Supplementary Materials

A. Detailed ConstNet Design and Considerations

In this section, we provide technical details for ConstNet architectures, and explain the motivations behind each architecture decision. The code used for this experiment is available in: https://github.com/yanivbl6/BeyondSigProp.

Zero initialization: To ensure that only the skip connections are taken into account for signal propagation, we initialize all weights in computation blocks bypassed by skip connection to zero. In gradient descent, two sequential blocks initialized to zero will not allow gradient propagation: the update of each weights will remain zero as long as the other operations are 0. To counter this, we replace the commonly used ResNet block which contains two chained operations, and use a single operation block instead, as seen in figure 2.

We note that the forward signal propagation can also be maintained by only initializing the last chained block to zero, as was done in fixup (Zhang et al., 2019), but this method does not align with our goal of examining initializations with identical features. Additionally, we adjust the position of the nonlinearities to allow a linear path for the backward signal, moving from the 'output' to the input.

Forcing identical features: For our goal of identical features to be reached, we initialize the first convolutional layer's filters $F = \mathrm{Mat}(W_{ij}) \in \mathbb{R}^{3\times 3}$ to $F_{kl} = \frac{1}{n_{\mathrm{in}}}\delta_{k1}\delta_{l1}, k, l \in 0, 1, 2$, where n_{in} is the number of input channels, so all there channels are being averaged. As we advance in a residual network from input to output, it is commonplace to increase the number of channels (a.k.a widening) while trimming their spatial dimensions (using strides of size larger than 1). The widening of a neural network does not allow simply using skips, as the expected output of the computational block has additional channels, and 1x1 convolution operations are used instead, so each output channel is a linear combination of the input channels. We initialize these filters to the constant $\frac{1}{n_{\mathrm{in}}}$, so they perform as an 'averaging operations' as well. Generally speaking, repeated averaging operations do not conserve signal propagation, and we take advantage of the fact that the number of those operations in the network is limited, and that only the first averaging operation causes a loss of information. One more possible issue is the usage of stride, as it may disturb the signal propagation. However, we show, in section 3.1, that under the assumption of spatial invariance, stride convolutions do not change the angles between features in expectation. Finally, we initialize the final fully connected layer to zero.

Depth and width selection: As mentioned, our channel sizes in the networks are based on Wide-Resnet, which has a similar number of parameters and operations. Since the number of channels is large, we expect the effects of feature diversity to be at a peak. Specifically, a randomly initialized Wide Res-Net has high number of features: the number of channels in the layers of the network increases from 16 to 640 at the final residual layers. A noteworthy design detail in ConstNet is that the activation in each skip is located before the convolution operation. This was done in order to have a non-linearity between the initial convolution and the following skippable convolution.

Batch Normalization: While the fixup initialization allow training neural networks without batch normalization, our default setup utilize batch-norm operations, positioned before the nonlinearities, as a tool for regularization. To avoid using batchnorm on the linear path, a scalar scale operator is added (Zhang et al., 2019) and a Mixup (Zhang et al., 2017) regularization has to be applied to prevent generalization loss. This method therefore requires an hyper-parameter search over the values of the Mixup parameter α and the learning rate. Notwithstanding the results (Yang et al., 2019), who has shown that batch-normalization is incompatible with our goal of maintaining signal propagation, batch-normalization of the spatial dimensions (per channel) will not have an effect over the channels in an initialized network, since the first Batch-norm operation is the only one that will have an effect, at the exact point of initialization.

Hyper-parameters: All runs with ConstNet, and the baseline runs with Wide-ResNet, were done for a model with 12 skip-able layers. We used SGD with momentum (0.9), batch size 128×2 GPUs, and with cross-entropy loss over the CIFAR-10 dataset. Unless specified otherwise, we trained the network for 200 epochs, with a learning rate decay at epochs $60, 120, 160$. Unless mentioned otherwise, the default learning rate was 0.03, dropout and mixup were disabled, Batchnorm was used and cudNN was non-deterministic. All parameters where chosen based on the default Wide-ResNet setup, with the exception of the learning rate, which was picked in consideration of network pruning results (Frankle & Carbin, 2018) (Lottery ticket method was shown to only work for residual networks with low learning rates). Results were averaged over 6 seeds. The number of parameters in the model was ~ 17M, and data was represented with 32-bit floating point.

B. Depth-Independent Signal Propagation in Convolutional Networks with Random Initialiation

Stability of backward features depends on the conditioning of the Jacobian between network states $J^{(\ell)} = \frac{\partial \tilde{\alpha}^{(\ell)}}{\partial \tilde{\alpha}^{(\ell-1)}}$, since the backward features take the form $\beta^{(\ell)} = \frac{\partial \mathscr{L}}{\partial \tilde{\alpha}^{(L-1)}} \frac{\partial \tilde{\alpha}^{(L-1)}}{\partial \tilde{\alpha}^{(\ell)}}$. In essence - stability of the backward features is ensured if $||J^{(\ell)}|| = 1$ and the variance of the singular values of $J^{(\ell)}$ is zero, where $||\cdot||$ denotes operator norm. Works that analyze signal propagation at the infinite width limit define *dynamical isometry* conditions that ensure the singular values of $J^{(\ell)}$ depend weakly on depth and all have absolute value close to 1.

In this section we briefly describe how to construct a convolutional network with random initialization and depth-independent forward and backward signal propagation (dynamical isometry). We combine elements of the approaches in (Xiao et al., 2018; Burkholz & Dubatovka, 2019). For simplicity consider a network that is given by a composition of convolutions, and we assume periodic boundary conditions and that the number of channels n is constant throughout. Given an input tensor $x \in \mathbb{R}^{S \times n/2}$ with S denoting a set of spatial dimensions, we define an augmented input

$$\overline{x} = \begin{pmatrix} x & 0 \end{pmatrix} \in \mathbb{R}^{S \times n}$$

If we assume the kernel is of size $K = (2k+1) \times (2k+1)$ and index these coordinates from $-k$ to k, we define a kernel by

$$\Delta \in \mathbb{R}^{2k+1 \times 2k+1}, \Delta_{ij} = \delta_{i0}\delta_{j0}. \tag{7}$$

This definition can be generalized trivially to settings where K has order different from 2. Initializing the biases at 0, the parameters of the ℓ-th convolutional layer are given by $W^{(\ell)} \in \mathbb{R}^{K \times n \times n}$, which we factorize as

$$W^{(\ell)} = \Delta \otimes \begin{pmatrix} U^{(\ell)} & -U^{(\ell)} \\ -U^{(\ell)} & U^{(\ell)} \end{pmatrix} \tag{8}$$

where $U^{(\ell)} \in \mathbb{R}^{n/2 \times n/2}$ is an orthogonal matrix drawn from a uniform distribution over $O(n/2)$ and $\Delta^{(\ell)} \in \mathbb{R}^K$ is the kernel defined above (we assume n is even). As before, we denote $\tilde{\alpha}^{(1)}_{\gamma,i} = \left[W^{(1)} \hat{*} \overline{x}\right]_{\gamma i} = \sum_{\kappa \in K} \sum_{j=1}^{n} W^{(1)}_{\kappa ij} x_{\gamma+\kappa,j}$. The pre-activations at a spatial location γ, arranged in a column vector $\left[W^{(1)} \hat{*} \overline{x}\right]^T_\gamma$, are thus given by

$$\begin{aligned}
\left[W^{(1)} \hat{*} \overline{x}\right]^T_\gamma &= \sum_{\kappa \in K} \Delta_\kappa \begin{pmatrix} U^{(1)} & -U^{(1)} \\ -U^{(1)} & U^{(1)} \end{pmatrix} \overline{x}^T_{\kappa+\gamma} \\
&= \begin{pmatrix} U^{(1)} & -U^{(1)} \\ -U^{(1)} & U^{(1)} \end{pmatrix} \overline{x}^T_\gamma = \begin{pmatrix} U^{(1)} & -U^{(1)} \\ -U^{(1)} & U^{(1)} \end{pmatrix} \begin{pmatrix} x^T_\gamma \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} U^{(1)} x^T_\gamma \\ -U^{(1)} x^T_\gamma \end{pmatrix}.
\end{aligned}$$

Applying a ReLU nonlinearity and another convolutional layer gives

$$\begin{aligned}
&\left[W^{(2)} \hat{*} \phi\left(W^{(1)} \hat{*} \overline{x}\right)\right]^T_\gamma \\
&= \sum_{\kappa \in K} \Delta_\kappa \begin{pmatrix} U^{(2)} & -U^{(2)} \\ -U^{(2)} & U^{(2)} \end{pmatrix} \phi\left(W^{(1)} \hat{*} \overline{x}\right)^T_{\kappa+\gamma} \\
&= \begin{pmatrix} U^{(2)} & -U^{(2)} \\ -U^{(2)} & U^{(2)} \end{pmatrix} \phi\left(\begin{pmatrix} U^{(1)} x^T_\gamma \\ -U^{(1)} x^T_\gamma \end{pmatrix}\right) \\
&= \begin{pmatrix} U^{(2)} & -U^{(2)} \\ -U^{(2)} & U^{(2)} \end{pmatrix} \begin{pmatrix} U^{(1)} x^T_\gamma \circ \left[U^{(1)} x^T_\gamma > 0\right] \\ -U^{(1)} x^T_\gamma \circ \left[U^{(1)} x^T_\gamma < 0\right] \end{pmatrix} \\
&= \begin{pmatrix} U^{(2)}\left(U^{(1)} x^T_\gamma \circ \left[U^{(1)} x^T_\gamma > 0\right] + U^{(1)} x^T_\gamma \circ \left[U^{(1)} x^T_\gamma < 0\right]\right) \\ -U^{(2)}\left(U^{(1)} x^T_\gamma \circ \left[U^{(1)} x^T_\gamma > 0\right] + U^{(1)} x^T_\gamma \circ \left[U^{(1)} x^T_\gamma < 0\right]\right) \end{pmatrix}
\end{aligned}$$

$$= \begin{pmatrix} U^{(2)}U^{(1)}x_\gamma^T \\ -U^{(2)}U^{(1)}x_\gamma^T \end{pmatrix}.$$

We thus obtain that for any two layers the pre-activations are simply related by a rotation:

$$\left(\tilde{\alpha}_\gamma^{(\ell+1)}\right)^T = \begin{pmatrix} U^{(\ell+1)} & 0 \\ 0 & U^{(\ell+1)} \end{pmatrix} \left(\tilde{\alpha}_\gamma^{(\ell)}\right)^T. \tag{9}$$

This preserves both norms and angles, and thus the covariance between pre-activations is invariant of depth, meaning

$$\left\langle \tilde{\alpha}_\gamma^{(\ell)}(x), \tilde{\alpha}_{\gamma'}^{(\ell)}(x') \right\rangle = \left\langle \tilde{\alpha}_\gamma^{(1)}(x), \tilde{\alpha}_{\gamma'}^{(1)}(x') \right\rangle.$$

Note that this holds surely, and not only in expectation over the weights as in the main text. As a result, the covariance between the hidden states themselves is also independent of depth.

We now consider backwards signal propagation. Since at a spatial location γ the pre-activation $\left(\tilde{\alpha}_\gamma^{(\ell)}\right)^T$ is a concatenation of two identical vectors with opposite sign, there are only $n/2$ independent degrees of freedom. The backward features are thus given by

$$\beta_{\eta j}^{(\ell)}(x) = \frac{\partial \mathscr{L}}{\partial \tilde{\alpha}_{\eta j}^{(\ell)}(x)} = \sum_{\gamma \in S} \sum_{i=1}^{n/2} \frac{\partial \mathscr{L}}{\partial \tilde{\alpha}_{\gamma i}^{(\ell+1)}(x)} \frac{\partial \tilde{\alpha}_{\gamma i}^{(\ell+1)}(x)}{\partial \tilde{\alpha}_{\eta j}^{(\ell)}(x)}$$

$$= \sum_{\gamma \in S} \sum_{i=1}^{n/2} \beta_{\gamma i}^{(\ell+1)}(x) \frac{\partial \tilde{\alpha}_{\gamma i}^{(\ell+1)}(x)}{\partial \tilde{\alpha}_{\eta j}^{(\ell)}(x)}$$

$$= \beta^{(L-1)}(x) J_{n/2}^{(L-1)} J_{n/2}^{(L-2)} \cdots \frac{\partial \tilde{\alpha}^{(\ell+1)}(x)}{\partial \tilde{\alpha}_{\eta j}^{(\ell)}(x)}$$

where

$$J_{n/2}^{(\ell)} \in \mathbb{R}^{S \times S \times n/2 \times n/2}, \left[J_{n/2}^{(\ell)}\right]_{\gamma \eta i j} = \frac{\partial \tilde{\alpha}_{\gamma i}^{(\ell)}(x)}{\partial \tilde{\alpha}_{\eta j}^{(\ell-1)}(x)}$$

(without this structure in the pre-activations the Jacobians would be defined as matrices in $\mathbb{R}^{S \times S \times n \times n}$). Since $\frac{\partial \tilde{\alpha}_{\gamma i}^{(\ell)}}{\partial \tilde{\alpha}_{\eta j}^{(\ell-1)}} = W_{\eta-\gamma, ij}^{(\ell)} \dot{\phi}(\tilde{\alpha}^{(\ell-1)})_{\eta, j} \mathbb{1}_{\eta-\gamma \in K}$, plugging in the form of the weight tensor from equation 8 gives

$$\frac{\partial \tilde{\alpha}_{\gamma i}^{(\ell)}}{\partial \tilde{\alpha}_{\eta j}^{(\ell-1)}} = \delta_{\gamma \eta} U_{ij}^{(\ell)} \dot{\phi}(\tilde{\alpha}^{(\ell-1)})_{\eta, j}$$

hence over the spatial dimensions $S \times S$, the tensor $J_{n/2}^{(\ell)}$ is simply a delta function, and since at a given spatial location the pre-activations are related according to eq. 9 we obtain

$$J_{n/2}^{(\ell)} = I_{S \times S} \otimes U^{(\ell)}$$

where $I_{S \times S}$ is shorthand for a product of delta functions over every spatial dimension. Since the $U^{(\ell)}$ are orthogonal, $J_{n/2}^{(\ell)}$ obeys the dynamical isometry conditions (all its singular values over the non-trivial dimensions have magnitude 1). Thus norms and angles between backward features are also independent of depth with this initialization.

Note that both forwards and backward features are stable surely, and not just in expectation over the weights which is the usual form of dynamical isometry results that study networks at the infinite width limit (which is predictive of the behavior of reasonably wide networks)(Schoenholz et al., 2016; Pennington et al., 2017). Therefore, a network initialized in this way will exhibit stable signal propagation at any width.

C. Depth-Independent Signal Propagation in ConstNet

Claim. *Let f be an L-layer ConstNet function as in eq. 4 and denote the scalar loss by $\mathscr{L}$. Then for any $0 \le \ell \le L-1$ we have*

$$\frac{\langle \beta^{(\ell)}(x), \beta^{(\ell)}(x') \rangle}{n_\ell} = \frac{\langle \beta^{(L-1)}(x), \beta^{(L-1)}(x') \rangle}{n_{L-1}}$$

$$\frac{\partial \mathscr{L}(x)}{\partial W^{(\ell)}_{\kappa ij}} = C_{ij} \cos(\theta_{\kappa,\ell})$$

$$\frac{\partial \mathscr{L}(x)}{\partial b^{(\ell)}_{i}} = C'_i$$

where $C_{ij}, C'_i > 0$ are constants that are independent of L (but depend on the functions P, g in the definition of the ConstNet function and on $\mathscr{L}$). $\theta_{\kappa,\ell}$ are constants that can depend on L.

Additionally, for translation invariant inputs, if we denote the spatial dimensions at layer ℓ by $S^{(\ell)}$ we have for any $\gamma, \gamma' \in S^{(\ell)}$

$$\tilde{\alpha}^{(\ell)}_{\gamma i}(x) \overset{d}{=} \tilde{\alpha}^{(0)}_{\gamma' 1}(x)$$

$$\frac{\langle \tilde{\alpha}^{(\ell)}(x), \tilde{\alpha}^{(\ell)}(x') \rangle}{\left|S^{(\ell)}\right| n_\ell} \overset{d}{=} \frac{\langle \tilde{\alpha}^{(0)}(x), \tilde{\alpha}^{(0)}(x') \rangle}{\left|S^{(0)}\right| n_0}$$

Proof. We assume that at the first layer we have

$$\tilde{\alpha}^{(0)}_{\gamma i} = \tilde{\alpha}^{(0)}_{\gamma j}$$

where $\tilde{\alpha}^{(0)}_{\gamma}$ is obtained by some affine transformation of the data. At any layer ℓ, at initialization either $\alpha^{(\ell)} = \mathrm{CB}_{0,0}(\alpha^{(\ell-1)})$ or $\alpha^{(\ell)} = \mathrm{WB}_{0,0,s}(\alpha^{(\ell-1)})$. In the former case we have

$$\tilde{\alpha}^{(\ell)}_{\gamma j}(x) = \tilde{\alpha}^{(\ell-1)}_{\gamma j}(x) = \tilde{\alpha}^{(\ell-1)}_{\gamma 1}(x)$$

for all γ, j, while in the latter case we have

$$\tilde{\alpha}^{(\ell)}_{\gamma j}(x) = \frac{1}{n}\sum_{j=1}^{n} \tilde{\alpha}^{(\ell-1)}_{\gamma s, j}(x) = \tilde{\alpha}^{(\ell-1)}_{\gamma s, 1}(x).$$

If we assume that there are p narrowing layers between 1 and ℓ, repeated application of the above equations gives

$$\tilde{\alpha}^{(\ell)}_{\gamma i}(x) = \tilde{\alpha}^{(0)}_{\gamma s^p, 1}(x).$$

If we denote the spatial dimensions at layer ℓ by $S^{(\ell)}$, since tranlation invariance of the inputs implies the same invariance of the pre-activations, we have

$$\tilde{\alpha}^{(\ell)}_{\gamma i}(x) = \tilde{\alpha}^{(0)}_{\gamma s^p, i}(x) = \tilde{\alpha}^{(0)}_{\gamma s^p, 1}(x) \overset{d}{=} \tilde{\alpha}^{(0)}_{\gamma' 1}(x)$$

for any γ, γ' and

$$\frac{\langle \tilde{\alpha}^{(\ell)}(x), \tilde{\alpha}^{(\ell)}(x') \rangle}{\left|S^{(\ell)}\right| n_\ell} = \frac{1}{\left|S^{(\ell)}\right| n_\ell} \sum_{\gamma \in S^{(\ell)}} \sum_{i=1}^{n_\ell} \tilde{\alpha}^{(\ell)}_{\gamma i}(x) \tilde{\alpha}^{(\ell)}_{\gamma i}(x')$$

$$= \frac{1}{\left|S^{(\ell)}\right|} \sum_{\gamma \in S^{(\ell)}} \tilde{\alpha}^{(0)}_{\gamma s, 1}(x) \tilde{\alpha}^{(0)}_{\gamma s, 1}(x')$$

$$\overset{d}{=} \frac{1}{\left|S^{(0)}\right|} \sum_{\gamma \in S^{(0)}} \tilde{\alpha}^{(0)}_{\gamma, 1}(x) \tilde{\alpha}^{(0)}_{\gamma, 1}(x') = \frac{\langle \tilde{\alpha}^{(0)}(x), \tilde{\alpha}^{(0)}(x') \rangle}{\left|S^{(0)}\right| n_0}.$$

We now consider a layer ℓ such that there are p narrowing layers between 1 and ℓ and q narrowing layers between ℓ and $L-1$. If we choose γ such that γ / s^q is a vector of integers (which we denote by $\gamma / s^q \in \mathbb{Z}^d$), we have

$$\beta^{(\ell)}_{\gamma j}(x) = \frac{\partial \mathscr{L}}{\partial \tilde{\alpha}^{(\ell)}_{\gamma j}(x)} = \frac{\partial \mathscr{L}}{\partial \tilde{\alpha}^{(L-1)}_{\gamma / s^q, 1}(x)} = \beta^{(L-1)}_{\gamma / s^q, 1}(x).$$

from which it follows that

$$\frac{\left\langle \beta^{(\ell)}(x), \beta^{(\ell)}(x') \right\rangle}{n_\ell} = \frac{1}{n_\ell} \sum_{\gamma \in S^{(\ell)}} \sum_{i=1}^{n_\ell} \beta^{(\ell)}_{\gamma i}(x) \beta^{(\ell)}_{\gamma i}(x')$$

$$= \sum_{\gamma \in S^{(\ell)}, \gamma/s^q \in \mathbb{Z}^d} \beta^{(L-1)}_{\gamma/s^q 1}(x) \beta^{(L-1)}_{\gamma/s^q 1}(x') = \sum_{\gamma \in S^{(L-1)}} \beta^{(L-1)}_{\gamma 1}(x) \beta^{(L-1)}_{\gamma 1}(x')$$

$$= \frac{\left\langle \beta^{(L-1)}(x), \beta^{(L-1)}(x') \right\rangle}{n_{L-1}}.$$

Additionally,

$$\frac{\partial \mathscr{L}}{\partial \tilde{\alpha}^{(L-1)}_{\gamma/s^q,1}(x)} = \frac{\partial \mathscr{L}}{\partial f(\tilde{\alpha}^{(L-1)}_{\gamma/s^q,1}(x))} \frac{\partial f}{\partial \tilde{\alpha}^{(L-1)}_{\gamma/s^q,1}(x)}$$

$$= \frac{\partial \mathscr{L}}{\partial P(\tilde{\alpha}^{(L-1)}_{\gamma/s^q,1}(x))} \frac{\partial P(\tilde{\alpha}^{(L-1)}_{\gamma/s^q,1}(x))}{\partial \tilde{\alpha}^{(L-1)}_{\gamma/s^q,1}(x)}$$

$$= \dot{\mathscr{L}}\left(P(\tilde{\alpha}^{(0)}_{\gamma s^p,1}(x))\right) \dot{P}(\tilde{\alpha}^{(0)}_{\gamma s^p,1}(x)).$$

Note that if γ/s^q is not a vector of integers, the location γ will not contribute to the loss and hence $\beta^{(\ell)}_{\gamma j}(x) = 0$. We have thus shown that the non-zero elements of $\alpha^{(\ell)}_i(x), \beta^{(\ell)}_i(x)$ can be written in terms of quantities that are independent of depth.

The gradients are given by

$$\frac{\partial \mathscr{L}(x)}{\partial W^{(\ell)}_{\kappa i j}} = \sum_{\gamma \in S^{(\ell)}} \beta^{(\ell)}_{\gamma i}(x) \alpha^{(\ell-1)}_{\gamma+\kappa, j}(x)$$

$$= \sum_{\gamma \in S^{(\ell)}, \gamma/s^q \in \mathbb{Z}^d} \dot{\mathscr{L}}\left(P(\tilde{\alpha}^{(0)}_{\gamma s^p,1}(x))\right) \dot{P}(\tilde{\alpha}^{(0)}_{\gamma s^p,1}(x)) * g\left(\tilde{\alpha}^{(0)}_{(\gamma+\kappa)s^p,1}(x)\right)$$

$$\frac{\partial \mathscr{L}(x)}{\partial b^{(\ell)}_i} = \sum_{\gamma \in S^{(\ell)}, \gamma/s^q \in S^{(L-1)}} \beta^{(\ell)}_{\gamma i}(x).$$

Note that the number of terms in this summation is $\left|S^{(L-1)}\right|$ and is thus independent of ℓ. The gradients depend on ℓ only through the relative shift between $\alpha^{(\ell-1)}_j(x)$ and $\beta^{(\ell)}_i(x)$, which is κs^p, and is thus bounded by product of the norms of these vectors which are independent of ℓ. It follows that we can write

$$\frac{\partial \mathscr{L}(x)}{\partial W^{(\ell)}_{\kappa i j}} = C_{ij} \cos(\theta_{\kappa,\ell}), \frac{\partial \mathscr{L}(x)}{\partial b^{(\ell)}_i} = C'_i.$$

Where C_{ij}, C'_i are independent of depth. □

Note that if C_{ij}, C'_i are equal to 0, which can be ensured with an appropriate choice of P, the gradient at initialization for all layers aside from the last will be 0 as well. This result ensures that during early stages of training, once gradients take non-zero values, they will behave in a stable manner even if the network is very deep. There is also no reason to expect $\cos(\theta_{\kappa,\ell})$ to decay with depth and lead to vanishing gradients.

If batch-norm parameters are also trained, or the map from the input to $\tilde{\alpha}^{(0)}$ is parametrized by trainable parameters, their gradients will also be insensitive to depth due to the insensitivity of the forward and backward features. Batch-norm will have no effect at initialization due to the symmetries of $\tilde{\alpha}^{(0)}$. Note also that for a reasonable batch size, if $\tilde{\alpha}^{(0)}(x)$ applies a convolution to x with a $W = \frac{1}{n_d} \mathbb{1}_{n_0} \mathbb{1}^T_{n_d} \times \Delta$ where Δ is the delta kernel defined in Appendix C, we expect the angles between the inputs averaged over the input channels to be preserved in the sense $\angle(\tilde{\alpha}^{(0)}(x), \tilde{\alpha}^{(0)}(x')) \approx \angle(\sum_\gamma x_\gamma, \sum_{\gamma'} x'_\gamma)$.

We also note that if the number of features was held constant at all layers (as in (Xiao et al., 2018) for instance) the norm of the backward features would be preserved as well, and the magnitude of the gradient elements would be completely independent of depth. The dependence is only a result of the widening layers and not of the ConstNet blocks themselves.

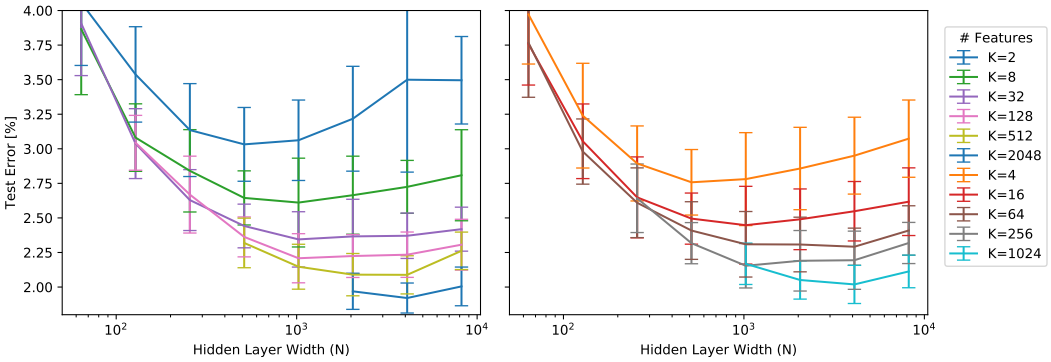

Figure 5. Full Test error results for a 2-layers fully connected network. The hidden layer features were initialized so the same random feature is represented N/K times.

D. ConstNet Features at Initialization are Equivalent to a Shallow Network

We consider the evolution of the network function f during full-batch gradient flow over a dataset of size d. Assuming $f(x) \in \mathbb{R}^{n_c}$, individual elements will evolve according to

$$\frac{\partial f_k(x)}{\partial t} = \sum_i \frac{\partial f_k(x)}{\partial \theta_i} \frac{\partial \theta_i}{\partial t} = -\sum_{ij} \frac{\partial f_k(x)}{\partial \theta_i} \frac{\partial \mathscr{L}(x_j)}{\partial \theta_i}$$

$$= -\sum_{jl} \left[\sum_i \frac{\partial f_k(x)}{\partial \theta_i} \frac{\partial f_l(x_j)}{\partial \theta_i} \right] \frac{\partial \mathscr{L}(x_j)}{\partial f_l(x_j)} \equiv \sum_{jl} \Theta_{kl}(x, x_j) \frac{\partial \mathscr{L}(x_j)}{\partial f_l(x_j)}.$$

where the tensor $\Theta \in \mathbb{R}^{n_c \times n_c \times d \times d}$ is known as the Neural Tangent Kernel (NTK). It is of interest because a sufficiently overparametrized neural network has the capacity to train with the NTK remaining essentially constant during training, enabling a detailed analysis of the dynamics of learning in this regime (Jacot et al., 2018; Lee et al., 2019; Arora et al., 2019). Training is thus essentially equivalent to kernel regression with respect to the kernel $\Theta \in \mathbb{R}^{n_c \times n_c \times d \times d}$, and if a standard random initialization is used this is a random feature kernel. While this is interesting from a theoretical perspective, it is also apparent that the real strength of deep neural networks is in learning features from data, and the standard operational regime of modern neural networks is one where Θ changes considerably during training (Chizat & Bach, 2018; Ghorbani et al., 2019).

The ConstNet architecture is interesting in this regard because the features it implements at initialization are particularly weak. They correspond to the features of a linear classifier. To show this, we consider a ConstNet function where the final layers of the network implement average pooling and an affine map, namely

$$f(x) = P(\alpha^{(L-1)}(x)) = W^{(L)} \frac{1}{\left|S^{(L-1)}\right|} \sum_{\gamma \in S^{(L-1)}} \alpha^{(L-1)}(x) + b^{(L)}$$

and all the previous layers are either convolutions or narrowing layers as described in section 3.1. The NTK takes the form

$$\Theta_{ij}(x, x') = \sum_k \frac{\partial f_i(x)}{\partial \theta_k} \frac{\partial f_j(x')}{\partial \theta_k}$$

$$= \sum_{\ell=1}^{L-1} \sum_{i_\ell j_\ell \gamma_\ell \gamma'_\ell \kappa} \beta^{(\ell)}_{\gamma_\ell i_\ell}(x) \alpha^{(\ell-1)}_{\gamma_\ell + \kappa, j_\ell}(x) \beta^{(\ell)}_{\gamma'_\ell i_\ell}(x') \alpha^{(\ell-1)}_{\gamma'_\ell + \kappa, j_\ell}(x')$$

$$+ \sum_{\gamma\gamma'} \beta^{(\ell)}_{\gamma_\ell i_\ell}(x) \beta^{(\ell)}_{\gamma'_\ell i_\ell}(x')$$

$$+ \delta_{ij} \left(\left\langle \frac{\sum_{\gamma \in S^{(L-1)}} \alpha^{(L-1)}_\gamma(x)}{\left|S^{(L-1)}\right|}, \frac{\sum_{\gamma' \in S^{(L-1)}} \alpha^{(L-1)}_{\gamma'}(x')}{\left|S^{(L-1)}\right|} \right\rangle + 1 \right)$$

where the last term is the contribution from the final layer. Note that since we initialize the final layer weights to zero, we have $\beta^{(\ell)}(x) = 0$ for $\ell < L$. In Appendix C we noted that if there are p narrowing layers in the network $\alpha_{\gamma i}^{(L-1)}(x) = \alpha_{\gamma s^p, i}^{(0)}(x)$ hence

$$\Theta_{ij}(x, x') = \delta_{ij}\left(\left\langle \frac{\sum_{\gamma \in S^{(L-1)}} \alpha_{\gamma s^p}^{(0)}(x)}{\left|S^{(L-1)}\right|}, \frac{\sum_{\gamma' \in S^{(L-1)}} \alpha_{\gamma' s^p}^{(0)}(x')}{\left|S^{(L-1)}\right|} \right\rangle + 1\right).$$

If we consider ConstNet without batch-normalization, which achieves a test accuracy of 95% on CIFAR-10 classification, we have $\tilde{\alpha}_{\gamma i}^{(0)}(x) = \sum_{j=1}^{3} x_{\gamma j}$. Therefore the kernel above is that of a linear model (since if the model was $f(x) = \theta^T x$, we would obtain $\Theta(x, x') = \langle x, x' \rangle$). The fact that ConstNet reaches this level of performance implies that there will be a massive performance gap between training ConstNet in the linear regime as in (Lee et al., 2019; Arora et al., 2019) and full nonlinear training. If we choose a more complex form for P the resulting NTK will still be identical to that of a shallow model (though not necessarily a linear one).

E. Replicating Features in a Single Hidden Layer

It is clear that having more than a single neuron representing the same feature in our final trained model is redundant — a group of neurons that are identical for all inputs will not contribute to a successful classification. We therefore ask the question: given that our network was initialized so the same neurons in a layer represent the same feature, will they diverge and contribute to the model accuracy during the training process?

To answer this question, we consider a 2-layers fully connected neural network:

$$\begin{gathered} X \in \mathbb{R}^d, W_1 \in \mathbb{R}^{N \times d}, W_2 \in \mathbb{R}^{\#\text{classes} \times N} \\ h(X, W_1) = \text{Relu}\left(W_1 X\right) \\ f(X, W_1, W_2) = \text{SoftMax}\left(W_2 h(x, W_1)\right) \end{gathered} \tag{10}$$

where we initialize the neural network as follow: the weight matrix W_2 is initialized using the standard He initialization (He et al., 2015), and for each number of features K, we initialize the temporary matrices $\tilde{W}_1 \in \mathbb{R}^{K \times d}, \widehat{W}_1 \in \mathbb{R}^{N \times d}$ using the He initialization, and initialize W_1 using $r = \frac{N}{K}$ replications of $\tilde{W}_1$, so

$$\forall i, W_1[i,:] = \tilde{W}_1[i \bmod k,:](1-\lambda) + \lambda \widehat{W}_1[i,:]. \tag{11}$$

where λ is a parameter simulating noise. For $\lambda = 0, K > 1$, this would result in each row in W_1 being repeated r times at initialization (and consequently, the same also applies to each hidden layer neuron). We identify two main possible causes for initially identical neurons to diverge during training: First, stochastic operations can result similar rows in W_1 receiving different gradient updates. The most commonly used operation that would achieve this is dropout, which will randomly mask neurons, so some neurons may freeze while their "replicas" change. The second cause is back-propagation itself: Even when not using dropout, the gradient of W_1 will be: $\frac{dL}{dW_{i,j}} = \frac{dL}{dh_i}\frac{dh_i}{dW_{i,j}} = \frac{dL}{dh_i} X_j$ and since $\frac{dL}{dh_i}$ depends on the corresponding column $(W_2^T)_i$ (which is random at initialization), the update gradient may be different for each row.

The training was done with standard SGD, learning rate of 0.1, with test accuracy measured after 7500 training steps and 30 seeds per sample.

F. Features Symmetry and Sub-networks

Network with identical features has inherently less unique sub-networks at initialization, but the there could be different levels of symmetries. In the case where all features have all-to-all connection with neighbouring layers (as was done in ConstNet and LeakyNet with '1' Init), all features at each layer are interchangeable. Therefore, when considering possible way to mask *neurons*, all sub-networks that mask the same amount of neurons at all layers are equivalent. For example, a randomly initialized neural network with L layers of width d can have up to 2^{dL} possible unique sub-networks, while a symmetrical features, all-to-all initialization ensures no more than $d^L = 2^{\log_2(d)L}$.

If the features are initialized to have a 1:1 connection with the following layer, as was done in the case LeakyNet with identity initialization, given a network of L layers if width d, we have 2^L ways to mask each feature, when only features with

the same masks at all layers are interchangeable. We can roughly approximate the number of sub-networks in this case by:

$$\left(\begin{array}{c} 2^L + d - 1 \\ d \end{array} \right) \propto \frac{\left(2^L - d\right)^{2^L}}{d^d \left(2^L\right)^{2^L}} \approx \frac{\left(2^L + d\right)^{2^L + d}}{d^d \left(2^L\right)^{2^L}} \overset{d \ll L}{\rightarrow} \frac{2^{dL}}{d^d}.$$

G. Perturbation Propagation

We illustrate in an idealized setting the effect of the I and '$\mathbb{1}$' initializations on propagation of a symmetry breaking signal. Recall that the tensors for these two initializations are given respectively by $\Delta \otimes I$ or $\Delta \otimes \frac{1}{n}\mathbf{1}\mathbf{1}^T$ where Δ is the delta kernel defined in eq. 7. As shown in appendix H, both of these choices of initialization are indistinguishable with respect to standard signal propagation as it applies to the features at initialization when these are symmetric.

Consider a feature tensor $\alpha \in \mathbb{R}^{S \times n}$ and a zero mean additive perturbation $\varepsilon \in \mathbb{R}^{S \times n}$ that breaks the symmetry between features. The source of such a perturbation can be non-deterministic GPU operations during parameter updates which will break symmetry after the first iteration of gradient descent. Applying a convolution gives

$$[\Delta \otimes I \hat{*} (\alpha + \varepsilon)]_{\gamma i} = [\alpha + \varepsilon]_{\gamma i}$$

$$\left[\Delta \otimes \frac{1}{n}\mathbf{1}\mathbf{1}^T \hat{*} (\alpha + \varepsilon)\right]_{\gamma i} = \left[\alpha + \sum_j \frac{1}{n}\varepsilon_{\gamma j}\right]_{\gamma i}.$$

We see that in the first instance the perturbation is propagated perfectly. In the second instance, if the perturbation components are independent, its norm will be reduced by about a factor of $\frac{1}{\sqrt{n}}$.

Propagation of the perturbation ε facilitates the breaking of symmetry in subsequent layers, and hence its attenuation hampers symmetry breaking. Thus even though from the perspective of propagation of the symmetric features the I and '$\mathbb{1}$' initializations are equivalent, the I initialization facilitates symmetry breaking signal propagation. Empirical evidence of this phenomenon is shown in figures 9 and 10. We measure the size of the symmetry breaking perturbation, given by the norm of the forward and backward features projected onto the complement of the all ones direction, divided by the feature norm itself for normalization. This is repeated for networks initialized with different combinations of the I and '$\mathbb{1}$' initializations. Generally, it can be seen that the relative norm of the symmetry breaking component increases during training. We also find that the relative size of the perturbation decreases sharply when passing through a '$\mathbb{1}$' initialized layer, yet is relatively unaffected by an I layer.

By measuring feature correlations as a function of layer, one can see the detrimental effect of the '$\mathbb{1}$' initialization on symmetry breaking even after training. These results are presented in figures 8. Networks with reduced symmetry breaking also achieve lower test accuracy as shown in table 2.

One can also conjecture based on these results that the combination of a zero initialization and skip connection as in ConstNet may be unique in the sense that the trainable parameters are initialized in a completely symmetric manner yet symmetry is easily broken. The '$\mathbb{1}$' initialization is symmetric but hampers symmetry breaking, while the I initialization is not symmetric (and it only preserves symmetry from previous layers since in LeakyNet the initial layer of the network induces a symmetry between features).

H. Leaky ReLU Signal Propagation

For simplicity we consider an L layer network of constant width at every layer and no batch normalization. The more general case with widening layers included can be handled in a similar manner to appendix C. Since batch normalization layers are applied after every block (which implements an identity map), their effect will simply be that after the first layer the normalized features are propagated.

We denote by $\alpha^{(0)}(x) \in \mathbb{R}^{S \times n}, \alpha^{(0)}_{\gamma i}(x) = \alpha^{(0)}_{\gamma j}(x)$ the input to the first LeakyNet block. As in the case of ConstNet, the input features are completely symmetric. Recall that the weight tensors of the network take the form $\Delta \otimes I$ or $\Delta \otimes \frac{1}{n}\mathbf{1}\mathbf{1}^T$ where Δ is the delta kernel defined in eq. 7, and the second factor acts on the channel indices. Due to the symmetry of the incoming features,

$$\Delta \otimes I \hat{*} \alpha^{(0)}(x) = \Delta \otimes \frac{1}{n}\mathbf{1}\mathbf{1}^T \hat{*} \alpha^{(0)}(x) = \alpha^{(0)}(x).$$

Defining by $\alpha^{(k)}(x)$ the features after k layers (where each LeakyNet block is composed of two layers), and recalling that $\sigma^\rho(-\frac{1}{\rho}\sigma^\rho(x)) = -x$, we find for every integer $k \leq L/4$

$$\begin{aligned}
\alpha^{(4k)}(x) &= \alpha^{(0)}(x) \\
\alpha^{(4k+1)}(x) &= -\frac{1}{\rho}\sigma^\rho(\alpha^{(0)}(x)) \\
\alpha^{(4k+2)}(x) &= -\alpha^{(0)}(x) \\
\alpha^{(4k+3)}(x) &= -\frac{1}{\rho}\sigma^\rho(-\alpha^{(0)}(x)).
\end{aligned}$$

Since we can express all features as simple functions of the initial features, norms of features and angles between them are trivially preserved (up to the application of single nonlinearity, which does not induce any dependence on the depth of the network).

Similarly, the backwards features cannot incur a dependence on depth, and their norms and angles between them are preserved up to the effect of a single non-linearity.

I. Comparison between symmetry breaking mechanisms

As seen in table 1, the hardware noise is a sufficient symmetry breaking mechanism for a ConstNet model initialized with identical features to perform as well as a similar model initialized with random initialization. It is, nevertheless, also apparent that not all breaking mechanisms perform equally well: When using 1% dropout as the sole symmetry breaking mechanism, there remains a gap of more than 1% between the model final accuracy to the accuracy of a randomly initialized model. The reason for dropout's failure is simple: On a standard setup, dropout is not being applied on the first convolutional layer, and it is therefore unable to break symmetry in this layer. Applying dropout on all layers does close this gap.

Another concern, is that due to our reliance hardware noise, it remains unclear how the same models will perform when used over different hardware. In table 3, we compare the accuracy of ConstNet models initialized with identical features and varying learning rates, using different GPUs. Our results show that the choice of hardware is indeed significant. Furthermore, they provide an insight concerning the effects of network quantization: Unlike the 32-bit floating point models tested before, models quantized to 16-bit, do not train successfully without the addition of dropout, as a complementary symmetry breaking mechanism. This result can be useful for the study of network-quantization, as it highlights a property of quantization that had not been studied — 32bit and 16bit training are typically expected to perform equally well for standard image-classification tasks. In this experiment, we used 200 epochs for training, which is sub-optimal (~ %0.5 drop baseline accuracy). Consequently, we can expect lower accuracy in cases where the symmetry breaking process is slow.

Hardware	Data-type	Computation Noise	Drop Rate:		
			0%	0.1%	1%
GeForce GTX-1080	FP32	✓	*94.09%	94.83%	*94.8%
GeForce GTX-1080	FP32	✗	24.96%	92.72%	92.99%
GeForce RTX-2080	FP32	✓	*77.81%	94.78%	94.82%
GeForce RTX-2080	FP32	✗	24.80%	92.69%	92.58%
GeForce RTX-2080	FP16	✓	38.61%	94.58%	94.62%
GeForce RTX-2080	FP16	✗	22.55%	91.99%	92.55%

Table 3. Experiment results (Test Accuracy) of training a ConstNet over the CIFAR10 dataset, with 200 epochs, 0 initialization ensuring identical features, and varying symmetry-breaking mechanisms. Dropout/ Computation noise by themselves are insufficient for successful training: the symmetry breaking in this case is too slow, and it come in the expense of the training process. Additionally, we can see a big gap between the effects of computation noise of different GPUs. When utilizing a 16-bit floating-point hardware architecture for a FP16 model, the hardware noise is insufficient and the training fails. *These results are discussed in more detail in appendix I.1.

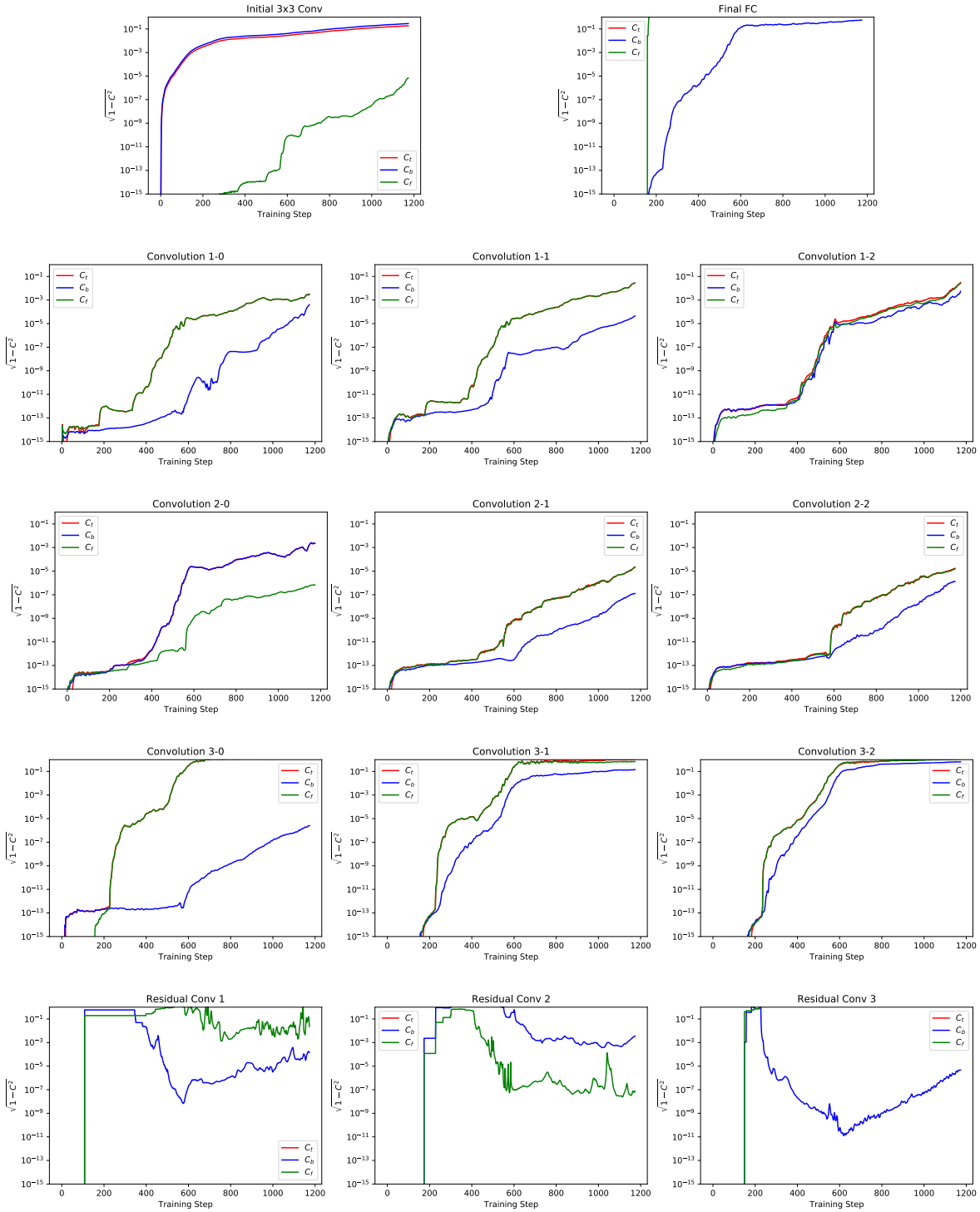

Figure 6. Feature symmetry in each of the layers ConstNet ('0'-init, 9 skip-able layers), on the first 3 epochs of training. The symmetry is measured using forward correlation C_f, backward correlation C_b, and total correlation C_t (Correlation between all filters), in all of the weight tensors in the network. Skip-able operations are numbered based on their groups (shown in figure 4), and position within the group: Tensors belong to the same groups, if the output of their respective operations is summed together (with or without passing through an activation operator). It is apparent that the forward correlations behave in a similar manner across operations for the 3×3 convolutions, while the residual convolutions tend to fluctuate and are less predictable. The backward correlations also behave in accordance to their group (though its group can be different from the group of the forward correlation in the same layer). e.g, the forward correlation of 'Convolution 3-0' is on group 3, while it's backward correlation, matches the behaviour of correlations group 2, to which it's input is connected.

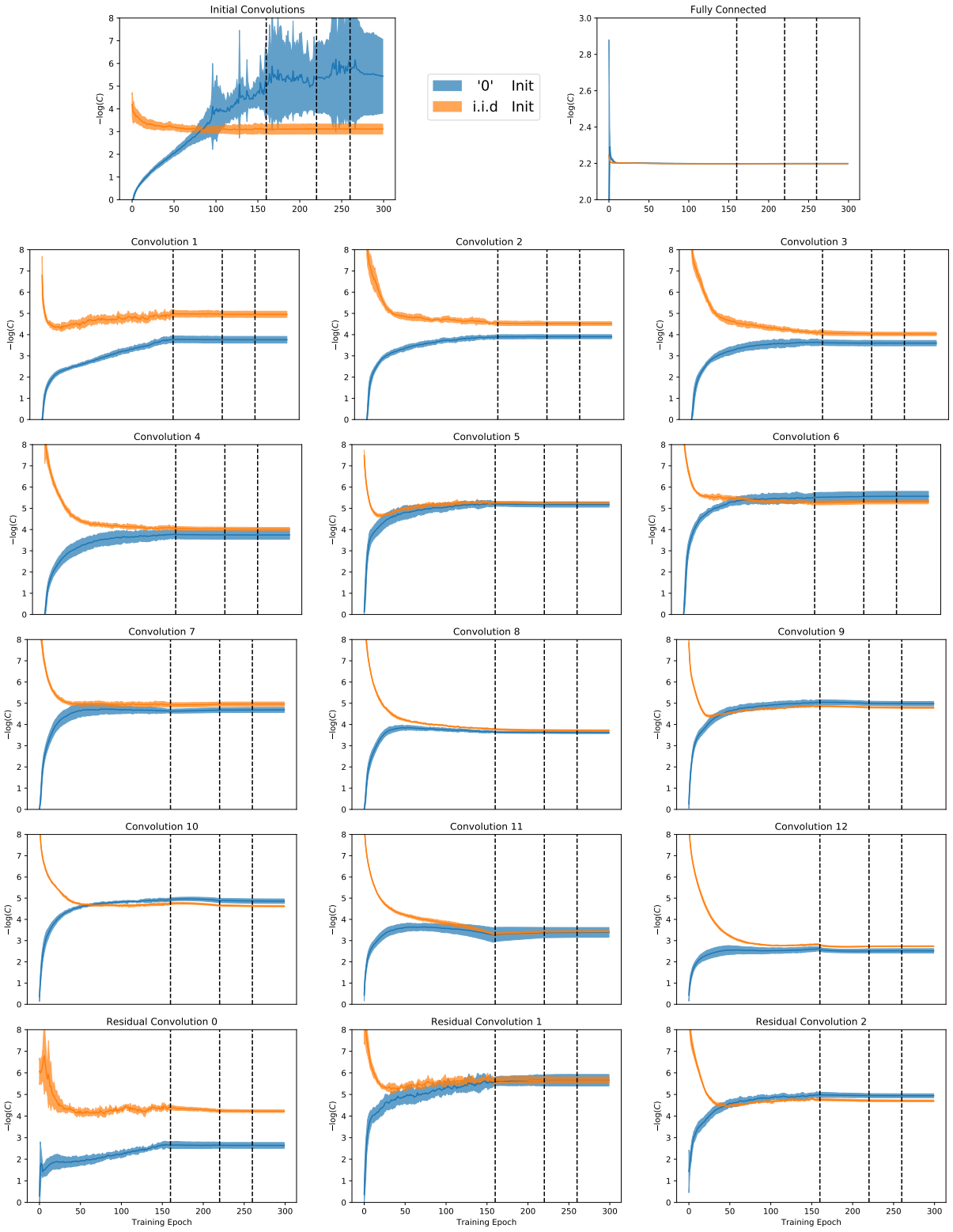

Figure 7. Natural logarithm of forward correlations for ConstNet during the entire training period. Average and error margin of 6 seeds per curve. The forward correlations of ConstNet with 0-init tend to converge to the same values measured for the same layer when ConstNet was initialized with independent, random initialization. Dashed lines mark the epochs in which the learning rate was lowered.

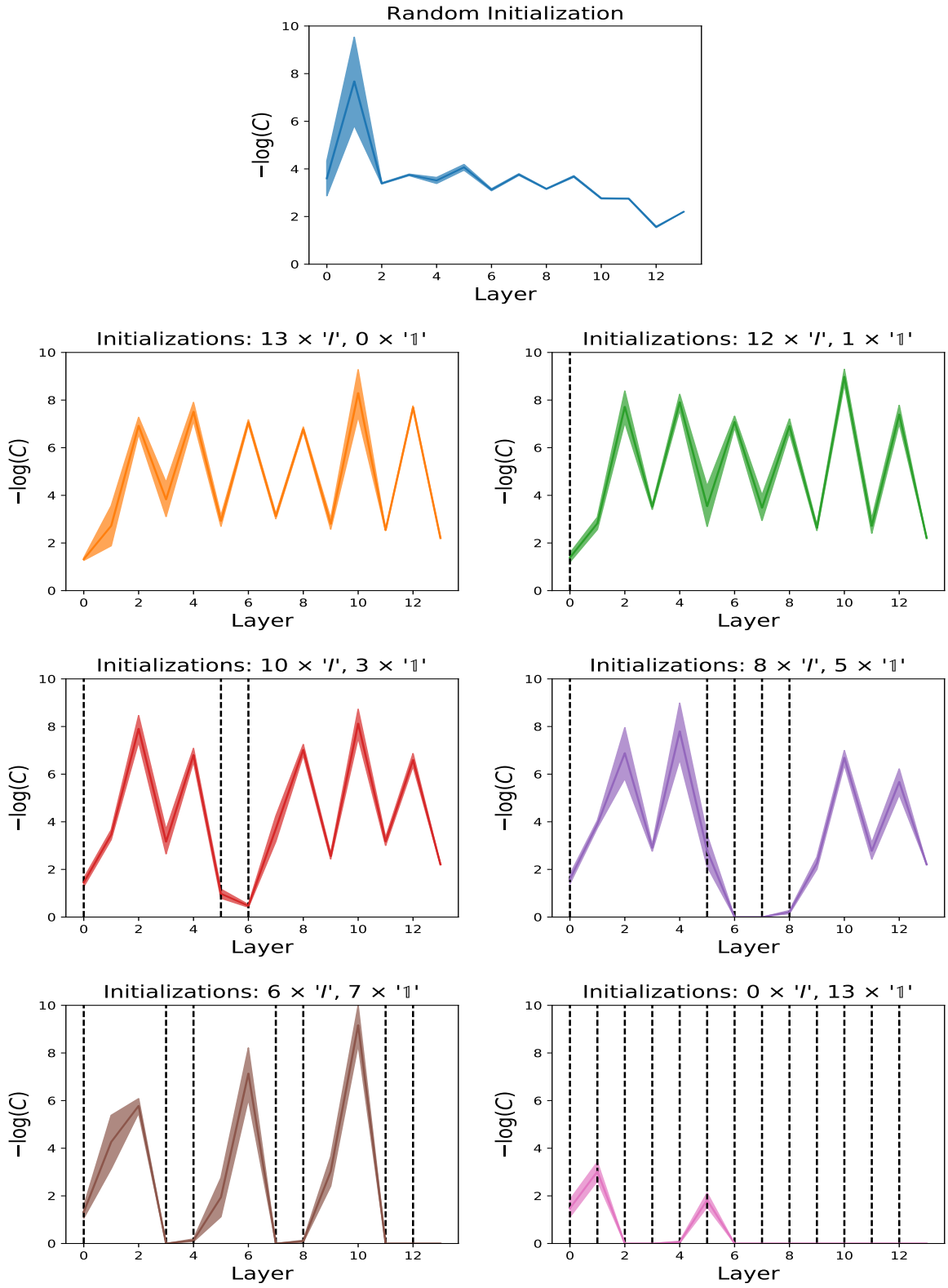

Figure 8. Natural logarithm of forward correlations in differently initialized leakyNets, at the end of training. Average and error margin of 4 seeds per curve. Unlike what we saw in ConstNet, the correlations in leakyNets initialized with mixed '$I/\mathbb{1}$' do not converge to the same values seen in the case of i.i.d. random He initialization (which eventually achieved higher test accuracy). Their inability to efficiently break symmetry can be explained by tracking the propagation of the perturbations from the initial features, as discussed in appendix G and can be seen in figures 9,10. Dashed lines mark the layers initialized with '$\mathbb{1}$'-init. Layer number 13 is the fully-connected layer, initialized to 0 on all instances.

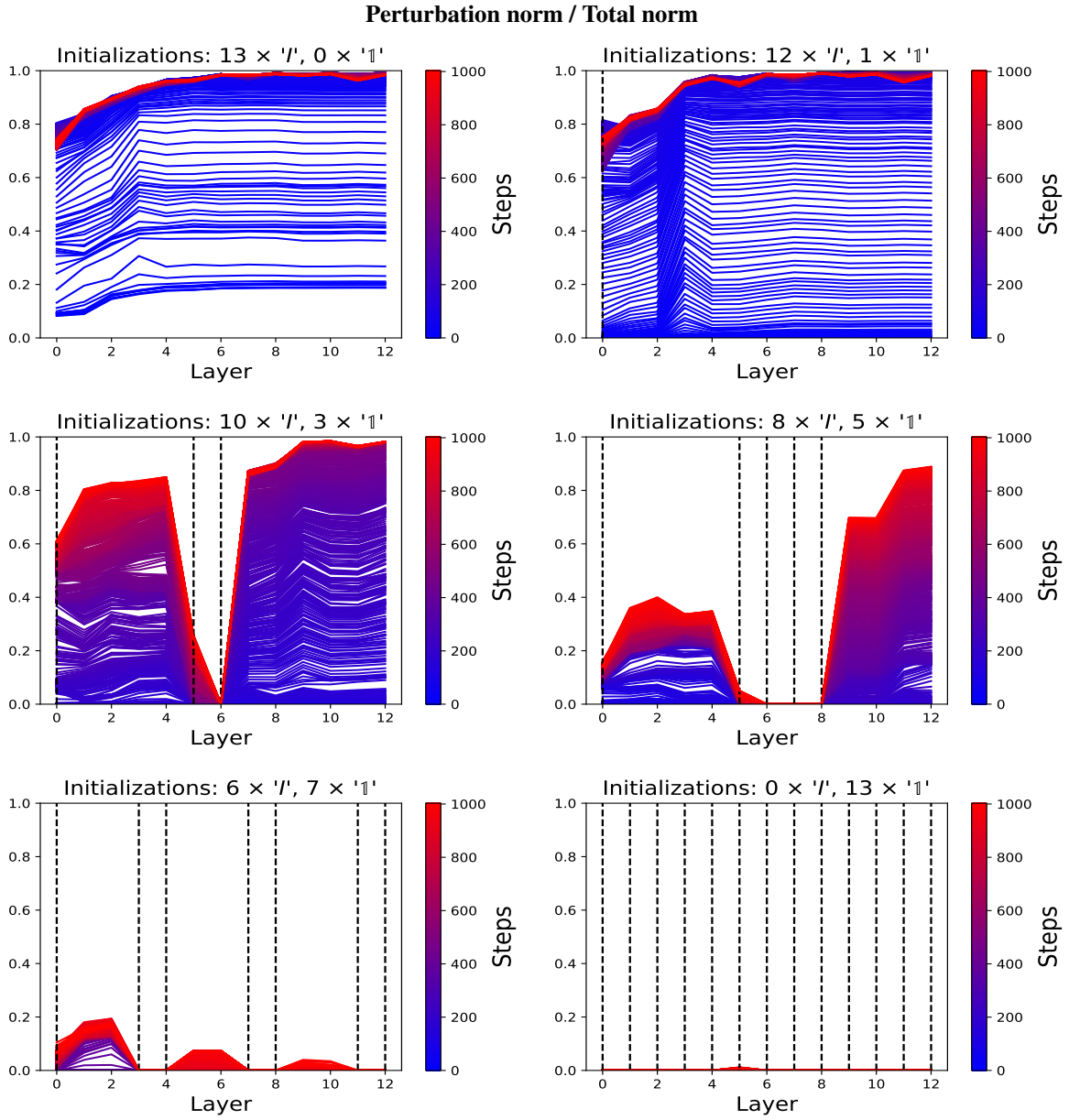

Figure 9. Symmetry breaking activation perturbation decays as it propagates forward through the network, if the $\mathbb{1}$ initialization was extensively used. The panels show the ratio between the perturbation norm $\left\|\Delta \otimes \left(I - \frac{1}{n}\mathbf{1}\mathbf{1}^T\right) \hat{*} \alpha^{(\ell)}(x)\right\|_F$ to the total norm $\left\|\alpha^{(\ell)}(x)\right\|_F$, for every layer ℓ and different initializations of leakyNet. Each panel also shows the evolution of the ratio at the first 1000 steps of training, as indicated by the color-map. Dashed lines mark the layers that were initialized with '$\mathbb{1}$'-init (while the other layers were initialized with I).

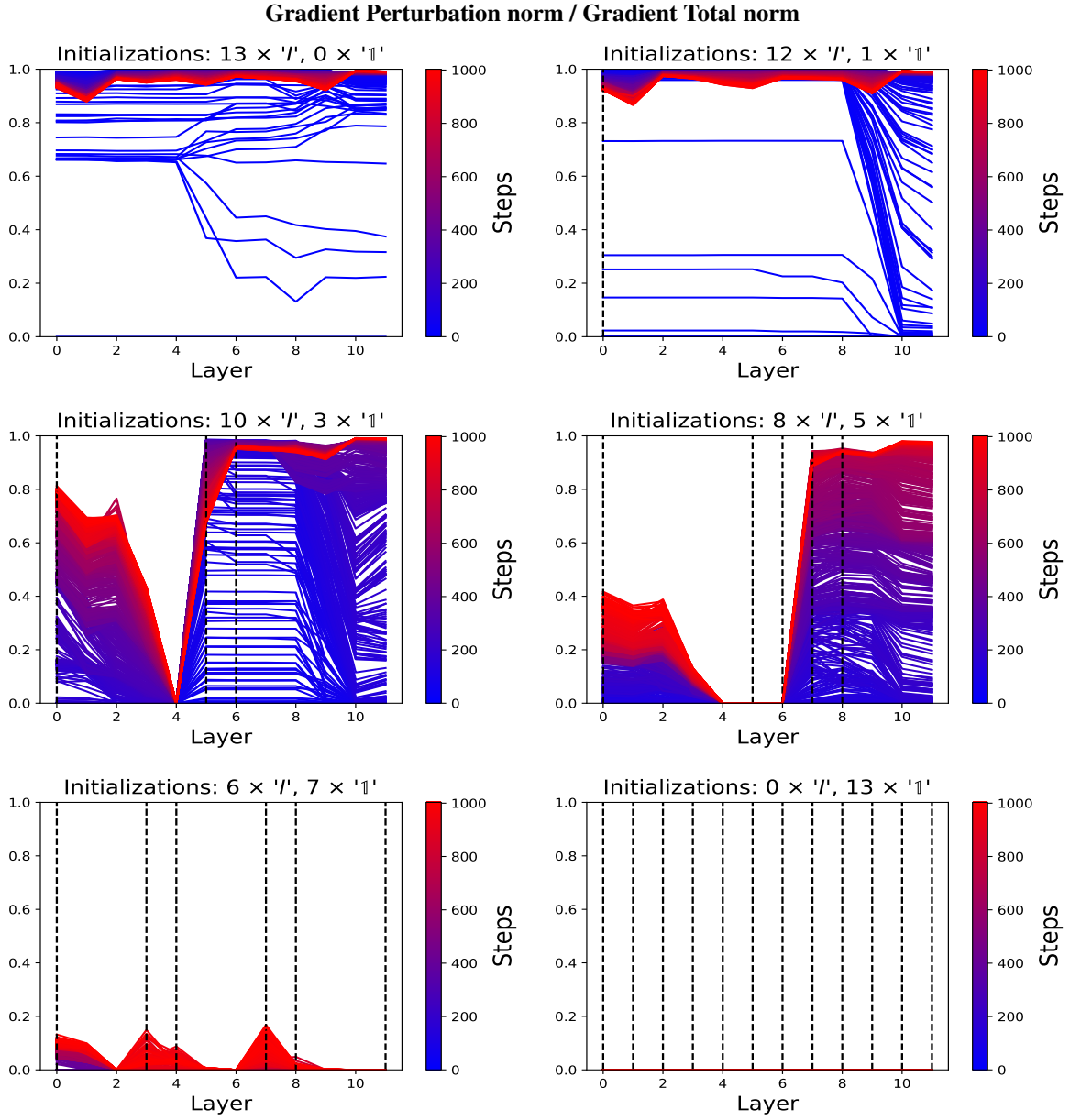

Figure 10. Symmetry breaking gradient perturbation decays as it propagates backwards through the network, due to the effect of the $\mathbb{1}$ initialization. As in figure 9, we measure the ratio of the gradient magnitude that is 'perturbed' from the mean-gradient over all channels. We thus measure the ratio between $\left\|\Delta \otimes \left(I - \frac{1}{n}\mathbf{1}\mathbf{1}^T\right) \hat{*} \frac{\partial \mathscr{L}(x,y)}{\partial \alpha^{(\ell)}(x)}\right\|_F$ to $\left\|\frac{\partial \mathscr{L}(x,y)}{\partial \alpha^{(\ell)}(x)}\right\|_F$. The perturbation in the gradients will directly influence the updates of the weight tensors, causing symmetry break (decrease forward correlation). In networks with symmetrical initialization ($\mathbb{1}$-init), it takes longer for the perturbation to grow– in both of the cases where the entire network was initialized with averaging initialization, the network remained symmetrical during the 1000 steps presented in this figure. Like before, dashed lines mark the layers that were initialized with '$\mathbb{1}$'-init (while the other layers were initialized with I).

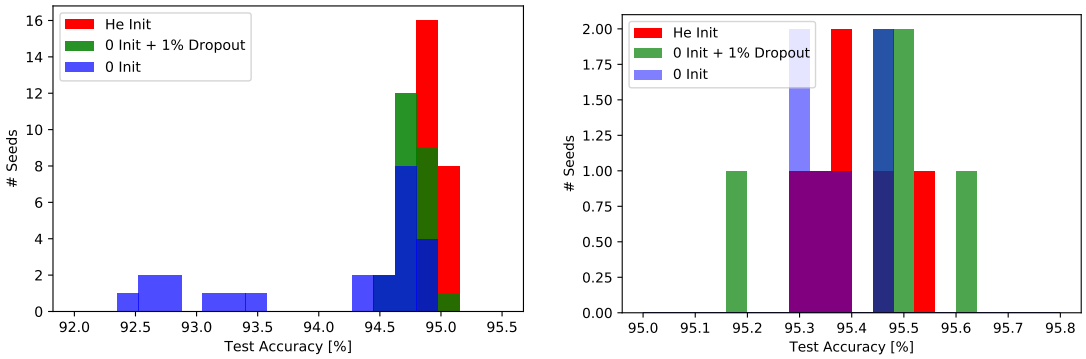

Figure 11. Distribution of results (Test accuracy) for training ConstNet with 200 epochs (left panel) and 300 epochs (right panel). Training ConstNet with hardware noise as the sole symmetry breaking mechanism is unreliable: On several instances, the symmetry breaking process was too slow for the model to fully recover during a 200 epochs training.

I.1. Limitations of symmetry breaking during training

As we can see in table 3, the negative effects of constant initialization are further highlighted in this experiment. In particular, we can see that the accuracy of 0 initialized ConstNet without dropout drops by more than %1.5 in comparison to its result for 300 epochs, as presented in table 1. A more thorough examination of the results shows the reason for this performance drop: On several cases (1/3 of the seeds tested), the hardware computation mechanism managed to break symmetry eventually, but spent too many epochs doing it. The distribution of the results for several cases can be seen in figure 11. When running the same model with the exact same configuration on a different GPU (Nvidia RTX), the degradation in performance is much more significant, with more than 1/2 of the experiments ending in failure ($< 80\%$).

J. Additional results for LeakyNet

In section 5, we presented LeakyNet, and examined the effect of mixing averaging ('$\mathbb{1}$') and identity ('I') initializations, by initializing arbitrary layers using different initializations. As complementary results, we were also interested to see how the results may be affected by using a linear mixture of the different initializations — so each of the 12 layers is initialized to $\gamma I + (1-\gamma)\mathbb{1}$ instead. Our results indicate that for any value of γ larger than 0, the training is equally successful, but is still significantly worse than training with random initialization. In this case, we can see that the forward correlations for each layer at the end of training are alternately similar/dissimilar, as can be seen in figure 12.

Data-set	Test accuracy [%]			
	Vanilla		Mixup ($\alpha = 1.0$)	
	'0'-Init	i.i.d-Init	'0'-Init	i.i.d-Init
Cifar10	94.82 ± 0.14	94.90 ± 0.06	96.22 ± 0.08	96.14 ± 0.01
Cifar100	77.02 ± 0.14	76.42 ± 0.20	79.08 ± 0.12	79.17 ± 0.10
CINIC	87.37 ± 0.02	87.50 ± 0.01	89.09 ± 0.05	89.21 ± 0.01

Table 4. **Generalization of the results**: Comparison of '0' and i.i.d initializations of ConstNet, for varying data-sets. In all runs, we used the training parameters as detailed in appendix A: specifically, we ran 2 seeds per configuration, with non- deterministic computation, and dropout with a drop rate of 1%. For all the data-sets and for all configurations, ConstNet achieved high accuracies, independent on the method of initialization (And thus, independent on the number of unique features at initialization).

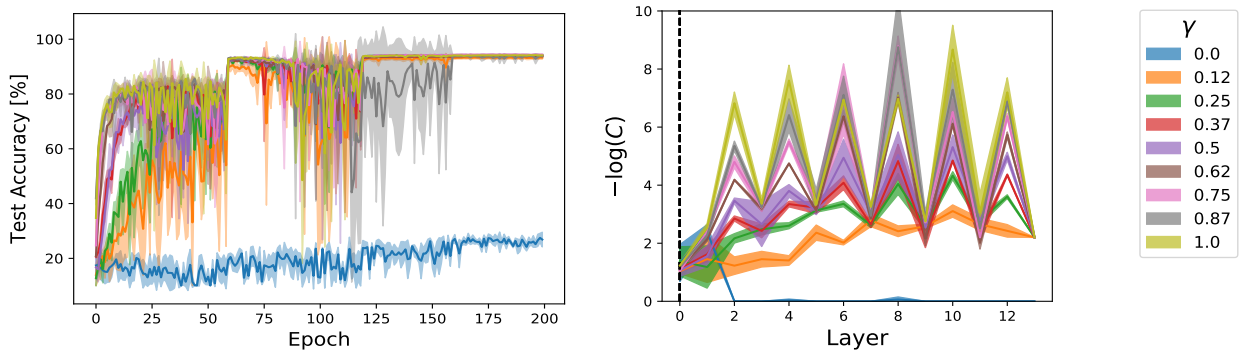

Figure 12. LeakyNets with layers initialized to a linear combination of $\mathbb{1}$ and 'I' initializations (3 seeds per configuration). The final accuracy was similar for all runs with $\gamma > 0$. The right panel describes the final forward correlation for all layers. With layer 0 being the initial convolution, only the odd layers have similar forward correlation values. The forward correlations of the even layers at the end of the run are highly dependent on their values at initialization.

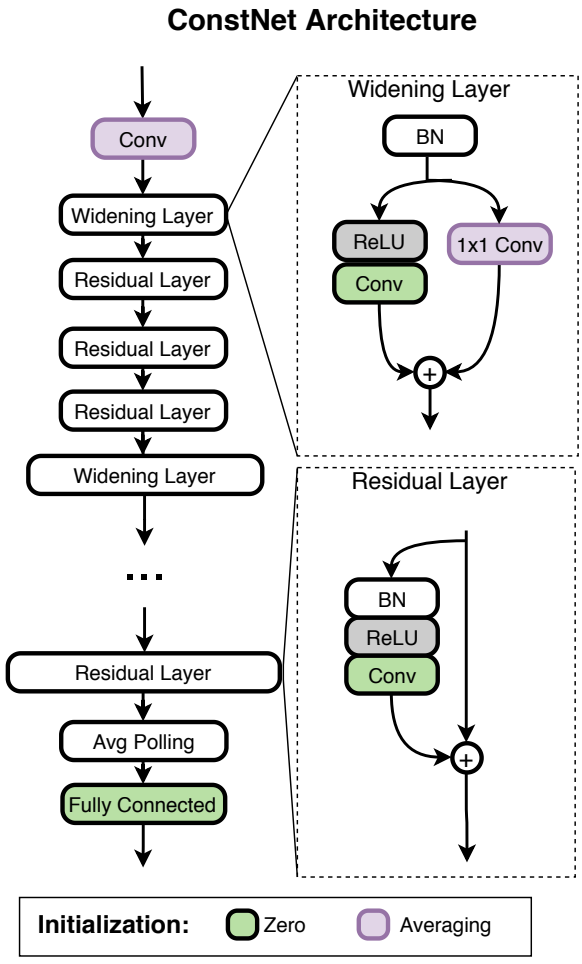

Figure 13. Detailed architecture of ConstNet.