Revealing the Utilized Rank of Subspaces of Learning in Neural Networks

Isha Garg [1] **Christian Koguchi** [1] **Eshan Verma** [1] **Daniel Ulbricht** [1]

Abstract

In this work, we study how well the learned weights of a neural network utilize the space available to them. This notion is related to capacity, but additionally incorporates the interaction of the network architecture with the dataset. Most learned weights appear to be full rank, and are therefore not amenable to low rank decomposition. This deceptively implies that the weights are utilizing the entire space available to them. We propose a simple data-driven transformation that projects the weights onto the subspace where the data and the weight interact. This preserves the functional mapping of the layer and reveals its low rank structure. In our findings, we conclude that most models utilize a fraction of the available space. For instance, for ViTB-16 and ViTL-16 trained on ImageNet, the mean layer utilization is 35% and 20% respectively. Our transformation results in reducing the parameters to 50% and 25% respectively, while resulting in less than 0.2% accuracy drop after fine-tuning. We also show that self-supervised pre-training drives this utilization up to 70%, justifying its suitability for downstream tasks.

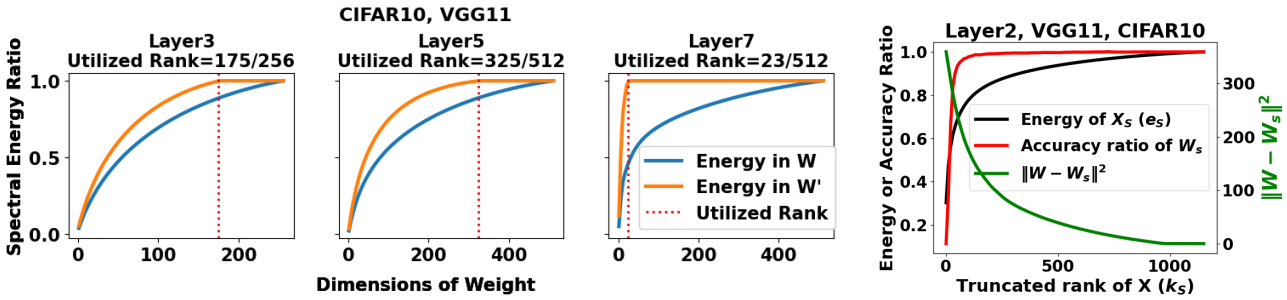
1. Introduction

The notion of 'capacity' of a network becomes less clear as we scale to large, deep neural networks. In practice, it is often thought of as a function of the number of parameters in the network. In this work, we shift out attention to the concept of *utilization*, which we define distinctly from model capacity in that it captures the interaction between both the complexity of a trained network and the dataset its trained on. We address utilization from a subspace perspective. Most learned weights appear to be full rank, suggesting we cannot trivially perform a low rank decomposition. In this work, we show that only a fraction of these dimensions interact with the data the weight operates on. We study the low rank decomposition of the input and output to the layers rather than the weights directly and find a simple modification that preserves the layer mapping by projecting the weight onto the subspaces of interaction. We refer to this as the *effective subspace* where learning occurred, and the dimension of this subspace as the *utilized rank* for that layer. This lower dimensional subspace allows for easy decomposition and efficiency by reducing the number of parameters and FLOPs. It also allows us to compare different networks in terms of their *Mean Layer Utilization* (MLU), a statistic that is informational for studying the structure of networks.

Suppose the input and output for a given layer live on subspaces S and T respectively. Then, projecting the input onto S and the output onto T is invariant in the forward pass up to some allowable L_2 error. We show that performing these two projections is similar to performing the forward pass with a transformed weight matrix W, with its row space projected onto S and its column space projected onto T. We upper-bound the error resulting from this transformation, and show that it can be driven down by controlling the spectral energies of the input and output subspaces. This transformation reveals the *utilized rank* of W, which we find to be far lower than the intrinsic rank of the original W. We determine the rank for a single layer by performing a binary search over the singular values of S and T to limit the resulting error from this transformation on the validation set. This allows us to find the utilized ranks of all layers without retraining, with a predictable and bounded accuracy drop that can easily be recovered via finetuning.

Studying the layerwise utilized rank of different network-dataset pairs suggests that most networks do not fully utilize the weight-space available to them. This means that a straightforward low rank decomposition can significantly reduce the number of parameters and FLOPs. For instance, we show that ViT variants trained on ImageNet only have 20% - 35% *mean layer utilization*, and can be decomposed to 25 to 48% of their original size while reducing the original FLOPs by between 13 to 33%. The resulting drop in accuracy after finetuning is less than 0.2%. We find that self-supervised pretraining uses the available space better ($MLU = 69\%$), making it suitable for multiple downstream tasks. We also study the effect of scaling the network and of increasing the dataset complexity.

[1]Apple. Correspondence to: Isha Garg <i_garg@apple.com>.



(a) Spectral energy spread of W and W'. The utilized rank becomes easily identifiable when we transform W to W'

(b) Projecting W onto S resulting from truncating the rank of X

Figure 1: Experiments on different layers of VGG11, CIFAR10

2. Methodology

2.1. Preliminaries: The Input and Output Subspaces

For simplicity, we consider a fully connected layer of a neural network. Let the input be to this layer be $X \in \mathbb{R}^{B\times d}$, where B is the batch size and each row vector $x \in \mathbb{R}^d$. Let $W^T \in \mathbb{R}^{d\times m}$ be the weight that maps X from to $Y \in \mathbb{R}^{B\times m}$. The corresponding forward pass can be written as:

$$Y = XW^T \tag{1}$$

For the first layer of a neural network, X is real data such as images. Similarly, the output of the layer would be dependent on the overlap between the input space and the column space of W, i.e. if the columns of W and X were orthogonal, the output would be zero. Generalizing to all layers, let S be the subspace of the input to the layer, with dimension k_S. The orthogonal complement of this subspace, $S_\perp$ is $d - k_S$ dimensional, and contains the space of inputs or activations not occupied by the real input. We can find this subspace using SVD, shown below

$$X = U_X \Sigma_X V_X^T \tag{2}$$

where Σ_X is a diagonal matrix of the d singular values σ_i and the first k_S rows of V_X^T represents a bases for S. We define the spectral energy ratio as $e_S = \frac{\sum_0^{k_S} \sigma_i^2}{\sum_0^d \sigma_i^2}$ such that we can preserve 99% of the spectral energy e.g. $e_s = 0.99$ with k_S equal to the number of singular values (squared) that contain 99% of the total energy. We construct the projection matrix P_S that projects X onto S, denoted by X_S as:

$$V_S := V_X[:k_S]; \quad V_{S_\perp} := V_X[k_S:] \tag{3}$$

$$P_S = V_S^T V_S; \quad P_{S_\perp} = V_{S_\perp}^T V_{S_\perp} \tag{4}$$

$$X_S = XP_S \quad X_{S_\perp} = XP_{S_\perp} \tag{5}$$

Similarly, let the subspace of the output be T and the spectral energy e_T correspond to the utilized rank k_T. Similar to equations for S, V_T contains the bases for T found from performing the SVD on Y and gives the projection matrix for $P_T \in \mathbb{R}^{m\times m}$. Further details for SVD computation are provided in appendix section 5.2.

2.2. The Weight Transformation and the Utilized Rank

In the forward pass equation 1, replacing X with X_S and Y with Y_T should result in the forward pass mapping remaining largely unaltered. This is equivalent to modifying W by projecting its column space onto S and its row space onto T, resulting in a modified W' as shown below:

$$Y \approx Y_T = YP_T \tag{6}$$

$$= XW^TP_T \tag{7}$$

$$\approx XP_SW^TP_T \tag{8}$$

$$\implies Y \approx XW'^T; \ \texttt{where} \ W' := P_SW^TP_T \tag{9}$$

We refer to the rank of W' as the *utilized rank* since **this transformation is data-dependent and captures the subspace overlap between the weight-space and the data-space**. In Figure 1a, we show the spectral energy distribution of W and W' for different layers of VGG11 trained on CIFAR10 data. From the figure, we can see that **the spectral energy of W has a wider distribution than W', obfuscating the true rank. Transforming W into W' compacts the spectral energy and allows us to identify the utilized rank more easily that naively applying the SVD directly**. For later layers, we note that the utilized rank is a small fraction of the available dimensions (23/512), highlighting the overparametrization of VGG architectures for CIFAR10. The resulting error from replacing W by W' in equation 1 can be upper-bounded by choosing appropriate dimensions for the input and output subspaces k_S and k_T.

$$\|E\|^2 = \|XW^T - X(P_TWP_S)^T\|^2 \tag{10}$$

$$\leq (1 - e_T)\|Y\|^2 + (1 - e_S)\|X\|^2\|W\|^2 \tag{11}$$

The proof utilizes the fact that the Frobenius norm is the sum of the square of singular values (see Appendix section 5.1).

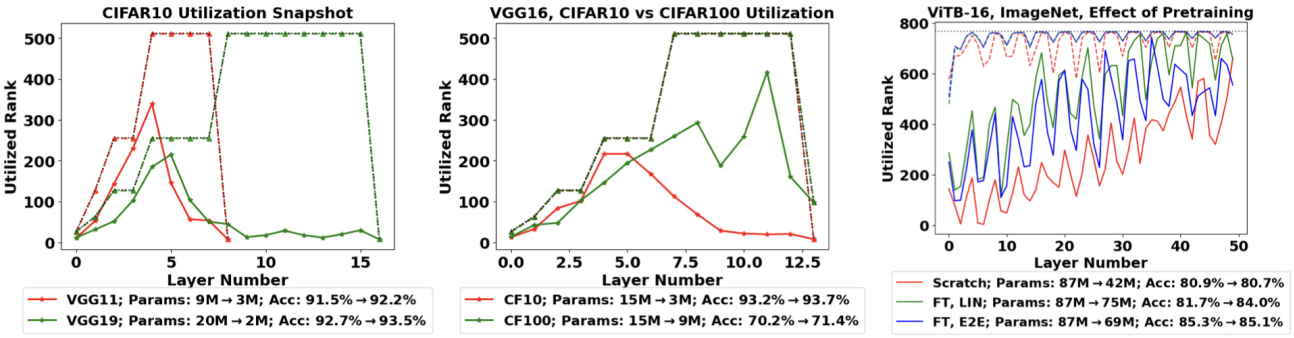


Figure 2: Utilization snapshots of different dataset-network pairs.The rank of the unaltered W is plotted for each layer in the dotted line, and the rank of the transformed W as the solid line. The brackets list the parameters and accuracy of the original and decomposed, finetuned network. FT refers to finetuning from SWAG [45], in a linear or end-to-end fashion.

How to choose k_S and k_T The error per layer is a function of e_S and e_T, and we use validation accuracy to inform us of the maximum k_S and k_T we can set before suffering a performance drop. In Figure 1b, we vary k_S for a single layer of VGG11 trained on CIFAR10 and plot the impact on e_S (black), the accuracy when we replace W by W_S as a ratio of the original accuracy (red), and the norm difference between W and W_S (green). For this layer, we note that when k_S reaches $\approx 200/1024$ dimensions, the transformed W_S does not result in an accuracy drop even though W_S differs significantly from W in norm (≈ 150). When $k_S = 200$ and $e_S = 0.8$, retaining only 80% of the energy was sufficient to achieve full accuracy. Hence, to maximize savings, we perform a binary search on e_S and e_T for each layer, while using validation accuracy drop as the signal to inform the stopping criterion. We call the accuracy drop tolerance for each transformation, (S or T projection for each layer) as ϵ, and set it to 0.1 for our experiments. After estimating $k_S,\ k_T$ that conforms to this ϵ error for all layers, the transformed network would have an accuracy drop = $2 \times \#\text{layers} \times \epsilon\%$, which scales with the depth of the network. However, **since we largely preserve the functional mapping of each layer, we find that finetuning is able to recover the allocated drop**. When finetuning, we decompose each layer into 2 layers of reduced rank to ensure that finetuning does not increase the searched rank.

2.3. Benefits of Studying the Utilized Ranks of Layers

Mean Layer Utilization: We describe the utilization statistic for a layer as the ratio of the rank of W' to the maximum rank possible. Suppose the utilized rank of a layer with $W \in \mathbb{R}^{m \times d}$ is r, then the layer utilization is $\frac{r}{\min(m,\ d)}$. The rank of W' is constrained to the rank of the product of $P_S W^T P_T$, so we can calculate the rank r for a given layer as $\min(k_S,\ k_T)$. A utilization close to 1 implies that the learnt column space of the weight overlaps fully with the subspace of the input to the layer, whereas a utilization close to 0 implies that the spaces are orthogonal, resulting in little to no signal being passed forward. The utilized rank depends on both the network architecture and the dataset, allowing us to capture a notion of capacity that is more informative than just the number of FLOPs or parameters. We average this score over all convolutional and linear layers, and call this the MLU (mean layer utilization) score of the network. **A higher MLU reveals that the network is well utilized, while a lower MLU allows for low-rank decomposition for efficiency.**

Savings in FLOPs and parameters: This low dimensionality of W results in a low rank decomposition that directly reduces memory and compute costs if the rank $r \leq \frac{m \times d}{m+d}$. Hence, for all layers that meet this criterion, we decompose the layer into 2 layers with weights of shapes $r \times d$ and $m \times r$, respectively. This reduces the total parameters and compute approximately by a factor of $\frac{(m \times d)}{r(m+d)}$.

Utilization Snapshot: To study the layer-specific dynamics of rank utilization, we chart the rank of the learned W, the utilized rank r, and the maximum rank possible at each layer as a utilization snapshot of a trained network. This can visualize the maximum per-layer utilization across various network and dataset combinations. We can also utilize this to understand the effects of different pretraining and finetuning techniques.

3. Results and Discussion

We perform experiments on VGG [44], ResNet [15], ViT [6], DeiT [48], Swin Transformer [32], and Resnet variants [57] on CIFAR10, CIFAR100 [28], and ImageNet [5]. We use pretrained ViTs and ResNets from torchvision [38] and DeiTs and SWIN transformers from TIMM [53][1]. We use Deepspeed [39] for profiling FLOPs with a batch size of

[1]For CIFAR, we use the architectures and hyperparameters from github.com/bearpaw/pytorch-classification

Architecture	Orig Acc (%)	Orig MLU (%)	Acc - Ours (%) (Δ)	True MLU (%)	Params Ratio	Flops Ratio
ViTB16	80.9	94	80.7 (-0.2)	35	0.48	0.33
ViTB32	75.7	94	75.8 (+0.1)	34	0.46	0.33
ViTL16	79.5	81	79.5 (+0.0)	20	0.25	0.13
ViTL32*	76.9	92	76.2 (-0.7)	26	0.36	0.26
DeiT - Tiny†	72.1 / 75.3	98	75.0 (-0.3)	86	0.99	0.99
DeiT - Small†	79.8 / 80.1	98	80.3 (+0.2)	74	0.89	0.89
DeiT - Base†	81.8 / 82.0	98	81.5 (-0.5)	49	0.64	0.65
SWIN - Tiny	81.2	98	81.3 (+0.1)	65	0.86	0.83
SWIN - Small	83.3	98	83.4 (+0.1)	60	0.81	0.77
SWIN- Base*	85.2	98	84.5 (-0.7)	66	0.86	0.83
SWIN - Large*	86.3	98	85.3 (-1.0)	53	0.74	0.70
ResNet34	73.2	99	72.2 -(1.0)	66	0.77	0.76
ResNet50	80.1	99	79.4 (-0.7)	60	0.83	0.74
ResNet101	81.5	99	80.5 (-1.0)	47	0.66	0.59
WideResNet50_2	81.2	99	80.6 (-0.6)	43	0.68	0.58
WideResNet101_2	82.3	99	81.7 (-0.6)	33	0.51	0.44

Table 1: Results for Utilized Rank Decomposition on ImageNet. ViT [6] and ResNet [15, 57] pretrained models from torchvision [38], DeiT [48] and SWIN[32] from TIMM [38] *implies $\epsilon = 0.05\%, 0.1\%$ otherwise. †Finetuning the original DeiT models results in improved performance.

32. We define the drop per layer at $\epsilon = 0.1\%$. For ViTL-32, Swin-Base, and Swin-Large, the finetuned accuracy drop for $\epsilon = 0.1\%$ was greater than 1%, and was reduced to 0.05%. We use SVD for calculating ranks. To rule out very small singular values arising from numerical errors, we assign the rank as the number of singular values that explain 99.99% spectral energy. Finetuning is done with each layer decomposed into two layers of reduced rank to ensure it does not increase rank. However, when reporting final savings, we decompose only those layers where matrix decomposition would result in a reduction in parameters. Finetuning hyperparameters are in Appendix section 5.4.

3.1. Utilization Statistics of Popular Networks

Studying layerwise utilization can help us understand the suitability of the model for the dataset. In Figure 2, left, we show the layer-utilization for VGG11 and VGG19, for the same dataset CIFAR10. We see that they achieve similar layer utilization, with a peak in utilization around layers 4-6 for the same task. While the original parameters grow from 9M to 20M, the utilized parameters stay stable around 2.5M. In Figure 2, center, we evaluate the effect of increasing dataset complexity on a static architecture to illustrate higher network utilization for CIFAR100 than CIFAR10. Not only is the utilization for CIFAR100 higher, but the utilization at higher layer numbers could indicate the usage of higher level features required to solve a more complex task.

From Tables 1 and 3, we note that the original models have close to $100\% MLU$, deceptively implying that all the space available for learning is well used. However, **upon decomposition, we find that the corresponding MLUs are quite low**, dipping to 20-35% for ViT variants on ImageNet. The fact that ViTs are too big for ImageNet has been noted previously, with the popularity of 'Tiny' variants.In fact, DeiT-Tiny utilizes space quite well (99% true MLU compared to ViTL-16's 20%), indicating that increasing size would indeed result in a gain in accuracy. We note that DeiT networks show improved performance when training for longer. For a fair comparison, we finetune DeiT pretrained models from TIMM using the same hyperparameters as ours, and compare against the finetuned models. Both this original and finetuned accuracy for DeiT models is reported.

3.2. Parameter and Compute Efficiency

In Figure 3, we study the effect of rank-decomposed and finetuned models on different architecture-dataset pairs. We plot the number of parameters against the accuracy, with the number of FLOPs represented by the sizes of the bubbles. We see that most networks shrink and move towards the top left corner when decomposed and finetuned, implying an increase in accuracy and decrease in number of parameters and FLOPs. From tables 1 and 3, we note that we can significantly reduce the size and FLOPs for most networks. For instance, VGG19 on CIFAR10 can be reduced to just 11% of the original size, consuming only 38% of the original FLOPs. Similarly, parameters reduce to 25% and FLOPs to 16% on ViTL-16 for ImageNet. On ImageNet, we see

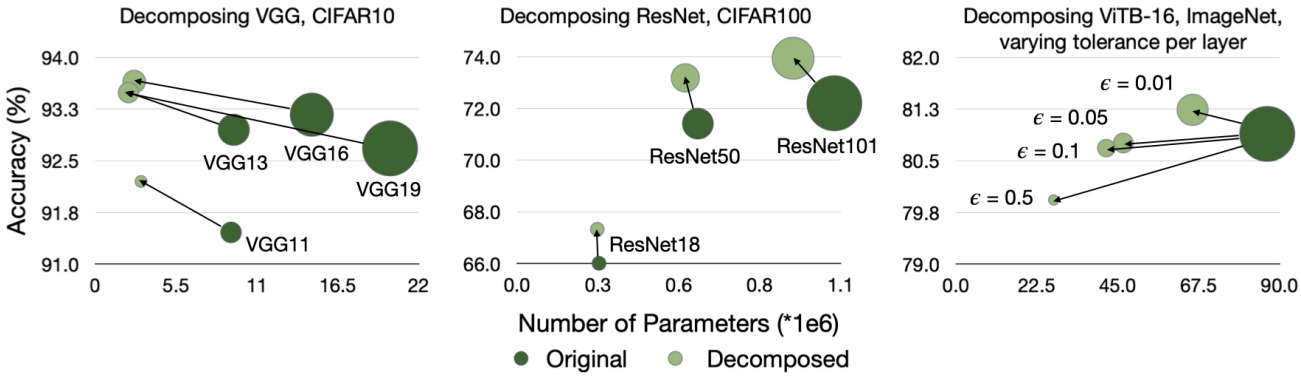


Figure 3: Visualizing the change in accuracy, number of parameters and FLOPs (size of bubble) of the decomposed, finetuned model. ϵ is the accuracy drop tolerance per layer during rank search.

drops and increases in accuracy of less than 1% On CIFAR, we note that finetuning accuracies never drop compared to original, sometimes increasing up to 2% over the baseline. We attribute this potentially to an increased regularization effect from using low rank weights for small datasets.

3.3. Scaling network size and dataset complexity

We show the effect of scaling a network in the same family for the same dataset in Figure 3, left, with numbers in Table 3. We see that VGG13, VGG16, and VGG19 all converge to very similarly sized models on CIFAR10 with a very similar accuracy upon decomposition, despite being different in their original format. This indicates that a bigger network is not necessarily beneficial for CIFAR10. However, we note that all networks report 10-20% higher MLU when we scale up the dataset complexity, going from CIFAR10 to CIFAR100, also seen in in Figure 3, center. This implies that the available capacity is being better utilized by larger datasets. Hence, our method serves to incorporate both the notion of capacity of the network, and its interaction with the complexity of the dataset.

3.4. Varying the acceptable accuracy drop per layer

We set the acceptable accuracy drop per layer, ϵ, to 0.1%, resulting in a total accuracy drop of $0.2\% \times$ #layers. In Figure 3, we show the effect of increasing or decreasing this hyperparameter for ViTB-16 (numbers in Appendix Table 5). Even when using a smaller drop of 0.01% per layer, we can still reduce the network to 76% of the parameters and 58% of the FLOPs while gaining 0.4% accuracy improvement, indicating that ViTB-16 is too large of a network for ImageNet. The smallest model resulting with $\epsilon = 0.5\%$ consumes only 31% of the parameters and 20% of the original FLOPs, and shows an accuracy drop of less than 1%. While ϵ should be tuned for every model and dataset pair, we find that 0.1% and 0.05% give good results across various architectures and datasets.

3.5. Effect of pretraining on ViTs

In Figure 2, right, we evaluate the impact of weakly supervised pretraining (SWAG [45]) on layer utilization on downstream tasks. All models start close to maximum rank shown in the dotted lines. [FT-LIN] refers to the network that was frozen after pretraining with only a linear head finetuned on ImageNet. The frozen weights learned from self supervised pretrained utilize the available space to the highest extent ($MLU = 69\%$), reflecting its suitability for downstream tasks. The model finetuned, end-to-end on ImageNet [FT-E2E] shows a drop in layer-utilization, especially at later layers, since it is altered for the classification task. Training a model from random initialization [scratch] yields a bespoke model for ImageNet and shows lower layer utilization ($MLU = 35\%$). The increase in accuracy for the LIN-FT network using our method is an unfair comparison, since we finetune end-to-end after finding the rank.

4. Conclusion

In this work, we proposed the *mean layer utilization*, a simple data-dependent metric for determining how efficiently a neural network learns a particular dataset. We do this by creating projection matrices for each layer to transform the learned weights onto a compact subspace dictated by the input and output activations with a controllable error that is upper bounded by the spectral energy of the input and output subspaces e_S and e_T. This compact representation reveals what we call the *utilized rank* of a matrix, which serves as a notion of capacity that includes both the network architecture and the dataset. Lastly, decomposing the layers onto these data-dependent subspaces naturally lend themselves to a simple weight matrix factorization which can easily be applied to various popular network architectures such as ViTs and ResNets achieving significant parameter reduction without compromising on downstream task performance.

References

[1] A. Aghajanyan, L. Zettlemoyer, and S. Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning, 2020.

[2] S. Ashkboos, M. L. Croci, M. G. do Nascimento, T. Hoefler, and J. Hensman. Slicegpt: Compress large language models by deleting rows and columns, 2024.

[3] N. Brown, A. Williamson, T. Anderson, and L. Lawrence. Efficient transformer knowledge distillation: A performance review, 2023.

[4] J.-F. Cai, E. J. Candes, and Z. Shen. A singular value thresholding algorithm for matrix completion, 2008.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[7] R. Feng, K. Zheng, Y. Huang, D. Zhao, M. Jordan, and Z.-J. Zha. Rank diminishing in deep neural networks, 2022.

[8] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[9] E. Frantar and D. Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot, 2023.

[10] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.

[11] K. Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5: 322–333, 1969. doi: 10.1109/TSSC.1969.300225.

[12] I. Garg, P. Panda, and K. Roy. A low effort approach to structured cnn design using pca. *IEEE Access*, 8:1347–1360, 2020. ISSN 2169-3536. doi: 10.1109/access.2019.2961960. URL `http://dx.doi.org/10.1109/ACCESS.2019.2961960`.

[13] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `https://proceedings.mlr.press/v9/glorot10a.html`.

[14] L. Grasedyck, D. Kressner, and C. Tobler. A literature survey of low-rank tensor approximation techniques, 2013.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[16] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network, 2015.

[17] E. Hoffer, R. Banner, I. Golan, and D. Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. *Advances in Neural Information Processing Systems*, 31, 2018.

[18] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models, 2021.

[19] M. Huh, H. Mobahi, R. Zhang, B. Cheung, P. Agrawal, and P. Isola. The low-rank simplicity bias in deep networks, 2023.

[20] Y. Idelbayev and M. A. Carreira-Perpinan. Low-rank compression of neural nets: Learning the rank of each layer. pages 8046–8056, 2020. doi: 10.1109/CVPR42600.2020.00807.

[21] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions, 2014.

[22] W. Johnson and J. Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemporary Mathematics*, 26:189–206, 01 1984. doi: 10.1090/conm/026/737400.

[23] S. R. Kamalakara, A. Locatelli, B. Venkitesh, J. Ba, Y. Gal, and A. N. Gomez. Exploring low rank training of deep neural networks, 2022.

[24] M. Khodak, N. Tenenholtz, L. Mackey, and N. Fusi. Initialization and regularization of factorized neural layers. *arXiv preprint arXiv:2105.01029*, 2021.

[25] M. Khodak, N. Tenenholtz, L. Mackey, and N. Fusi. Initialization and regularization of factorized neural layers, 2022.

[26] H. Kim, M. U. K. Khan, and C.-M. Kyung. Efficient neural network compression, 2019.

[27] H. Kim, M. U. K. Khan, and C.-M. Kyung. Efficient neural network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[28] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009. URL `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

[29] C. Li, H. Farkhoor, R. Liu, and J. Yosinski. Measuring the intrinsic dimension of objective landscapes, 2018.

[30] Y. Li, Y. Liang, and A. Risteski. Recovery guarantee of weighted low-rank approximation via alternating minimization, 2016.

[31] L. Liebenwein, A. Maalouf, O. Gal, D. Feldman, and D. Rus. Compressing neural networks: Towards determining the optimal layer-wise decomposition, 2021.

[32] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021.

[33] R. K. Mahabadi, J. Henderson, and S. Ruder. Compacter: Efficient low-rank hypercomplex adapter layers, 2021.

[34] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, and B. Bossan. Peft: State-of-the-art parameter-efficient fine-tuning methods, 2022.

[35] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. pages 807–814. Omnipress, 2010. ISBN 9781605589077.

[36] M. B. Noach and Y. Goldberg. Compressing pre-trained language models by matrix decomposition. pages 884–889. Association for Computational Linguistics, 12 2020. URL `https://aclanthology.org/2020.aacl-main.88`.

[37] S. Oymak, Z. Fabian, M. Li, and M. Soltanolkotabi. Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian. *arXiv preprint arXiv:1906.05392*, 2019.

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[39] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3406703. URL `https://doi.org/10.1145/3394486.3406703`.

[40] O. Roy and M. Vetterli. The effective rank: A measure of effective dimensionality. In *2007 15th European Signal Processing Conference*, pages 606–610, 2007.

[41] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. pages 6655–6659, 2013. doi: 10.1109/ICASSP.2013.6638949.

[42] S. Schotthöfer, E. Zangrando, J. Kusch, G. Ceruti, and F. Tudisco. Low-rank lottery tickets: finding efficient low-rank neural networks via matrix differential equations. *Advances in Neural Information Processing Systems*, 35:20051–20063, 2022.

[43] P. Sharma, J. T. Ash, and D. Misra. The truth is in there: Improving reasoning in language models with layer-selective rank reduction. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=ozX92bu8VA`.

[44] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[45] M. Singh, L. Gustafson, A. Adcock, V. de Freitas Reis, B. Gedik, R. P. Kosaraju, D. Mahajan, R. Girshick, P. Dollár, and L. van der Maaten. Revisiting weakly supervised pre-training of visual perception models, 2022.

[46] X. Suau, L. Zappella, and N. Apostoloff. Filter distillation for network compression, 2019.

[47] Y. Sui, M. Yin, Y. Gong, J. Xiao, H. Phan, and B. Yuan. Elrt: Efficient low-rank training for compact convolutional neural networks, 2024.

[48] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention, 2021.

[49] M. Tukan, A. Maalouf, M. Weksler, and D. Feldman. Compressed deep networks: Goodbye svd, hello robust low-rank approximation, 2020.

[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023.

[51] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma. Linformer: Self-attention with linear complexity, 2020.

[52] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li. Coordinating filters for faster deep neural networks, 2017.

[53] R. Wightman. Pytorch image models, 2019.

[54] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

[55] A. Yaguchi, T. Suzuki, S. Nitta, Y. Sakata, and A. Tanizawa. Decomposable-net: Scalable low-rank compression for neural networks. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/447. URL http://dx.doi.org/10.24963/ijcai.2021/447.

[56] W. Yang. pytorch-classification. https://https://github.com/bearpaw/pytorch-classification, 2017. Accessed: 2023-06-01.

[57] S. Zagoruyko and N. Komodakis. Wide residual networks, 2017.

[58] E. Zangrando, P. Deidda, S. Brugiapaglia, N. Guglielmi, and F. Tudisco. Neural rank collapse: Weight decay and small within-class variability yield low-rank bias, 2024.

[59] G. Zhang, C. Wang, B. Xu, and R. Grosse. Three mechanisms of weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1lz-3Rct7.

[60] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization, 2018.

[61] Y. Zhang, E. Chuangsuwanich, and J. Glass. Extracting deep neural network bottleneck features using low-rank matrix factorization. pages 185–189, 2014. doi: 10.1109/ICASSP.2014.6853583.

5. Appendix

5.1. Upper-bounding the error from transforming W to W'

We note that the projection matrices are symmetric since $P_S^T = (V_S^T V_S)^T = P_S$. We use these to express the error from transforming W to W' in terms of the perpendicular spaces.

$$
\begin{aligned}
E &= XW^T - XW'^T && (12)\\
&= XW^T - X(P_T W P_S)^T && (13)\\
&= XW^T - (XP_S)W^T P_T && (14)\\
&= XW^T - X_S W^T P_T && (15)\\
&= XW^T - (X - X_{S_\perp})W^T P_T && (16)\\
&= (XW^T - XW^T P_T) + X_{S_\perp} W^T P_T && (17)\\
&= (Y - Y_T) + X_{S_\perp}(P_T W)^T && (18)\\
&= Y_{T_\perp} + X_{S_\perp} W_T^T && (19)
\end{aligned}
$$

Since the Frobenius norm of a matrix, squared, is the sum of its singular values, squared, our definition of S, T implies the following relations:

$$
\begin{aligned}
X &= X_S + X_{S_\perp}; & Y &= Y_T + Y_{T_\perp}; && (20)\\
\|X_S\|^2 &= e_S \|X\|^2; & \|Y_T\|^2 &= e_T \|Y\|^2; && (21)\\
\|X_{S_\perp}\|^2 &= (1 - e_S)\|X\|^2; & \|Y_{T_\perp}\|^2 &= (1 - e_T)\|Y\|^2 && (22)
\end{aligned}
$$

where all norms refer to Frobenius norm. Additionally, we know that $\|A + B\|_F^2 = Tr\left((A + B)^T (A + B)\right) = \|A\|_F^2 + \|B\|_F^2 + 2Tr(A^T B)$. Since trace is invariant to cyclic permutation and transpose), we have $Tr(A^T B) = Tr(AB^T)$. Putting all this together, we can upper bound the error in equation 19 as follows.

$$
\begin{aligned}
\|E\|^2 &= \|Y_{T_\perp}\|^2 + \|X_{S_\perp} W_T^T\|^2 + 2Tr(Y_{T_\perp} W_T X_{S_\perp}^T) && (23)\\
&= \|Y_{T_\perp}\|^2 + \|X_{S_\perp} W_T^T\|^2 + 2Tr(Y P_{T_\perp} P_T W X_{S_\perp}^T) && (24)\\
&= \|Y_{T_\perp}\|^2 + \|X_{S_\perp} W_T^T\|^2 + 2Tr(Y (P_{T_\perp} P_T) X_{S_\perp}^T) && (25)\\
&= \|Y_{T_\perp}\|^2 + \|X_{S_\perp} W_T^T\|^2 + 0 && (26)\\
&= \|Y_{T_\perp}\|^2 + \|X P_{S_\perp} W_T^T\|^2 && (27)\\
&\leq \|Y_{T_\perp}\|^2 + \|X_{S_\perp}\|^2 \|W_T^T\|^2 && (28)\\
&\leq \|Y_{T_\perp}\|^2 + \|X_{S_\perp}\|^2 \|W^T\|^2 && (29)\\
&= (1 - e_T)\|Y\|^2 + (1 - e_S)\|X\|^2 \|W\|^2 && (30)
\end{aligned}
$$

The trace in equation 26 reduces to zero since we multiply two matrices in orthogonal spaces, resulting in zero. The last inequality in equation 29 arises from applying triangle inequality on W.

$$
\begin{aligned}
W &= W_T + W_{T_\perp} && (31)\\
\|W\|^2 &= \|W_T\|^2 + \|W_{T_\perp}\|^2 + 2Tr(W_T W_{T_\perp}) && (32)\\
\|W\|^2 &= \|W_T\|^2 + \|W_{T_\perp}\|^2 + 0 && (33)\\
\|W\|^2 &\geq \|W_T\|^2 && (34)
\end{aligned}
$$

5.2. Details of SVD to find bases

For computational ease, we perform the SVD of $X^T X$, which directly gives us the bases and the square of the singular values. This only require storing the sum of $X^T X$ at each layer, which can be parallelized over multiple batches of forward passes. We do not need to store the outputs of a layer, since we can find $T^T T$ from pre and post multiplying the saved $X^T X$ with W^T and W respectively, and then performing SVD on this smaller matrix. For CIFAR datasets, we use the entire training dataset to perform PCA, and for ImageNet, we choose 200 samples per class, resulting in 20,000 samples. Because this computation is parallelizable across batches and requires only forward passes, the cost of finding bases and ranks of a space is negligible. Note The same analysis will hold for bias/convolutional layer with the input being the flattened patches convolved into the filters. The addition of bias back into the analysis also does not alter the subspaces under consideration, since we only look at each layer's input and output in isolation from all other layers.

5.3. Computational Overhead of Binary Search for Rank

There are three main overheads: performing SVD at each layer, weight transformation and binary search on dimensions. We perform highly parallelized SVD on the entire training dataset of CIFAR, or 20,000 samples for ImageNet, and performing SVD for all layers takes lesser time than a training epoch in most cases. Each choice of e_S and e_T results in an analytical weight transformation from just 2 matrix multiplications, and we only need to perform a validation pass for each level of binary search to find the direction of binary search. There are a few hyperparamters that can be optimized to speed this up, such as size of data to perform SVD on, maximum levels of binary search, and conditions to quit search on, such as acceptable accuracy drop and limiting the change in dimensions between consecutive iterations.

The most expensive part of our computation is the validation accuracy checks for binary search for rank. Let the weight matrix at a layer be $m \times d$ dimensional, with L layers in the network. For the first projection on S, we perform SVD on a $d \times d$ matrix, and a binary search on the resulting d singular values. Each level of binary search performs one projection to get W' and one validation accuracy check. This means that we have O(log d) validation accuracy check. Similarly for the output, we have O(log m) accuracy checks, bringing the total to $L \times O(m \times d)$ accuracy checks. For ViTB-16, the largest layers are 768×3072, and there are approximately 50 linear layers. This means that we perform ~ 1000 valiation accuracy checks for this network. It took us 7.5 hours on a machine with 8 A100 GPUs to calculate the utilized rank of all layers via this binary search.

5.4. Hyperparameters for Finetuning

After performing binary search on all layers of the network, we decompose each linear and convolutional layer into two consecutive layers (without non-linearity in between) so that we can finetune while preserving the searched rank. We initialize the two layers to the left and right matrices arising from SVD on the weight (with either one appropriately scaled by the singular values). We then perform a grid search on the following parameters for finetuning: learning rate, weight decay and EMA (exponential moving average) decay. When we use EMA, we start averaging the model for EMA from the beginning of finetuning. For all other hyperparameters, we used the same as the base repository that we took the model from.

	ViTs, SWINs, DeiTs on ImageNet	ResNets on ImageNet	All architectures on CIFAR
Optimizer	adamW	SGD, LR decayed by 0.1 at 30 and 60	SGD, LR decayed by 0.1 at 50, 100, 130
Batch Size	512	32	32
Epochs	300	90	200
Learning Rate	[0.0003,0.0001]	[0.1, 0.01, 0.001]	[0.1,0.01,0.001]
Weight Decay	[0.3,0]	[0,0.0001]	[50, 100, 130]
EMA Decay	[0.85, 0.9, 0.95]	NA	NA

Table 2: Hyperparameters for finetuning the decomposed models.

5.5. CIFAR Results

Here we present the numbers corresponding to the graphs in Figures 3 for CIFAR10 and CIFAR100 on VGG and ResNet architecutre variants. All results correspond to networks decomposed and finetuned to respect the rank found from binary search.

		Orig Acc (%)	Orig MLU (%)	Acc - Ours (%) (Δ)	True MLU (%)	Params Ratio	Flops Ratio
CIFAR10	VGG11	91.5	98	92.5 (+1.0)	47	0.34	0.60
	VGG13	92.9	98	93.5 (+0.6)	47	0.24	0.60
	VGG16	93.2	99	93.6 (+0.4)	44	0.18	0.54
	VGG19	92.7	99	93.6 (+0.9)	32	0.11	0.38
	ResNet18	90.9	95	91.4 (+0.5)	80	0.86	0.92
	ResNet50	92.8	96	93.1 (+0.3)	64	0.78	0.78
	ResNet101	93.2	96	94.1 (+0.9)	51	0.73	0.64
CIFAR100	VGG11	66.9	99	67.4 (+0.5)	64	0.78	0.72
	VGG13	70.2	99	71 (+0.8)	68	0.76	0.77
	VGG16	70.2	99	71.4 (+1.2)	62	0.61	0.71
	VGG19	70.2	99	71.8 (+1.6)	51	0.38	0.68
	ResNet18	66.0	96	67.8 (+1.8)	90	0.98	0.99
	ResNet50	71.4	96	73.5 (+2.1)	80	0.93	0.92
	ResNet101	72.2	96	73.5 (+1.3)	63	0.87	0.77

Table 3: Results for Utilized Rank Decomposition on CIFAR dataset for different architectures.

5.6. Pretraining on ViTB-16

Here, we present the results of analyzing VitB-16 architecture trained from scratch on ImageNet and finetuned from a model pretrained in a self-supervised fashion [45]. All results correspond to networks decomposed and finetuned to respect the rank found from binary search.

	Orig Acc (%)	Orig ALU (%)	Proj Acc (%) (Δ)	True ALU (%)	Params Ratio	Flops Ratio
ViTB16 - scratch	80.9	94	80.7 (-0.2)	35	0.48	0.33
ViTB16, FT, Lin	81.7	97	$84.0^{\dagger}$	69	0.87	0.74
ViTB16, FT, E2E	85.3	97	85.1 (-0.2)	57	0.79	0.54

Table 4: Results for Utilized Rank Decomposition for ViTB-16 trained with and without self supervised training [45] † The increase in accuracy for linear models after finetuning with decomposed layers is an unfair comparison since the original network only finetuned the linear head.

5.7. ViTB-16 with different accuracy drop tolerance, ϵ

Here, we present the results of analyzing ViTB-16 architecture trained from scratch with varying accuracy drop tolerance per layer, per transformation. All results correspond to networks decomposed and finetuned to respect the rank found from binary search.

	Acc (%)	ALU (%)	Params Ratio	Flops Ratio
ViTB16: Original	80.9	93.8	1.00	1.00
ViTB16: $\epsilon = 0.01$	81.2	57.9	0.76	0.58
ViTB16: $\epsilon = 0.05$	80.8	38.7	0.54	0.37
ViTB16: $\epsilon = 0.1$	80.7	34.6	0.48	0.33
ViTB16: $\epsilon = 0.5$	79.9	22.5	0.31	0.20

Table 5: ViTB-16 pretrained network from torchvision, analyzed for dimensions with varying ϵ (percentage accuracy drop tolerance per transformation per layer).

5.8. Literature Review

Determining the rank of learning subspaces has garnered a lot of interest due to its theoretical implications on capacity and generalization of neural networks and its application to model compression. Theoretically rigorous works that find the rank of learning subspaces often show results on small networks and are unable to scale due to computational intractability [29, 37]. On large-scale networks, the low rank nature is assumed and empirically shown to give good results. Many of them exploit the low rank nature of weight matrices to reduce the number of parameters in neural networks by factoring the learned weights in each layer into products of low rank matrices [58, 7, 19, 55, 21, 23, 31, 33, 12, 46, 61].

Different approaches define *intrinsic* rank differently. Most works find the rank of the weight matrices using matrix factorizations like the SVD [55, 21]. Some works constrain this rank statically based on the singular values of the W matrix [27, 24, 23, 31, 19, 26], while others learn the rank as part of the optimization procedure [20, 42, 52]. Rather than predefining the rank via a factorization, another technique that has been used is to construct approximate low rank projection matrices by leveraging the distributional Johnson-Lindenstrauss lemma [22, 51] via random projections. These approaches differ from ours in that we project our weights onto the subspaces produced from the input and output activations, which is a architecture-dependent *and* data-dependent approach. Low rank projection based approaches have been applied to transformers [50] in the past by projecting the weights onto a low rank subspace such as in Linformer [51] which sought to reduce the $O(n^2)$ self-attention complexity or SliceGPT [2] which uses PCA projections to prune large language models.

LoRA [18] has become the de-facto standard of finetuning large models on downstream tasks. It assumes that the weight updates are low rank and can be restricted to a low-dimensional subspace. It does not restrict the rank of the final, fused weights. We differ from LoRA in that we study and limit the rank of the weights. Our formulation remains compatible with LoRA finetuning on downstream tasks. In parallel, there are many works that achieve efficiency by quantization, pruning, and knowledge distillation [10, 9, 16, 3]. In this work, we focus on efficiency via low rank decompositions, and expect that our resulting networks to remain compatible with many of these techniques.