iUNets - Fully invertible U-Nets for Memory-Constrained Applications

Christian Etmann, joint work with Rihuan Ke & Carola-Bibiane Schönlieb

30th June 2020

U-Nets

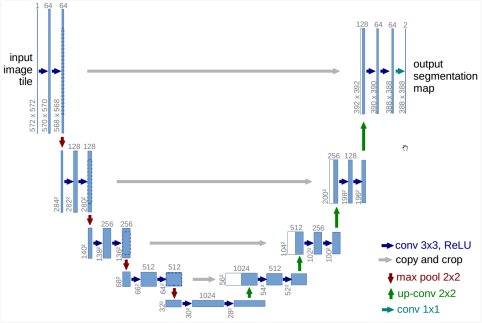


Figure 1: The 'classic' U-Net (Ronneberger et al.)

- U-Nets standard design principle for image-to-image tasks (segmentation, inverse problems in imaging etc., same for 3D.)
- High dimensionalities: prohibitive memory requirements
- This work: drop-in replacement for U-Nets (2D & 3D) with *far* lower memory footprint

U-Nets

Papers from Google Docs file which use U-Net-like architectures (5 out of 13):

- *Multi-Scale Learned Iterative Reconstruction* (Hauptmann et al., 2019)
- *Noise2Inverse: Self-supervised deep convolutional denoising for linear inverse problems in imaging* (Hendriksen et al., 2020)
- *NETT: Solving Inverse Problems with Deep Neural Networks* (Li et al., 2020)
- *Learning the invisible: a hybrid deep learning-shearlet framework for limited angle computed tomography* (Bubba et al., 2019)
- *Deep Bayesian Inversion* (Adler & Öktem, 2018)

Invertible networks for memory-restricted applications

Why do we need to store activations?
Let $x_i = \Phi_i(x_{i-1})$ be the output of a neural network's layer with nonlinear mapping $\Phi_i : \mathbb{R}^d \to \mathbb{R}^d$, where θ_i are the layer's parameters. Let further $\mathcal{L} = \ell(x_i)$ be the loss of the network. Then

$$\nabla_{\theta_i}\mathcal{L} = \left(\frac{\mathrm{d}x_i}{\mathrm{d}\theta_i}\right)^* \cdot \nabla_{x_i}\mathcal{L} = \left(\frac{\mathrm{d}\Phi_i(x_{i-1})}{\mathrm{d}\theta_i}\right)^* \cdot \nabla_{x_i}\mathcal{L}$$

provides the weight-gradient necessary for the training of the network.
For the calculation, we need

- the gradient of of the loss with respect to the output node, i.e. $\nabla_{x_i}\mathcal{L}$ and
- the input node x_{i-1} for the calculation of the derivative $\frac{\mathrm{d}\Phi_i(x_{i-1})}{\mathrm{d}\theta_i}$.

Invertible networks for memory-restricted applications

- Solution: if Φ_i is bijective and we have x_i in memory, we can simply *reconstruct*

$$x_{i-1} = \Phi_i^{-1}(x_i)$$

  from the next layer's activation x_i
- If the whole network is invertible, we can employ a memory-efficient backpropagation
- Successively reconstruct activations and calculate gradients from the last back to the first layer
- Memory requirement *independent* of the network's depth!
- How to construct these layers?
- Restriction: dimensionality may not change throughout network. For many applications: invertibility only up to last layer.

Invertible networks via additive coupling layers

- Additive coupling layers divide activation by channels: $x_i \cong (u_i, v_i)$
- Forward mapping $\Phi_i : (u_{i-1}, v_{i-1}) \mapsto (u_i, v_i)$ defined by

$$\begin{aligned} u_i &= v_{i-1} \\ v_i &= u_{i-1} + f_{\theta_i}(v_{i-1}) \end{aligned} \tag{1}$$

  for (almost) arbitrary f_{θ_i}, e.g. several conv layers
- Inverse can be computed algebraically via

$$\begin{aligned} u_{i-1} &= v_i - f_{\theta_i}(u_i) \\ v_{i-1} &= u_i \end{aligned} \tag{2}$$

- Other concepts employ numerical inversion schemes (*Invertible Residual Networks*, Behrmann et al., 2019)

Invertible U-Nets with Learnable Up- and Downsampling

Invertible Up- and Downsampling

- Normally, downsampling inherently non-invertible, because the dimensionality is decreased
- Classical methods: bi-/trilinear or bi-/tricubic interpolation, nearest neighbors, max pooling...
- ...unless we increase (decrease) the number of channels at the same time as decreasing (increasing) the spatial dimensionality, e.g. invertible function

$$\mathbb{R}^{C \times H \times W} \rightarrow \mathbb{R}^{4C \times H/2 \times W/2}$$

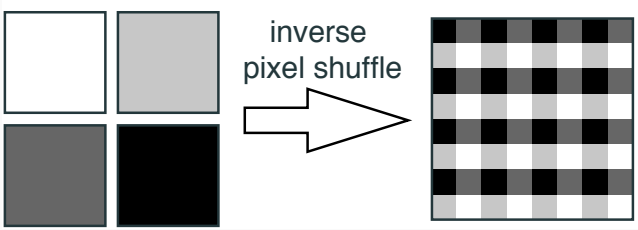
- invertible upsampling is inverse to invertible downsampling

Existing Invertible Up- and Downsampling

Figure 2: Straightforward methods for invertibly downsampling: *pixel shuffle* and *2D Haar transform*.

We could try to build an invertible U-Net with these. There is a problem though...

Problems of these methods



- Inverse pixel shuffle will lead to checkerboard artefacts
- When applying conv layers with the standard 3-by-3 kernel, we will get Moiré patterns
- Either we get artefacts or very non-diverse input features (probably both)
- Haar transform might work better, but it would be best if we could *learn* a suitable invertible upsampling operation

Learnable Invertible Up- and Downsampling

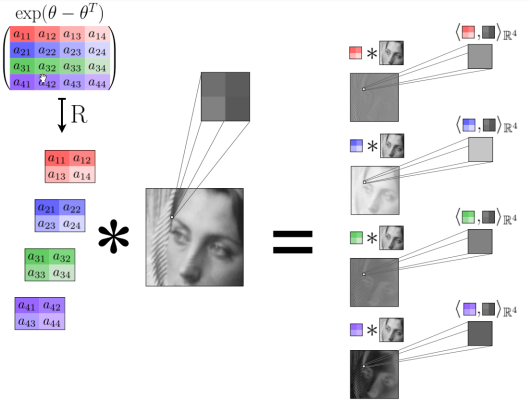


Figure 3: Idea: create orthogonal convolutional operator, which realizes invertible downsampling. Its inverse (=invertible upsampling) is given by the corresponding transposed convolution.

Examples

Figure 4: Original image, randomly initialized learnable invertible downsampling, invertible downsampling optimised for sparsity (1-norm minimisation)

Mathematical background

- The matrix exponential map

$$\exp : \mathfrak{so}(\sigma, \mathbb{R}) \to \mathrm{SO}(\sigma, \mathbb{R}) \tag{3}$$

  from Lie algebra $\mathfrak{so}(\sigma, \mathbb{R})$ of skew-symmetric matrices ($M^T = -M$) to Lie group $\mathrm{SO}(\sigma, \mathbb{R})$ is surjective.

- We can thus represent any special orthogonal matrix via $\exp(\theta - \theta^T)$ for some square matrix θ

- e.g. Haar transform:

$$\theta = \frac{\pi}{4} \begin{pmatrix} 0 & 0 & -1 & -1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

- Alternatives: Householder transforms, Givens rotations, Cayley transforms

Invertible U-Net

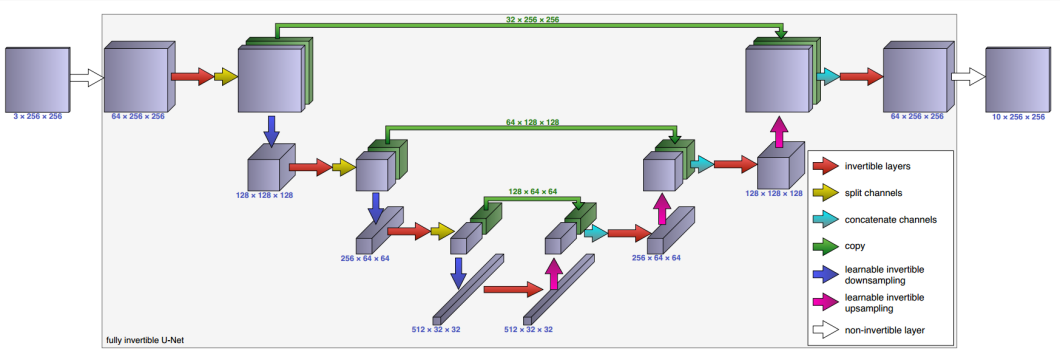
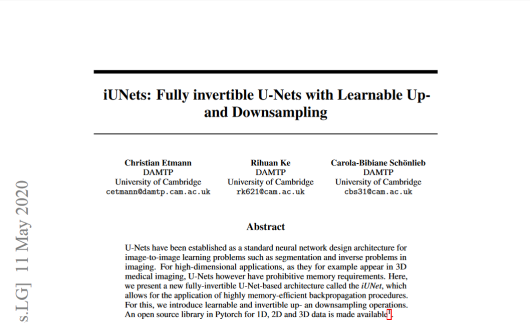


Figure 5: Segmentation example (RGB data to 10 classes). We increase and later decrease the number of channels, but can use the memory-efficient backpropagation inbetween.

Invertible U-Net



s.LG] 11 May 2020

iUNets: Fully invertible U-Nets with Learnable Up- and Downsampling

Christian Etmann
DAMTP
University of Cambridge
cetmann@damtp.cam.ac.uk

Rihuan Ke
DAMTP
University of Cambridge
rk621@cam.ac.uk

Carola-Bibiane Schönlieb
DAMTP
University of Cambridge
cbs31@cam.ac.uk

Abstract

U-Nets have been established as a standard neural network design architecture for image-to-image learning problems such as segmentation and inverse problems in imaging. For high-dimensional applications, as they for example appear in 3D medical imaging, U-Nets however have prohibitive memory requirements. Here, we present a new fully-invertible U-Net-based architecture called the *iUNet*, which allows for the application of highly memory-efficient backpropagation procedures. For this, we introduce learnable and invertible up- an downsampling operations. An open source library in Pytorch for 1D, 2D and 3D data is made available[3].

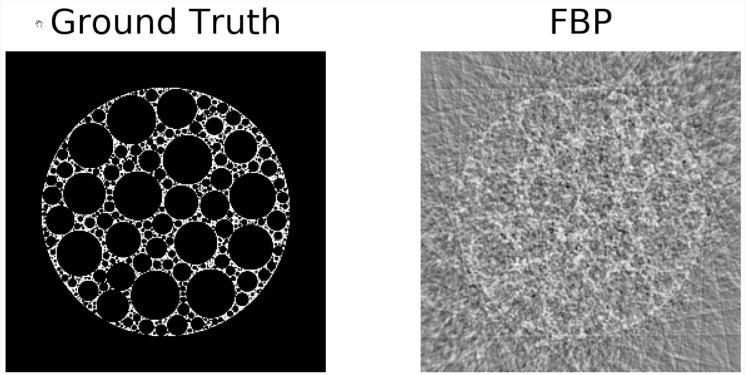
- Preprint released
- Implemented in Pytorch: `www.github.com/cetmann/iunets`
- Don't do it in Tensorflow...

Memory Consumption & Runtimes

Table 1: Memory-efficient (ME) vs conventional backpropagation, $64 \times 512 \times 512$-images. 4 downsampling operations, each Φ_i^L and Φ_i^R parametrized by δ coupling layers.

	Peak memory consumption			Runtime		
	ME	Conventional	Ratio	ME	Conventional	Ratio
$\delta = 5$	0.85 GB	3.17 GB	26.8 %	1.94 s	1.16 s	167 %
$\delta = 10$	1.09 GB	5.90 GB	18.4 %	4.10 s	2.45 s	167 %
$\delta = 20$	1.57 GB	11.36 GB	13.8 %	6.82 s	3.67 s	186 %
$\delta = 30$	2.06 GB	16.82 GB	12.2 %	10.63 s	5.13 s	207 %

Experiment 1: Learned Post-Processing from CT Reconstructions



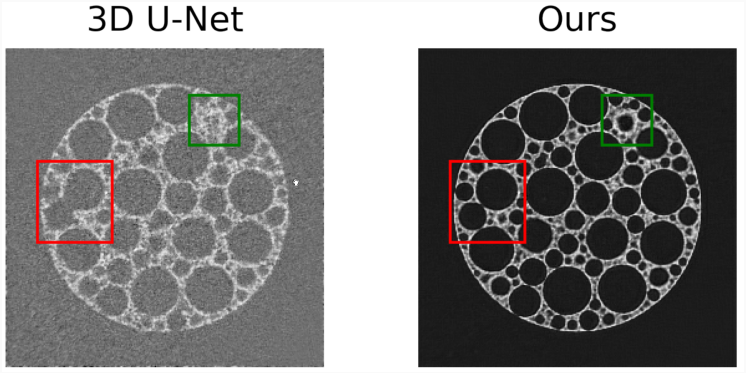
- Artificial data: foam phantoms, undersampled parallel beam CT with slanted+perturbed axis (simulated at 512^3), Poisson noise
- learned post-processing with (invertible) U-Net at 256^3 from filtered backprojection

Experiment 1: Learned Post-Processing from CT Reconstructions

Table 2: Results of learned post-processing experiments. Here, 'scales' indicates the number of different resolutions, whereas 'channel blowup' denotes the number of feature maps before reverting to one feature map again.

scales	channel blowup	3D U-Net		3D iUNet	
		SSIM	PSNR	SSIM	PSNR
4	4	0.302	13.29	0.568	14.00
4	8	0.416	13.89	0.780	14.99
8	4	0.236	12.42	0.768	15.10
8	8	0.425	13.92	0.829	15.82
8	16	-	-	**0.854**	**16.11**

Experiment 1: Learned Post-Processing from CT Reconstructions



Experiment 2: BraTS 2018

Table 3: Results on BraTS2018 validation set (acting as the test set)

	Dice score				Sensitivity			
	ET	WT	TC	avg	ET	WT	TC	avg
U-Net	0.770	**0.901**	0.828	0.833	0.776	0.914	0.813	0.834
iUNet-16	0.767	0.900	0.809	0.825	0.779	0.916	0.798	0.831
iUNet-32	0.782	0.899	0.825	0.835	0.773	0.908	0.824	0.835
iUNet-64	**0.801**	0.898	**0.850**	**0.850**	**0.796**	**0.918**	**0.829**	**0.848**

- Brain Tumour Segmentation Challenge, MRI data
- $160 \times 192 \times 128$ crop, 4 channels

Experiment 3: Normalising Flows

Let the random variable z have probability density function q, for which we will write $z \sim q(z)$. For any diffeomorphism f, it holds that

$$x := f^{-1}(z) \sim q(z) \cdot \left| \det \frac{\mathrm{d}f^{-1}(z)}{\mathrm{d}z} \right|^{-1}$$

due to the change-of-variables theorem.

$\Rightarrow$ Probability density of x (denoted p), the log-likelihood of x can be expressed as

$$\log p(x) = \log q(f(x)) + \log \left| \det \frac{\mathrm{d}f(x)}{\mathrm{d}x} \right| . \tag{4}$$

Experiment 3: Normalising Flows

Let f be parametrised by an invertible neural network. Given training data $(x_n)_{n=1}^N$, the minimiser of

$$\min_{\theta \in \Theta} -\frac{1}{N} \left(\sum_{n=1}^{N} -\log q(f(x_n)) + \log \left| \det \left. \frac{\mathrm{d}f(x)}{\mathrm{d}x} \right|_{x=x_n} \right| \right). \tag{5}$$

is a maximum likelihood estimator of the training data under the neural network.

Experiment 3: Normalising Flows

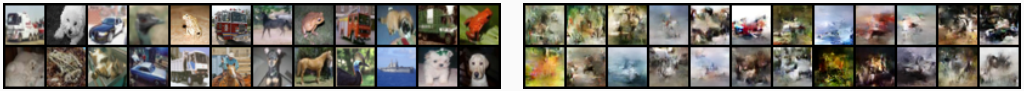


Figure 6: Left: real images. Right: generated images.

Results on CIFAR-10:

- Test NLL: 3.60 bits/dim
- RealNVP comparison: 3.49 bits/dim

Thank you!