Towards Efficient Recurrent Visual Models

David S. Hippocampus*
Department of Computer Science
Cranberry-Lemon University
Pittsburgh, PA 15213
hippo@cs.cranberry-lemon.edu

Abstract

It is well known that the human visual cortex is rich in feedback loops. Besides their biological plausibility, recurrent visual architectures have significant practical advantages over their purely feed-forward counterparts. Recurrence in vision models would allow for reducing the number of parameters and therefore combating over-fitting. In recurrent networks, the amount of computation (readout time) could be chosen dynamically based on the complexity of the scene. Also, the representational power of such models is increased significantly over feed-forward networks with similar parameter count. Despite these potential advantages, feed-forward architectures still outperform their similar cost counterparts by a large margin. In this paper, we study the activation dynamics of deep residual networks and the effect of relaxed weight sharing techniques in the context of computer vision. Following recent works of He et al and Liao and Poggio, we think that a better understanding of recurrent residual networks could lead to more efficient models for computer vision.

1 Introduction

Motivated by the recent introduction of residual networks by He et al [He+15], Liao and Poggio [LP16] proposed that by sharing the weight between its residual blocks, a deep residual network is turned into a recurrent network with fixed readout time. They conjecture that the surprising effectiveness of deep networks can be attributed to the fact that they can emulate recurrent networks. We think that this view has strong practical merits beyond its biological inspiration. Efficient recurrent vision models have a lot of attractive features including their relatively low parameter count and their strong representational power. Also, artificial recurrent architectures allow for efficient, natural switching between low-cost and high quality modes depending on the complexity of the visual input and the circumstances. This would prove especially useful in resource constrained environments such as mobile or embedded vision.

On the other hand [LP16] fails to provide strong empirical evidence for the qualitative superiority of recurrent architectures for practical computer vision tasks. Although they manage to train moderately (up to twenty layers) deep residual networks with weight sharing, their experimental findings seem to support the hypothesis that the sheer parameter count is the best predictor of the recognition power of the neural network, regardless of its depth. This raises the exciting question whether this is an inherent flaw of the idea or an artifact of the employed training methodology: even if the space of efficient recurrent parameter sets might be large, commonly employed optimization methods could be too weak to find those solutions in reasonable time.

In this paper, we look at this problem from various angles. First, we hypothesize that with more refined optimization methods, tailored specifically towards vertically weight-shared networks, the quality

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

gap between similar depth feed-forward and recurrent architectures can be bridged significantly. One interesting candidate for such a training method – parameter sharing relaxation – was introduced by Kaiser and Sutskever in [KS15]. Their particular purpose was to improve the training dynamics of recurrent architectures in the context of learning to execute parallel algorithms. Instead of forcing the residual network blocks to be completely identical, a quadratic loss term is added to promote the closeness of parameters. The network is then gradually 'cooled' into a fully weight-shared network by increasing the weight of this loss term during training. This gives the system more degrees of freedom to explore the loss landscape during the earlier phases of the training, while the end result is still a fully weight-shared network, or synonymously in our case, a recurrent network with a fixed readout time.

Here we explore the applicability of this idea to vision architectures and find that parameter sharing relaxation consistently outperforms direct weight sharing for all network sizes studied here. Even more importantly, using this technique, the performance does not suffer with increased depth. Although this represents an important step in the right direction, it seem that the usefulness of parameter sharing relaxation has clear limits in this particular context and does not give a satisfactory solution for training very high quality recurrent vision architectures. Improving on the above minimal baseline is a relatively low bar for such complex and slow-to-train systems, and our results are mixed when it comes to actually reaping the benefits of recurrence. In line with [Eig+13], even with our current training methodology, we still find that total parameter count is a strong predictor of the performance of our various recurrent and non-recurrent systems. The beneficial effect of recurrence is second order compared to this, and diminishes with increased depth, at least with the optimization techniques at hand.

This motivates our second line of attack: a novel visualization of the activation evolution of deep residual and recurrent architectures. We hope that the insights gained by these studies will help us to gain insights about their working which can in turn lead to improved training and optimization techniques. Also it can guide our architectural choices. One beneficial effect of working with residual networks is that the residual connections enforce a temporal consistency between the activation outputs of adjacent blocks establishing a natural matching between their components. This allows us to put all the activation vectors into the same space and study their behavior by dimension reduction techniques.

2 Related work

Stochastic depth residual networks were introduced recently by Huang et al. [Hua+16]. The idea is to keep only a random subset of the residual blocks for each minibatch and replace the rest by the identity function (or equivalently: zero out the residual activations of the dropped blocks). This means that the trained networks are shallow for each minibatch, but the final network is deep during evaluation. An interesting consequence of this training procedure is that the residual blocks cannot have reliable information about their exact position in the stack. This promotes temporal invariance of their behavior. In other words, a residual block must be capable of usefully transforming its input regardless of its exact temporal position, suggesting that neighboring residual blocks are forced to have similar input-output behavior. The logical conclusion of this reasoning that this represents another form of relaxed weight sharing, which is exactly what we set out to achieve.

The idea that weight-shared deep neural networks can be interpreted as recurrent networks is not new, for example [Eig+13] studies recurrent networks from this view point, but it is not immediately obvious how their findings generalize to residual networks [He+15] and the presence of batch normalization [IS15], as these techniques enabling the training of deeper networks were not available at the time.

Much more recently, Liao and Poggio [LP16] investigated various weight-shared convolutional architectures inspired by residual networks. The architecture in their work that is most directly comparable to ours involves a 3-level residual network, with weight-sharing between the blocks within levels. (Figure 4. (D) in their paper, with results on Figure 6.) Here the number of non-padding residual blocks per level is 3. As we will see, the direct weight sharing approach does not scale to significantly deeper networks that this, and we will employ parameter sharing relaxation to overcome this limitation.

3 Approach

3.1 Parameter sharing relaxation

The goal of the parameter sharing relaxation technique [KS15] is to aid the finding of an optimal weight-shared system by letting the optimization pass through non-weight-shared states. Instead of forcing a set of variables to be completely identical, a loss term is added, promoting closeness. For each variable, the loss term is the squared distance between the variable, and the average of the set. The scalar coefficient of these loss terms is called the relaxation pull.

The network is gradually 'cooled' into a fully weight-shared network by increasing the relaxation pull during training. This gives the system more degrees of freedom to explore the loss landscape during the earlier phases of the training, while the end result is still a fully weight-shared network, or synonymously in our case, a recurrent network with a fixed readout time.

As a final step of the training, the weights are 'consolidated', that is, each is replaced with their average. We observe that the consolidation step not only does not degrade performance, but in a statistically non-significant majority of the cases we have encountered, it improves it. Presumably, the deviations from the average are noise at the final phase of the training, and averaging out these deviations acts as a form of regularization.

An obvious downside of parameter sharing relaxation compared to direct weight sharing is the increased memory requirement. On the other hand, the running time of the two approaches is comparable, as the calculation of the parameter sharing relaxation loss is dominated by the calculation of convolutions. The memory- and running time requirements of the two approaches are identical during evaluation.

In a non-weight-shared residual network, the weight matrices of different convolutional layers are approximately orthogonal with high probability. Thus, with a low relaxation pull at the first phase of training, the weight matrices are approximately orthogonal as well. As the relaxation pull increases, the cosine distance between the relaxedly shared weight matrices goes down close to zero. We have found that when the angle between the weight vectors becomes smaller than degree 1, the consolidation step does not measurably degrade accuracy anymore.

3.2 Stochastic depth

We employ the recent stochastic depth residual network technique by Huang et al. [Hua+16]. During stochastic depth training, for each minibatch we randomly drop a subset of residual blocks, and bypass them with the identity function. The resulting networks are shallow during training, and deep during evaluation.

The probability of a block being switched off is called the death rate. As Huang et al, we use a linearly decaying death rate. That is, the death rate is 0 for the first residual block, and linearly increases to 0.5 until we reach the last residual block. As with the conceptually similar dropout, we have to add a multiplicative term to the output of the residual blocks during evaluation, to compensate for their death rate.

Our original motivation for employing stochastic depth training was our expectation that it facilitates weight sharing by enforcing a 'temporal invariance' property. That is, a residual block must be capable of usefully transforming its input regardless of its exact position in the stack of residual blocks, which suggests that neighboring residual blocks must have similar input-output behavior.[2] Nevertheless, at the medium network depths we have experimented with, stochastic depth training did not change accuracy or training time in our systems in a statistically quantifiable way, neither for the the standard residual networks nor for the weight-shared ones. We give a post hoc explanation for this finding in the next section in our ablation experiments, and highlight some differences between networks trained with- and without stochastic depth.

[2] More accurately, our interest in weight-sharing between residual blocks came from our expectation that stochastic depth training promotes such similarity of input-output behavior.

Table 1: Comparing parameter counts and test error for different weight sharing methods. Sharing between the blocks of the middle level. 18 residual blocks per level.

Method	Parameter count	Test error
Non-shared	1732010	6.67
Relaxed	1437098	6.67
Raw	1437098	7.78
Minimal	1434026	7.23

4 Experiments

Three level network architecture. We use the same network architecture in all our experiments as the one employed by [Hua+16] in their CIFAR-10 experiments. This architecture consists of three levels, each containing stacked residual blocks. We refer to these three levels as *lower*, *middle* and *upper* levels, the lower being the closest to the input. The structure of a residual block is conv-batchnorm-ReLU-conv-batchnorm, followed by the element-wise addition and a ReLu nonlinearity.

The three levels differ only in the number of convolutional filters and feature map sizes. The number of filters are 16, 32 and 64, respectively. We use zero padding to match dimensions between levels of different sizes. Padding blocks are excluded from weight sharing.

At the top of the network there is a non-recurrent part (post-net, in the terminology of [LP16]). This consists of a global average pooling and a fully connected layer. As we have measured, and our representation trajectory visualizations will suggest, a simpler post-net would already be sufficient: at the last residual block, the representation already linearly separates the image classes.

Training on CIFAR-10. The CIFAR-10 dataset [Kri09] consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. We apply a very mild form of input augmentation utilizing only horizontal flipping and per feature normalization.

For a better comparison we use the same training setup as in [Hua+16]. The models are trained for 500 epochs with SGD, with a mini-batch size of 128. The initial learning rate is 0.1, and is divided by a factor of 10 at epochs 250 and 375. The weight decay is set to 10^{-4}. We use a momentum of 0.9 and Nesterov momentum with dampening of 0.

With relaxed weight sharing, the relaxation pull is also initialized at 10^{-4} and is gradually increased to 10^{-1} following an exponential function.

4.1 Parameter sharing at the middle level

In this experiment, we turn on relaxed weight sharing between the blocks of the middle level of a residual network. Like in [LP16], only convolutional layers are shared, padding blocks are not shared, and no sharing takes place between the two convolutional layers of a single block.

We compare the performance of this *relaxed* model to several baselines:

1. The *non-shared* baseline is the model with the same topology, without weight sharing.
2. The *minimal* baseline is the model that has only a single non-padding block at the middle level.
3. The *raw* baseline is the model with the same topology as the *non-shared* baseline, but with traditional weight sharing turned on between middle level blocks.

4.2 Parameter sharing on all levels

The minor difference between the parameter count of the shared and minimal models is due to the non-shared batch normalization parameters.

The error curves all show accuracy degradation during the middle part of training. This phenomenon is characteristic of all high accuracy residual networks, and especially pronounced for stochastic

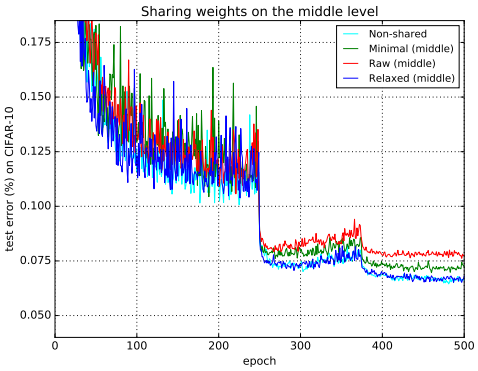


Figure 1: Test error on CIFAR-10 comparing relaxed weight sharing on the middle level to different baselines.

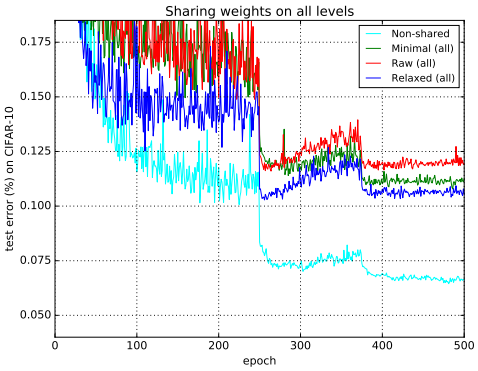


Figure 2: Test error on CIFAR-10 comparing relaxed weight sharing on all levels to different baselines.

depth networks. To the best of our knowledge, the exact cause of this phenomenon is unknown, and it is not necessarily a sign of detrimental overfitting. Expectedly, the relaxed model shows the highest accuracy drop, as in that model, there is an explicit trade-off between weight sharing and accuracy, due to gradually increased relaxation pull.

The accuracy of the recurrent models stays below the non-shared model significantly, which is to be expected, considering the smaller parameter count. The only minor exception is the relaxed model with weight sharing only at the middle level, which has similar performance to a non-shared model with a somewhat larger parameter count.

While at block count 3 direct weight sharing confidently outperforms the minimal baseline (Table 3), this is not the case at block count 18 (Tables 1 and 2). Relaxed weight sharing outperforms the minimal baseline in each of our experiments.

4.3 Effect of readout time

Importance of stochastic depth. Albeit the 'temporal invariance' phenomenon arising in networks with stochastic depth was our primary motivation to investigate weight sharing in residual networks, our subsequent experiments showed that training with or without the stochastic depth technique does not influence the accuracy of the weight shared network. However, stochastic depth has an impact on how the information is organized across residual blocks in a trained network. Accordingly, we observe that applying weight sharing techniques result in networks that show different behavior regarding readout time and accuracy.

Table 2: Comparing parameter counts and test error for different weight sharing methods. Sharing on all three levels. 18 residual blocks per level.

Method	Parameter count	Test error
Non-shared	1732010	6.67
Relaxed	179114	10.63
Raw	179114	12.04
Minimal	168266	11.18

Table 3: Comparing parameter counts and test error for different weight sharing methods. Sharing on all three levels. 3 residual blocks per level.

Method	Parameter count	Test error
Non-shared	363146	8.26
Relaxed	169610	10.06
Raw	169610	10.29
Minimal	168266	11.25

Ablation experiments. In our ablation experiments we switch off individual and consecutive residual blocks to gain a better insight into the importance of specific residual blocks and to study the effect of readout time.

We conduct ablation experiments in the following ways:

1. In the *top-to-bottom ablation* experiment we switch off residual blocks one after another starting from the *upper* part of the middle level, thus emulate decreasing readout times.
2. In the *bottom-to-top ablation* experiment we switch off residual blocks one after another starting from the *lower* part of the middle level.
3. In the *single ablation* experiment we switch off *individual* residual blocks independently.

We run all of our experiments using levels with 18 residual blocks. When utilized, stochastic depth training is used with linearly decaying death rates. Figure-3 presents the results of our ablation experiments performed on the middle level.

Bottom-to-top ablation experiments clearly show that networks without weight sharing are much more sensitive to the elimination of the lower residual blocks. Also, stochastic depth training averages out individual contributions of residual blocks, resulting in models requiring larger readout time for the same accuracy.

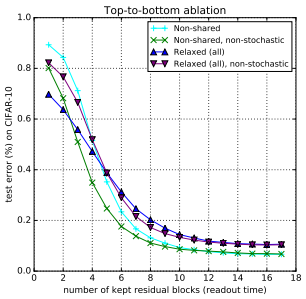
The effect of 'temporal invariance' forced by both weight sharing and stochastic depth training result in smoother decaying curves.

In stochastic depth training lower residual blocks take part in the training with larger probability, resulting in networks where the lower blocks are forced to contribute more to the overall representational power of a level. This force opposes the effect of relaxed weight sharing.

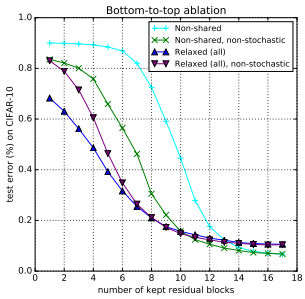
4.4 Visualizing temporal patterns of neural activations

Within a single level, the internal representations of the network live within a common vector space. This means that we can talk about the trajectories of these representations, as the addition of the appropriate residuals as deltas repeatedly takes place. The shape and relative position of these trajectories provides insights about the internal workings of the residual network. This is especially true for weight-shared networks, where the residual only depends on the current position in the representation space, and not on the readout time, which makes these networks naturally interpretable as first-order difference equations.

Fixing a random set of 100 training examples, we present a two-dimensional representation of their trajectories. In the *unnormalized* version, a PCA is applied to the 100 times $N + 1$ points on the trajectories, where N is the block count of the level. Each dot corresponds to the input or output of a



(a) *Top-to-bottom* ablation



(b) *Bottom-to-top* ablation

Figure 3: Ablation experiments on the middle level.

block (with the identity mapping added). Adjacent blocks are connected with an arrow, and dots are colored according to their CIFAR class.

This visualization immediately shows that there is a 'drift' taking place, with residuals having a constant bias that does not visibly contribute to the representational power of their output.[3] Hence we also provide a *normalized* version, where we subtract the sample-average activations, discarding the drift.

Both the non-shared and the shared networks have relatively continuous trajectories. There are no immediately visible individual bursts of neural activations that could signal the recognition of some specific visual pattern. (The same is true for non-stochastic residual networks, we omit their visualizations here.) Zooming into the details of the weight-shared and non-weight-shared network visualizations,[4] we can see that the weight-shared networks have significantly smoother trajectories. Inspecting the CIFAR class of trajectory endpoints of the last level, we can see that classes are linearly separable even before the top non-residual layers. Each of the three levels has its own distinct large-scale pattern, and this pattern is very robust to repeated training runs, and somewhat robust to the choice of weight-sharing versus non-weight-sharing.

5 Discussion

It's tempting to interpret the above results by conjecturing that our recurrent network learns a kind of universal computation. Under this interpretation, a single residual block can be seen as implementing a generic computational step, similar to the transition rule of a Turing machine. In fact, that is exactly the interpretation favored by Liao and Poggio, who write:

> A radical conjecture would be: the effectiveness of most of the deep feedforward neural networks, including but not limited to ResNet, can be attributed to their ability to approximate recurrent computations that are prevalent in most tasks with larger t than shallow feed-forward networks.

Although this kind of learned emergent behavior would be very interesting from a theoretical perspective, our current preferred interpretation of our observations is somewhat different: We believe that the emphasis should be on representations (activations), rather than computation. We conjecture that what happens in these recurrent networks is a 'holographic' compression of representations.

As a stylized example, we can imagine a deep non-recurrent network that has mouth- and eye-sensitive neurons at the 5th layer, and face-sensitive neurons at the 6th layer. For this setup to work, the output of the 5th layer must have an interpretation in terms of mouth and eye patterns. In contrast to this, in a recurrent network the interpretations must be shared across all layers. But that's not necessarily a strict limitation: As an unrealistic existence proof, we can image a single neuron whose output is

[3] In principle, the between-block ReLUs break the translation invariance, but that appears to be a weak effect, and as recently shown by [He+16], between-block nonlinearities are in no way a necessary part of residual networks.

[4] Only visible in electronic form.

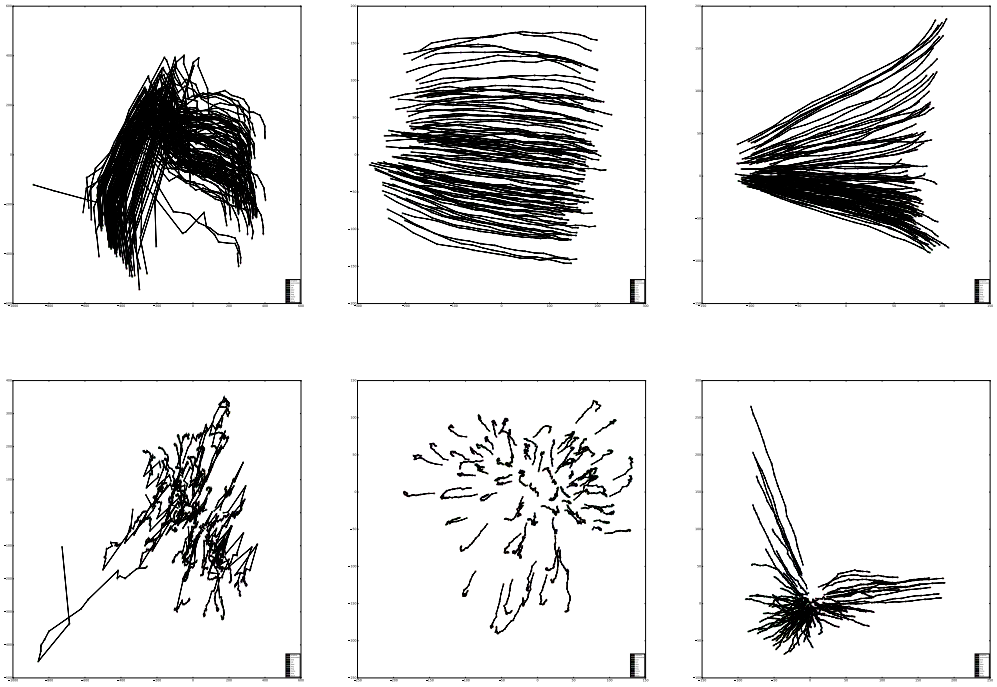


Figure 4: Activation trajectories for lower, middle and upper level of a non-weight-shared stochastic depth residual network. First row: unnormalized, second row: normalized.

incremented in each recurrent time step, and the relevant neurons follow mouth-sensitive behavior if the output of the timer neuron is 5, and follow face-sensitive behavior if the output of the timer neuron is 6. Needless to say, the hypothetical timer neuron is not the preferred mechanism of recurrent networks to achieve shared representations. Rather, we conjecture that our networks learn to roughly and fuzzily partition the representation space into subsets, and on each of these subsets, the block's behaviour is optimized for a narrow band of preferred readout time.

Future work is required to turn these alternative interpretations into falsifiable predictions. One possible avenue towards this goal is understanding the probability distribution of activations of the recurrent network for different values of time t.

References

[Kri09] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[Eig+13] David Eigen et al. "Understanding Deep Architectures using a Recursive Convolutional Network". In: *CoRR* abs/1312.1847 (2013). URL: `http://arxiv.org/abs/1312.1847`.

[He+15] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). URL: `http://arxiv.org/abs/1512.03385`.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. 2015, pp. 448–456. URL: `http://jmlr.org/proceedings/papers/v37/ioffe15.html`.

[KS15] Lukasz Kaiser and Ilya Sutskever. "Neural GPUs Learn Algorithms". In: *CoRR* abs/1511.08228 (2015). URL: `http://arxiv.org/abs/1511.08228`.

[He+16] Kaiming He et al. "Identity Mappings in Deep Residual Networks". In: *arXiv preprint arXiv:1603.05027* (2016).

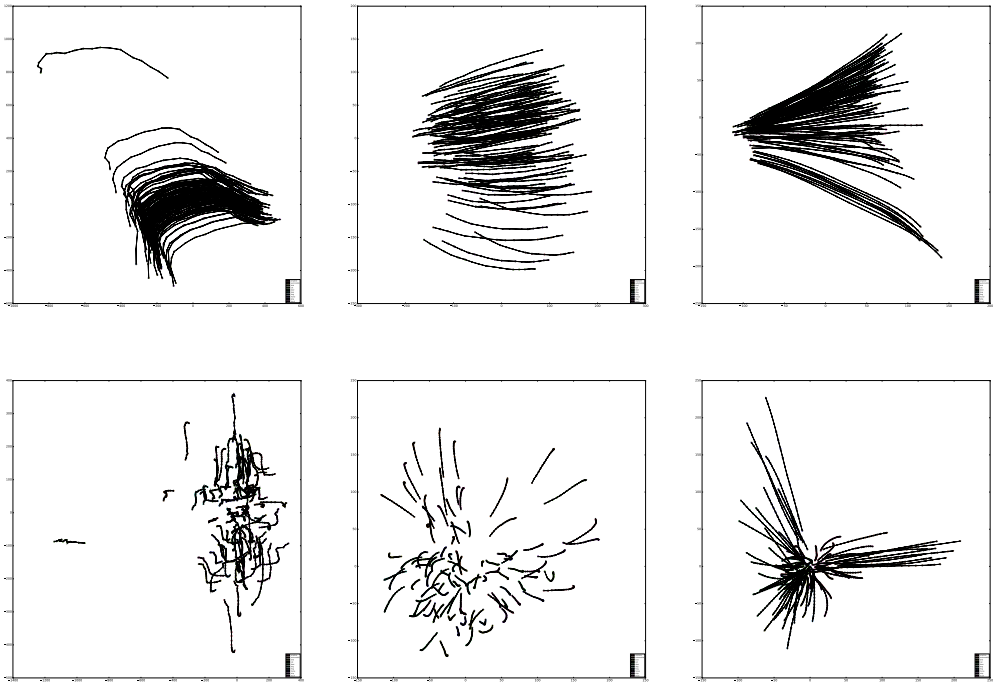


Figure 5: Activation trajectories for lower, middle and upper level of a relaxedly weight-shared residual network. First row: unnormalized, second row: normalized.

[Hua+16] Gao Huang et al. “Deep Networks with Stochastic Depth”. In: *CoRR* abs/1603.09382 (2016). URL: http://arxiv.org/abs/1603.09382.

[LP16] Q. Liao and T. Poggio. “Bridging the Gaps Between Residual Learning, Recurrent Neural Networks and Visual Cortex”. In: *ArXiv e-prints* (Apr. 2016). arXiv: 1604.03640 [cs.LG]. URL: http://arxiv.org/abs/1604.03640.