Towards a More Complete Theory of Function Preserving Transforms

Michael Painter

Abstract

In this paper, we develop novel techniques that can be used to alter the architecture of a neural network, while maintaining the function it represents. Such operations are known as function preserving transforms and have proven useful in transferring knowledge between networks to evaluate architectures quickly, thus having applications in efficient architectures searches. Our methods allow the integration of residual connections into function preserving transforms, so we call them `R2R`. We provide a derivation for `R2R` and show that it yields competitive performance with other function preserving transforms, thereby decreasing the restrictions on deep learning architectures that can be extended through function preserving transforms. We perform a comparative analysis with other function preserving transforms such as `Net2Net` and Network Morphisms, where we shed light on their differences and individual use cases. Finally, we show the effectiveness of `R2R` to train models quickly, as well as its ability to learn a more diverse set of filters on image classification tasks compared to `Net2Net` and Network Morphisms.

1. Forword

I completed the majority of this work for a project for the CS231n: Deep Learning for Computer Vision course at Stanford. The original premise of the project was to use the function-preserving transforms described below to perform an efficient architecture search. This was an optimistic project given the only compute resources I had were Microsoft Azure credits. At the start of my PhD in 2018 I spent some time writing this work up for an ICML conference submission but subsequently didn't manage to find time to work on this further, and I don't think I will be able to find the time to work on this for the foreseeable future.

I submitted this work to ICML2019 and received scores of {`Accept, Weak Accept, Weak Reject, Reject`}, where the main strengths highlighted were the mathematical derivations and the main concerns had to do with the performance of the networks used in the evaluation. I spent a small amount of time in 2019 starting to work on addressing these concerns and working on getting experimental results with larger-scale neural networks and datasets. Unfortunately as mentioned above, I did not find time to finish addressing the concerns with experimental results.

In retrospect, the experimental section could have been made clearer and the results presented more cleanly. For a keen reader, I can provide some additional context for the experimental section. The experiments are supposed to be *toy experiments* where the networks used are very small compared to any network that would achieve a state-of-the-art performance. What I intended to demonstrate was if you take a network architecture A, you can train the network to the *same* performance (not better) by starting with a smaller architecture B, and then using function preserving transforms mid-training to transform architecture A into architecture B. This should allow you to train a network to the same performance, hopefully using less floating point operations (i.e. less GPU time). The plots don't show this too clearly, as each plot only shows curves for one training run of each architecture. If the curves used even three or five training runs, then Figure 8 should demonstrate this effect (see that in the first plot, the red curve is always on top, but not in the second, noting that they correspond to the same training runs with different x-axes).

Given recent developments using extremely large neural networks with billions and trillions of parameters, I'm guessing that there may be some interest in trying to train them more efficiently, so I'm publishing this work to arxiv on the odd chance that someone may find this work useful. The idea of using symmetry in the initialisation of new weights in a function-preserving transform should easily generalize to more recent attention and transformer architectures.

The only updates I've made to this paper for this arxiv version are: (1) changing the authorship from anonymous; (2) adding this foreword (and shifting the introduction; (3) adding a link to the code in Appendix A; (4) a small comment about symmetry breaking in Appendix B.

Finally, I've copied the abstract onto the next page to somewhat preserve the original formatting of the paper, as I don't

Correspondence to: Michael Painter <mpainter.thts@gmail.com>.

have to conform to any page limit ☺.

Abstract

In this paper, we develop novel techniques that can be used to alter the architecture of a neural network, while maintaining the function it represents. Such operations are known as function preserving transforms and have proven useful in transferring knowledge between networks to evaluate architectures quickly, thus having applications in efficient architectures searches. Our methods allow the integration of residual connections into function preserving transforms, so we call them `R2R`. We provide a derivation for `R2R` and show that it yields competitive performance with other function preserving transforms, thereby decreasing the restrictions on deep learning architectures that can be extended through function preserving transforms. We perform a comparative analysis with other function preserving transforms such as `Net2Net` and Network Morphisms, where we shed light on their differences and individual use cases. Finally, we show the effectiveness of `R2R` to train models quickly, as well as its ability to learn a more diverse set of filters on image classification tasks compared to `Net2Net` and Network Morphisms.

2. Introduction

Function preserving transforms (FPTs), also known as *network morphisms*, provide a method for transferring the performance of a *teacher network*, f, to a *student network*, g, with a different (typically larger) architecture. This is assured by computing the initial parameters θ_s for the student network from the parameters of the teacher θ_t, such that they are *function preserving*: $\forall x.\ f(x;\theta_t) = g(x;\theta_s)$.

FPTs allow us to dynamically increase the capacity of a network, during training, without degrading performance. This enables us to leverage an already trained teacher network and perform a fast evaluation of new architectures, without incurring the overhead of training them from scratch. In particular, FPTs have applications in efficient architecture searches (Cai et al., 2018; Jin et al., 2018). Many architectures can be evaluated quickly by repeatedly applying FPTs, thereby amortizing the high cost of training from a random initialization over many architecture evaluations.

Existing methods for performing FPTs include `Net2Net` (Chen et al., 2015) and Network Morphing (Wei et al., 2016). `Net2Net` provides techniques for *widening* hidden layers in the network and for introducing new hidden layers (*deepening*). Conversely, Network Morphing proposes alternative techniques for widening and deepening, as well as methods for *kernel size morphing* and *sub-net morphing*.

Previous FPTs do not handle residual connections (He et al., 2016), however, they are frequently used to train deep networks faster and more reliably, making them commonplace in the deep learning community. Motivated by this, we propose `R2R`, consisting of two new FPTs `R2WiderR` and `R2DeeperR`, which allow neural networks with residual connections to be morphed.

We believe that the two largest barriers preventing FPTs from being used beyond architecture searches are the complexity overhead of understanding and implementing the transform, alongside a lack of flexibility in the architectures that the FPTs can be applied to. We aim to address these issues by introducing simple to understand and implement FPTs and by facilitating an increase in the number of architectures that FPTs can be applied to.

Our contributions are as follows: (1) proposing `R2WiderR` and `R2DeeperR`, novel FPTs that are compatible with residual connections; (2) performing a comparative analysis of `R2R`, `Net2Net` and Network Morphism; (3) introducing a novel evaluation to show that FPTs can be used to train networks to convergence with greater computational efficiency on the Cifar-10 (Krizhevsky & Hinton, 2009).

3. Definitions and notation

Say tensor T has shape $(\alpha \times \beta \times \gamma)$ to formally denote that $T \in \mathbb{R}^{\alpha\times\beta\times\gamma}$. Also consider a convolutional kernel W and a volume x with shapes $(C_o \times C_i \times k_h \times k_w)$ and $(C_i \times h \times w)$ respectively. Let $W_{ij} * x_k$ be a 2D convolution defined in any standard way, which may include (integer) stride, padding and dilation for example. For notational convenience assume that the spatial dimensions are preserved (i.e. $W_{ij} * x_k$ has shape $(h \times w)$). The arguments made in later sections are valid without the need for this assumption, albeit with a slightly more complex shape analysis.

In convolutional neural networks, a convolution is often considered to operate between 4D and 3D tensors, such that $W * x$ has shape $(C_o \times h \times w)$. This complex *convolution product* can also be defined using matrix notation and the 2D convolution as follows:

$$
\begin{aligned}
W * x &\overset{\text{def}}{=} \begin{bmatrix} W_{11} & W_{12} & \dots & W_{1C_i} \\ W_{21} & W_{22} & \dots & W_{2C_i} \\ \vdots & \vdots & \ddots & \vdots \\ W_{C_o1} & W_{C_o2} & \dots & W_{C_oC_i} \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{C_i} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{m=1}^{C_i} W_{1m} * x_m \\ \sum_{m=1}^{C_i} W_{2m} * x_m \\ \vdots \\ \sum_{m=1}^{C_i} W_{C_om} * x_m \end{bmatrix}. \qquad (1)
\end{aligned}
$$

W_{ij} and x_k are 2D matrices, with shapes $(k_h \times k_w)$ and $(h \times$

w) respectively. *Block convolutions*, can be considered in a similar fasion to block matrix multiplication. For example, if A, B, C, D have shape $(4, 2, 5, 5)$ and a, b have shape $(2, 100, 100)$, then we may write:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} * \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} A * a + B * b \\ C * a + D * b \end{bmatrix}. \tag{2}$$

Note that if $k_h = k_w = h = w = 1$, then the convolution operation above reduces to matrix multiplication (with some unnecessary additional dimensions). Because of this reduction, we will discuss our function preserving transform only in a convolutional setting, and the corresponding formulation using linear layers follows.

Consider a standard formulation of a convolutional neural network (ignoring residual connections for now). Let the i^{th} intermediate volume be $x^{(i)}$ with shape $(C^{(i)} \times h^{(i)} \times w^{(i)})$, let the i^{th} convolutional kernel be $W^{(i)}$ with shape $(C^{(i)} \times C^{(i-1)} \times k_h^{(i)} \times k_w^{(i)})$ and corresponding bias $b^{(i)}$ with shape $(C^{(i)})$. Consider also a non-linearity $\sigma^{(i)}_{\rho^{(i)}} : \mathbb{R}^{C^{(i)} \times h^{(i-1)} \times w^{(i-1)}} \to \mathbb{R}^{C^{(i)} \times h^{(i)} \times w^{(i)}}$ that is applied to the i^{th} intermediate volume. Parameters $\rho^{(i)}$ are included in the definition of $\sigma^{(i)}_{\rho^{(i)}}$ to indicate that there may be some parameters associated with the non-linearity. Given the above definitions, $x^{(i+1)}$ and $x^{(i)}$ can be computed using the following equations:

$$x^{(i)} = \sigma^{(i)}_{\rho^{(i)}} \left(W^{(i)} * x^{(i-1)} + b^{(i)} \right) \tag{3}$$

$$x^{(i+1)} = \sigma^{(i+1)}_{\rho^{(i+1)}} \left(W^{(i+1)} * x^{(i)} + b^{(i+1)} \right). \tag{4}$$

In this work *non-linearity* refers to any part of the network other than convolutions and linear (matrix) operations. This includes batch norm (Ioffe & Szegedy, 2015), max pooling and average pooling layers. In our experiments, we compose ReLUs, pooling layers and batch norm into the non-linearity, hence batch norms parameters would be included in $\rho^{(i)}$.

Additionally, the non-liniarity $\sigma^{(i)}_{\rho^{(i)}}$ is assumed to operate on each channel independently. That is, for any input x with shape $(C^{(i)} \times h^{(i)} \times w^{(i)})$ we can write:

$$\sigma^{(i)}_{\rho^{(i)}} \left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{C^{(i)}} \end{bmatrix} \right) = \begin{bmatrix} \sigma^{(i)}_{\rho^{(i)}_1}(x_1) \\ \sigma^{(i)}_{\rho^{(i)}_2}(x_2) \\ \vdots \\ \sigma^{(i)}_{\rho^{(i)}_{C^{(i)}}}(x_{C^{(i)}}) \end{bmatrix}. \tag{5}$$

4. Related work

We provide an overview of the only two other function preserving transform methods to date (to the best of our knowledge). Then, we consider a few existing applications of these function preserving transformations.

4.1. Net2Net

`Net2Net` introduces two function preserving transforms: `Net2WiderNet` for adding more hidden units into the network layers and `Net2DeeperNet` for expanding the network with additional hidden layers. To widen layer the i^{th} dense layer in a network, $h^{(i)} = \psi(W^{(i)} h^{(i-1)})$, from n to q hidden units using `Net2WiderNet`, the weight matrices $W^{(i)}$ and $W^{(i+1)}$ are replaced with $U^{(i)}$ and $U^{(i+1)}$ where:

$$U^{(i)}_{j,k} = W^{(i)}_{g(j),k}, \quad U^{(i+1)}_{h,j} = \frac{1}{|\{x \mid g(x) = g(j)\}|} W^{(i+1)}_{h,g(j)} \tag{6}$$

and where

$$g(j) = \begin{cases} j, & \text{for } j \leq n \\ \text{random sample in } \{1, 2, \dots n\}, & \text{for } n < j \leq q. \end{cases}$$

`Net2WiderNet` duplicates columns from $W^{(i)}$ to widen a hidden layers and obtain $U^{(i)}$ and then adjusts for their replication in $U^{(i+1)}$ to be function preserving.

`Net2DeeperNet` can be used to increase the number of hidden layers in a network by replacing a layer $h^{(i)} = \psi(W^{(i)} h^{(i-1)})$ with two layers $h^{(i)} = \psi(V^{(i)} \psi(W^{(i)} h^{(i-1)}))$. The weight matrix for the newly added layer $V^{(i)}$ is initialized to the identity matrix. `Net2DeeperNet` only works with idempotent activation functions ψ, where $\psi(I\psi(v)) = \psi(v)$. While, ReLU activations satisfy this property, sigmoid and tanh do not.

4.2. Network morphism

Network Morphism (Wei et al., 2016) generalizes `Net2Net`, and considers a set of *morphisms* between neural networks that can be used to also increase the kernel size in convolutional layers and to introduce more Network In Network style subnetworks (Lin et al., 2013).

(Wei et al., 2016) derive their morphisms in a principled way, by introducing the *network morphism equation*, which provides a sufficient condition for new parameters must satisfy in order for the morphism to be function preserving. To widen a network (ignoring non-linearities) their equations reduce down to solving the equation:

$$\begin{aligned} W^{(i)} * W^{(i-1)} &= \begin{bmatrix} W^{(i)} & U^{(i)} \end{bmatrix} * \begin{bmatrix} W^{(i-1)} \\ U^{(i-1)} \end{bmatrix} \\ &= W^{(i)} * W^{(i-1)} + U^{(i)} * U^{(i-1)}, \end{aligned} \tag{7}$$

for new parameters $U^{(i)}, U^{(i-1)}$. They solve this by setting either $U^{(i)}$ or $U^{(i-1)}$ to be zero, and we will refer to

widening a network in this way as `NetMorph`. They also generalize the equations to include non-linearities.

(Wei et al., 2016) also define kernel size morphing, which zero pads a convolutional kernel with zeros in the spatial dimension, and subnet morphism, which in essence adapts `Net2WiderNet` to operate on sebnetworks.

4.3. Architecture search

Architecture search algorithms can be considered to simultaneously optimize over neural network architectures and their hyper-parameters. This usually involves training many models and comparing their performance, which can be very computationally expensive, requiring hundreds if not thousands of *GPU days*. To optimize over architectures a number of methods have been considered such as using evolutionary strategies or reinforcement learning (Real et al., 2018; Zoph & Le, 2016). Moreover, to lower the computational costs *efficient architecture searches* have been developed.

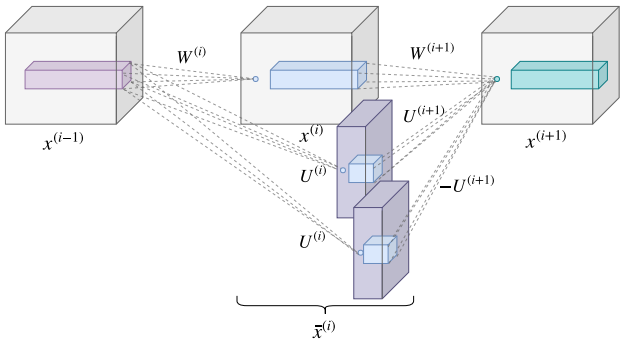

Figure 1. An overview of `R2WiderR`. The purple volumes represent $x_L^{(i)}$ and $x_R^{(i)}$ in the widened volume $\bar{x}^{(i)}$. We label the convolution operations with their initializations, and it shows that $x_L^{(i)}$ and $x_R^{(i)}$ cancel each other out in $x^{(i+1)}$.

`Net2Net` and Network Morphism have both been used in *efficient architecture searches*, which aim to reduce the computation time needed to evaluate architectures, thus making architecture search a more approachable technique (Cai et al., 2018; Jin et al., 2018). In parallel to using FPTs, there have been other efforts to making architecture search efficient, such as using a differentiable encoding of architectures and using gradient based algorithms, and using parameter sharing (Liu et al., 2018; Pham et al., 2018).

5. R2R

R2R consists of two FPTs: `R2WiderR`and `R2DeeperR`, which provide alternative methods for increasing the size of hidden layers and for adding hidden layers in a neural network.

5.1. R2WiderR

To increase the capacity of a network, some intermediate hidden volume $x^{(i)}$ can be *widened* by increasing its number of channels. We propose `R2WiderR`, a novel way of adding $2E$ channels by padding $x^{(i)}$ with $x_L^{(i)}$ and $x_R^{(i)}$, of shapes $(E \times h^{(i)} \times w^{(i)})$, thus obtaining a new volume $\bar{x}^{(i)}$. To assure that the value of $x^{(i+1)}$ does not change, we initialize new parameters such that $x_L^{(i)} = x_R^{(i)}$, and that the contributions of $x_L^{(i)}$ and $x_R^{(i)}$ cancel each other out when computing $x^{(i+1)}$. This idea is visualized in figure 1.

Let $U^{(i)}$ be an arbitrary tensor with shape $(E \times C^{(i-1)} \times k_h^{(i-1)} \times k_w^{(i-1)})$, let $c^{(i)}$ be an arbitrary tensor with shape (E) and let $U^{(i+1)}$ be an arbitrary tensor, with shape $(C^{(i+1)} \times E \times k_h^{(i)} \times k_w^{(i)})$. Additionally, let $\bar{\rho}^{(i)}$ include all the parameters in $\rho^{(i)}$ and any additional parameters required, such as batch norm's affine parameters, as discussed previously. We define the new kernels and biases as follows:

$$\bar{W}^{(i)} = \begin{bmatrix} W^{(i)} \\ U^{(i)} \\ U^{(i)} \end{bmatrix}, \ \bar{b}^{(i)} = \begin{bmatrix} b^{(i)} \\ c^{(i)} \\ c^{(i)} \end{bmatrix}, \tag{8}$$

$$\bar{W}^{(i+1)} = \begin{bmatrix} W^{(i+1)} & U^{(i+1)} & -U^{(i+1)} \end{bmatrix}. \tag{9}$$

The intermediate volume $\bar{x}^{(i)}$ can now be computed:

$$\bar{x}^{(i)} = \begin{bmatrix} x^{(i)} \\ x_L^{(i)} \\ x_R^{(i)} \end{bmatrix} = \sigma_{\bar{\rho}^{(i)}}^{(i)} \left(\bar{W}^{(i)} * x^{(i-1)} + \bar{b}^{(i)} \right) \tag{10}$$

$$= \begin{bmatrix} \sigma_{\bar{\rho}^{(i)}}^{(i)} (W^{(i)} * x^{(i-1)} + b^{(i)}) \\ \sigma_{\bar{\rho}^{(i)}}^{(i)} (U^{(i)} * x^{(i-1)} + c^{(i)}) \\ \sigma_{\bar{\rho}^{(i)}}^{(i)} (U^{(i)} * x^{(i-1)} + c^{(i)}) \end{bmatrix}. \tag{11}$$

In equation (11) it is clear that we have $x_L^{(i)} = x_R^{(i)}$. Thus,

$$\begin{aligned} &\bar{W}^{(i+1)} * \bar{x}^{(i)} \\ &= W^{(i+1)} * x^{(i)} + U^{(i+1)} * x_L^{(i)} - U^{(i+1)} * x_R^{(i)} \\ &= W^{(i+1)} * x^{(i)}. \end{aligned} \tag{12}$$

Using equation (12) to compute the $(i+1)^{\text{th}}$ layer, we obtain:

$$x^{(i+1)} = \sigma_{\rho^{(i+1)}}^{(i+1)} \left(\bar{W}^{(i+1)} * \bar{x}^{(i)} + b^{(i+1)} \right). \tag{13}$$

To conclude, the `R2WiderR` transformation defines new kernels and biases as in (8) and replaces equations (3), (4) with the equations (10) and (13) in the neural network. We provide a full derivation of `R2WiderR`, in appendix B.

5.2. Residual connections in `R2WiderR`

We outline how to adapt residual connections to account for `R2WiderR` operations. Figure 2 gives a schematic overview of the idea. We consider a simplified case where we do not handle (spatial) shape matching, or *down-sampling*, and only show how to adapt a residual connection for a single application of `R2WiderR`. We provide a more general argument that shows how to handle down-sampling and repeated applications of `R2WiderR` in appendix B.

Suppose that we have a residual connection from the ℓ^{th} intermediate layer to the i^{th} layer, which we write as

$$x^{(i)} = \mathcal{F}^{(i)}_{\theta^{(i)}}\left(x^{(i-1)}\right) + x^{(\ell)}, \tag{14}$$

where $\theta^{(i)} = \{W^{(i)}, b^{(i)}, \rho^{(i)}\}$ and

$$\begin{aligned} x^{(i)} &= \mathcal{F}^{(i)}_{\theta^{(i)}}\left(x^{(i-1)}\right) \\ &\overset{\text{def}}{=} \sigma^{(i)}_{\rho^{(i)}}\left(W^{(i)} * x^{(i-1)} + b^{(i)}\right). \end{aligned} \tag{15}$$

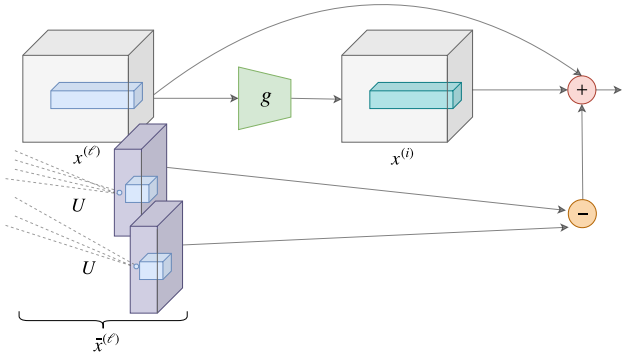

Figure 2. A schematic of how to adapt simple residual connections for the `R2WiderR` operation, in the simple case. The purple volumes indicate the new parameters in $\bar{x}^{(\ell)}$, and g represents the composition of layers $\ell + 1$ to $i - 1$ in the network.

If we apply `R2WiderR` to $x^{(\ell)}$, then equation (11) gives:

$$\bar{x}^{(\ell)} = \begin{bmatrix} x^{(\ell)} \\ x^{(\ell)}_L \\ x^{(\ell)}_R \end{bmatrix} = \begin{bmatrix} \mathcal{F}^{(\ell)}_{\theta^{(\ell)}}\left(x^{(\ell-1)}\right) \\ \mathcal{F}^{(\ell)}_{\theta^{(\ell)}_L}\left(x^{(\ell-1)}\right) \\ \mathcal{F}^{(\ell)}_{\theta^{(\ell)}_R}\left(x^{(\ell-1)}\right) \end{bmatrix}, \tag{16}$$

with $\theta^{(\ell)}_L = \{W^{(\ell)}_L, b^{(\ell)}_L, \rho^{(\ell)}_L\} = \{W^{(\ell)}_R, b^{(\ell)}_R, \rho^{(\ell)}_R\} = \theta^{(\ell)}_R$ and $x^{(\ell)}_L = x^{(\ell)}_R$. Therefore, if we let

$$r\left(\bar{x}^{(\ell)}\right) = x^{(\ell)} + x^{(\ell)}_L - x^{(\ell)}_R, \tag{17}$$

then we may use

$$x^{(i)} = \mathcal{F}^{(i)}_{\theta^{(i)}}\left(x^{(i-1)}\right) + r\left(x^{(\ell)}\right) \tag{18}$$

in place of equation (14), when computing $x^{(i)}$.

5.3. Zero initializations

Before we define `R2DeeperR`, we first consider *zero initializations*, that is, how we can initialize a network such that its output is always zero, while still being able to train it. Thus, so we aim to have as many arbitrary parameters as possible for the zero initialization. Formally, we want to find parameters θ_f, such that $\forall x.\ f(x; \theta_f) = 0$.

Let $g_{\theta^{(1:n)}}$ be an arbitrary neural network, with parameters $\theta^{(1:n)}$, and with n layers formed using equations similar to (3) and (4). Let $x^{(n)} = g_{\theta^{(1:n)}}(x^{(0)})$. To create a constant zero output, we will add an additional two layers on the output, using the same trick we used to derive `R2WiderR`.

By letting

$$W^{(o_1)} = \begin{bmatrix} U^{(o_1)} \\ U^{(o_1)} \end{bmatrix}, \quad W^{(o_2)} = \begin{bmatrix} U^{(o_2)} & -U^{(o_2)} \end{bmatrix}, \tag{19}$$

$$b^{(o_1)} = \begin{bmatrix} c^{(o_1)} \\ c^{(o_1)} \end{bmatrix}, \quad b^{(o_2)} = \begin{bmatrix} c^{(o_2)} & -c^{(o_2)} \end{bmatrix}, \tag{20}$$

where $U^{(o_1)}, U^{(o_2)}, b^{(o_1)}, b^{(o_2)}$ have shapes $(C^{(o_1)} \times C^{(n)} \times k^{(n)}_h \times k^{(n)}_w)$, $(C^{(o_2)} \times C^{(o_1)} \times k^{(o_1)}_h \times k^{(o_1)}_w)$, $(C^{(o_1)})$ and $(C^{(o_2)})$ respectively, the additional two layers become:

$$x^{(o_1)} = \sigma^{(o_1)}_{\rho^{(o_1)}}\left(W^{(o_1)} * x^{(n)} + b^{(o_1)}\right), \tag{21}$$

$$x^{(o_2)} = \sigma^{(o_2)}_{\rho^{(o_2)}}\left(W^{(o_2)} * x^{(o_1)} + b^{(o_2)}\right). \tag{22}$$

Expanding and simplifying, similar to section 5.1, gives:

$$x^{(o_2)} = \sigma^{(o_2)}_{\rho^{(o_2)}}(0). \tag{23}$$

Therefore, provided we chose $\sigma^{(o_2)}$ such that $\sigma^{(o_2)}_{\rho^{(o_2)}}(0) = 0$, then we have a zero initialized network, with output $x^{(o_2)}$ and $n + 2$ layers. See appendix B for a full derivation.

5.4. `R2DeeperR`

We propose a novel method for deepening a network using a residual block, initialized to be an identity function. Let z_θ be a network that is zero initialized in the way described in section 5.3, and therefore we have for all x that $z_\theta(x) = 0$. Adding an x to each side then gives

$$z_\theta(x) + x = x. \tag{24}$$

Let $x^{(e)}$ be a new volume we wish to add between the consecutive volumes $x^{(i)}$ and $x^{(i+1)}$. For simplicity assume that $x^{(i)}$ and $x^{(i+1)}$ are computed as described in equations (3) and (4), however, they be generalized to include residual connections. The layerwise operation now becomes:

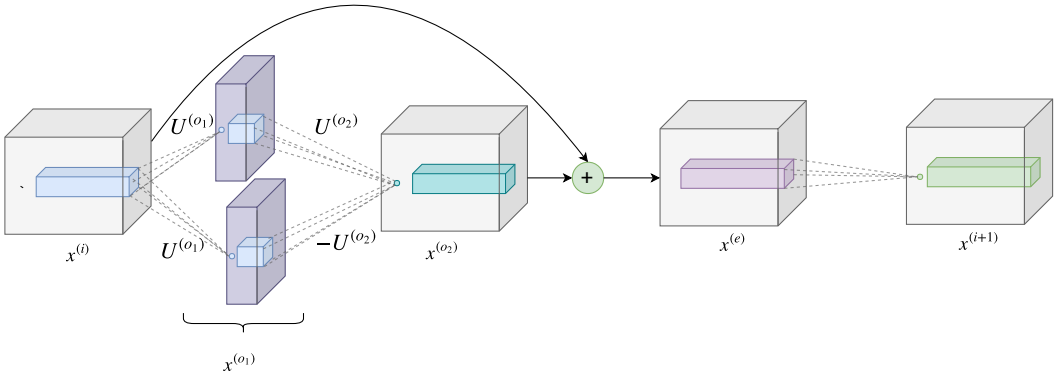

Figure 3. A schematic of the `R2DeeperR` operation. Before `R2DeeperR` is applied, $x^{(o_1)}$ and $x^{(o_2)}$ can be ignored, and the network would compute $x^{(i+1)}$ directly from $x^{(i)}$. After `R2DeeperR` is applied $U^{(o_1)}$ is used to make two identical volumes, which cancel each other out in volume $x^{(o_2)}$, so that $x^{(o_2)} = 0$. The residual connection assures thaen that $x^{(e)} = x^{(i)}$.

$$x^{(i)} = \sigma^{(i)}_{\rho^{(i)}} \left(W^{(i)} * x^{(i-1)} + b^{(i)} \right), \tag{25}$$

$$x^{(e)} = z_\theta \left(x^{(i)} \right) + x^{(i)}, \tag{26}$$

$$x^{(i+1)} = \sigma^{(i+1)}_{\rho^{(i+1)}} \left(W^{(i+1)} * x^{(e)} + b^{(i+1)} \right). \tag{27}$$

From equations (24) and (24) we must have that $x^{(e)} = x^{(i)}$. Substituting $x^{(e)}$ for $x^{(i)}$ in equation (4), the $(i+1)^{\text{th}}$ layer, gives equation (27).

6. A comparison of function preserving transforms

This section discusse similarities and differences between each of the FPTs `R2R`, `Net2Net` and `NetMorph`, with the aim of providing insight into their use cases. A summary of differences can be seen in table 6.

6.1. Non-linearities

Each of the FPTs makes assumptions about the form of non-linearities they can be applied with. This restricts what activation functions can be used. Let ψ denote a non-linearity used in the networks that we are applying the FPTs to. `R2WiderR`, `Net2WiderNet` and `NetMorph` each need the assumption that the non-linearity *maintains channel dependencies*, as described in equation (5). That is, if ψ is applied to volume x with some i, j such that $i \neq j$ and $x_i = x_j$, then we require $\psi(x)_i = \psi(x)_j$.

`R2DeeperR`, as stated in section 5.3, requires the last non-linearity in the introduced residual block to have a fixed point at zero $\psi(0) = 0$. Whereas, in `Net2DeeperNet` every non-linearity needs to be idempotent, $\psi(\psi(\cdot)) = \psi(\cdot)$. (Chen et al., 2015) incorporate batch normalization by setting its affine parameters to invert the normalization and produce and identity function (for a specific mini-batch).[1]

(Wei et al., 2016) manage to avoid these problems by introducing *P-activations*, which use a parameter to interpolate between an identity function (which satisfies all the properties discussed above) and any arbitrary non-linearity during training. This method could be applied to any of the FPTs.

6.2. Residual connections

`R2R` not only allows the incorporation of residual connections in FPTs, but necessarily requires residual connections for `Net2DeeperNet`. Our deepening operation is fundamentally different to those considered by (Chen et al., 2015) and (Wei et al., 2016), who approach the problem as one of matrix/convolution decomposition, to split one network layer into two. The `Net2DeeperNet` operation makes use of the matrix decomposition $A = IA$, where I is the identity matrix, whereas (Wei et al., 2016) consider that more complex, and dense, decompositions could be used. In contrast, `R2DeeperR` introduces a completely new identity function in the middle of a neural network, using the residual connection to provide the identity. `R2DeeperR` allows for a dense initialization of new parameters, and introduces an entire residual block for each application of `R2DeeperR`, rather than introducing an extra layer for each application.

We note that both `Net2WiderNet` and `NetMorph` could be adapted to allow for residual connections, similar to section 5.2. For `NetMorph` they could be introduced in almost identically as for `R2WiderR`, however, for `Net2WiderNet`it is not straightforward how to do so.

Although it has been shown that residual connections are not necessary to train deep networks, although they are still widely used to help train networks stably and quickly

[1] Technically it will always be possible to find an input x to a batch norm layer, bn, such that $\text{bn}(x) \neq x$. The closest we can get is finding affine parameters such that $\mathbb{E}[\text{bn}(x)] = x$.

Table 1. An overview of the differences between R2R, Net2Net (Chen et al., 2015) and Network Morphism (Wei et al., 2016). We use MCD as a shorthand for saying that a function maintains channel dependencies.

	R2R	NET2NET	NETMORPH
ACTIVATION FUNCTIONS (DEEPEN)	MCD+FIXED POINT AT ZERO	IDEMPOTENT	N/A
ACTIVATION FUNCTIONS (WIDEN)	MCD	MCD	MCD+FIXED POINT AT ZERO
FUNCTION PRESERVING	$\surd$	IN EXCPECTATION	$\surd$
REQUIRES NOISE	$\times$	$\surd$	$\times$
RESIDUAL CONNECTIONS	$\surd$	$\times$	$\times$
DEGREES OF FREEDOM	HALF	NONE	HALF
PRESERVES EXISTING PARAMETERS	$\surd$	$\times$	$\surd$
NUMBER OF NEW LAYERS (DEEPEN)	≥ 2	ANY	N/A
NUMBER OF NEW CHANNELS (WIDEN)	EVEN	ANY	ANY

(Szegedy et al., 2017).

6.3. Preservation of parameters

A useful property for the transformation is whether it preserves already existing parameters, i.e any parameters that existed in the teacher network are not changed in the student network. This property holds for R2R and NetMorph, however, it does not for Net2Net. This property allows the teacher and student networks to co-exist with shared parameters, potentially allowing for efficient training of multiple models of different sizes.

6.4. Visualizing the transforms

To help demonstrate the differences between R2WiderR Net2WiderNet and NetMorph we visualize the weights in the first layer of a a three layer convolutions network, initially containing 16 filters. We train the network on the Cifar-10 classification task (Krizhevsky & Hinton, 2009) until convergence. Then we apply R2WiderR, Net2WiderNet or NetMorph, to add another 16 filters to the first layer, giving a total of 32. After widening we train again until convergence and compare the weights to when the network was widened. The visualizations can be seen in figure 4.

Because in R2WiderR and NetMorph there is freedom to choose how about half of the new parameters are initialized, we tended to see some more randomness in the training of the new filters and the filters they converge to. In contrast, in Net2WiderNet there is no freedom in the parameter initialization, and we found the *orientation* of the new filters often did not change from they initial set-up. However, the colors being detected typically changed. As an example, if a new filter from Net2WiderNet was initialized as a pink/green vertical edge detector, after further training it may become a blue/orange edge detector.

Finally the visualizations for R2WiderR indicate that the new 16 filters were able to learn different filter orientations and that their symmetry was broken without having to add more noise (as required in Net2Net). However, we found that the filters learned by R2WiderR were often noisy themselves. With NetMorph we found that the new filters tended to be quite noisy and/or contain relatively small weights, where the latter is depicted in figure 4. We think that this noise in R2WiderR and the small-weights in NetMorph could be a result of a faint *training signal*, after the teacher network has already learned to classify many examples in the training set correctly.

For all three FPTs, the first 16 filters learned tend to still be present in the larger set of 32 filters after widening and training to convergence again. However, in R2WiderR initialising the weights with a larger scale compared to the existing filters results in loosing the first 16 filters.

7. Experiments

We consider training ResNet-18 networks on Cifar-10 to compare the performance of the FPTs. In appendix C we define the ResNetCifar-10(r) and ResNetCifar-18(r) architectures in more detail, where the standard ResNet architectures (He et al., 2016) have been adapted for the smaller images size in Cifar-10. r denotes that we use r times as many filters in every convolutional layer, We chose r to ensure that the performance on Cifar-10 is limited in the teacher network, so that the student network has something to improve upon.

In all of our experiments we initialize the free parameters with standard deviation equal to the numerical standard deviation of the existing weights in the kernel for widening and the standard deviation of the weights in kernel prior when deepening. Specifically, if we wanted to initialize new weights with numerical variance s^2 then we sampled from the uniform distribution $U(-\frac{s}{\sqrt{3}}, \frac{s}{\sqrt{3}})$.

All training hyper-parameters are detailed in appendix C.

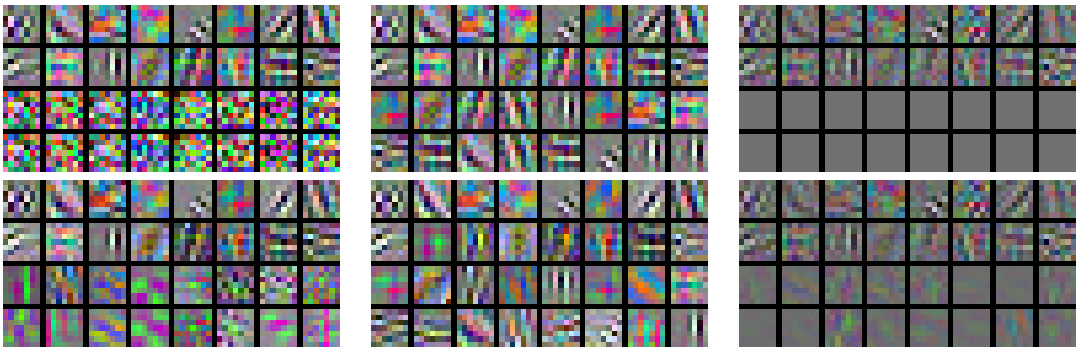

Figure 4. Visualization of weights from a 7×7 convolution layer trained on Cifar-10. From left to right we have `R2R`, `Net2Net` and `NetMorph` schemes. On the top row we have visualizations immediately after the FPT operation, and the bottom row shows the filters after they have all been trained to convergence.

7.1. Network convergence tests

In line with prior work we perform tests similar to (Chen et al., 2015), and we compare the converged accuracies when using different FPTs. To test the widening operators we compare performance of ResNetCifar-18(2^{-3}) networks on Cifar-10. To begin with, a teacher network, with $\sqrt{2}$ fewer channels (i.e. a ResNetCifar-18($2^{-3.5}$) network) is trained to convergence. This network is then used as a teacher network for each of `R2WiderR`, `Net2WiderNet` and `NetMorph`, and the respective student networks are trained to convergence. Additionally, we also compare with a student using a random padding, and with a ResNetCifar-18(2^{-3}) initialized completely randomly.

We found that all three FPTs are competitive and converged to at least as good score as the randomly initialized networks. To apply `Net2WiderNet` we had to remove the residual connections from ResNetCifar-18(2^{-3}). We found that this particular network, similar to (He et al., 2016), could not reach the same performance as the one with residual connections; thus, we believe that it would be unfair, given this data, to draw any direct conclusions on the performance of `Net2Net` versus `R2R`. Validation curves are shown in figure 5 and final validation accuracies are in table 2.

To test deepening operators we compare the performance of ResNetCifar-18(2^{-3}) networks on Cifar-10. This time we use a ResNetCifar-10(2^{-4}) for the teacher network, and then use either `R2DeeperR` or `Net2DeeperNet`to add eight layers. We also compare with random padding and a randomly initialized ResNetCifar-18(2^{-3}). Validation curves are in figure 6, and final accuracies in table 2.

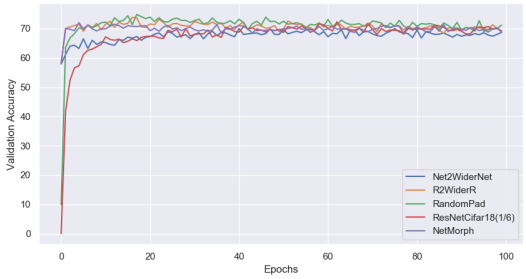

Figure 5. Validation curves comparing a student network using each of `Net2WiderNet`, `R2WiderR`, `NetMorph`, with baselines of random padding and training the ResNet "from scratch".

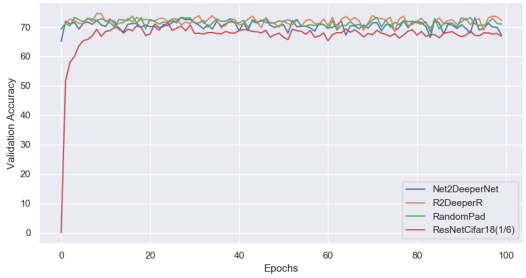

Figure 6. Validation curves comparing a student network using each of `Net2DeeperNet`, `R2DeeperR`, `NetMorph`, with baselines of random padding and training the ResNet "from scratch".

We also observed that if the teacher network overfits during training, then the student has minimal improvement over the teacher, as can be seen in figure 7. This can be explained by the weak training signal due to the teacher having overfitted, and it therefore struggles to improve on the initial performance. Due to this phenomenon we conclude that regularization to preventing overfitting is essential for a student

Table 2. Final validation scores for student networks at the *end* of training. All networks reach similar performance.

	VALIDATION ACCURACY
RESNETCIFAR(2^{-3})	69.9%
R2WIDERR STUDENT	69.9%
NET2WIDERNET STUDENT	69.6%
NETMORPH STUDENT	69.8%
RANDOMPADWIDEN STUDENT	70.2%
R2DEEPERR STUDENT	72.1%
NET2DEEPERNET STUDENT	70.5%
RANDOMPADDEEPEN STUDENT	71.5%

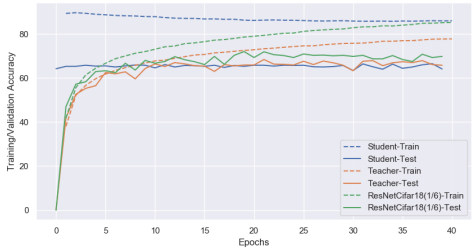

Figure 7. In this example we training curves where the teacher network (a ResNetCifar-18($2^{-3.5}$)) was allowed to overfit. The student network (ResNetCifar-18(2^{-3}) is unable to make any additional performance gains, and a randomly initialized ResNetCifar-18(2^{-3}) is able to reach a better converged performance.

network to perform better than the teacher.

7.2. Faster training tests

We also consider if FPTs can be used to train a network to convergence faster than initializing the network from scratch, while still achieving the same performance.In these experiments we train a ResNetCifar-18($2^{-3.5}$) or ResNetCifar-10(2^{-3}) teacher network and then appropriately widening or deepening the network using `R2R`, `Net2Net` or Network Morphism in the middle of training. Each network is then trained until it converges.

In figure 8 we see that training is slower with respect to the number of training updates. However, when we consider the actual number of floating point operations (FLOPs) we find that using the FPTs was faster. This suggests that FPTs can be used in similar training procedures to trade off between computational and sample complexities of training.

These experiments also indicate that we need to be careful when initializing new parameters in `R2R`. We observed that if the new weights were initialized with large values with respect to existing weights in the network it lead to instability in the training, and a drop in performance immediately after the widen. We illustrate this case in the Appendix D.

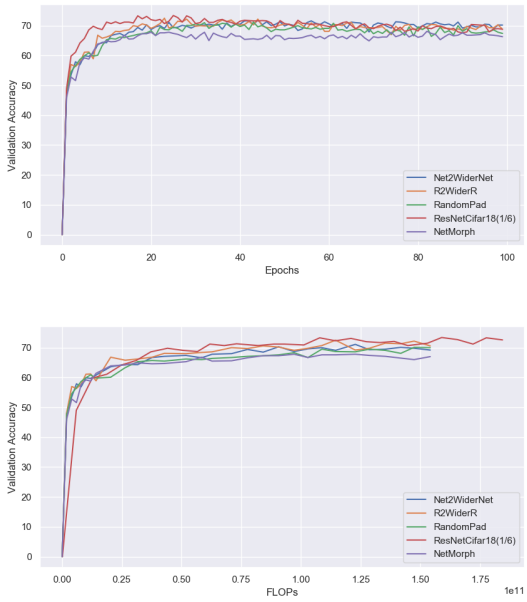

Figure 8. Validation curves comparing a network using each of `Net2DeeperNet`, `R2DeeperR`, to deepen at 25 epochs. We compare with baselines of random padding and training the ResNet "from scratch". Similar results for widening operations are omitted for space. *Top*: Plots with respect to the number of epochs passed (validation score with respect to sample complexity). *Bottom*: Plots with respect to the number of FLOPs used (validation score with respect to computational complexity, only 30 epochs shown).

The results obtained in this section indicate that FPTs can be used to train our ResNetCifar-18 networks with a lower computational cost and similar performance.

8. Conclusion

In this work we have introduced new FPTs: `R2WiderR` and `R2DeeperR`, that incorporate residual connections into the theory of FPTs. We derived how they preserve the function represented by the networks, despite altering the architecture. We then provided an in depth discussion on the differences, similarities and use-cases of `R2R`, `Net2Net` and `NetMorph`. Experiments conducted on Cifar-10 demonstrated that all three FPT schemes have similar performance, thereby allowing a wide range of neural network architectures to have FPTs applied to them. Finally, we also demonstrated that FPTs can be used to trade off between sample complexity and computational complexity of training.

References

Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. Efficient architecture search by network transformation. AAAI, 2018.

Chen, T., Goodfellow, I., and Shlens, J. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Jin, H., Song, Q., and Hu, X. Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282*, 2018.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Lin, M., Chen, Q., and Yan, S. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, pp. 12, 2017.

Wei, T., Wang, C., Rui, Y., and Chen, C. W. Network morphism. In *International Conference on Machine Learning*, pp. 564–572, 2016.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

A. Code

Code implemented in PyTorch (whichever version was current at the start of 2019): https://github.com/MWPainter/Deep-Neuroevolution-With-SharedWeights—Ensembling-A-Better-Solution.

As alluded to in the foreword, this code may have changes not represented in this paper, as I did some amount of work after the submission.

B. A full derivation of `R2R`

In this section we provide an extended descriptions of the steps we went through to derive `R2R` so that the padding produces a function preserving transform. In particular we provide a more principled approach by finding equations that must hold for the transformation/padding to be function preserving and then solving them. Solutions can then be found by observation, yielding the methods presented in section 5.

Additional note for arxiv version: I recall having some maths (that I never got around to writing up) that suggested that the symmetry introduced in the zero-initializations should be broken after some number of backups (if I remember correctly it did rely on either the input/output data not being symmetric in some sense that I cannot recall). This can be observed in Figure 4 where the symmetry is in fact broken. However, in retrospect, some of the new filters are still quite symmetrical, so adding a small amount of noise to any new parameters may be helpful for symmetry breaking.

B.1. `R2WiderR`

Recall equations (3) and (4) which we used to define the operation of a neural network. Suppose that we want to widen $x^{(i)}$ by padding with $x_L^{(i)}$ and $x_R^{(i)}$, of shapes $(E \times h^{(i)} \times w^{(i)})$, to produce a new volume $\bar{x}^{(i)}$. Let $W_L^{(i)}, W_R^{(i)}$ with shape $(E \times C^{(i-1)} \times k_h^{(i-1)} \times k_w^{(i-1)})$, $W_L^{(i+1)}, W_R^{(i+1)}$ with shape $(C^{(i+1)} \times E \times k_h^{(i)} \times k_w^{(i)})$, and $b_L^{(i)}, b_R^{(i)}$ with shape (E) be the new parameters introduced. Also, we introduce new parameters $\rho_L^{(i)}, \rho_R^{(i)}$ for additional channels required in the non-linearity $\sigma^{(i)}$, and let $\bar{\rho}^{(i)} = \rho^{(i)} \cup \rho_L^{(i)} \cup \rho_R^{(i)}$.

Given this padding, the i^{th} intermediate volume in the neural network is now $\bar{x}^{(i)}$ and has shape $((C^{(i)} + 2E) \times h^{(i)} \times w^{(i)})$. Let the new $(i+1)^{\text{th}}$ intermediate volume be denoted $\bar{x}^{(i+1)}$, which has the same shape as $x^{(i+1)}$ of $(C^{(i+1)} \times E \times k_h^{(i)} \times k_w^{(i)})$.

Equations (28) and (29) below recall the operation of the neural network before the padding, and equations (30) and (31) show the new operation after the padding.

$$x^{(i)} = \sigma^{(i)}_{\rho^{(i)}}\left(W^{(i)} * x^{(i-1)} + b^{(i)}\right) \tag{28}$$

$$x^{(i+1)} = \sigma^{(i+1)}_{\rho^{(i+1)}}\left(W^{(i+1)} * x^{(i)} + b^{(i)}\right) \tag{29}$$

$$\begin{bmatrix} x^{(i)} \\ x_L^{(i)} \\ x_R^{(i)} \end{bmatrix} = \bar{x}^{(i)} = \sigma^{(i)}_{\bar{\rho}^{(i)}}\left(\begin{bmatrix} W^{(i)} \\ W_L^{(i)} \\ W_R^{(i)} \end{bmatrix} * x^{(i-1)} + \begin{bmatrix} b^{(i)} \\ b_L^{(i)} \\ b_R^{(i)} \end{bmatrix}\right) \tag{30}$$

$$\bar{x}^{(i+1)} = \sigma^{(i)}_{\bar{\rho}^{(i+1)}}\left(\begin{bmatrix} W^{(i+1)} & W_L^{(i+1)} & W_R^{(i+1)} \end{bmatrix} * \bar{x}^{(i)} + b^{(i+1)}\right) \tag{31}$$

By expanding $\bar{x}^{(i)}$ in equation (31) we obtain

$$\bar{x}^{(i+1)} = \sigma^{(i)}_{\bar{\rho}^{(i+1)}}\left(\begin{bmatrix} W^{(i+1)} & W_L^{(i+1)} & W_R^{(i+1)} \end{bmatrix} * \begin{bmatrix} x^{(i)} \\ x_L^{(i)} \\ x_R^{(i)} \end{bmatrix} + b^{(i+1)}\right) \tag{32}$$

$$= \sigma^{(i)}_{\bar{\rho}^{(i+1)}}\left(W^{(i+1)} * x^{(i)} + W_L^{(i+1)} * x_L^{(i)} + W_R^{(i+1)} * x_R^{(i)} + b^{(i+1)}\right). \tag{33}$$

So it is sufficient that $W_L^{(i+1)} = -W_R^{(i+1)}$ and $x_L^{(i)} = x_R^{(i)}$ for $x^{(i+1)} = \bar{x}^{(i+1)}$. Then, recalling equation (5), our assumption

about the form of the non-lineariy, we have

$$\begin{bmatrix} x^{(i)} \\ x_L^{(i)} \\ x_R^{(i)} \end{bmatrix} = \sigma^{(i)}_{\bar{\rho}^{(i)}} \left(\begin{bmatrix} W^{(i)} \\ W_L^{(i)} \\ W_R^{(i)} \end{bmatrix} * x^{(i-1)} + \begin{bmatrix} b^{(i)} \\ b_L^{(i)} \\ b_R^{(i)} \end{bmatrix} \right) = \begin{bmatrix} \sigma^{(i)}_{\rho^{(i)}}(W^{(i)} * x^{(i-1)} + b^{(i)}) \\ \sigma^{(i)}_{\rho_L^{(i)}}(W_L^{(i)} * x^{(i-1)} + b_L^{(i)}) \\ \sigma^{(i)}_{\rho_R^{(i)}}(W_R^{(i)} * x^{(i-1)} + b_R^{(i)}) \end{bmatrix}. \tag{34}$$

And so if we set $W_L^{(i)} = W_R^{(i)}$, $b_L^{(i)} = b_R^{(i)}$ and $\rho_L^{(i)} = \rho_R^{(i)}$, then we have $x_L^{(i)} = x_R^{(i)}$.

In conclusion, if we have $\sigma^{(i)}$ that satisfies the assumption in equation (5), and set $W_L^{(i)} = W_R^{(i)}, b_L^{(i)} = b_R^{(i)}, \rho_L^{(i)} = \rho_R^{(i)}, W_L^{(i+1)} = -W_R^{(i+1)}$ (where we can choose $W_R^{(i)}, \rho_R^{(i)}, W_R^{(i+1)}, b_R^{(i)}$ arbitrarily), then the transform is function preserving. We also note that `R2WiderR` can be considered a non-trivial solution to the network morphism equations defined by Wei *et al.* (2016).

B.2. Residual connections

In section 5.2 we outlined how residual connections can be adapted for `R2WiderR`, however, we did not cover how to deal with either widening a volume multiple times or how to handle down-sampling. In this section we generalize the argument to handle both of these cases.

For convenience let $\theta^{(i)} = \{W^{(i)}, b^{(i)}, \rho^{(i)}\}$ and define $\mathcal{F}$ as a shorthand for equation (28) as follows:

$$x^{(i)} = \mathcal{F}^{(i)}_{\theta^{(i)}}\left(x^{(i-1)}\right) \stackrel{\text{def}}{=} \sigma^{(i)}_{\rho^{(i)}}\left(W^{(i)} * x^{(i-1)} + b^{(i)}\right). \tag{35}$$

Consider if `R2WiderR` is applied to an intermediate volume $x^{(\ell)}$, producing the two equal volumes $x_L^{(\ell)}$ and $x_R^{(\ell)}$. If $x^{(\ell)}$ was used as part of a residual connection, then we need to take into account how to handle $x_L^{(\ell)}$ and $x_R^{(\ell)}$ over that residual connection. We can re-write equation (34) as:

$$\bar{x}^{(\ell)} = \begin{bmatrix} x^{(\ell)} \\ x_L^{(\ell)} \\ x_R^{(\ell)} \end{bmatrix} = \begin{bmatrix} \mathcal{F}^{(\ell)}_{\theta^{(\ell)}}\left(x^{(\ell-1)}\right) \\ \mathcal{F}^{(\ell)}_{\theta_L^{(\ell)}}\left(x^{(\ell-1)}\right) \\ \mathcal{F}^{(\ell)}_{\theta_R^{(\ell)}}\left(x^{(\ell-1)}\right) \end{bmatrix}, \tag{36}$$

Let $r^{(i)}_{\phi^{(i)}}$, with parameters $\phi^{(i)}$, denote a function that *reshapes* a tensor to have shape $(C^{(i)} \times h^{(i)} \times w^{(i)})$, where the input tensor shape can depend on $\phi^{(i)}$. The purpose of $r^{(i)}$ is a parameterized function that is used over the residual connection. To give some examples of what $r^{(i)}$ could be, if we have $(C^{(i)}, h^{(i)}, w^{(i)}) = (C^{(\ell)}, h^{(\ell)}, w^{(\ell)})$ then $r^{(i)}$ would typically be an identity function, if $C^{(\ell)} < C^{(\ell)}$ the $r^{(i)}$ may perform a zero padding and if either $C^{(\ell)} > C^{(i)}$, $h^{(\ell)} > h^{(i)}$ or $w^{(\ell)} > w^{(i)}$ then $r^{(i)}$ must be some function that downsamples, typically a convolution (and why parameters are needed in $r^{(i)}$ for this to be a general argument).

Now, consider a residual connection from the ℓ^{th} intermediate volume to the i^{th} intermediate volume, formally written as:

$$x^{(i)} = \mathcal{F}^{(i)}_{\theta^{(i)}}\left(x^{(i-1)}\right) + r^{(i)}_{\phi^{(i)}}\left(x^{(\ell)}\right). \tag{37}$$

Let $\phi_L^{(i)}$ be some arbitrary parameters, such that $r^{(i)}_{\phi_L^{(i)}}$ takes tensors with the same shape as $x_L^{(\ell)}$. Define $\phi_R^{(i)}$ similarly for $x_R^{(i)}$. We can now define $\bar{r}^{(i)}$ to be used as a new residual connection from $\bar{x}^{(\ell)}$:

$$\bar{r}^{(i)}\left(\bar{x}^{(\ell)}\right) = r^{(i)}_{\phi^{(i)}}\left(x^{(\ell)}\right) + r^{(i)}_{\phi_L^{(i)}}\left(x_L^{(\ell)}\right) - r^{(i)}_{\phi_R^{(i)}}\left(x_R^{(\ell)}\right). \tag{38}$$

As in `R2WiderR` we have $\theta_L^{(i)} = \theta_R^{(i)}$ in equation (36), we have $x_L^{(\ell)} = x_R^{(\ell)}$, and so by setting $\phi_L^{(i)} = \phi_R^{(i)}$ initially, we have

$\bar{r}^{(i)}(\bar{x}^{(\ell)}) = r^{(i)}_{\phi^{(i)}}(x^{(\ell)})$. Which gives

$$x^{(i)} = \mathcal{F}^{(i)}_{\theta^{(i)}}\left(x^{(i-1)}\right) + r^{(i)}\left(x^{(\ell)}\right) \tag{39}$$

$$= \mathcal{F}^{(i)}_{\theta^{(i)}}\left(x^{(i-1)}\right) + \bar{r}^{(i)}\left(\bar{x}^{(\ell)}\right). \tag{40}$$

and shows that $\bar{r}^{(i)}$ can be used in place of the

Finally, we return to what we added in this section over section 5.2. Our more general argument included non-trivial residual connections, that can include arbitrary parametereized functions and handle down-sampling over the connection. Moreover, because the residual connection includes an arbitrary function, it allows for a recursive application of these rules, and therefore can handle multiple applications of `R2WiderR`over a residual connection.

B.3. Zero initializations

Here we provide a full derivation of our zero initializations that we use in `R2DeeperR`. Let $g_{\theta^{(1:n)}}$ be an arbitrary neural network, with parameters $\theta^{(1:n)}$ and let $x^{(n)} = g_{\theta^{(1:n)}}(x^{(0)})$ be the output of this network. We will add two additional laters on the output, $x^{(o_1)}$ and $x^{(o_2)}$, with shapes $(C^{(o_1)} \times h^{(o_1)} \times w^{(o_1)})$ and $(C^{(o_2)} \times h^{(o_2)} \times w^{(o_2)})$ respectively.

Assuming that $C^{(o_1)}$ and $C^{(o_2)}$ are even, let $W^{(o_1)}_L, W^{(o_1)}_R$ have shape $(C^{(o_1)}/2 \times C^{(n)} \times h^{(n)} \times w^{(n)})$ and let $W^{(o_2)}_L, W^{(o_2)}_R$ have shape $(C^{(o_2)} \times C^{(o_1)}/2 \times h^{(n)} \times w^{(n)})$. Also let $b^{(o_1)}_L, b^{(o_2)}_R$ have shape $(C^{(o_1)}/2)$ and let $b^{(o_2)}$ have shape $(C^{(o_2)})$. Additionally define $\rho^{(o_1)} = \{\rho^{(o_1)}_L, \rho^{(o_1)}_R\}$ and $\rho^{(o_2)} = \{\rho^{(o_2)}_L, \rho^{(o_2)}_R\}$. Now, define new parameters for the network:

$$W^{(o_1)} = \begin{bmatrix} W^{(o_1)}_L \\ W^{(o_1)}_R \end{bmatrix}, \tag{41}$$

$$W^{(o_2)} = \begin{bmatrix} W^{(o_2)}_L & W^{(o_2)}_R \end{bmatrix}, \tag{42}$$

$$b^{(o_1)} = \begin{bmatrix} b^{(o_1)}_L \\ b^{(o_1)}_R \end{bmatrix}. \tag{43}$$

The computation for new layers o_1 and o_2 is then:

$$x^{(o_1)} = \sigma^{(o_1)}_{\rho^{(o_1)}}\left(W^{(o_1)} * x^{(n)} + b^{(o_1)}\right), \tag{44}$$

$$x^{(o_2)} = \sigma^{(o_2)}_{\rho^{(o_2)}}\left(W^{(o_2)} * x^{(o_1)} + b^{(o_2)}\right). \tag{45}$$

Expanding the definitions of the new params and multiplying out we get:

$$x^{(o_2)} = \sigma^{(o_2)}_{\rho^{(o_2)}}\left(W^{(o_1)}_L * \sigma^{(o_1)}_{\rho^{(o_1)}_L}\left(W^{(o_1)}_L * x^{(n)} + b^{(o_1)}_L\right) + W^{(o_2)}_R * \sigma^{(n+1)}_{\rho^{(n+1)}_R}\left(W^{(n+1)}_R * x^{(n)} + + b^{(o_1)}_R\right) + b^{(o_2)}\right). \tag{46}$$

If we set $W^{(o_1)}_L = W^{(o_1)}_R, b^{(o_1)}_L = b^{(o_1)}_R, W^{(o_2)}_L = -W^{(o_2)}_R$ and $\rho^{(o_1)}_L = \rho^{(o_1)}_R$, where $W^{(o_1)}_R, b^{(o_1)}_R, W^{(o_2)}_R$ and $\rho^{(o_1)}_R$ can be arbitrary, then equation (46) above reduces to

$$x^{(o_2)} = \sigma^{(o_2)}_{\rho^{(o_2)}}\left(b^{(o_2)}\right). \tag{47}$$

As $x^{(o_2)}$ will be the output volume, if $\sigma^{(o_2)}_{\rho^{(o_2)}}(b^{(o_2)}) = 0$, then we have found a zero initialization. In particular, we use ReLU activations in our experiments, which satisfy $\sigma^{(o_2)}_{\rho^{(o_2)}}(0) = 0$, in which case we set $b^{(o_2)} = 0$ as well.

We note that there is a restriction on the output non-linearity, which we requires that there must exist some value x such that $\sigma^{(o_2)}_{\rho^{(o_2)}}(x) = 0$. If our output non-linearity does not have this property of having at least one zero in its range, then we cannot provide a zero initialization. For example, this means that we cannot use a sigmoid activation function on the output of our zero initialized network.

Table 3. Definition of architectures used for the visualizations in figure 4.

LAYER NAME	OUTPUT SIZE	SMALLCONV	SMALLCONV(WIDENED)
CONV1	32×32	7×7, 16	7×7, 32
POOL1	16×16	2×2 MAX POOL, STRIDE 2	
FC1	1×1	150-D FC	
	1×1	10-D FC	

Table 4. Definition of architectures used for the tests on Cifar-10. We use the same notation as He *et al.* (2016), where each block is defined in square brackets, each layer is specified by the kernel size and number of filters, and there is a residual connection over every block. We use the floor function, $\lfloor \cdot \rfloor$, for the number of channels, to allow for non-integer r. All 3×3 convolutions use a padding of 1, and 7×7 convolutions use a padding of 3. All convolutions use a stride of one, except from CONV1, CONV2_1 and CONV3_1 which use a stride of two.

LAYER NAME	OUTPUT SIZE	RESNETCIFAR-10(r)	RESNETCIFAR-18(r)
CONV1	16×16	7×7, $64r$, STRIDE 2	
CONV2_X	8×8	3×3 MAX POOL, STRIDE 2	
		$\begin{bmatrix} 3 \times 3, & \lfloor 64r \rfloor \\ 3 \times 3, & \lfloor 64r \rfloor \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, & \lfloor 64r \rfloor \\ 3 \times 3, & \lfloor 64r \rfloor \end{bmatrix} \times 4$
CONV3_X	4×4	$\begin{bmatrix} 3 \times 3, & \lfloor 128r \rfloor \\ 3 \times 3, & \lfloor 128r \rfloor \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, & \lfloor 128r \rfloor \\ 3 \times 3, & \lfloor 128r \rfloor \end{bmatrix} \times 4$
	1×1	AVG POOL, 10-D FC, SOFTMAX	

B.4. `R2DeeperR`

Given the derivation of zero initializations we just covered, the discussion in section 5.4 is a sufficient derivation, as given zero initializations `R2DeeperR` follows quickly. We still keep this section for completeness of appendix B.

C. Architectures and hyper-parameters

To be as transparent as possible with how we ran our experiments, in tables 3 and 4 we provide descriptions of the ResNet architectures we used, using the same notation as (He et al., 2016) in the original ResNet paper. Moreover, for every test run we provide all training parameters in table 5. Apart from the *teacher networks*, which were heavily regularize to avoid the problem described in section 7.1, hyper-paramters were chosen using a hyper-parameter search for each network.

For all of our tests, inputs to the neural networks were color normalized. All of the training and validation accuracies computed for Cifar-10 were using a single view and the entire 32×32 image as input, with no cropping. We performed no other augmentation.

In all tests, a widen operation consists of multiplying the number of filters by a factor of 1.5 , therefore, when a widen operation is applied to ResNetCifar($\frac{1}{8}$) it is transformed to a ResNetCifar($\frac{3}{16}$) network. We only ever applied a deepen operation to ResNetCifar10(r), transforming it into a corresponding ResNetCifar18(r).

All remaining training details are specified as hyper-parameters in table 5.

Table 5. Here we specify all hyper-parameters used in our tests. In the experiment name we use the following naming convention: [Test]-[NetworkName]. For the network convergence tests, we abbreviate the prefix to NCT, and for the faster training tests, we abbreviate the prefix to FTT. The prefixes that we use for the visualizations, overfitting (figure 7) and bad initializations (figure 9) are abbreviated to VIZ, OF and BI.

EXPERIMENT NAME	(STARTING) ARCHITECTURE	BATCH SIZE	OPTIMIZER	WEIGHT DECAY
VIZ-R2R	SMALLCONV	128	ADAM	10^{-3}
VIZ-NET2NET	SMALLCONV	128	ADAM	10^{-3}
VIZ-NETMORPG	SMALLCONV	128	ADAM	10^{-3}
OF-TEACHER	RESNETCIFAR18($\frac{1}{8}$)	128	ADAM	0
OF-STUDENT	OF-TEACHER	128	ADAM	3×10^{-3}
OF-RAND	RESNETCIFAR18($\frac{3}{16}$)	128	ADAM	3×10^{-4}
BI-HE	RESNETCIFAR18($\frac{1}{8}$)	128	ADAM	10^{-3}
BI-STD	RESNETCIFAR18($\frac{1}{8}$)	128	ADAM	10^{-3}
NCT-WIDENTEACHER	RESNETCIFAR18($\frac{1}{8}$)	128	ADAM	5×10^{-3}
NCT-R2WIDERR	NCT-WIDENTEACHER	128	ADAM	10^{-2}
NCT-NETMORPH	NCT-WIDENTEACHER	128	ADAM	10^{-2}
NCT-RANDPADWIDEN	NCT-WIDENTEACHER	128	ADAM	10^{-2}
NCT-WIDENTEACHER(NORESIDUAL)	RESNETCIFAR18($\frac{1}{8}$)	128	ADAM	5×10^{-3}
NCT-NET2WIDERNET	NCT-WIDENTEACHER(NORESIDUAL)	128	ADAM	10^{-2}
NCT-DEEPENTEACHER	RESNETCIFAR18($\frac{1}{8}$)	128	ADAM	5×10^{-3}
NCT-R2DEEPERR	NCT-DEEPERTEACHER	128	ADAM	10^{-2}
NCT-RANDPADDEEPER	NCT-DEEPERTEACHER	128	ADAM	10^{-2}
NCT-DEEPERTEACHER(NORESIDUAL)	RESNETCIFAR18($\frac{1}{8}$)	128	ADAM	5×10^{-3}
NCT-NET2DEEPERNET	NCT-DEEPERTEACHER(NORESIDUAL)	128	ADAM	10^{-2}
FFT-R2WIDERR	RESNETCIFAR18($\frac{1}{8}$)	128	ADAM	10^{-2}
FFT-NETMORPH	RESNETCIFAR18($\frac{1}{8}$)	128	ADAM	10^{-2}
FFT-RANDPADWIDEN	RESNETCIFAR18($\frac{1}{8}$)	128	ADAM	10^{-2}
FFT-NET2WIDERNET	RESNETCIFAR18($\frac{1}{8}$)	128	ADAM	10^{-2}
FFT-R2DEEPERR	RESNETCIFAR18($\frac{1}{8}$)	128	ADAM	10^{-2}
FFT-RANDPADDEEPER	RESNETCIFAR18($\frac{1}{8}$)	128	ADAM	10^{-2}
FFT-NET2DEEPERNET	RESNETCIFAR18($\frac{1}{8}$)	128	ADAM	10^{-2}

EXPERIMENT NAME	TRANSFORMATIONS	LEARNING RATE	LR DROPS	EPOCHS TRAINED
VIZ-R2R	[WIDEN@150 EPOCHS]	10^{-3}	N/A	600
VIZ-NET2NET	[WIDEN@150 EPOCHS]	10^{-3}	N/A	600
VIZ-NETMORPH	[WIDEN@150 EPOCHS]	10^{-3}	N/A	600
OF-TEACHER	N/A	10^{-3}	N/A	250
OF-STUDENT	[WIDEN@0 EPOCHS]	10^{-3}	[$\frac{1}{5}$@0 EPOCHS]	250
OF-RAND	N/A	10^{-3}		250
BI-HE	[WIDEN@20 EPOCHS]	10^{-3}	[$\frac{1}{5}$@20 EPOCHS]	120
BI-STD	[WIDEN@20 EPOCHS]	10^{-3}	[$\frac{1}{5}$@20 EPOCHS]	120
NCT-WIDENTEACHER	N/A	3×10^{-3}	N/A	250
NCT-R2WIDERR	[WIDEN@0 EPOCHS]	3×10^{-3}	[$\frac{1}{5}$@0 EPOCHS]	250
NCT-NETMORPH	[WIDEN@0 EPOCHS]	3×10^{-3}	[$\frac{1}{5}$@0 EPOCHS]	250
NCT-RANDPADWIDEN	[WIDEN@0 EPOCHS]	3×10^{-3}	[$\frac{1}{5}$@0 EPOCHS]	250
NCT-WIDENTEACHER(NORESIDUAL)	N/A	3×10^{-3}	N/A	250
NCT-NET2WIDERNET	[WIDEN@0 EPOCHS]	3×10^{-3}	[$\frac{1}{5}$@0 EPOCHS]	250
NCT-DEEPERTEACHER	N/A	3×10^{-3}	N/A	250
NCT-R2DEEPERR	[DEEPEN@0 EPOCHS]	3×10^{-3}	[$\frac{1}{5}$@0 EPOCHS]	250
NCT-RANDPADDEEPER	[DEEPEN@0 EPOCHS]	3×10^{-3}	[$\frac{1}{5}$@0 EPOCHS]	250
NCT-DEEPERTEACHER(NORESIDUAL)	N/A	3×10^{-3}	N/A	250
NCT-NET2DEEPERNET	[DEEPEN@0 EPOCHS]	3×10^{-3}	[$\frac{1}{5}$@0 EPOCHS]	250
FFT-R2WIDERR	[WIDEN@25 EPOCHS]	3×10^{-3}	[$\frac{1}{5}$@25 EPOCHS]	250
FFT-NETMORPH	[WIDEN@25 EPOCHS]	3×10^{-3}	[$\frac{1}{5}$@25 EPOCHS]	250
FFT-RANDPADWIDEN	[WIDEN@25 EPOCHS]	3×10^{-3}	[$\frac{1}{5}$@25 EPOCHS]	250
FFT-NET2WIDERNET	[WIDEN@25 EPOCHS]	3×10^{-3}	[$\frac{1}{5}$@25 EPOCHS]	250
FFT-R2DEEPERR	[DEEPEN@25 EPOCHS]	3×10^{-3}	[$\frac{1}{5}$@25 EPOCHS]	250
FFT-RANDPADDEEPER	[DEEPEN@25 EPOCHS]	3×10^{-3}	[$\frac{1}{5}$@25 EPOCHS]	250
FFT-NET2DEEPERNET	[DEEPEN@25 EPOCHS]	3×10^{-3}	[$\frac{1}{5}$@25 EPOCHS]	250

D. Weight initialization in `R2R`

In figure 9 we compare what happens if we use a more standard He initialization for new parameters in the network, compared to matching the standard deviations of local weights as described in section 7.2.

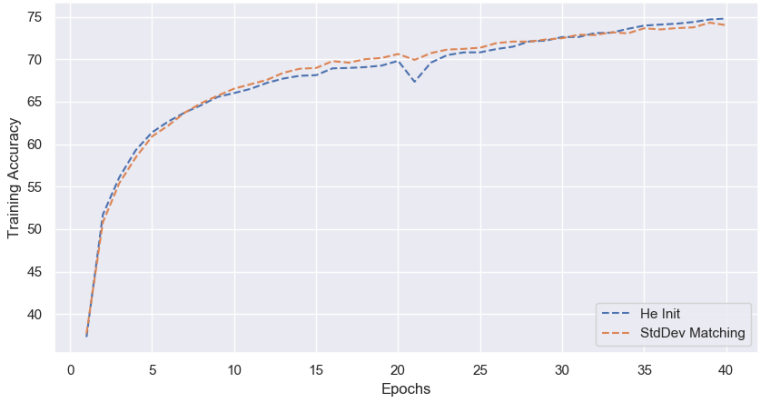

Figure 9. Two networks widened at 20 epochs. Weights using He initialization are much larger than existing parameters, and causes some instability if used. We can observe similar effects with deepening.

E. Future work

Although we have discussed `R2R`, `Net2Net` and Network Morphism in depth in this work, there are still some properties that we would have liked to consider. For example, we typically need to change the learning rate and weight decay, and be careful about the initialization scale of new parameters. We would have liked to consider the interplay between the learning rate, weight decay, weight initialization and optimizer, and provided (theoretical) guidance on how to manipulate these hyper-parameters best when performing an FPT.

Additionally, we think that the freedom in the initializations could allow more complex iniilization schemes to be considered. For example, a hybrid system between `R2WiderR` and `Net2WiderNet` could be considered, where new channels in `R2WiderR`are initialized as copies of other channels. Alternatively, learning good initializations has been considered in the meta-learning problem (Finn et al., 2017), and so maybe 'good' initializations could be learned in some problems.

Finally, Network Morphism also considers kernel size morphing and subnet morphing, and we have not consdered these transformations in this work. We note that our zero initialization from section 5.3 could be used as an alternative subnet morphing scheme to that considered by (Wei et al., 2016).

References

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Wei, T., Wang, C., Rui, Y., and Chen, C. W. Network morphism. In *International Conference on Machine Learning*, pp. 564–572, 2016.