Pretraining a Neural Network before Knowing Its Architecture

Boris Knyazev [1]

Abstract

Training large neural networks is possible by training a smaller *hypernetwork* that predicts parameters for the large ones. A recently released Graph HyperNetwork (GHN) trained this way on one million of smaller ImageNet architectures is able to predict parameters for large unseen networks such as ResNet-50. While networks with predicted parameters lose performance on the source task, the predicted parameters have been found useful for fine-tuning on other tasks. We study if fine-tuning based on the same GHN is still useful on novel strong architectures that were published after the GHN had been trained. We found that for recent architectures such as ConvNeXt, GHN initialization becomes less useful than for ResNet-50. One potential reason is the increased distribution shift of novel architectures from those used to train the GHN. We also found that the predicted parameters lack the diversity necessary to successfully fine-tune parameters with gradient descent. We alleviate this limitation by applying simple post-processing techniques to predicted parameters before fine-tuning them on a target task and improve fine-tuning of ResNet-50 and ConvNeXt.

1. Graph HyperNetworks for Initialization

Initialization of deep neural nets is critical to make them converge fast and to a generalizable solution (Glorot & Bengio, 2010; He et al., 2015; Knyazev et al., 2017). When training a neural net on small training data, an effective approach to initialize it is to pretrain it on a large dataset (Huh et al., 2016; Kolesnikov et al., 2020), such as ImageNet (Russakovsky et al., 2015). The architectures of neural nets keep evolving due to efforts of humans (Dosovitskiy et al., 2020; Liu et al., 2022) and neural architecture search (Elsken et al., 2019). So practitioners often need to rerun a costly pretraining procedure on their large in-house data[1] for every newly architecture discovered by the community to initialize it this way (Figure 1).

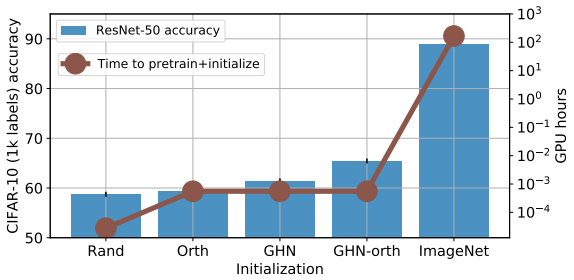

Figure 1. Motivating example showing a large amount of GPU hours (note the log scale) required to pretrain a novel architecture on ImageNet-1k to use it for initialization. To reduce the computational burden arising in practice when having own large pretraining dataset, we study the strategy of reusing an already trained Graph HyperNetwork (GHN) (Knyazev et al., 2021) for novel architectures, such as ConvNeXt (Liu et al., 2022), even if the architectures are discovered after the GHN was trained. GHN-orth is the proposed initialization. GPU hours are based on Knyazev et al. (2021). See Table 1 for the reported accuracies.

In the long term, a more efficient approach to initialize neural nets may be to train a Graph HyperNetwork (GHN) (Zhang et al., 2018; Knyazev et al., 2021) that can predict parameters for different architectures, including those yet to be discovered. GHN $H_{\mathcal{D}}$ parameterized by θ needs to be trained only once on a large pretraining dataset $\mathcal{D}$ (e.g. ImageNet). It can then predict parameters $\mathbf{w}$ in fractions of a second for arbitrary architectures a: $\mathbf{w} = H_{\mathcal{D}}(a;\theta)$. GHNs can predict parameters for much larger architectures than seen during training such as ResNet-50 (He et al., 2016). While networks with predicted parameters lose performance on the source task $\mathcal{D}$, the predicted parameters have been found useful as initialization for fine-tuning on other tasks (Knyazev et al., 2021). Such an initialization compared favorably to random-based initialization methods (He et al., 2015).

We study if initialization based on the already available trained GHN is still useful on novel strong architectures that were found after the GHN had been trained. We consider a

[1]Samsung - SAIT AI Lab, Montreal, Canada. Boris Knyazev's homepage: <http://bknyaz.github.io/>.

First Workshop of Pre-training: Perspectives, Pitfalls, and Paths Forward at ICML 2022, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

[1]E.g. Google's JFT-300M and Facebook's IG-1B-Targeted.

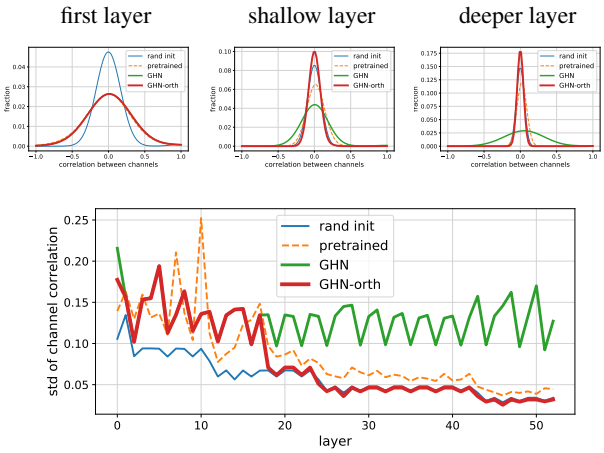

Figure 2. The similarities between the channels of convolutional layers in ResNet-50 measured as Pearson's correlation. **Top**: distribution of correlation for selected layers. **Bottom**: standard deviation of correlation distribution for all layers. Lower correlation (around 0) is expected to be beneficial since it implies the parameters are linearly independent and, thus, can facilitate convergence and generalization (Saxe et al., 2013; Wang et al., 2020). We found that channels of the parameters predicted by Graph HyperNetworks (GHNs) (Knyazev et al., 2021) are highly correlated making their fine-tuning challenging. We alleviate this issue by post-processing (Section 3) predicted parameters (GHN-orth). We show the benefit of our post-processing in Section 5.

real example of such a situation by evaluating the released GHN of Knyazev et al. (2021) on a recent ConvNeXt architecture (Liu et al., 2022). We found that for ConvNeXt the parameters predicted by the GHN become less useful for initialization and fine-tuning than for earlier architectures such as ResNet-50. One potential reason for that is an increased distribution shift of novel architectures from those used to train the GHN. We also analyzed predicted parameters (Section 2) and found that when initializing networks with predicted parameters, fine-tuning performance can be improved by reducing (Section 3) the similarities of predicted parameters.

2. Analysis of Parameters Predicted by GHNs

Since the GHN predicts parameters from a highly-compressed low-dimensional representation (Knyazev et al., 2021), we hypothesize that the predicted parameters may be highly correlated to each other. The benefit of random initialization is that all parameters are drawn independently from some probability distribution, e.g. Gaussian: $\mathbf{w}_i \sim \mathcal{N}$, where i is an index of the individual scalar value of the parameter tensor. In the GHN case, the parameters become conditional on a latent representation of the input computational graph $\mathbf{z}$: $\mathbf{w}_i \sim p(\mathbf{w}_i|\mathbf{z})$. To verify if the predicted parameters are highly correlated, we computed Pearson's correlation between channels for a given layer of a given architecture. We compared these correlations between networks with predicted parameters, initialized randomly and pretrained on ImageNet. We found that predicted parameters have generally much higher correlations with each other compared to other initializations (Figure 2). Methods such as random initialization and orthogonal regularization enforce statistical and linear independence of neural network parameters making them converge to a better solution in terms of generalization (Arora et al., 2019; Bansal et al., 2018; Wang et al., 2020). As predicted parameters are highly correlated, their fine-tuning may be difficult with stochastic gradient descent and non-convex problems. We therefore propose to decorrelate predicted parameters without destroying their pretraining power.

3. Post-processing of Predicted Parameters

In a given neural net with the parameters predicted by GHNs, post-processing is performed for each l-th layer independently from other layers. We denote the parameters of the l-th layer as $\mathbf{w}_l$. Parameters of convolutional layers are 4D, $\mathbf{w}_l \in \mathbb{R}^{K\times C\times H\times W}$, while in certain post-processing steps a matrix (2D) form is required. Following (Wang et al., 2020), to transform 4D to 2D, $\mathbf{w}_l$ is first reshaped to $\mathbf{w}_l \in \mathbb{R}^{K\times CHW}$ and then transposed if $K < CHW$. Post-processing consists of two steps: conditional noise addition (Section 3.1) and orthogonal re-initialization (Section 3.2).

3.1. Conditional Noise Addition

In addition to the channels of parameters being highly correlated (Figure 2), we found that many parameters are identical because the GHN of Knyazev et al. (2021) copies the same tensor multiple times to make sure the shapes of the predicted and target parameters match. Furthermore, the orthogonal re-initialization step introduced next in Section 3.2 is not beneficial or applicable to some layers (e.g. first layers or batch normalization layers). Therefore, to break the symmetry of identical parameters in all layers, we first add the Gaussian noise to all parameters:

$$\tilde{\mathbf{w}}_l = \mathbf{w}_l + \mathcal{N}\Big(0, \beta\sigma(r(\mathbf{w}_l))\Big), \tag{1}$$

where $r(\mathbf{w}_l) \in \mathbb{R}^{K\times K}$ is the correlation between the channels of the parameters $\mathbf{w}_l$ (Figure 2), $\sigma(\cdot)$ is the standard deviation, while β is a scaling factor shared across all layers. This way, the noise is added conditionally on the layer statistics to ensure that all layers are perturbed relatively equally.

3.2. Orthogonal Re-initialization

For fully-connected and convolutional layers starting from a certain depth $l \geq L$ we perform orthogonal re-initialization. We perform the same steps as in orthogonal initializa-

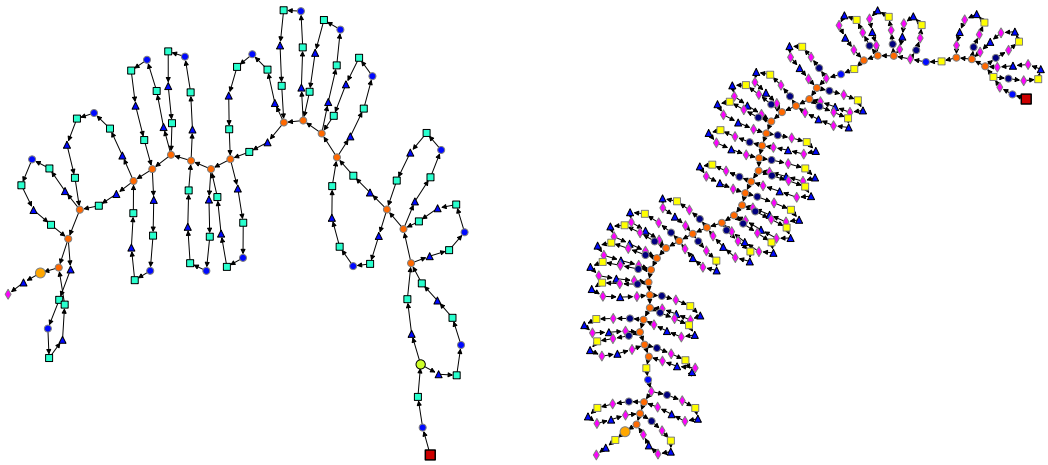

Figure 3. Computational graphs of ResNet-50 (left) and ConvNeXt (right) used as inputs to the GHN to predict their parameters. Each node corresponds to a certain operation such as convolution, pooling, layer normalization, etc. For the detailed description of node shapes and colors see (Knyazev et al., 2021).

tion (Saxe et al., 2013)[2], but where initial parameters $\tilde{\mathbf{w}}_l$ are predicted by GHNs (with some noise added according to (1)) rather than drawn randomly from a Gaussian distribution. Specifically, we first perform QR-decomposition $\tilde{\mathbf{w}}_l = \mathbf{QR}$ to find orthogonal matrix $\mathbf{Q}$ and upper-triangular matrix $\mathbf{R}$. We then obtain new parameters $\tilde{\mathbf{w}}$ as following:

$$\tilde{\mathbf{w}}_l = \mathbf{Q} \odot \text{sign}\Big(\text{diag}(\mathbf{R})\Big). \tag{2}$$

Parameters $\tilde{\mathbf{w}}_l$ are (transposed and) reshaped back to $K \times C \times H \times W$ and used instead of the original $\mathbf{w}_l$.

4. Experimental Setup

We evaluate our parameter post-processing techniques on the fine-tuning task on CIFAR-10 image classification (Krizhevsky et al., 2009) with 1000 (100 labels per class) labels as in (Knyazev et al., 2021). For ImageNet-based and GHN-based initializations we replace the last classification layer and fine-tune all layers. We fine-tune two neural nets: ResNet-50 and ConvNeXt (the base variant (Liu et al., 2022)) (Figure 3). We identified five important hyperparameters: optimizer, initial learning rate, number of epochs, weight decay and input image size. The hyperparameters are selected from the following values:

- optimizer: SGD, AdamW (Loshchilov & Hutter, 2017);
- learning rate: $\{0.025, 0.01, 0.005, 0.0025, 0.001\}$ for SGD and $\{0.004, 0.001, 0.0004\}$ for AdamW;
- number of epochs: 50, 100, 200, 300;
- weight decay: 0.0001, 0.001, 0.01, 0.05;
- image size: 224×224, 32×32 (original CIFAR-10 image size).

[2]E.g. see PyTorch implementation (Paszke et al., 2019)

The batch size is fixed to 96 for ResNet-50 as in (Zhang et al., 2018; Knyazev et al., 2021) and to 48 for ConvNeXt to fit into the memory of GPUs available to us. The cosine learning rate schedule is used in all experiments as in (Zhang et al., 2018; Knyazev et al., 2021). For our method (GHN-ORTH), we have additional hyperparameters: layer L from which to perform orthogonal re-initialization (2) and level of noise β added to parameters in (1). We tune all hyperparameters on the held-out validation set of 5,000 images.

Baselines As a baseline, we use random initialization (He et al., 2015) standard for ResNets, orthogonal initialization (Saxe et al., 2013) and GHN-2 from (Knyazev et al., 2021) (denoted as GHN in this paper). Orthogonal initialization (Saxe et al., 2013) is based on the same equation as (2) but applied to the randomly-initialized parameters drawn from the Gaussian distribution. As the oracle initialization we use ImageNet pretrained models. For fair comparison, we tune hyperparameters the same way for all methods. The experiments are run three times with different random seeds. Mean and standard deviation of the accuracy on the test set of CIFAR-10 is reported in Table 1.

Experiments are done using the GHN code base of Knyazev et al. (2021): `https://github.com/facebookresearch/ppuda/tree/main/ppuda`.

Table 1. Classification accuracies (mean and standard deviation across 3 runs) on reduced CIFAR-10 with 1000 labels.

INITIALIZATION	RESNET-50 (HE ET AL., 2016)	CONVNEXT (LIU ET AL., 2022)
# PARAMS	23.5M	87.6M
RAND INIT (HE ET AL., 2015)	58.7±0.5	48.3±0.8
ORTH INIT (SAXE ET AL., 2013)	59.5±0.4	51.8±0.1
GHN-2 (KNYAZEV ET AL., 2021)	61.4±0.2	51.9±0.8
GHN-2-ORTH (OURS)	65.4±0.5	53.5±0.3
IMAGENET PRETRAINED (HE ET AL., 2016)	89.0±0.1	95.1±0.3

5. Results

Our initialization based on predicted parameter post-processing (GHN-ORTH) improves on the direct competitor GHN-2 by 4.0 and 1.6 absolute percentage points for ResNet-50 and ConvNeXt respectively (Table 1). These results demonstrate the importance of proposed parameter post-processing. GHN-ORTH also outperforms ORTH-INIT confirming that our post-processing preserved useful structure in predicted parameters. GHN-ORTH is significantly inferior to ImageNet-based initialization. However, GHN-ORTH takes only fractions of a second to initialize for ResNet-50, ConvNeXt and potentially many other upcoming neural architectures in the future. In contrast, pretraining on ImageNet or large in-house datasets available to practitioners can take days or weeks for every novel architecture, especially given their increasing scale.

Applying our post-processing steps to ImageNet-pretrained models have not been found helpful and reduced fine-tuning results. This can be explained by the fact that the parameters of ImageNet-pretrained models are not highly correlated (Figure 2). While our post-processing can make them more linearly and statistically independent, it can also damage high-quality filters.

6. Discussion

The benefit of GHN-ORTH and GHN is lower on ConvNeXt than on ResNet-50 (Table 1). The capacities of these architectures (in terms of the number of trainable parameters) are not that different to explain this difference. We argue that even though ConvNeXt is composed of largely the same primitive operations[3] that compose the training architectures of GHNs (DeepNets-1M (Knyazev et al., 2021)), the compositions of these primitives in ConvNeXt are quite different

[3]One of the operations used in ConvNeXt was not supported by the GHNs, so we did not predict their parameters, which accounted for a small percentage w.r.t. the total number of parameters in ConvNeXt. There are also some operations without trainable parameters such as GELU nonlinearities or permutation of dimensions that are not explicitly modeled by GHNs and not included in the input computational graphs.

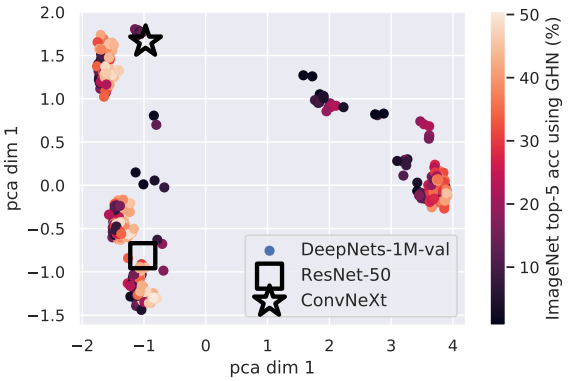

Figure 4. PCA-based projection of 32-dimensional latent representations of input computational graphs to two dimensionalities. These representations are computed based on Knyazev et al. (2021) on 500 validation architectures of DeepNets-1M.

compared to the other architectures in DeepNets-1M and ResNet-50. Such a difference in compositions may create a significant distribution shift confusing the GHN and making it predict poor parameters. To visualize this effect, we first extracted latent representations of input computational graphs from the validation architectures of DeepNets-1M as well as of ResNet-50 and ConvNeXt the same way as in Knyazev et al. (2021). We then projected these representations using the principal component analysis (PCA) into two dimensionalities and color coded the architectures with the accuracies of the corresponding networks with predicted parameters (Figure 4). This visualization reveals distinct clusters for lower and higher performant architectures in GHN's latent space. While ResNet-50 is located closely to the clusters with higher performant architectures, ConvNeXt is grouped together with a few low performant architectures. A relatively outlying latent representation of ConvNeXt may be explained by either the lack of similar architectures in the training set of DeepNets-1M or due to the difficulty of training the GHN on this kind of architecture. Understanding these reasons better may lead to more advances in GHNs and may potentially bridge the gap between computationally-intensive pretraining of networks

with SGD and almost zero-cost parameter prediction in the transfer learning scenarios.

Alternative to our approach, efficient pretraining of large networks is possible by first pretraining a smaller version of the network and then growing it (Chen et al., 2015; Evci et al., 2022). However, parameter prediction using GHNs is even more efficient (assuming the GHN already exists) as it does not require pretraining networks. We also have not compared to numerous initialization methods such as (Mishkin & Matas, 2015; Zhang et al., 2019; Huang et al., 2020; Zhang et al., 2019; Dauphin & Schoenholz, 2019; Zhu et al., 2021; Elsken et al., 2020), which is left for future work.

Acknowledgements

We would like to thank Diganta Misra, Bharat Runwal, Marwa El Halabi, Yan Zhang and Graham Taylor for the useful discussion and feedback. Resources used in preparing this research were provided by Calcul Québec (www.calculquebec.ca), Compute Canada (www.computecanada.ca) and Machine Learning Research Group at the University of Guelph (www.gwtaylor.ca).

References

Arora, S., Du, S., Hu, W., Li, Z., and Wang, R. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pp. 322–332. PMLR, 2019. 2

Bansal, N., Chen, X., and Wang, Z. Can we gain more from orthogonality regularizations in training deep networks? *Advances in Neural Information Processing Systems*, 31, 2018. 2

Chen, T., Goodfellow, I., and Shlens, J. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015. 5

Dauphin, Y. and Schoenholz, S. S. Metainit: Initializing learning by learning to initialize. 2019. 5

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019. 1

Elsken, T., Staffler, B., Metzen, J. H., and Hutter, F. Meta-learning of neural architectures for few-shot learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12365–12375, 2020. 5

Evci, U., Vladymyrov, M., Unterthiner, T., van Merriënboer, B., and Pedregosa, F. Gradmax: Growing neural networks using gradient information. *arXiv preprint arXiv:2201.05125*, 2022. 5

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010. 1

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015. 1, 3, 4

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016. 1, 4

Huang, X. S., Perez, F., Ba, J., and Volkovs, M. Improving transformer optimization through better initialization. In *International Conference on Machine Learning*, pp. 4475–4483. PMLR, 2020. 5

Huh, M., Agrawal, P., and Efros, A. A. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016. 1

Knyazev, B., Barth, E., and Martinetz, T. Recursive autoconvolution for unsupervised learning of convolutional neural networks. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2486–2493. IEEE, 2017. 1

Knyazev, B., Drozdzal, M., Taylor, G. W., and Romero Soriano, A. Parameter prediction for unseen deep architectures. *Advances in Neural Information Processing Systems*, 34, 2021. 1, 2, 3, 4

Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., and Houlsby, N. Big transfer (bit): General visual representation learning. In *European conference on computer vision*, pp. 491–507. Springer, 2020. 1

Krizhevsky, A. et al. Learning multiple layers of features from tiny images. 2009. 3

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., and Xie, S. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022. 1, 2, 3, 4

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 3

Mishkin, D. and Matas, J. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015. 5

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 3

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3): 211–252, 2015. 1

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013. 2, 3, 4

Wang, J., Chen, Y., Chakraborty, R., and Yu, S. X. Orthogonal convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11505–11515, 2020. 2

Zhang, C., Ren, M., and Urtasun, R. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018. 1, 3

Zhang, H., Dauphin, Y. N., and Ma, T. Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019. 5

Zhu, C., Ni, R., Xu, Z., Kong, K., Huang, W. R., and Goldstein, T. Gradinit: Learning to initialize neural networks for stable and efficient training. *arXiv preprint arXiv:2102.08098*, 2021. 5