Residual Networks for Tiny ImageNet

Hansohl Kim
Stanford University
hansohl@stanford.edu

Abstract

Residual networks are powerful tools for image classification, as demonstrated in ILSVRC 2015 [5]. We explore the application of pretrained residual networks to the Tiny ImageNet Challenge, with the goal of both accuracy and minimal retraining time. We achieve high performance with relatively little finetuning on an 18-layer Facebook residual network in Torch [3] through single-layer training and rescaling of filter weights to obtain well-initialized parameters for layers that must be adapted or replaced.

We achieve 31.1% test error, 24.99% validation error, and 8.14% top-5 validation error, matching the accuracy of the pretrained network on its original task (full ImageNet). These results were obtained with a model that underwent only 32 epochs of retraining, demonstrating high performance with little computation. The pretrained residual network was highly flexible and adapted quickly to the required changes for Tiny ImageNet, but it was also highly vulnerable to poorly-initialized finetuning. Residual connections may amplify these strengths and weaknesses, which may be of interest in future efforts to adapt pretrained residual networks to new tasks.

1. Introduction

Convolutional Neural Networks have demonstrated remarkable successes in the field of computer vision, approaching and surpassing human-level accuracy in image classification tasks such as the ImageNet Challenge [5, 6]. Much of this progress is built not only on the increasing complexity and capacity of networks, but also on the development and use of new techniques that allow for effective training of larger and deeper networks [3, 5, 6, 7, 8]. As such, classification challenges such as ILSVRC (ImageNet Large Scale Visual Recognition Challenge) have served as testing grounds and driving forces for the development of efficient and effective networks for computer vision.

We address the Tiny ImageNet 200 Challenge, an image classification dataset sampled from the larger ImageNet data. As in full ImageNet classification, the objective is to attain maximum classification accuracy over a diverse range of image inputs. Although image classification is a well-studied field relative to components of ILSVRC such as localization and detection, the Tiny ImageNet dataset provides an environment compact enough to facilitate experimentation with limited resources, while still providing depth as a significant challenge.

While (as discussed later) the downsampling of images and classes in Tiny ImageNet presents problems that differentiate it from the original ILSVRC classification challenge, the general techniques and methods developed for convolutional networks in ILSVRC should still largely apply. With limited time and computational resources, we were motivated to explore the adaptation of networks pretrained on ImageNet to the Tiny ImageNet Challenge as an experiment in the efficacy of transfer learning from state-of-the-art models to achieve high performance with limited resources.

In particular, we leverage pretrained residual networks, which held the ImageNet classification accuracy record until February 2016 [5] and incorporated several recent developments in convolutional neural networks in a unified architecture. The techniques used in these residual networks represent significant advancements since the previous Tiny ImageNet Challenge, and are results of the progress made in the full ILSVRC that we now apply to Tiny ImageNet. These pretrained models adapt well to Tiny ImageNet and exhibit very high performance on the classification challenge when carefully finetuned in stages. The strong results obtained with only 10-15 hours of retraining on pretrained models indicate high capability in this case for finetuning existing residual networks, and suggest properties of retraining residual networks that may generalize.

2. Related Work

2.1. Tiny ImageNet

The Tiny ImageNet Challenge has occured previously, in 2015. As such, the previous year's submissions offer significant insight into the challenge. We first consider the results obtained as a benchmark for our own performance. The optimal publicized approaches in Tiny Ima-

geNet 2015 achieved reported test error rates between 45 to 60% [2, 4, 10, 14], with only one achieving below 50% [10].

These approaches were non-residual convolutional networks, largely based on linear AlexNet or VGG-like models, as residual networks did not exist. Approaches varied from experimenting with activation types [4, 10] to the effects of progressively deeper networks and ensembles [10, 14]. All included analysis of the Tiny ImageNet dataset itself and concluded that data augmentation techniques, such as random cropping, were vital to the challenge, as well as dropout to reduce overfitting [2, 4, 10, 14]. We note that in the most successful approach, the primary factor in reducing error rate was the use of deeper networks with longer training [10]. Furthermore, we note that these networks were limited to at most 16 layers (and often fewer), and that insufficient training time and computational resources were common considerations.

In considering our approach, we took into account the role of network depth in these previous submissions, as well as the commonly raised issue of data augmentation for Tiny ImageNet. In keeping with [3] and [9], we employ the standard random crop and flip procedure used to train residual networks, as well as color and lighting variation. We pursued pretrained networks as a method to obtain depth while allowing time for experimentation and training to convergence. We also noted the unimpressive performance of PReLU units reported by [10] and retained ReLU.

2.2. Residual Networks

Among pretrained networks we focus on residual networks. This model was introduced by [5] in 2015 for the ImageNet Challenge, and it incorporates several recently developed techniques. As a result, the residual network we employ here for Tiny ImageNet and our usage of it draw upon significant related literature.

The primary addition introduced by [5] is the residual connection. While many earlier convolutional networks used a simple linear structure of layers, recent research has experimented with branching and merging layers [5, 12, 13]. Residual networks use shortcut connections between blocks in the sequential layers to skip through the network. These shortcut connections allow largely unimpeded backpropagation of the loss gradient through deep networks, improving the viability of very deep networks [5].

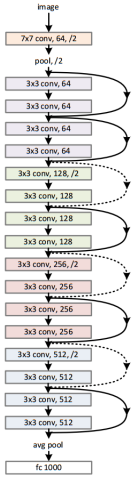
Residual networks also make liberal use of (spatial) batch normalization layers [8], which accelerate training by addressing shifting input distributions to layers during the training process. These layers force inputs corresponding to each feature to conform to a unit Gaussian distribution. Means and variances are calculated across batches, and a running average is kept for use during testing. [8] claims this significantly speeds training when applied prior to activations by reducing sensitivity to initialization and allowing for increased learning rates. [5] eschews dropout following the advice of [8], which claims dropout is largely redundant with batch normalization. We therefore do the same.

Finally, [5] largely implements the suggestions of [11] to rely almost exclusively on convolutional layers. The residual network architecture includes a single max-pool, average-pool, and fully connected layer each. Spatial downsampling is handled primarily by strided convolutions, while removing the Fully Connected ReLU layers found at the end of both AlexNet and VGG-net removes a large portion of the parameters and greatly reduces complexity.

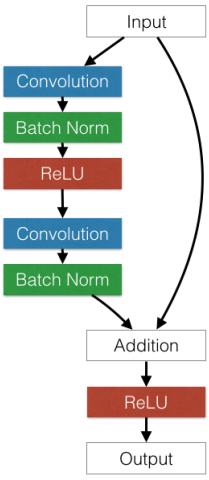
The initial residual networks in [5] won ILSVRC 2015 and attracted significant attention. The architecture was later implemented by Facebook [3] for further exploration, allowing public access to pretrained 18, 34, 59, and 101-layer residual networks. [3] experiments with the placement of the residual connections, the depth of the networks, and the training algorithms used. We refer to [3] and opt to use simple SGD with Nesterov Momentum, while noting that the differences in performance from the various depth networks are relatively small compared to the additional effort required to train deeper models.

3. Methods

3.1. Model Architecture



(a) Architecture (edited from [5])



(b) Residual Block [3]

Figure 1: 18-Layer Residual Network

Our pretrained models are obtained from [3] and are based on the Residual Networks of [5]. A residual block is depicted in figure 1b, where the shortcut residual connection from the input is added prior to the final ReLU layer. Our primary model is the minimal 18-layer pretrained

model, which is depicted in figure 1a. We note that spatial downsampling is achieved by a stride of 2 in the first conv layer of a block, and that shortcut connections also use strided convolutions as in [5].

While our pretrained model is defined by [3], we make certain adjustments to the structure to adapt it to Tiny ImageNet. The primary adjustment is the replacement of the final 256 to 1000 fully connected classification layer with a 256 to 200 fully connected layer. This is required for our smaller number of classes. We also later consider replacing the initial 7x7 stride 2 convolution and 2x2 max-pool with a single 3x3 stride 1 convolution.

3.2. Implementation Framework and Computation

We implement our experiments in the Torch framework [1]. Torch is based on the Lua scripting language and is highly modular, allowing for simple modification of models for experimentation. Torch also has good access to recently developed models and techniques, and the publicly released pretrained Facebook residual networks are also implemented in Torch.

Retraining was done with an AWS G2.2xlarge instance, which provided access to an NVIDIA GRID K520 GPU.

3.3. FC Layer Retraining

Adapting the pretrained residual networks required replacing the final fully connected (fc) layer to match the reduced number of classes (200 rather than 1000). The adapted model was then trained using the same techniques employed by [5] and [3], with stochastic gradient descent operating through standard backpropagation. The class scores calculated by the final layer were input into a softmax function for a cross-entropy loss. The softmax function normalizes scores across all classes to sum to 1.

$$Softmax(\mathbf{z}_j) = \frac{e^{\mathbf{z}_j}}{\sum_k e^{\mathbf{z}_k}} \tag{1}$$

This allows interpretation of each class score as the (unnormalized) log probability of a class given the input image. Then the we consider the probability that the class y is j based on incoming features $\mathbf{x}$ and weights $\mathbf{w}$ in the fully connected (final) layer to be:

$$P(y = j|\mathbf{x}; \mathbf{w}) = \frac{e^{\mathbf{x}^T \mathbf{w}_j}}{\sum_k e^{e^{\mathbf{x}^T \mathbf{w}_k}}} \tag{2}$$

Which allows prediction of class based on the highest probability, as well as the definition of the loss as:

$$Loss_j = -\log(Softmax(\mathbf{x}^T \mathbf{w}_j)) \tag{3}$$

Allowing us to backpropagate the loss (depending on the target) through the network. As in [3], we use Nesterov momentum with stochastic gradient descent.

We note, however, that important nuances arise in the manner of retraining. We first considered the straightforward "naive" method of adapting the pretrained network to Tiny ImageNet, where we simply reset the final layer and retrain the entire network on the Tiny ImageNet data. Here we allowed backpropagation through the entire network.

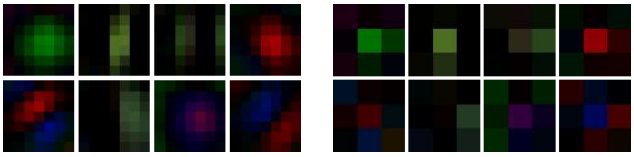
We then considered the case of fixing the first 17 layers at their pretrained values and only retraining the reset final layer. Then we only allow backpropagation through the final layer, and this is equivalent to viewing the first 17 layers as a fixed pretrained feature extractor feeding a linear softmax classifier.

Finally, we considered merging the two methods in a staged pattern. We first fix the first 17 layers and train the reset final layer to convergence. We then release the first 17 layers for finetuning once the final layer has been well initialized.

3.4. Image Size Adaptation

In addition to the reduced number of classes, another major adaptation required for Tiny ImageNet is the reduced image size and corresponding resolution. ImageNet data consists of 256x256 pixels, while our pretrained networks accept 224x224 pixel crops. Tiny ImageNet data, however, consists of 64x64 pixel images.

We initially bypassed this issue by scaling the input images to 256x256 with bicubic interpolation. This allows for 224x224 crops for the network. However, this introduces significant potential for error and scaling artifacts, as well as harmful noise and non-existent texture.



(a) Original 7x7 (b) Scaled to 3x3

Figure 2: Sample of Initial Convolution Filter Weights

We therefore attempted to address this scale-by-4 issue by replacing the initial 7x7 stride 2 convolutional and 2x2 max-pool layers with a single 3x3 stride 1 convolutional layer. This allows for 56x56 crop input while maintaining the spatial size for each interior layer, so the final convolution still produces a 7x7 spatial output and each layer is observing features at the expected scale.

Again, we attempted a blank initialization of this first layer as well as a meaningful initialization. For the initialization we scaled the 7x7 filter weights themselves using bicubic interpolation, as the initial convolution filters consist of basic Gabor-patterns (figure 2).

4. Dataset

4.1. Tiny ImageNet 200

The Tiny ImageNet Challenge is essentially a downsampled version of the ImageNet Challenge for classification. The 2012 ImageNet Classification Dataset consists of over 1.2 million labeled training examples with 1000 classes [5]. The state-of-the-art (December 2015) top-5 test error is 3.57%, while the top-1 validation error is 19.38% [5].

In contrast, Tiny ImageNet consists of 200 classes, each with 500 training examples, as well as 10000 images in each of the validation and testing sets. The images are also downsampled from 256x256 pixels to only 64x64 pixels. As noted by [2] and [4], this sampling can actually pose significant problems, as it is possible for the reduction process to distort relevant information or even crop it out completely. Furthermore, 500 training examples per class is far fewer than in CIFAR 10 (5000) or ImageNet (1000). It is therefore difficult to assert that the smaller size of Tiny ImageNet should easily allow accuracy rates exceeding those achieved on ImageNet, since the downsampling may introduce new difficulties as well. We may, however, expect somewhat similar accuracy rates, as the two challenges do share many common aspects.

Figure 3: From [4]: "Note the scaling artifacts prominent in (a) and (d), the loss of texture in (c) and (g), the loss of crucial information due to cropping in (b), (f), and (h), and the difficulty of locating small objects in (c)"

4.2. Data Augmentation

With relatively few training images per class, we rely heavily on data augmentation to counter overfitting and allow for a generalizable model. We train our networks with randomized 224x224 crops of input images (56x56 for our later models with size adaptation) that are also randomly flipped (horizontal reflection). Additional augmentation is provided by [3], following the example of [5] (first described in [9]) in the form of color and lighting jitter, introducing another form of regularization. This jitter averages to 0, allowing us to run testing unaffected.

(a) Single Center Crop (b) Part of a 10-Crop

Figure 4: Test Image Crop Methods

The randomized crops, however, must be addressed during testing, and the method used can affect the performance of a model. We considered several different schemes to account for random crop training at test time.

The default test method from [3] is a single, centered crop of the test image. This requires only a single pass through the network, but also discards potentially significant information around the edges. Resizing the full image to the crop size allows for a similar effect, but without removing the edge information. Resizing, however, poses potential problems in warping the information.

In addition, we employ a 10-Crop method, a standard procedure that takes four crops from each corner, as well as one from the center, and then performs a horizontal flip on each to pass 10 crops from a single test image through the network. The classification scores are then combined to determine the most likely class. In keeping with the interpretation of score as log probability, we sum the exponentials of the scores from each crop.

We also introduce a 12-Crop, where we rescale the full image to crop size and include the scaled full image and its horizontal reflection in the 10-Crop scheme. This has potential usefulness in Tiny ImageNet, where the extremely small image sizes often result in significant loss of information from further cropping. The additions to the 12-Crop allow the network an opportunity to view the whole image.

5. Experiments and Results

5.1. FC Layer Retraining

The first change to the pretrained network was the replacement of the final, fully connected layer. The new fully connected layer was not initialized beyond the default Torch behavior. Naive finetuning, where we subsequently allowed the entire network to retrain on the Tiny ImageNet data, converged to 48% validation error within 9 epochs. On the GRID K520 GPU, this corresponded to only 4 hours.

Recalling the previous submission benchmarks of 60-45% validation error, we observe that even a straightforward approach to applying pretrained residual networks achieves good results with minimal computation.

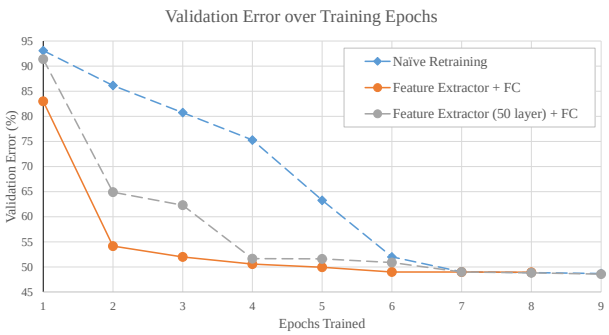


Figure 5: FC Layer Retraining Methods

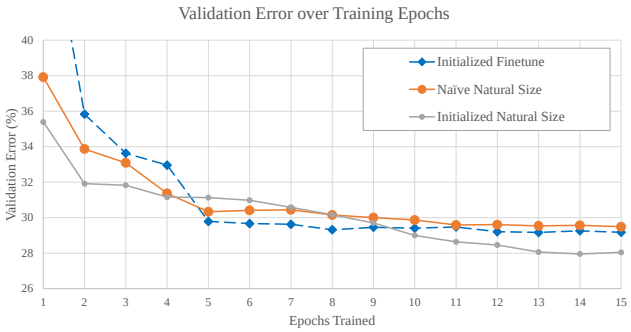


Figure 6: Initial Convolution Retraining Methods

We note, however, that fixing the first 17 layers as a feature extractor and training only the fc layer achieved equally competitive results and converged more quickly (figure 5). The 50-layer pretrained fixed feature extractor also converged more quickly than the 18-layer naive finetuning. There was, however, no notable difference in accuracy observed between the feature extractors, so the 18-layer model was selected for all further developments, as the 50-layer network was significantly slower to train and was greatly limited in batch size by the memory requirements of the parameters. The 18-layer feature extractor achieved 50% validation error in only 54 minutes on the GRID K520 GPU.

We would expect an optimization over a single layer to perform poorly compared to an optimization over all layers, unless the remaining 17 layers were already completely optimal for Tiny ImageNet. We therefore view the equal performance obtained by the naive finetuning and the fixed feature extractor as evidence that the naive approach interferes significantly with the finetuning process, as expected. Large loss gradients backpropagating from the complete retraining of the fc layer are potentially out of scale relative to the finely pretrained parameters in the preceding 17 layers. Such wild gradients may greatly perturb these highly trained parameters from their optimal values, ruining some of the benefit of using pretrained networks.

We therefore expected the staged approach, where we initially fixed 17 layers and retrained prior to full network finetuning, to achieve lower error than the previous approaches. This staged, well-initialized finetuning broke the 45% validation error barrier and dropped to below 30% in 5 epochs (figure 6 - blue). The improvement in accuracy was expected, but the magnitude of the improvement was not. This discrepancy suggests that these pretrained residual networks are highly susceptible to the negative effects of poor (or arbitrary) initialization in altered layers. We also note, however, that the residual networks readjust and (approximately) converge quickly in all cases, potentially aided by the residual connections introduced to provide efficient and coherent training across layers.

5.2. Image Size Adaptation

The significant effects of initialization were also observed in the first layer. To allow the pretrained network - trained on 224x224 pixel crops - to train on the natural 56x56 pixel crop size of Tiny ImageNet, we replaced the initial 7x7 stride 2 layer with a 3x3 stride 1 layer and removed a max-pool. This was an experiment on the adaptability of the pretrained residual network, and we had no prior reason to expect improvement from these changes.

Figure 6 depicts the retraining of the replaced convolutional layer compared against the well-initialized finetune model achieved at the end of Section 5.1 (blue). In the "naive" approach, we apply the scaling of the 7x7 filter weights to 3x3 filter weights discussed in Section 3.4 but proceed to retrain the entire network at once. The validation error increases, but is at only 38% after a single epoch. The network is able to largely recover from the alteration and achieve similar accuracy on the natural 56x56 crops rather than crops scaled to 224x224. The scaling of Gabor filters in section 3.4 therefore seems to be an effective tool in this situation. However, this method does not manage to completely regain its prior accuracy.

The "well-initialized" approach is largely identical to section 5.1. By fixing all other layers and only training the initial convolutional layer, we provide an improved initialization for full network finetuning. The model trained with this method not only regains its prior accuracy, but actually exceeds it, reducing validation error by almost 2%.

Not depicted in figure 6 is the true naive approach, with no initialization of the new convolutional layer at all. This method immediately diverged above 50%, and we note that significant error in the first layer can cascade and be amplified through the remaining layers, perturbing pretrained parameters. The validation error did not return to below 50% within the allotted 15 epochs. These results suggest that the scaled initialization from Section 3.4 is significantly better than blank re-initialization, and enforce our understanding of the vulnerabilities of our pretrained residual network.

(a) (b) (c) (d)

Figure 7: Images Vulnerable to Crop Type

5.3. Test-Time Cropping

The various test-time cropping methods discussed in Section 4.2 offered a final source of improvement for both validation and test performance. We previously noted that data augmentation was particularly important with only 500 images per class. However, simple changes in test-time behavior to account for our data augmentation crops changed validation performance by 2 to 3% (Table 1).

We note that the default center crop method we had be using performs poorly in both validation and testing. Figure 7 contains example images that were particularly susceptible to reclassification as the crop methods were changed. Both full and center crop methods classified (a) as a spider and (b) as volleyball. Both center crop and 10-Crop classified (c) as an apron and (d) as a lake (note the presence of a lighthouse in the far left).

While these mistakes qualitatively make some sense, these images appear to represent a problematic balance between attention to global and local detail. The full and center crop methods are more attuned to global details, capturing the large features such as the shape of the Christmas stockings where crop methods fail. As a consequence though, these methods are less capable of detecting important local clues. 10-Crop can easily distinguish the deer in (a) and the lamp in (b), and the crops focused on those objects contribute a strong signal to the overall decision. We note, however, that both (c) and (d) represent shortcomings of 10-Crop (as well as the center crop), as the inability to capture the entire image risks ignoring or missing important details. Any of the 10 crops of (c) lose a significant portion of the stocking shape, while fewer than half of the 10 crops of (d) contain any trace of the lighthouse.

The 12-Crop method was an attempt to address these problems. The 10 original crops allow for attention to local detail, while the additional scaled full images allow the ability to avoid cropping any useful information. While this approach still fails in several extreme cases (such as (d), which is an unfortunate combination of a very localized detail that is not particularly strong in any crop), it provides consistent improvement to both validation and test accuracy. We therefore choose the 12-Crop method over the other systems. We note that this optimality may be unique to the smaller Tiny ImageNet data, where the low resolution and small picture size increase the possibility of several local objects blending together or crops discarding vital details, necessitating the full image.

As an additional note, we experimented with different methods of combining the crop predictions in 10-Crop and 12-Crop, but found that the sum of e^{score} remained optimal for our network and test data (albeit by less than 1%), possibly reinforcing our view of the trained class score predictions as unnormalized log probabilities of classes. We therefore continue using the exponential sum to average the contributions.

Table 1: Test-Time Crop Methods. Results from model with FC initialized finetuning but no size adaptation (224x224)

Error Rate (%)	Val Top-5	Val Top-1	Test Top-1
Full (Resize)	8.97	28.8	34.4
Center Crop	9.94	28.9	34.2
10-Crop	9.10	27.1	32.2
12-Crop	8.48	26.8	32.0

5.4. Final Model

Table 2: Test-Time Crop Methods. Results from model with FC initialized finetuning but no size adaptation (224x224)

Error Rate (%)	Val Top-5	Val Top-1	Test Top-1
Naive FC	23.65	48.69	N/A
Feat. Extr. FC	24.31	48.93	N/A
Feat. Extr. FC (50)	24.14	48.56	N/A
Initialized FC	8.48	26.83	32.1
Final Model	8.14	24.99	31.1

The final model for our Tiny ImageNet Challenge submission was an "initialized finetuning" model for both the final fc layer and the initial convolutional layer. It used 12-Crop testing and exponential summing of crop scores. The model achieved 31.1% test error, 24.99% validation error, and 8.14% top-5 validation error. The fc layer initialization was 5 epochs, the fc layer finetuning was 10 epochs, the convolutional layer initialization was 4 epochs, and the final finetuning was 13 epochs. 32 epochs of training were therefore required for this model, 9 of which were significantly faster single-layer training epochs. On the GRID K520 GPU this slightly under 13 hours.

Training hyperparameters were initially set by randomized search, validating over three epochs of training. Learning rate began at 0.01 and was reduced when the running average of validation error over epochs did not improve notably over three epochs. We note, however, that the hyperparameters from [3] and [5] were near optimal to achieve our results.

6. Conclusions and Future Work

By achieving 31.1% test error with only 32 epochs of retraining, we have demonstrated that pretrained residual networks for ImageNet can be efficiently adapted to the Tiny ImageNet Challenge with relatively little retraining while achieving very high accuracy. We note that the original validation error rate for the 18-layer Facebook residual network on ImageNet was 30.6% using single crop testing. Our final model achieved 27.1% validation error and 32.0% test error when restricted to center crop testing. We are therefore approaching equality between the accuracy rate on the original task and the adapted task.

Perhaps most noteworthy is the short retraining time required. Careful and staged initialization and retraining of the replaced layers prior to full model finetuning allowed us to train our final model in fewer than 13 hours on a publicly available (priced) Amazon GRID K520 GPU. The speed of readjustment from major layer changes was perhaps the most surprising aspect of finetuning these residual networks, particularly after noting that even the miniaturized 18-layer network is deeper than previous Tiny ImageNet architectures based on AlexNet or VGG-net. If this speed and ability to adapt generalize to other residual networks and tasks, finetuning of pretrained residual networks could represent an efficient and powerful tool for fast prototyping and experimentation.

Future work could therefore be focused on exploring whether the flexibility of residual networks observed here is applicable in other similar situations. Determining whether the residual connections themselves significantly improve retraining and finetuning efficiency, in addition to their demonstrated ability to improve initial training efficiency, may be of particular interest.

An equally important question may be whether the same residual connections amplify the negative effects of poor initialization in naive retraining. Poor initialization was observed to significantly impair our networks' performance, and we were forced to address this with initialization techniques. If the residual connections serve to facilitate the disturbance of pretrained parameters from their optimal ranges in early retraining, residual networks may merit more care than usual in finetuning. In this case, experiments with a strongly varying set of individual learning rates for each layer or staged freezing of layers while finetuning may yield useful results.

References

[1] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning, 2011. In *Biglearn, NIPS Workshop*, number EPFL-CONF-192376.

[2] S. Feng and L. Shi. Kaminet - a convolutional neural network for tiny imagenet challenge. *CS 231N*, 2015.

[3] S. Gross and M. Wilber. Training and investigating residual nets. *http://torch.ch/blog/2016/02/04/resnets.html*, 2016.

[4] L. Hansen. Tiny imagenet challenge submission. *CS 231N*, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv:1502.01852*, 2015.

[7] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*, 2012.

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

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

[10] Y. Le and X. Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 2015.

[11] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv:1412.6806*, 2015.

[12] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv:1505.00387*, 2015.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv:1409.4842*, 2014.

[14] L. Yao and J. Miller. Tiny imagenet classification with convolutional neural networks. *CS 231N*, 2015.