Feature Space Saturation during Training

Mats L. Richter[1]
matrichter@uni-osnabrueck.de
Justin Shenk[2]
shenkjustin@gmail.com
Wolf Byttner[3]
wolf@byttner.org
Anna Wiedenroth[1]
awiedenroth@uni-osnabrueck.de
Mikael Huss[4]
mikael.huss@gmail.com

[1] Institute of Cognitive Science
University of Osnabrueck
Osnabrueck, Germany

[2] VisioLab
Rheinsberger Str. 76/77
Berlin, Germany

[3] Rapid Health Ltd
London, UK

[4] Peltarion
Holländargatan 17
Stockholm, Sweden

Abstract

We propose layer saturation - a simple, online-computable method for analyzing the information processing in neural networks. First, we show that a layer's output can be restricted to an eigenspace of its covariance matrix without performance loss. We propose a computationally lightweight method that approximates the covariance matrix during training. From the dimension of its relevant eigenspace we derive *layer saturation* - the ratio between the eigenspace dimension and layer width. We show evidence that saturation indicates which layers contribute to network performance. We demonstrate how to alter layer saturation in a neural network by changing network depth, filter sizes and input resolution. Finally we show that pathological patterns of saturation are indicative of parameter inefficiencies caused by a mismatch between input resolution and neural architecture.

1 Introduction

In recent years various techniques have been proposed for exploring the properties of neural network layers. Understanding how neural networks process information and how this processing may be influenced is vital for designing more efficient and better performing neural architectures. The works of Zeiler *et al*. [20], Szegedy *et al*. [21] and Yosinski *et al*. [18] are examples of experimental work that show the boundaries and limits of generalization and transferability of features. Recent works by Raghu *et al*. [9] and Alain *et al*. [1] propose techniques that allow for a deeper analysis of networks on a layer wise level.

The common problem with these and other techniques for analyzing the properties of neural networks is their complexity and computational inefficiency, which makes them impractical to use in neural architecture development or in more quantitative studies [1, 9, 22].

This work shows, that a simple, on-line computable property like the covariance matrix of the layer outputs is able to give interesting insights into the dynamics of the inference

process. To enable practical application, we provide a technique to efficiently compute the covariance matrix. We show how to use Principal Component Analysis (PCA) to project the output of all layers into low-dimensional spaces while not negatively affecting predictive performance. We refer to these subspaces as *relevant eigenspaces*.

Based on these findings, we derive *saturation* as a metric for analyzing the dynamics of the inference process. Similar to the work of Alain *et al*. [1], saturation can be thought of as a level indicator or thermometer, showing the complexity of the processing in the respective layers. By analyzing the distribution of saturation values within the network, we identify the "tail pattern" as a pathological symptom of a parameter-inefficient inference process. Finally, we propose simple, saturation-based strategies for altering the neural architecture to resolve such parameter inefficiencies.

2 Related work

In this work, we are interested in analyzing convolutional neural network models layer by layer. The most notable inspiration for this work is SVCCA by Raghu *et al*. [9] as well as the follow-up work by Morcos *et al*. [7], who use singular value decomposition for comparing the learned features of different models and layers. Another inspiration for this paper is Montavon *et al*. [6], in which kernel PCA with radial basis functions alongside linear classifiers are used to perform their analysis. Functionally similar to saturation are logistic regression probes proposed by Alain *et al*. [1], which will be utilized in this work as well in order to relate saturation patterns to parameter-inefficiencies caused by unproductive layers. Saturation was initially proposed by Shenk [13] and applied to model parameterization by Shenk *et al*. [14]. The saturation metric is used by our follow-up work to study the role of the input resolution in neural network training [10]. Further follow-up publications are basing proposed design guidelines for neural architectures on insights gained by a saturation-based analysis [11, 12].

3 Layer eigenspaces

In this section, we will explore the properties of the variance eigenspaces of each layer's feature space in order to motivate the derivation of the metric saturation. First, the methodology of computing the layer wise variance eigenspaces during training is described. This is followed by an experimental part, where we demonstrate that the eigendirections of the highest variance contain most of the information required to solve the trained classification task. We will further show that relevant eigenspaces exist and that they contain fewer dimensions than the original feature spaces of the network. Based on these findings, we will introduce saturation as a metric for studying the inference dynamics of neural network models.

3.1 Computing variance eigenspaces and relevant eigenspaces

Below is a brief discussion of the method we use to compute variance eigenspaces and relevant eigenspaces in our experiments. We apply PCA on the layer output to determine the eigenspace of the layer's features. Then we sum the largest eigenvalues that explain a percentage δ of the layer output variance. The space spanned by these eigenvectors is the

variance eigenspace. In this way, we find candidate eigenspaces for the layer. This process is described in section 3.2.

We establish that the layer's output is contained in the eigenspace as follows. We project all validation set output vectors into the space and determine whether the network's validation performance changes. To do so, we add special projection layers that only change the output at validation time. This we call a *projected network*. We then apply Student's paired t-test to determine if the network validation performance difference between the projected and the normal network is statistically significant. We pick the smallest δ such that there is not a statistically significant difference (p<.01). The details are in section 3.3. We consider the eigenspaces of projected networks with no statistically significant changes in performance as an approximation of the *relevant eigenspaces*.

3.2 Finding layer eigenspaces

A common problem of many analysis tools for neural networks is, that they are resource intensive to compute. For example, logistic regression probes by Alain and Bengio [1] and SVCCA by Raghu *et al.* [9] can require significantly more computation time and RAM than the training of the model, especially for large datasets and models. For practical application, this is a significant drawback. Ideally, the analysis can be conducted during training with little computational overhead to minimize the cycle time of experiments. For this reason, we propose an on-line algorithm for computing the covariance matrix during the regular forward pass. In this section, we are interested in finding a subspace of the layer output space to which we can restrict layer output vectors $z_{l,i}$ without changing the network's validation performance. We use PCA on the layer output matrix $A_l := (z_{l,1}, ..., z_{l,n})$ of n samples at training time, thus $A_l \in \mathbb{R}^{n \times w}$ where w is the layer width. To do this efficiently, we compute the covariance matrix $Q(Z_l, Z_l)$, where $Z_l := \sum_{i=1}^{n}(z_{l,i})/n$, using the covariance approximation algorithm between two random variables X and Y with n samples:

$$Q(X,Y) = \frac{\sum_{i=1}^{n} x_i y_i}{n} - \frac{(\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n^2} \tag{1}$$

We make this computation more efficient by exploiting the shape of the layer output matrix A_l: We compute $\sum_{i=1}^{n} x_i y_i$ for all feature combinations in layer l by calculating the running squares $\sum_{b=0}^{B} A_{l,b}^T A_{l,b}$ of the batch output matrices $A_{l,b}$ where $b \in \{0, ..., B-1\}$ for B batches. We replace $\frac{(\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n^2}$ by the outer product $\bar{A}_l \bigotimes \bar{A}_l$ of the sample mean $\bar{A}_l$. This is the running sum of all outputs $z_{l,k}$, where $k \in \{0, ..., n\}$ at training time, divided by the total number of training samples n. The final formula for covariance approximation is then:

$$Q(Z_l, Z_l) = \frac{\sum_{b=0}^{B} A_{l,b}^T A_{l,b}}{n} - (\bar{A}_l \bigotimes \bar{A}_l) \tag{2}$$

Since we only store the sum of squares, the running mean and the number of observed samples, we require constant memory and computation is done batch-wise. The algorithm requires roughly the same number of computations as the processing of a forward pass of the respective layer does; thus we compute saturation after every epoch. The variables are reset at the beginning of each epoch to minimize the bias induced by weight updates during training. Our algorithm uses a thread-save common value store on a single compute device or node, which furthermore allows to update the covariance matrix asynchronous when the network is trained in a distributed manner.

In convolutional layers, we treat every kernel position as an individual observation.[1] The advantage of this strategy is that no information is lost, while keeping Q at a manageable size. This strategy was proposed by Raghu *et al.* [9] after their initial publication and Garg *et al.* [3], who use it in their PCA-based pruning strategy for CNNs.

To determine the eigenspace E_l^k such that the projection of output vectors $z_{l,i}$ to E_l^k is as lossless as possible, we find $k := arg\,max(\frac{\sum\lambda_k}{N}) \leq \delta$ where $\sum\lambda_k$ is the sum of the largest k eigenvalues.[2] This technique is similar to how Raghu *et al.* [9] apply singular value decomposition in SVCCA - that study fixes δ to 99% of the variance. Pruning strategies by Garg *et al.* [3] and Chakraborty *et al.* [2] settle on δ of 99.9% and 99% respectively.

3.3 Exploring the properties of projected networks

Our approximation of the *relevant eigenspace* is by the nature of PCA a linear approximation. Since neural networks are non-linear models, it is not guaranteed that a linear subspace can capture the information relevant for the inference process accurately. Therefore, we demonstrate experimentally in this section the usefulness of PCA for this application. First, we study the effects of different values for δ and show that variance eigenspaces contain more information necessary for the inference process than randomly chosen orthonormal subspaces of equal dimensionality. We will then demonstrate, on VGG13 and VGG19, that δ can be chosen such that the variance eigenspaces in all layers of the network are relevant eigenspaces. To study the effect of different values for δ we introduce PCA-Layers, inserted after any non-output (fully connected and convolutional) layer l. At training time PCA-Layers are pass-through layers. At testing time they project the output of the preceding layer A_l into the variance eigenspace E_l^k. This is done by multiplying A_l with the projection matrix, $P_{E_l^k} = E_l^k(E_l^k)^T$. For convolutional layers, we compute the (1×1) convolution $A_l * vec(P_{E_l^k})$. The net effect is to turn a dataset problem into a network parameter one; we study the properties of samples by changing the projection parameters.

First, we study how well the eigenspaces are able to maintain the predictive performance of the model compared to random orthonormal subspaces of equal dimensionality. We train 20 different variations of VGG-style networks on CIFAR10 [5].[3] The *relative performance* of a network for a value of δ is the ratio between the test accuracy with enabled and disabled PCA-Layers. We indirectly control the dimensionality of E_l^k with δ, which is set globally for the entire network.

The results in Fig. 1 show that the eigenspaces of the layer outputs are able to maintain the information better than equally sized random orthonormal projections. Relative performance given lower values of δ also degrades slower than random projections.

Next, we explore the upper bounds of this projection by finding a $\delta < 1$ for a trained model such that the difference in predictive performance to the model with disabled PCA-Layers is insignificant. For each network we compare a network's projected and unprojected CIFAR10 *validation set* performance. We use the eigenspace E_l^k computed during training.

[1] This turns an output-tensor of shape (samples $\times$ height $\times$ width $\times$ filters) into a data matrix of shape (samples $\cdot$ height $\cdot$ width $\times$ filters).

[2] In order to achieve more accurate results on small networks, we treat the threshold as soft. If $max(\sum\lambda_k) > \delta \leq max(\sum\lambda_k) + 0.02$ an additional dimension is added. If $dim E_l^k = 0$, because a single dimension exceeds the δ-threshold, we set $E := \{v_1\}$.

[3] In order to include networks of different width (filter sizes) and depth (number of layer) we trained VGG[11,13,16,19] as well as variations of all those architectures with filter sizes reduced by a factor of [2,4,8,16]. All models were trained on a batch size of 128 using the Adam optimizer and a learning rate of 0.001 for 30 epochs.

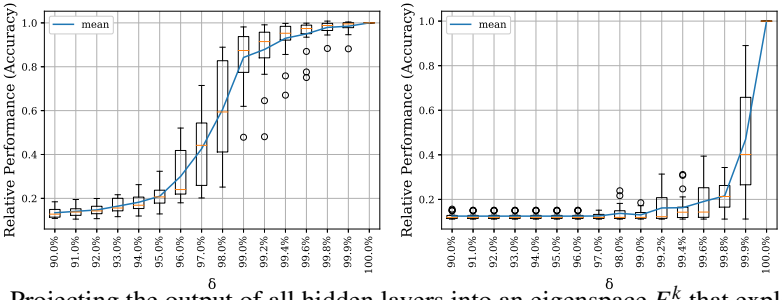


Figure 1: Projecting the output of all hidden layers into an eigenspace E_l^k that explains δ % of the variance (left) maintains predictive performance better than projecting the layers into a random orthonormal subspace of the same size (right).

We test whether a network's performance changes relative to the unprojected performance. Applying Student's two-tailed t-test to VGG13 (Table 1) shows that at $\delta = 0.996$ we cannot

δ	$\mu_{p-\bar{p}}$	σ	t	p
0.9999	-0.0004	0.0008	-2.42	0.023
0.9998	-0.0005	0.0009	-2.81	0.010
0.999	**-0.0017**	0.0016	-5.50	0.000
0.998	-0.0017	0.0022	-3.92	0.001
0.996	-0.0005	0.0030	-0.91	0.371
0.994	0.0037	0.0043	4.45	0.000
0.99	0.0178	0.0136	6.68	0.000

Table 1: Based on $n = 26$ trained VGG13 models, we find that $\delta = 0.996$ allows us to approximate the relevant eigenspace in all layers, such that the performance p becomes indistinguishable from the unprojected model. This is based on a t-statistic (student's two-tailed t-test), $\mu \neq 0$ (p<.01), selected δ at $\alpha = 0.1$. $\mu \neq 0$ in *italics*. $\mu_{p-\bar{p}}$ is the average pairwise distance between the test accuracies of unprojected and projected networks.

distinguish the projected network from the base network at significance p< .1. However, at $\delta = 0.999$, the projected network **outperforms** the base network by 0.17% at significance p< .01. This means we can **increase** validation performance by **restricting** layer output to a subspace known at training time.

VGG13's validation performance improves in the range $\delta \in [0.998, 0.9998]$, with a maximum improvement 0.17%.[4] This shows that there are noisy or not generalizing feature dimensions in the layer output; we use this to develop the concept of *layer saturation* and improve network performance further.

We also study saturation behavior in VGG19 (Table 2). At $\delta = 0.9995$, the t-value is insignificant. We need 1841 ± 86 of 5860 dimensions to describe the data. Thus, we can **remove** two out of three dimensions without changing network performance.

[4]The full results are available in the supplementary material.

δ	t	p	μ_{Sat}	σ_{Sat}	$\sum_l dimE_l^k$
0.9997	-2.82	0.008	51.2	0.7	2071 ± 93
0.9996	-1.28	0.208	48.8	0.6	1938 ± 88
0.9995	**-0.352**	0.727	47.1	0.7	1841 ± 86
0.9994	2.18	0.035	45.6	0.7	1766 ± 84
0.9993	2.62	0.012	44.5	0.7	1705 ± 83

Table 2: The table depicts VGG19 t-statistic, as well as the mean and standard deviation of saturation and the sum of intrinsic dimensions ($\sum_l dimE_l^k$). It is also possible to find a relevant eigenspace for VGG19. Based on $n = 40$ trained models, the relevant eigenspace of the projected network can be computed by using the depicted values of δ using the same $alpha$ as in Table 1. Interestingly, from the 5860 dimension in all feature spaces combined, only up to 2071 ± 93 are used to construct all relevant eigenspaces in the network.

3.4 Saturation

Based on previous results, we are interested in analyzing the sequence of eigenspaces that the data traverses during the forward pass. For this purpose, we propose layer saturation:

$$s_l = \frac{dim\, E_l^k}{dim\, Z_l} \tag{3}$$

Intuitively this *layer saturation* ratio represents the proportion of spatial dimensions occupied by the information in a layer l. Therefore we can think of saturation as a level indicator which shows the fraction of useful dimensions in the output space. We can analyze and compare the inference dynamics of multiple networks by plotting the saturation level of each network layer.

4 Exploring the Properties of Saturation

4.1 Saturation patterns are stable over different model runs

The first question we want to answer is whether s_l is stable enough to observe meaningful patterns. To investigate, we train VGG16 and ResNet18 100 times using the same setup as in section 3.3.

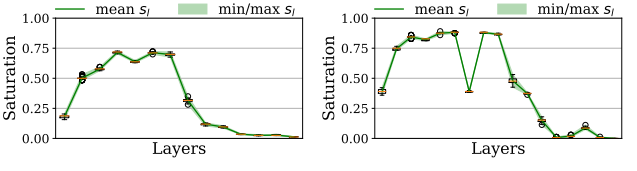


Figure 2: Saturation only deviates slightly for VGG16 (left) and ResNet18 (right) between training using the same setup for 100 training, which allows us to analyze saturation-patterns with little bias caused by random fluctuations.

From the result in Fig. 2 we observe that for ResNet18 as well as VGG16 the emerging saturation patterns are stable. In fact, the standard deviation σ_s of VGG16's saturation is 0.281 while the standard deviation of the accuracy-performance σ_{acc} from the same model

is significantly higher at 0.511, while both values are bound in [0, 1]. The same can be said for ResNet18, where $\sigma_s = 0.353$ and $\sigma_{acc} = 0.523$. Based on these observations, we can conclude that saturation is sufficiently stable to allow for the analysis of convolutional neural networks.

4.2 Saturation and Network Width

We find [11] that models with lower average saturation s_μ tend to perform better than architecturally similar models with higher s_μ (see later in Fig. 7). There are two main factors influencing s_μ [11]: Problem difficulty, increasing s_μ, and the width of the network, decreasing s_μ. The width of a network is the number of filters or units in each layer. Effectively, more difficult problems require more capacity in each layer and thus more computational resources to be processed effectively. Therefore, finding a sweet spot for s_μ for a given architecture and dataset by scaling its width optimizes the efficiency of the model for the given setup.

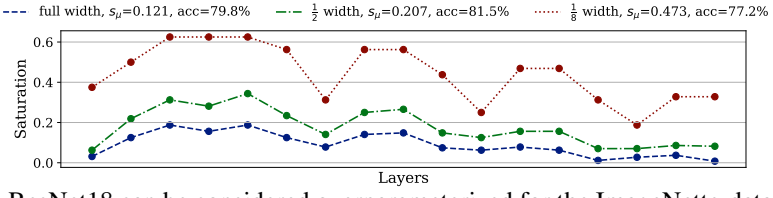


Figure 3: ResNet18 can be considered overparameterized for the ImageNette-dataset, which is apparent from the low saturation level (blue line). Halving the number of filters also halves the computational and memory footprint while slightly improving performance (green line). Reducing the width of the network too much results in a poor performing, underparameterized model (red line). The saturation of all layers (dots) are depicted in the order of the forward pass from input to output.

To demonstrate this, we train ResNet18 on ImageNette, a 10-class subset of ImageNet, using an 224×224 input resolution. To demonstrate over-, under- and well- parameterized variants of the model, we scale its width by a factor of 1, $\frac{1}{8}$ and $\frac{1}{2}$ respectively. From the results in Fig. 3, we can see that ResNet18 with $\frac{1}{2}$ width provides the best performance while requiring half the memory and FLOPS compared to ResNet18 with full width. We attribute the slightly poorer predictive performance of the full-width-model to overfitting.

From additional experiments on VGG and ResNet-style models, we find that an average saturation s_μ of roughly 20% to 30% delivers the best performance in the tested scenarios, assuming all networks have roughly the same relative distribution in saturation across layers and follow the conventional pyramidal structure of modern classifiers [4, 15, 17]. Models with s_μ below this interval provide approximately similar performance at reduced efficiency, while models with higher s_μ will degrade in predictive performance, as figure 3 and the results of our follow-up work [11] demonstrate.

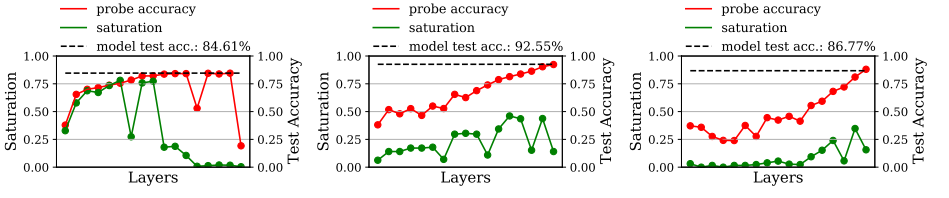
By adjusting the scaling of the network's width based on the current s_μ, the network can be optimized in an informed manner. While it is possible to manipulate individual layers similarly, we advise against it for multiple reason. First, the saturation of individual layers is also subject to noise induced by some components like (1×1) convolutions, downsampling and skip connections, which makes clear rules of action hard to quantify. Second, this practice can theoretically mask true inefficiencies like tail-patterns (see sections 4.3 and 5). For example, strongly overparameterizing the productive part of the model brings the saturation

of these layers down until tail-patterns in the (less overparameterized) unproductive layers become hard to locate.

4.3 Saturation patterns provide insight into the distribution of the inference process across the network

We move on to investigate the emerging saturation patterns when viewing the saturation levels of the individual layers in order of the forward pass. To accomplish this, we use logistic regression probes by Alain *et al.* [1]. Logistic regression probes measure the intermediate quality of the solution in a layer by training a logistic regression classifier on its output solving the same task as the model, allowing us to measure solution progress. The size and complexity of extracted features increases with every consecutive convolutional layer due to the expansion of the receptive field. We investigate whether this influences saturation and whether this can be connected to some patterns observed in logistic regression probes.

We train ResNet18 on Cifar10 (deliberately choosing a low-resolution dataset to avoid side effects caused by the addition of information by increasing the resolution) using three distinct input resolutions: 32×32 (Cifar10 native), 224×224 (ResNet18's design resolution) and 1024×1024 (intentionally over-sized). We compute the test accuracy of logistic regression probes and saturation on every convolutional layer.



(a) 32×32 (Cifar10 native) (b) 224×224 (model standard) (c) 1024×1024

Figure 4: The input resolution changes how the inference is distributed in the model, observable from probe accuracy and saturation patterns. Even distribution and best accuracy is achieved at the design resolution of the model (224×224 pixels).

In Fig. 4 we observe three distinct patterns in saturation and probe performances. At Cifar10's native resolution of 32×32 pixels, we observe a high saturated sequence of layers followed by a low saturated sequence. Only the high saturated sequence is contributing qualitatively to the inference process according to the logistic regression probes. This pattern is inverted, when drastically over-sizing the resolution to 1024×1024 pixels. The best predictive performance and most even distribution of the inference process is achieved when training ResNet18 on its original design resolution of 224×224 pixels. According to the logistic regression probes, we can see that low saturated subsequences of layers are indicative of unproductive layers in the network. We refer to this pattern indicating a parameter inefficiency as *tail pattern*. From additional experiments on a variety of models and datasets (see supplementary material for detailed results) we deduct a definition of a tail pattern:

A tail is a subsequence of at least 3 consecutive layers in a feed-forward neural architecture with an average saturation at least 50% lower relative to the average saturation of the rest of the network.

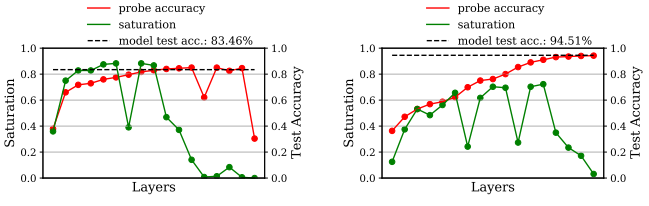
This definition is imperfect and does not fit all patterns that we would visually classify

as similar to a tail pattern. However, to test the implications of the presence of such patterns, it is necessary to have a more rigorous definition than purely visual observation.

The experiment of this section demonstrates that we can use the saturation values of the individual layers to gain insights on how the inference process is distributed among the networks layers. The insights shown in this work are expanded on upon in our follow-up works [10, 12], where we explain these inefficiencies experimentally with the expansion of the receptive field.

5 Optimizing Convolutional Neural Networks with Saturation

In this section, we focus on the practical application of saturation for optimizing neural architectures. While, as previously demonstrated, it is possible to remove a tail pattern and therefore parameter inefficiency by altering the input resolution of the model (see Fig. 4), an increase in resolution scales quadratic with the memory footprint and the FLOPs required per forward pass, making it an expensive solution. We therefore present two simple strategies for altering the neural architecture on a few selected examples as an alternative way of removing this inefficiency. The first strategy is to bring the receptive field of the feature extractor closer to the input resolution. To optimize ResNet18 for a 32×32 pixel resolution, we replace the first two layers, which are sometimes referred to as "stem" by a 3×3 convolution, with stride size 1. By removing the initial downsampling layers, we reduce the receptive field size of ResNet18 from 435×435 to 109×109 pixels.

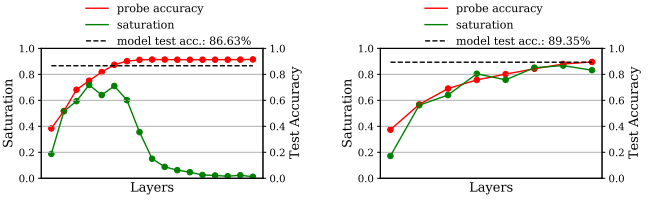


(a) ResNet18 (b) ResNet18 (no Stem)

Figure 5: Reducing the size of the receptive field (here by removing the two initial downsampling layers) removes the tail pattern and improves the test accuracy on Cifar10 at native resolution.

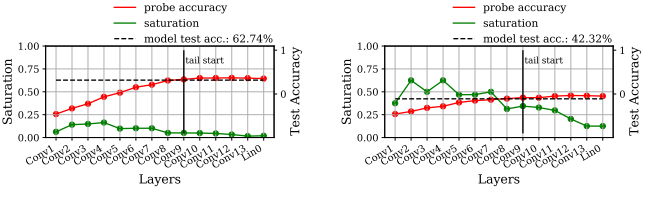
In Fig. 5 we can see an improved version of ResNet18 that was trained on Cifar10 at 32×32 pixel resolution. The performance of the model drastically increases to over 94% accuracy and the tail pattern is removed. The computations required for a forward pass of a single image increase from 0.04 GFLOPS to 0.56 GFLOPS. For comparison, increasing the input resolution to the original resolution of ResNet18, which is 224×224 pixels, yields the same result but requires 1.83 GFLOPs per forward pass per image.

Another, more simple possibility to remove the tail pattern is to remove the layers of the tail-pattern and retrain the model. A quick test on VGG19 (see Fig. 6) demonstrates that retraining the truncated model reduced the number of parameters from 20,170,826 to 2,363,338, the FLOPS needed per forward pass from 399 MFLOPS to 229 MFLOPS. The accuracy meanwhile improved from 86.63% to 89.35%.



(a) VGG19 (b) VGG19 (truncated)

Figure 6: Removing the tail layers of VGG19 and retraining the truncated model reduced the computational and memory footprint, meanwhile improving the performance slightly. Both models are trained on Cifar10 at native resolution.



(a) VGG16 full width (b) VGG16 $\frac{1}{8}$ width

Figure 7: The width of the VGG16 trained on the ImageWoof dataset at 32×32 pixel input resolution does not change the size of the tail, meaning that inefficiencies caused by low-saturated tails and under-parameterized layers are independent.

Generally, the inefficiencies of *width* (too high / low s_μ) and *depth* (tail pattern) can be considered independent, see Fig. 7. Both models are trained on the ImageWoof-dataset, another 10-class subset of ImageNet, and the images were downsampled to 32×32 pixels. The resulting difference in saturation and predictive performance does not affect the number of inactive layers caused by the tail pattern. Based on these observations, we conclude that width and depth can be treated as mostly independent factors when optimizing an architecture for a fixed input resolution, as long as the saturation is distributed similarly. For this reason, optimizing the depth should be done first to guarantee an even distribution of saturation. Then the width can be scaled to put s_μ into the sweet-spot of $s_\mu \in (20\%, 30\%)$ (see Section 4.2).

6 Conclusion

In this work, we propose a novel, on-line computable metric "saturation" for analyzing neural networks layer by layer. We base the development of saturation on the results of experiments that demonstrate the relevance of low-dimensional eigenspaces of the hidden layer output for the quality of the inference process. We show that saturation yields reliable and reproducible results. Over a series of experiments, we show that saturation can be used to detect two independent kinds of parameter-inefficiency in neural network architectures: Tail-patterns indicating unproductive layers, and over- or underparamterization that can be detected by the average saturation of the network. Based on these findings, we propose simple strategies and guidelines to remove those inefficiencies from the neural architecture, gaining predictive performance and efficiency.

References

[1] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=HJ4-rAVtl.

[2] Indranil Chakraborty, Deboleena Roy, Isha Garg, Aayush Ankit, and Kaushik Roy. PCA-driven hybrid network design for enabling intelligence at the edge. *ArXiv*, abs/1906.01493, 2019.

[3] Isha Garg, Priyadarshini Panda, and Kaushik Roy. A low effort approach to structured CNN design using PCA. *CoRR*, abs/1812.06224, 2018. URL http://arxiv.org/abs/1812.06224.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[5] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-10 (Canadian Institute for Advanced Research). 2010. URL http://www.cs.toronto.edu/~kriz/cifar.html.

[6] Grégoire Montavon, Klaus-Robert Müller, and Mikio L. Braun. Layer-wise analysis of deep networks with gaussian kernels. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1678–1686. Curran Associates, Inc., 2010. URL http://papers.nips.cc/paper/4061-layer-wise-analysis-of-deep-networks-with-gaussian-kernels.pdf.

[7] Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 5732–5741. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7815-insights-on-representational-similarity-in-neural-networks-with-canonical-correlation.pdf.

[8] Quynh Nguyen, Mahesh Chandra Mukkamala, and Matthias Hein. Neural networks should be wide enough to learn disconnected decision regions. *CoRR*, abs/1803.00094, 2018.

[9] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. SVCCA: singular vector canonical correlation analysis for deep learning dynamics and interpretability. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6076–6085, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/dc6a7e655d7e5840e66733e9ee67cc69-Abstract.html.

[10] Mats L. Richter, Wolf Byttner, Ulf Krumnack, Anna Wiedenroth, Ludwig Schallner, and Justin Shenk. (Input) Size Matters for CNN Classifiers. In Igor Farkaš, Paolo Masulli, Sebastian Otte, and Stefan Wermter, editors, *Artificial Neural Networks and Machine Learning – ICANN 2021*, pages 133–144, Cham, 2021. Springer International Publishing. ISBN 978-3-030-86340-1.

[11] Mats L. Richter, Leila Malihi, Anne-Kathrin Patricia Windler, and Ulf Krumnack. Exploring the properties and evolution of neural network eigenspaces during training, 2021. URL https://arxiv.org/abs/2106.09526.

[12] Mats L. Richter, Julius Schöning, Anna Wiedenroth, and Ulf Krumnack. Should You Go Deeper? Optimizing Convolutional Neural Network Architectures without Training by Receptive Field Analysis. In *International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2021 (in print). URL https://arxiv.org/abs/2106.12307.

[13] Justin Shenk. *Spectral Decomposition for Live Guidance of Neural Network Architecture Design*. Unpublished master's thesis, University of Osnabrück, 2018.

[14] Justin Shenk, Mats L. Richter, Anders Arpteg, and Mikael Huss. Spectral analysis of latent representations. *CoRR*, abs/1907.08589, 2019. URL http://arxiv.org/abs/1907.08589.

[15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://arxiv.org/abs/1409.1556.

[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. URL http://arxiv.org/abs/1512.00567.

[17] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019. URL http://arxiv.org/abs/1905.11946.

[18] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3320–3328. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf.

[19] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.

[20] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10590-1.

[21] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *CoRR*, abs/1611.03530, 2016. URL http://arxiv.org/abs/1611.03530.

[22] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921–2929, 2016.