

[3] Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Joern-Henrik Jacobsen. Invertible residual networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 573–582, 09–15 Jun 2019.

[4] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174, 2016.

[5] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[6] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *CoRR*, abs/1410.8516, 2014.

[7] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *International Conference on Learning Representations (ICLR)*, 2017.

[8] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1243–1252, 2017.

[9] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In *Advances in Neural Information Processing Systems (NIPS) 30*, pages 2214–2224. 2017.

[10] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael Cree. Regularisation of neural networks by enforcing lipschitz continuity. *arXiv preprint arXiv:1804.04368*, 2018.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Proceedings of the 14th European Conference on Computer Vision (ECCV)*, pages 630–645, 2016.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[14] Jörn-Henrik Jacobsen, Arnold W.M. Smeulders, and Edouard Oyallon. i-revnet: Deep invertible networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[15] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems (NeurIPS) 31*, pages 10215–10224. 2018.

[16] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems (NIPS) 29*, pages 4743–4751. 2016.

[17] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-10 (Canadian Institute for Advanced Research).

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS) 25*, pages 1097–1105. 2012.

[19] Matthew MacKay, Paul Vicol, Jimmy Ba, and Roger B Grosse. Reversible recurrent neural networks. In *Advances in Neural Information Processing Systems (NeurIPS) 31*, pages 9029–9040. 2018.

[20] James Martens and Ilya Sutskever. *Training Deep and Recurrent Networks with Hessian-Free Optimization*, pages 479–535. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[21] Dmytro Mishkin and Jiri Matas. All you need is a good init. In *International Conference on Learning Representations (ICLR)*, 2016.

[22] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML)*, pages 1530–1538, 2015.

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, Dec 2015.

[24] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016.

[25] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *International Conference on Computer Vision (ICCV)*, 2019.

[26] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations (ICLR)*, 2018.

[27] Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. In *International Conference on Learning Representations (ICLR)*, 2019.

# A Experimental Details

## A.1 CIFAR-10

In our CIFAR-10 [17] experiments, we follow the same training scheme used in [27], where we train all of the different ResNet architectures with mini-batches of size 128 and optimized using SGD with momentum and weight decay. The initial learning rate is set to 0.1, weight decay to 0.0005 and momentum to 0.9. The learning rate is dropped by 0.1 at 100 and 150 epochs and we train for a total of 200 epochs. We reduce the learning rate of the scale and bias for Fixup and RevUp by $10\times$ as performed in [27]. We also apply the standard preprocessing and data augmentation techniques [12] on the CIFAR-10 dataset.

## A.2 ImageNet

For our ImageNet experiments, we trained all architectures for 100 epochs using SGD with momentum and weight decay at a batch size of 256. The initial learning rate is set to 0.1, weight decay to 0.0001 and momentum to 0.9. The learning rate is dropped by a factor of 0.1 at 30, 60 and 90 epochs. We apply the standard preprocessing and data augmentation techniques on ImageNet [18]. Also, similar to the CIFAR-10 experiments, we reduce the learning rate of the scale and bias for Fixup and RevUp by $10\times$ as performed in [27].

# B Memory Consumption

The memory consumption of RevUp, vanilla ResNet and ResNet with Fixup were measured on a NVIDIA V100 GPU (with 16 GB memory) for the plots in Figure 6 and Figure 7. We used PyTorch's CUDA package to obtain the allocated GPU memory after each residual block $t$ in a cumulative manner during both `forward` and `backward` passes of the network. The memory complexity increases with depth, $L$, in the forward pass for standard ResNets and ResNet with Fixup. Then during the backward pass, the memory allocated for each residual block is succesively freed from $t_9$ to $t_1$.

One of the advantages of RevUp networks is that memory complexity is constant during the forward and backward passes because the memory of the input activations $\mathbf{x}_t$ is freed after computing the output of each residual block $g_{\theta_t}$ except for the downsampling stages where we checkpoint the input activations to later be used during the backward pass. We also observe drops in memory usage at the downsampling stages, for example $t \in \{3, 6, 9\}$ in the right plot of Figure 7 for RevUp-20. This is because during the forward pass of block $t$ with activation sizes $\mathcal{A}_t$ we have at least $2 * \mathcal{A}_t$ in memory to hold the input and the output of the block that we are calculating (could be more if we hold intermediate values too, but let us assume it is $P * \mathcal{A}_t$). At block 6, for instance, we have one checkpoint of size $\mathcal{A}_3$ in addition to however many temporary copies we use during the forward pass. Therefore, the memory usage at block 6 is $P * \mathcal{A}_6 + (1 * \mathcal{A}_3)$ versus $P * \mathcal{A}_3$ at block 3. During the downsampling stage, the spatial dimensions $(H, W)$ are decreased by a factor of 2 and the number of channels is increased by a factor of 2. Thus, when we perform downsampling, $A_6 = \frac{\mathcal{A}_3}{2}$ which results in a flat line at $P = 3$ and for $P > 2$ memory decreases in subsequent blocks.

**ImageNet:** The right plot in Figure 6 presents the memory usage when processing a mini-batch size of 256 ImageNet samples ($224 \times 224$ input size) with 18 layer residual networks.

**CIFAR-10:** The left and right plots in Figure 7 reflect the memory usage when processing a mini-batch of 128 CIFAR-10 samples ($32 \times 32$ input size) with 20 and 32 layer residual networks, respectively. In all cases, we observe that the memory consumption for RevUp stays constant, whereas the memory complexity increases with depth for ResNet and ResNet + Fixup architectures.

## B.1 4K Resolution Image

We perform a memory test on a RevUp-50 architecture with a 4K resolution image of size $4096 \times 2160$ at a batch size of 1, in order to determine if we can fit a deep residual network, one that is commonly used in practice, and the activations of the full-resolution image onto GPU memory. We perform this test because there is an increasing demand for super high resolution images, however, most of the approaches in the literature either downsample the high resolution image or crop it into

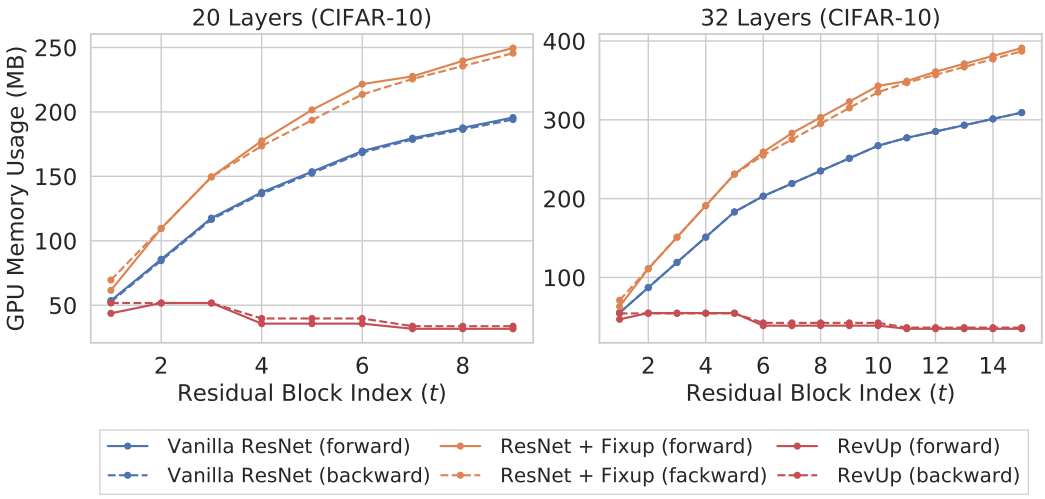



Figure 7: The GPU memory usage during the forward and backward passes of residual networks at depths of **(left)** 20 layers and **(right)** 32 layers for a batch of 128 CIFAR-10 samples. The memory complexity remains $\mathcal{O}(1)$ with our RevUp architectures.

small patches for separate processing due to GPU memory limits. The RevUp-50 architecture was able to process a $4096 \times 2160$ sample on a Volta V100 GPU where the peak memory usage was 4525.58 MB which occurs during the backward pass (see right plot in Figure 6).

## C Related Work

**Invertible Neural Networks:** There have been a number of recent works on designing more flexible neural network architectures that are invertible. Gomez et al. [9] proposed reversible residual networks (RevNet) to limit the memory overhead of backpropagation. RevNet architectures can be viewed as series of a composition of two sets of additive coupling layers similar to NICE and RealNVP. Jacobsen et al. [14] proposed i-RevNet which built modifications based on a cascade of homeomorphic layers to improve RevNet's theoretical properties and practical performance. MacKay et al. [19] proposed reversible GRU (RevGRU) and reversible LSTM (RevLSTM) which applied the idea of RevNets to build reversible recurrent networks (RNNs). However, all of these networks require architectural modifications to their vanilla counterpart due to the additive coupling layers. Recently, Behrmann et al. [3] proposed i-ResNet, which enforces invertibility by using spectral-normalization to constrain the Lipschitz constant of each residual block to be less than unity and applying the fixed-point theorem to compute its inverse. The advantage of this approach is that invertibility can be achieved without modifying the architecture with additive coupling layers which perform channel-wise splitting and add additional parameters to the standard ResNet. However, i-ResNet does make architectural changes to the vanilla ResNet which we aim to solve with our RevUp architecture.

**Training Residual Networks without Normalization:** A number of initialization techniques have been proposed in the literature to train residual networks such as Layer-sequential unit-variance (LSUV) initialization [21], looks-linear (LL) initialization [2] or stabilizing training in convolutional sequence to sequence modeling [8]. Recently, Zhang et al. [27] introduced a new approach called *fixed-update initialization* (Fixup) which solves the exploding and vanishing gradient problem at the beginning of training via properly rescaling a standard initialization. Moreover, Zhang et al. [27] showed that residual networks of different depths can be stably trained without normalization and can achieve competitive test performance against residual networks with normalization on image classification and machine translation tasks. Given that BN layers are not invertible and most of the Lipschitz constants of the layers in ResNets trained with BN can be well above one [3], we employ Fixup initialization to allow for invertibility and still be able to train residual networks of various depths.