DECOUPLING THE LAYERS IN RESIDUAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose the Warped Residual Network using a parallelizable warp operator for forward and backward propagation to distant layers that trains faster than the original residual neural network. We apply a perturbation theory on residual networks and decouple the interactions between residual units. The resulting warp operator is a first order approximation of the output over multiple layers. The first order perturbation theory exhibits properties such as binomial path lengths and exponential gradient scaling found experimentally by Veit et al. (2016). We show that the Warped Residual Networks learn invariant models by breaking the redundancy in the weights caused by local symmetries in the input. The proposed network can outperform or achieve comparable predictive performance to the original residual network with the same number of parameters, while achieving a significant speedup on the total training time. We find that as the layers get wider, the speedup in training time (44% with the widest architecture) is closer to the optimal speedup of 50% for skipping over one residual unit. Our architecture opens up a new research direction for methods to train a K layer ResNets in $O(1)$ time as opposed to $O(K)$ time with forward and backward propagation.

1 INTRODUCTION

Deep Convolution Neural Networks (CNN) have been used in image recognition tasks with great success. Since AlexNet (Krizhevsky et al., 2012), many other neural architectures have been proposed to achieve start-of-the-art results at the time. Some of the notable architectures include, VGG (Simonyan & Zisserman, 2015), Inception (Szegedy et al., 2015) and Residual networks (ResNet)(He et al., 2015).

Training a deep neural network is not an easy task. As the gradient at each layer is dependent upon those in higher layers multiplicatively, the gradients in earlier layers can vanish or explode, ceasing the training process. The gradient vanishing problem is significant for neuron activation functions such as the sigmoid, where the gradient approaches zero exponentially away from the origin on both sides. The standard approach to combat vanishing gradient is to apply Batch Normalization (BN) (Ioffe & Szegedy, 2015) followed by the Rectified Linear Unit (ReLU) (Hahnloser et al., 2000) activation. More recently, skip connections (Srivastava et al., 2015) have been proposed to allow previous layers propagate relatively unchanged. Using this methodology the authors in (Srivastava et al., 2015) were able to train extremely deep networks (hundreds of layers) and about one thousand layers were trained in residual networks (He et al., 2015).

As the number of layers grows large, so does the training time. To evaluate the neural network's output, one needs to propagate the input of the network layer by layer in a procedure known as forward propagation. Likewise, during training, one needs to propagate the gradient of the loss function from the end of the network to update the model parameters, or weights, in each layer of the network using gradient descent. The complexity of forward and propagation is $O(K)$, where K is the number of layers in the network. To speed up the process, one may ask if there exist a shallower network that accurately approximates a deep network so that training time is reduced. In this work we show that there indeed exists a neural network architecture that permits such an approximation, the ResNet.

Residual networks typically consist of a long chain of residual units. Recent investigations suggest that ResNets behave as an ensemble of shallow networks (Veit et al., 2016). Empirical evidence supporting this claim includes one that shows randomly deactivating residual units during training (similar to drop-out (Srivastava et al., 2014)) appears to improve performance (Huang et al., 2016). The

results imply that the output of a residual unit is just a small perturbation of the input. In this work, we make an approximation of the ResNet by using a series expansion in the small perturbation. We find that merely the first term in the series expansion is sufficient to explain the binomial distribution of path lengths and exponential gradient scaling experimentally observed by Veit et al. (2016). The approximation allows us to effectively estimate the output of subsequent layers using just the input of the first layer and obtain a modified forward propagation rule. We call the corresponding operator the warp operator. The backpropagation rule is obtained by differentiating the warp operator. We implemented a network using the warp operator and found that our network trains faster on image classification tasks with predictive accuracies comparable to those of the original ResNet.

A lot can be learned about deep neural networks when the layers are decoupled. When the data exhibit rotational symmetry, same copies of filters that only differ by rotations are learned (Dieleman et al., 2016). This creates a redundancy that limits the possible generalizations that can be learned in the network. We generalize this to the case of arbitrary symmetries, where symmetries in the input give rise to a set of weights under the same symmetry transformations. This kind of symmetries are local, in the sense that the transformation on the input induces a change only on the weights acting on the input but not any other weights. We find that the warp generator breaks the symmetry of the weights. The series approximation automatically generates a convolution layer with flipped filters from another layer so the rotational symmetry is broken, resulting in a more rotation-invariant model.

Symmetries in neural networks have been discussed in the literature. Many of which breaks symmetries explicitly by adding extra layers. For example, Gens & Domingos (2014) employed Lie groups to design a network invariant to the affine group. Cohan & Welling (2016) added G-convolution layers in their network. Badrinarayanan et al. (2016) considered the shape of the loss manifold to design an optimization scheme to handle symmetries, and Orhan & Pitkow (2017) proposed skip connections breaking certain symmetries. Little, if not all, has discussed about the relationship between learning and symmetry breaking in the weights. Through the Warped Residual Network, this paper provides insights to learning via symmetry breaking. Further, this is the first successful attempt to decouple layers in a deep neural network using a perturbative theory leading to a novel parallelizable network architecture.

Our contributions in this work include

- We analytically investigate the properties of ResNets. In particular, we show that the first order term in the Taylor series expansion of the layer output across K residual units has a binomial number of terms, which are interpreted as the number of paths in Veit et al. (2016), and that for ReLU activations the second and higher order terms in the Taylor series vanish with probability one.
- Based on the above-mentioned analysis, we propose a novel architecture, WarpNet, which employs a warp operator as a parallelizable propagation rule across multiple layers at a time. The WarpNet is an approximation to a ResNet with the same number of weights.
- We investigate WarpNet and provide insights about how breaking the weight symmetry, or redundancies in the weights, assists generalization.
- We show that WarpNet achieves comparable predictive performance to the original ResNet while achieving about 40% speedup.

The organization of this paper is as follow. In Section 2 we analyze the properties of ResNet and show that the binomial path length arises from a Taylor expansion to first order. In Section 3 we describe Warped Residual Networks. In Section 3.2, we study symmetries in the network. In Section 4 we show that WarpNet can outperform the original ResNet.

2 Properties of ResNets

In this section we show that recent numerical results (Veit et al., 2016) is explained when the perturbation theory is applied to ResNets. Consider the input $\mathbf{x}_i$ of the i-th residual unit and its output $\mathbf{x}_{i+1}$, where

$$\mathbf{x}_{i+1} = \mathbf{h}_i(\mathbf{x}_i) + \mathbf{F}_i(\mathbf{x}_i, \mathbf{W}_i). \tag{1}$$

Typically, $\mathbf{h}(\mathbf{x}_i)$ is taken to be an identity mapping, $\mathbf{h}_i(\mathbf{x}_i) = \mathbf{x}_i$. When the feature maps are down sampled, $\mathbf{h}$ is usually taken to be an 1×1 convolution layer with a stride of 2. The functions $\mathbf{F}_i$ is a combination of convolution, normalization and non-linearity layers, so that $\mathbf{W}_i$ collectively represents the weights of all layers in $\mathbf{F}_i$.

We now show that $\mathbf{h}$ must be close to the identity, up to $O(\epsilon) \ll 1$, when the output of a residual unit is similar to the input and the weights are trained $\mathbf{W}_i = \mathbf{W}_i^*$. Suppose that $\mathbf{x}_{i+1} \simeq \mathbf{x}_i$, motivated by the numerical results (Veit et al., 2016) that removing a single residual unit does not affect the performance drastically. Then $\mathbf{x}_i = \mathbf{h}_i(\mathbf{x}_i) + \mathbf{F}(\mathbf{x}_i, \mathbf{W}_i^*) + O(\epsilon)$. Since this holds true for arbitrary input and $\mathbf{h}$ and $\mathbf{F}$ are assumed to be linearly independent, we must have $\mathbf{h}_i(\mathbf{x}_i) = \mathbf{x_i} + O(\epsilon)$ and $\mathbf{F}_i(\mathbf{x}_i, \mathbf{W}_i^*) = O(\epsilon) \ll 1$. An equivalent solution exists where $\mathbf{F}_i$ is the identity and $\mathbf{h}_i \sim O(\epsilon)$ which we ignore. Conversely, if the identity mapping is assumed as in a typical residual unit, then $\mathbf{F}_i(\mathbf{x}_i, \mathbf{W}_i^*) \sim O(\epsilon)$ from empirical observations such as those by Greff et al. (2016).[1]

Perturbative feature map flow First, we show that the interpretation of ResNets as an ensemble of subnetworks is accurate up to the first order in $\mathbf{F}$ with identity mapping. The identity mapping causes $\mathbf{F}$ to be small compared to the input $\mathbf{x}$, one can approximate the output of a chain of residual units by a series expansion. For instance, the output of two residual units $\mathbf{x}_3$ is related to the input of the first unit by the following (we call the process where $\mathbf{x}_k$ is expressed in terms of $\mathbf{x}_{k-1}$ an iteration. The following equations show two iterations).

$$\begin{aligned}
\mathbf{x}_3 &= \mathbf{x}_2 + \mathbf{F}_2(\mathbf{x}_2, \mathbf{W}_2^*) \\
&= \mathbf{x}_1 + \mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1^*) + \mathbf{F}_2(\mathbf{x}_1 + \mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1^*), \mathbf{W}_2^*) \\
&= \mathbf{x}_1 + \mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1^*) + \mathbf{F}_2(\mathbf{x}_1, \mathbf{W}_2^*) + \mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1^*)\mathbf{F}_2'(\mathbf{x}_1, \mathbf{W}_2^*) + O(\epsilon^2), \qquad (2)
\end{aligned}$$

where $\mathbf{F}_2'(\mathbf{x}_1, \mathbf{W}_2^*)$ denotes the partial derivative of $\mathbf{F}_2$ with respect to $\mathbf{x}_1$. A Taylor series expansion in powers of $\mathbf{F}_1$ was performed on $\mathbf{F}_2$ in the second line above.[2] The $O(\epsilon^2)$ term arises from the Taylor series expansion. Equation (2) can be interpreted as an ensemble sum of subnetworks.We will show below that the $O(\epsilon^2)$ term can be ignored because the first order term in the Taylor series is a good approximation. It can be shown that across K residual units,

$$\mathbf{x}_{K+1} = \mathbf{x}_1 + \sum_{k=1}^{K} \binom{K}{k} (\mathbf{F}'(\mathbf{x}_1, \mathbf{W}^*))^{k-1} \mathbf{F}(\mathbf{x}_1, \mathbf{W}^*) + O(\epsilon^2). \qquad (3)$$

We have made the last equation simpler, solely for simplicity, by setting $\mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1^*) = \mathbf{F}_2(\mathbf{x}_1, \mathbf{W}_2^*) = \mathbf{F}$ and $\mathbf{W}_1^* = \mathbf{W}_2^* = \mathbf{W}^*$. Our calculations do not use this simplification. Henceforth when an expression $\mathbf{F}(\mathbf{x}, \mathbf{W}^*)$ occurs, it is understood that all weights are set equal $\mathbf{W}_i^* = \mathbf{W}^*$. After the ResNet is trained, each residual unit perturbs the input feature maps little by little as they pass through the ResNet in such a way that the final feature maps minimize training loss. This is precisely the behavior illustrated in (Greff et al., 2016). Therefore, a the first order term in the perturbation series is sufficient to give a good approximation across layers in ResNet.

Exponential gradient scaling Similarly, one observes that the gradient is the sum from all subnetwork contributions, including the identity network. The magnitudes of subnetwork gradients for an 110 layer ResNet have been measured by (Veit et al., 2016). If one takes $\mathbf{F}$ to have ReLU non-linearity with input $\mathbf{x}_R$ having a continuous probability density function $p(\mathbf{x}_R)$, then $\mathbf{F}''(\mathbf{x}, \mathbf{W}^*) = 0$ with probability one as the probability of $\mathbf{x}_R = 0$ is zero. The non-trivial gradient can be expressed as $\binom{K}{k}(\mathbf{F}'(\mathbf{x}, \mathbf{W}^*))^k$. The average gradient norm for each subnetwork of depth k as estimated by Veit et al. (2016) is given by $||\mathbf{F}'(\mathbf{x}, \mathbf{W}^*)||^k$. This proves the numerical results that the gradient norm decreases exponentially with subnetwork depth as reported in (Veit et al., 2016).

Higher order perturbations In order to prove the validity of a first order perturbation theory we need to show that the second and higher order terms are negligible. This is a good approximation, as the expected gradient norm across one residual unit was shown to be of order 10^{-6} (Veit et al., 2016). In fact, for ReLU non-linearity, the second order perturbations vanish with probability 1.

[1] See figure 3 in (Greff et al., 2016), where the input converges slowly to the optimum as it passes through each residual unit. This suggests that $\mathbf{F}$ is small.

[2] The Taylor expansion for multivariate functions is $f(\mathbf{x} + \mathbf{a}) = f(\mathbf{x}) + \mathbf{a} \cdot \nabla_{\mathbf{x}} f(\mathbf{x}) + \cdots$

The second order perturbation terms all contain the Hessian $\mathbf{F}''(\mathbf{x})$. But after the network is trained, the only non-linear function in $\mathbf{F}$, ReLU, is only non-linear at the origin[3]. Suppose the input to the ReLU non-linearity $\mathbf{x}_R$ has a continuous probability density function in each of its elements $(x_R)_i$, then the probability of $(x_R)_i = 0$ is zero for all element i. Therefore all second order perturbations vanish with probability 1. The same argument applies to higher orders. The first order perturbation theory is *almost surely* an exact theory for ResNets with ReLU non-linearity.

Gradient decoupling Suppose $\mathbf{F}_{1,2}$ are two non-identity mappings in two residual units and let $\mathbf{W}^{(1)}$ be the weights of any layer in the first residual unit. Let $\mathbf{x}_1$ be the input of this two residual unit chain and $\mathbf{x}_3$ be its output. Consider the partial derivative with respect to the weights in the first unit from Equation (2)

$$\frac{\partial \mathbf{x}_3}{\partial \mathbf{W}_1} = \frac{\partial \mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1)}{\partial \mathbf{W}_1} + \frac{\partial \mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1)}{\partial \mathbf{W}_1} \mathbf{F}_2'(\mathbf{x}_1, \mathbf{W}_2^*),$$

evaluated at $\mathbf{W}_1 = \mathbf{W}_1^*$. Let $\partial_{\mathbf{W}^{(1)}}$ denotes the partial derivatives above. Note that the factor of 2 is absent in the first term as $\partial_{\mathbf{W}_1} \mathbf{F}_2 = 0$. Suppose the expected value of $||\mathbf{F}_2'(\mathbf{x}_1)|| \sim O(\epsilon)$, with $\epsilon \simeq 10^{-6}$ which is suggested by the results in Veit et al. (2016). Then at the critical point, the above equation must vanish, which implies $\partial_{\mathbf{W}_1} \mathbf{F}_1 \simeq 0$. This is an important result, which suggests that to the first non-vanishing order in gradient norm, the local minima is *independent* of parameters in subsequent residual units.

The case for K residual units is similar but the coupling between residual units is enhanced by the sum of subnetworks. The weight $\mathbf{W}_j$ appears only after the $(K-j+1)$-th iteration in Taylor series expansion. Differentiating Equation (3) with respect to $\mathbf{W}_j$, then setting all weights to equal as before, the gradient with respect to the parameters $\mathbf{W}^{(j)}$ in the j-th residual unit $\mathbf{F}_j$ is

$$\frac{\partial \mathbf{x}_{K+1}}{\partial \mathbf{W}_j} = \frac{\partial \mathbf{F}_j(\mathbf{x}_j, \mathbf{W}_j)}{\partial \mathbf{W}_j} + \sum_{k=2}^{K-j+1} \binom{K-j}{k-1} \left[(\mathbf{F}'(\mathbf{x}_j, \mathbf{W}))^{k-1} \frac{\partial \mathbf{F}_j(\mathbf{x}_j, \mathbf{W}_j)}{\partial \mathbf{W}_j} \right].$$

The new binomial coefficient comes from the fact that the derivative only picks out subnetworks containing $\mathbf{F}_j$. This is equivalent to picking one black ball from an urn of $K-j+1$ balls containing one black ball and $K-j$ white balls with k samples without replacement. The probability is governed by the hypergeometric distribution. Note that the derivative with respect to $\mathbf{W}_j$ only acts on $\mathbf{F}$ but not on its derivative $\mathbf{F}'$. This is because the j-th unit is the first unit after $K-j+1$ iterations, and that the first unit is $\mathbf{F}$. Also note that the above expression depends upon weights in earlier units through the forward propagated input $\mathbf{x}_j$.

All aforementioned properties apply only after the ResNets are trained. However, if an approximation in the network is made, it would still give similar results after training. We show in the following sections that not only our network trains, its performance can surpass the original ResNet. The reason is that the approximation breaks the symmetries in the weights, such as those from rotations, in the network.

3 Warped Residual Network

The Warped Residual Network (WarpNet) is an approximation to the residual network, where K consecutive residual units are compressed into one warp layer. The computation in a warp layer is different from that in a conventional neural network. It uses a warp operator to compute the output (i.e., x_{K+1}) of the layer directly from the input (i.e., x_1), as shown in Equation (3). The number of weights in a warped layer is the same of the one in the original residual network for K consecutive residual units. For instance, the weights W_1, W_2 and W_K are present in a warped layer. But these weights can be used and updated in parallel due to the use of the warp operator. Below we first describe the forward and backward propagation rules used in warped residual network, and then discuss the symmetry breaking property of the WarpNet.

[3]Batch normalization layers are non-linear during training, due to the scaling by the sample variance.

3.1 FORWARD AND BACKWARD PROPAGATION ACROSS WARP OPERATORS

This section shows the propagation rules of the Warped Residual Network using the warp operator T_{warp}. The expression for T_{warp} is derived from Equation 3 by dropping the negligible $O(\epsilon)$ term, that is, by using the Taylor series expansion to the first order:

$$\mathbf{x}_{K+1} = \mathbf{x}_1 + \sum_{k=1}^{K} \binom{K}{k} (\mathbf{F}'(\mathbf{x}_1, \mathbf{W}^*))^{k-1} \mathbf{F}(\mathbf{x}_1, \mathbf{W}^*).$$

Note that T_{warp} can be calculated in a parallelizable manner for all K. This is shown in Figure 1 with $K = 2$, where [4]

$$\begin{aligned} \mathbf{x}_3 &= T_{warp}^{K=2}(\mathbf{x}_1, \mathbf{W}_1, \mathbf{W}_2) \\ &= \mathbf{x}_1 + \mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1) + \mathbf{F}_2(\mathbf{x}_1, \mathbf{W}_2) + \mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1)\mathbf{F}_2'(\mathbf{x}_1, \mathbf{W}_2), \end{aligned} \tag{4}$$

where $\mathbf{W}_i$ corresponds to the weights in the i-th residual unit in the original ResNet.

Now we derive the backpropagation rules. Suppose that the upstream gradient $\partial L/\partial \mathbf{x}_5$ is known and we wish to compute $\partial L/\partial \mathbf{W}_1$ for gradient descent. We first back propagate the gradient down from $\mathbf{x}_5$ to $\mathbf{x}_3$. With $\mathbf{x}_5 = T_{warp}(\mathbf{x}_3)$, we can derive the backpropagated gradient

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{x}_3} &= \frac{\partial L}{\partial \mathbf{x}_5} \cdot \frac{\partial \mathbf{x}_5}{\partial \mathbf{x}_3} \\ &= \frac{\partial L}{\partial \mathbf{x}_5} \cdot \left[I + \frac{\partial \mathbf{F}_3(\mathbf{x}_3, \mathbf{W}_3)}{\partial \mathbf{x}_3} + \frac{\partial \mathbf{F}_4(\mathbf{x}_3, \mathbf{W}_4)}{\partial \mathbf{x}_3} + \frac{\partial \mathbf{F}_3(\mathbf{x}_3, \mathbf{W}_3)}{\partial \mathbf{x}_3} \frac{\partial \mathbf{F}_4(\mathbf{x}_3, \mathbf{W}_4)}{\partial \mathbf{x}_3} \right], \end{aligned}$$

where we have set the derivative of $\mathbf{F}_4'$ to zero for ReLU non-linearities. Note that we have removed all BN layers from $\mathbf{F}_4'$ in our implementation. One sees that the same kind of parallelism in the warp operator is also present for back propagation. The above equals $\partial L/\partial \mathbf{x}_3$. Now we can evaluate the weight gradient for updates

$$\frac{\partial L}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \mathbf{x}_3} \cdot \frac{\partial \mathbf{x}_3}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \mathbf{x}_3} \cdot \left[\frac{\partial \mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1)}{\partial \mathbf{W}_1} + \frac{\partial \mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1)}{\partial \mathbf{W}_1} \frac{\partial \mathbf{F}_2(\mathbf{x}_1, \mathbf{W}_2)}{\partial \mathbf{x}_1} \right].$$

Similarly for the update rule for $\mathbf{W}_2$. Rules for the all other weights in WarpNet can be obtained in the same way,

$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \mathbf{x}_3} \cdot \frac{\partial \mathbf{x}_3}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \mathbf{x}_3} \cdot \left[\frac{\partial \mathbf{F}_2(\mathbf{x}_1, \mathbf{W}_2)}{\partial \mathbf{W}_2} + \mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1) \frac{\partial^2 \mathbf{F}_2(\mathbf{x}_1, \mathbf{W}_2)}{\partial \mathbf{x}_1 \partial \mathbf{W}_2} \right].$$

The weights $\mathbf{W}_1$ and $\mathbf{W}_2$ can be updated in parallel independently. The derivatives $\partial \mathbf{F}_2(\mathbf{x}_1, \mathbf{W}_2)/\partial \mathbf{x}_1$ (in $\partial L/\partial \mathbf{W}_1$) and $\partial \mathbf{F}_2(\mathbf{x}_1, \mathbf{W}_2)/\partial \mathbf{x}_1$ (in $\partial L/\partial \mathbf{W}_1$) are already computed in the forward pass which could be saved and reused. Furthermore, all derivatives needed in $\partial L/\partial \mathbf{x}_3$ can also be computed in the forward pass.

3.2 SYMMETRY BREAKING IN WARPED RESIDUAL NETWORKS

In this section we will use our warp operator as an example to show that the warp operator breaks rotational symmetry. For simplicity, we focus on the first warp operator in the chain of residual units, with the corresponding layers in the residual units $\mathbf{F}_1$ and $\mathbf{F}_2$. The same behaviors apply to all other warp operators in the network. Breaking this symmetry corresponds to breaking the redundancy in the weights, resulting in a higher level of generalization.

First consider $\mathbf{x}_2 = \mathbf{x}_1 + \mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1^*)$ as the first output from a residual unit with $\mathbf{x}_i$ being a $D \times D$ feature map. For simplicity, assume that $\mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1^*) = conv(\mathbf{x}_1, \mathbf{W}_1^*)$ is just a convolution layer with the respective weights. This can be done as long as the residual units contain only layers that do not affect rotation - their operations commute.

From the observation that same copies of rotated filters are learned when the inputs are symmetric (Dieleman et al., 2016), we let the weights have the same rotation as the inputs, so that the input

[4] This equation is from Equation (2) after dropping the negligible $O(\epsilon^2)$ terms.

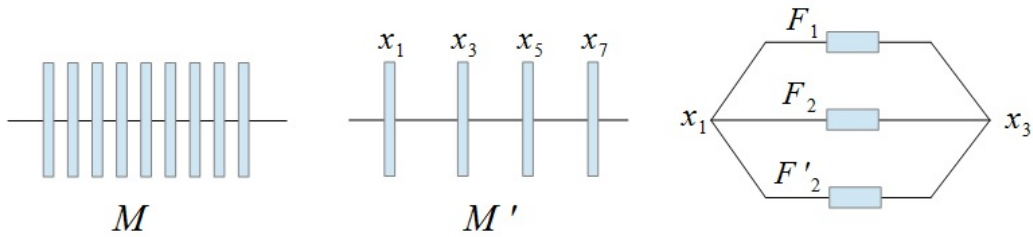

Figure 1: Forward propagation in WarpNet. Left, the original residual network where each block corresponds to a residual unit. Middle, the corresponding Warped Network with indices determined by the architecture of M. The diagram on the right shows the parallelism allowed in the forward pass of WarpNet. Note that $\mathbf{F}_2$ shares the same weights with $\mathbf{F}'_2$, as they come from the same residual unit in the original network M. The precise form for the warp operator is used for computation as shown in Equation 4.

$\mathbf{x}$ and weights $\mathbf{W}^*$ becomes $Q\mathbf{x}$ and $Q\mathbf{W}^*$, respectively, where the orthogonal matrices Q satisfy $Q^TQ = I$. Another way to see that this is sensible is to consider rotating the whole data set by Q. Then the learned weight would be rotated by Q, too. However, this is a *global* transformation as the input changes the weights over the entire network. This is equivalent to a change in the coordinate basis and the weights would be the same as the original untransformed network. Therefore we enforce the rotation (and all other symmetry transformations) to be a *local* transformation, where only the weights in the immediately layer operating on the input are transformed with the input. Such that a rotation on the input $Q\mathbf{x}_i$ induces a rotation on the corresponding weight $Q\mathbf{W}_i$ and other weights $\mathbf{W}_{j\neq i}$ remain unchanged. Henceforth we will only consider local symmetries and not the global ones as it does not reduce the redundancy in the weights.

Under a rotation, the output of the convolution layer is $Q\mathbf{F}(\mathbf{x}, \mathbf{W}^*)$. This can be seen by rotating both the feature map and filters on the 2D plane. So we have $\mathbf{F}(Q\mathbf{x}, Q\mathbf{W}^*) = Q\mathbf{F}(\mathbf{x}, \mathbf{W}^*)$ and rotational symmetry is preserved by the shared symmetry between the weights and the input. To avoid this redundancy the shared rotational symmetry must be broken.

Our network learns a rotation-invariant model by breaking the rotational symmetry within the warp operator. We will use the simplest case $K = 2$ as an example. We have

$$\mathbf{x}_3 = \mathbf{x}_1 + \mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1^*) + \mathbf{F}_2(\mathbf{x}_1, \mathbf{W}_2^*) + \mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1^*)\mathbf{F}'_2(\mathbf{x}_1, \mathbf{W}_2^*).$$

We showed in the last paragraph that weights in $\mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1^*)$ and $\mathbf{F}_2(\mathbf{x}_1, \mathbf{W}_2^*)$ can be rotated so that each term preserve rotational symmetry,

$$\begin{aligned} \mathbf{F}_1(Q\mathbf{x}_1, Q\mathbf{W}_1^*) &= Q\mathbf{F}_1(\mathbf{x}_1, \mathbf{W}_1^*) \\ \mathbf{F}_2(Q\mathbf{x}_1, Q\mathbf{W}_2^*) &= Q\mathbf{F}_2(\mathbf{x}_1, \mathbf{W}_2^*). \end{aligned}$$

However, since $\mathbf{F}'_2(\mathbf{x}_1, \mathbf{W}_2^*)$ is the gradient of the convolution, which is convolution with a flipped $\mathbf{W}_2^*$, rotated by 180 degrees, $\mathbf{F}'_2 = conv(\cdot, Q_\pi \mathbf{W}_2^*)$. There is only a rotation of 180 degrees of the feature map that can simultaneously the preserve rotational symmetry of both $\mathbf{F}_2(\mathbf{x}_1, \mathbf{W}_2^*)$ and $\mathbf{F}'_2(\mathbf{x}_1, \mathbf{W}_2^*)$. To see this, consider a rotation in the input such that $\mathbf{F}_1 \rightarrow Q\mathbf{F}_1$. The third term becomes $\mathbf{F}_1\mathbf{F}'_2 = conv(Q\mathbf{F}_1, Q_\pi \mathbf{W}_2^*)$. The rotational symmetry is preserved if and only $Q = Q_\pi$ so that $\mathbf{F}_1\mathbf{F}'_2 \rightarrow Q_\pi \mathbf{F}_1 \mathbf{F}'_2$. Thus rotational symmetry is broken down to a symmetry corresponding to a rotation of 180 degrees.

4 EXPERIMENTS

In this section we discuss our implementation of the WarpNet architecture and the experimental results. In order to ensure the validity of the series expansion we replace the 1×1 convolution layers on skip connections by an average pooling layer and a concatenate layer to reduce the spatial dimensions of feature maps and mutiply their channels. We adopt a wide residual architecture (Zagoruyko & Komodakis, 2016). The average pooling and concatenate layers were used to that the identity mapping is preserved across dimensionality changing units. The convolution blocks

Table 1: Network architecture

$H \times W \times C$	Conv blocks
$32 \times 32 \times 3$	Input
$32 \times 32 \times 16$	Conv-BN-ReLU
Stage 1	Concatenate
$32 \times 32 \times 16k_w$	$[T_{warp}^{(K=2)}] \times N_{warp}$
Stage 2	Avg_pool(2x2)-Concatenate
$16 \times 16 \times 32k_w$	$[T_{warp}^{(K=2)}] \times N_{warp}$
Stage 3	Avg_pool(2x2)-Concatenate
$8 \times 8 \times 64k_w$	$[T_{warp}^{(K=2)}] \times N_{warp}$
$1 \times 1 \times 64k_w$	BN-ReLU-Average_Pooling(8,8)
	Fully Connected(# of classes)

$\mathbf{F}$ comprised of the following layers, from input to output, BN-Conv-BN-ReLU-Conv-BN (Han et al., 2016). The neural architecture of Warped Residual Networks is shown in Table 1. The layers $[T_{warp}^{(2)}] \times N_{warp}$ represent forward propagating N_{warp} times, such that $\mathbf{x}_{i+K} = T_{warp}^{(K)}(\mathbf{x}_i, \mathbf{W}_i)$ and the indices i correspond to the indices in the original residual network.

We found that the computation bottleneck arises from the BN layers in $\mathbf{F}'_2$. The reason being the gradient of BN layers contains an averaging operation that is expensive to compute. In our final implementation we removed all BN layers in $\mathbf{F}'_2$ from our network. This results in a departure from our series approximation but it turns out the network still trains well. This is because the normalizing layers are still being trained in $\mathbf{F}_{1,2}$.

Using Tensorflow, we implemented a network with $k_w = 4$, $K = 2$ and $N_{warp} = 3$, where k_w is the widening factor (Zagoruyko & Komodakis, 2016), K is the warp factor and with the scheme shown in Figure 1. We employ Tensorflow's automatic differentiation for backpropagation, where the gradients are calculated by sweeping through the network through the chain rule.

We use common data augmentation techniques, namely, whitening, flipping and cropping. Comparison of test errors with the original ResNets is shown in Table 2. We find that our network outperforms the wide ResNet in CIFAR-10, and within standard deviation on CIFAR-100. The reason is likely that there are much fewer weight redundancy for CIFAR-100. The number of feature maps per class is 10 times fewer in CIFAR-100 than in CIFAR-10.

We trained for 80000 iterations, or 204 epochs. We took training batch size of 128. Initial learning rate is 0.1. We used two sets of hyperparameter (A,B). For set A, the learning rate drops by a factor of 0.1 at epochs 100 and 150. The weight decay for set A is 0.0002. For set B, the learning rate drops by a factor of 0.1 at epochs 60, 120, and 160, with a weight decay of 0.0005.

To exploit the parallelism of the our architecture and reduce the memory footprint at the same time, we computed $\mathbf{F}_1$ on one GPU, $\mathbf{F}_2$ and $\mathbf{F}'_2$ at the same time on another GPU because they share the same weights. We observed significant speed up in both forward and backward propagation. To measure the computation time, we first generated a random batch of data of dimensions $(128 \times 32 \times 32 \times 3)$. The time for forward propagation is the time to obtain the output of the network using this generated batch of data. The total time is the time to fully process a batch during training. The backward propagation time is calculated using the difference between the total time and forward propagation. The relative speed up with respect to the original ResNet at various k_w values is recorded in Table 3. We find that the computation time is relatively constant with respect to the network depth, N_{warp}. The speedup appears to scale with increasing widening factor k_w and sizable with just a warp factor of $K = 2$.

The Warped Residual Network is able to outperform the residual network. The reason is that the perturbative series expansion generates extra layers to learn invariant models. For example, a convolution with flipped filters appears in the forward propagated gradient term. This results in a change in the loss landscape so that the Warped Network is more likely to learn a rotation-invariant model.

Table 2: Test errors (%) between the ResNet and its approximation, the Warped Residual Networks (WarpNets). The ResNet here has the same pooling and concatenate structure as the WarpNet in Table 1, but with each warp operator replaced by two residual units.

	# of parameters	CIFAR-10	CIFAR-100
Wide ResNet 38-4-A	9.3M	5.45	23.1
WarpNet 38-4-A	9.3M	5.12	23.2
Wide ResNet 38-4-B	9.3M	5.42	22.8
WarpNet 38-4-B	9.3M	5.13	22.3

Table 3: Relative speed up with respect to ResNet, given by $(t_{res} - t_{warp})/t_{res}$, where t_{res} and t_{warp} are the time taken for either forward propagation or total time with ResNet and WarpNet, respectively. $N_{warp} = 3$ in all experiments.

	Forward	Backward	Total time
$k_w = 4$	31%	43%	33%
$k_w = 8$	35%	43%	41%
$k_w = 16$	37%	47%	44%

5 Concluding Remarks

We made significant progress in understanding deep neural networks in this work, producing a parallelizable neural network architecture that can produce comparable results to the original ResNet while having significant speedup due to the parallelism. The major contributions comes from two fronts. First, we showed that a residual network can be decoupled into individual residual units that weakly couples to each other. Second, we modelled the redundancies in the weights caused by symmetries in the input as local symmetry transformations and found that WarpNet is able to reduce more weight redundancies than ResNet.

We proposed a novel type of depth-wise parallelism in neural networks derived from the Residual network. In principle, all layers in Resnet can be skipped and trained in parallel with the warp operator in WarpNet. As a proof of concept, we implemented a WarpNet that skips every other layer and we were able to obtain speedup. Our work opens up a new research direction for the architecture that can train all ResNet layers with the time of just a few. This is not a trivial task, as the number of parallelizable computation grows exponentially with the number of layers skipped. We believe that some kind of approximations can be further made, just as we have done so by removing all BN layers in $\mathbf{F}'_2$. As deep networks can spend days, if not weeks, to train. Research in methods that can reduce the training time required while maintaining performance is of the utmost importance.

Acknowledgments

We thank Wenxin Xu for providing his code for ResNet at `https://github.com/wenxinxu/resnet_in_tensorflow`.

References

V. Badrinarayanan, B. Mishra, and R. Cipolla. Symmetry-invariant optimization in deep networks. 2016. `https://arxiv.org/abs/1511.01754`.

T. S. Cohan and M. Welling. Group equivariant convolution networks. *International Conference on Machine Learning*, 2016.

S. Dieleman et al. Exploiting cyclic symmetry in convolutional neural networks. 2016. `https://arxiv.org/abs/1602.02660`.

R. Gens and P. M. Domingos. Deep symmetry networks. *In ICLR*, 2014.

K. Greff, R. K. Srivastava, and J. Schmidhuber. Highway and residual networks learn unrolled iterative estimation. 2016. `https://arXiv.org/abs/1612.07771`.

R. Hahnloser et al. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405:947–951, 2000.

D. Han, J. Kim, and J. Kim. Deep pyramidal residual networks. 2016. https://arXiv.org/abs/1610.02915.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, 2015. https://arxiv.org/abs/1512.03385.

G. Huang et al. Deep networks with stochastic depth. *CoRR*, 2016. https://arXiv.org/abs/1603.09382.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 448–456, 2015.

A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012.

E. Orhan and X. Pitkow. Skip connections eliminate singularities. 2017. https://arxiv.org/abs/1701.09175.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

N. Srivastava et al. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. 2015. https://arxiv.org/abs/1512.03385.

C. Szegedy et al. Going deeper with convolutions. *CVPR*, 2015.

A. Veit, Wilber M., and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. 2016. https://arXiv.org/abs/1605.06431.

S. Zagoruyko and N. Komodakis. Wide residual networks. 2016. https://arXiv.org/abs/1605.07146.