

UNIVERSITY OF OSNABRÜCK
INSTITUTE OF COGNITIVE SCIENCE

DISSERTATION

Towards Efficient Convolutional Neural Architecture Design

EXPLORING THE PROPERTIES OF NEURAL ARCHITECTURES USING SPECTRAL
DECOMPOSITION AND LOGISTIC REGRESSION PROBES

April 21, 2022

Author:
Mats L. RICHTER

Supervisor:
Prof. Dr. Gunther
HEIDEMANN

Disputation
April 20, 2022

Contents

Contents	i
List of Tables	vi
List of Figures	viii
Abbreviations	xxvi
Symbols	xxvii
1 Introduction	1
1.1 Motivation	1
1.2 On the Usage of the Third-Person Plural	2
1.3 Structure of this Work	2
2 Basics	4
2.1 Deep Convolutional Neural Network Classifiers	4
2.2 Benchmarks	5
2.2.1 MNIST	5
2.2.2 Cifar10 / Cifar100	7
2.2.3 ImageNet	9
2.3 The Convolution Operation	11
2.4 Conventions in Training and Evaluation Methodology	13
2.4.1 Loss	14
2.4.2 Preprocessing and Data Augmentation	15
2.4.3 Optimizers	17
2.4.4 Models	19
2.4.5 General Methodology of Training and Evaluation	20
2.4.6 Evaluation Metrics and Methodology	21
2.5 Metrics	21
3 The Current State of Convolutional Neural Architecture Design	24
3.1 Design Criteria	24
3.1.1 Predictive Performance	24
3.1.2 Memory Footprint	26
3.1.3 Computational Efficiency	26
3.1.4 Parameter Efficiency	27

3.2	Basic Structural Conventions in Convolutional Neural Network Architectures	28
3.2.1	Classifier	29
3.2.2	Feature Extractor	29
3.2.3	Pyramidal Architectures	30
3.2.4	Fully Convolutional Neural Networks	31
3.3	Conventions in Neural Architecture Design	32
3.3.1	Convolutional Layers	32
3.3.2	Stem	35
3.3.3	Building Block-Style Feature Extractors	36
3.3.4	Skip Connections	36
3.3.5	Bottlenecks and Inverted Bottlenecks	39
3.3.6	Attention Mechanisms	41
4	Analyzing Convolutional Neural Networks	48
4.1	Related Work	48
4.2	Feature Space Analysis	51
4.3	Logistic Regression Probes	54
4.4	The Receptive Field	57
4.4.1	Minimum and Maximum Receptive Field	59
5	Problem Description	61
5.1	Developing a Practical Technique for Analyzing Convolutional Neural Networks	61
5.2	Diagnosing and Resolving Architectural Inefficiencies in Convolutional Neural Networks	62
6	Open-Source Software and Technical Foundation	64
6.1	Delve: Framework for Experiment Control	64
6.1.1	Overview	64
6.1.2	Experiment Control and Integration in the Training Loop	66
6.1.3	Covariance Approximation	67
6.1.4	Supported Layer-Based Information	69
6.1.5	Automated Analysis, Logging and Saving	70
6.1.6	PCA-Layers	70
6.2	PHD-Lab: Experiment Automation Framework	72
6.2.1	Overview	72
6.2.2	Automated Assembly and Execution of Experiment-Setups	74
6.2.3	Configuration	74

6.2.4	Logging	75
6.2.5	Logistic Regression Probes	78
6.2.6	Recovery and Reproducibility	78
7	Analyzing Neural Network Layers using Principal Component Analysis	80
7.1	General Concept and Methodology of the Experiments	80
7.2	Demonstrating That Projected Networks can Maintain Some Predictive Quality.	82
7.2.1	Methodology	83
7.2.2	Results	85
7.3	Investigating the Information Content of Projected Networks .	87
7.3.1	Methodology	87
7.3.2	Results	89
7.4	Finding Relevant Eigenspaces	89
7.4.1	Methodology	90
7.4.2	Results	91
7.5	Conclusion	91
8	Understanding the Behavior of Layer-Based Analysis Tools	93
8.1	Saturation	93
8.2	Evolution and Stability of Saturation Patterns	94
8.2.1	Methodology	95
8.2.2	Results	95
8.3	Logistic Regression Probe Ablation Study	96
8.3.1	Methodology	97
8.3.2	Results	99
8.3.3	Implications for Experimental Setups Using Logistic Regression Probes	102
8.4	Studying the Average Saturation of Convolutional Neural Architectures	102
8.4.1	Methodology	103
8.4.2	Results	104
8.5	Understanding Saturation Patterns	105
8.5.1	Methodology	105
8.5.2	Results	106
8.6	Conclusion	110

9	Exploring the Relationship Between Input Resolution and Neural Architecture	112
9.1	The Effect of Input Resolution and Details on the Predictive Performance	113
9.1.1	Methodology	114
9.1.2	Results	115
9.2	The Role of the Size of Discriminatory Features in the Relation of Model and Input Resolution	115
9.2.1	Methodology	116
9.2.2	Results	117
9.3	Receptive Field Size and Tail Patterns	118
9.3.1	Methodology	119
9.3.2	Results	121
9.4	The Impact of Residual Connections on the Relation of Receptive Field Size and Tail Patterns	125
9.4.1	Methodology	125
9.4.2	Results	126
9.5	Conclusion and Implications for Neural Architecture Design .	128
9.5.1	A Priori	128
9.5.2	Post Hoc	129
10	Predicting Architectural Inefficiencies in Convolutional Neural Networks	130
10.1	Attention Mechanisms and Their Effect on the Inference Process	131
10.1.1	Methodology	131
10.1.2	Results	132
10.2	Characteristics of Multipath Architectures	135
10.2.1	Methodology	137
10.2.2	Results	138
10.3	b_{min} can Predict Tail Patterns in Networks With Residual Connections	140
10.3.1	Methodology	140
10.3.2	Results	141
10.4	Conclusion	142
11	Implications for Neural Architecture Design	143
11.1	On the Goals of Architecture Optimization	143
11.2	Methodology	144
11.3	Optimizing the Width of Neural Architectures	145

11.4	Optimizing the Depth of Neural Architectures	149
12	Summary and Future Work	153
	References	155
A	Full <i>t</i>-Test Tables of VGG11, VGG13, VGG16 and ResNet18	167
B	On the Evolution of Saturation Patterns in Fully Connected and Convolutional Neural Networks During Training	170
	B.1 Methodology	170
	B.2 Results	171
C	Different Types of Tail Patterns - A Brief Explanation	174
D	Resolution Shifts the Distribution of the Inference Process on Different Models	177
	D.1 VGG16 - Cifar10	177
	D.2 VGG16 - MNIST	178
	D.3 ResNet18 - MNIST	179
	D.4 ResNet18 - TinyImageNet	180
	D.5 ResNet50 - Cifar10	181
E	Object Size Experiments on MNIST and ResNet models	182
	E.1 ResNet18 - Cifar10	182
	E.2 VGG16 - MNIST	183
	E.3 ResNet50 - Cifar10	184
	E.4 Reproducing Tail Patterns on ImageNet and iNaturalist	185
F	Predicting the Border Layers in Different Scenarios Using VGG	189
	F.1 Results	189
	F.2 VGG-models $\frac{1}{8}$ Filter Size - Cifar10	190
	F.3 VGG11 / VGG16 - TinyImageNet	191
	F.4 DenseNet18, 65 - Cifar10	192
G	Receptive Field Analysis With Partial Solution Heatmaps (Compilations from Chapter 9)	193

List of Tables

1	Convolutional Autoencoder Architecture.	84
2	Hyperparameters for the convolutional autoencoder.	84
3	When projecting all layers l of an autoencoder into their reduced eigenspaces E_l^k , the reconstructed examples are still recognizable, even when only a fraction of the dimensionality of the feature space is used in every layer. It is also worth noting that an eigenspace with only 53.3% of the feature space dimensionality is needed in the bottleneck of the autoencoder, where the feature space has the lowest dimensionality, to explain 99.9% of its variance. The images depicted here can also be found in Richter et al. (2021c).	86
4	VGG13 t -statistic, $\mu \neq 0$, selected δ at $\alpha = 0.01$ (n=26). $\mu \neq 0$ in <i>italics</i> . Note that projections for some values of δ improve performance.	91
5	Sum of projections in ResNet18 (n=15). $\mu_{diff} \neq 0$ (p=.95) in bold . μ_{diff} refers to the average difference in performance between the projected network at threshold value δ and the unprojected network. We also provide the standard-deviation of the accuracy σ_{sample} over all samples. $\mu(\sum dim E_l^k)$ refers to the average number of eigendirections in the entire network \pm the $\sigma(\sum dim E_l^k)$	92
6	Hyperparameters for the ablation study.	97
7	Hyperparameters of training setups for all models trained on Cifar10 in this experiment.	103
8	Hyperparameters for training ResNet18 on datasets of varying difficulty.	105
9	Hyperparameters for the resolution experiments.	114

10	Relative Top1-Accuracy: The table shows the predictive performance of trained models relative to a baseline model. The baseline uses the same model and architecture and is trained on images resized to 224×224 pixels. Training the models on 32×32 pixel images results in less than half the predictive performance in all tested scenarios relative to the baseline. Resizing the 32×32 images back to 224×224 results in a significant recovery of lost performance in all tested scenarios, despite the fact that no information is added by upscaling. These results were previously published in Richter et al. (2021a).	115
11	Hyperparameters for the experiments conducted on the VGG-family of networks and the modified VGG and ResNet models.	119
12	Hyperparameters used for training the ResNet variants with different attention mechanisms.	132
13	Hyperparameters used for training the ResNet variants with different attention mechanisms.	145
14	Sum of Projections in VGG11 (n=15). $\mu \neq 0$ (p=.99) in bold . . .	167
15	Sum of projections in VGG13 (n=26). $\mu \neq 0$ ($\alpha = 0.01$) in bold .	168
16	Sum of projections in VGG19 (n=40). $\mu \neq 0$ (p=.99) in bold . . .	169

List of Figures

1	A simple hypothetical example of a cat-dog-classifier: The classifier receives an input image and predicts a label from a set of discrete symbols.	4
2	The dataset consists of 10 classes of hand-written digits, the images are 28×28 binary images, which can be considered a very low dimensional visual domain compared to other popular datasets like Cifar10 and ImageNet. Due to its simplicity, MNIST is mostly used for proof of concept work.	6
3	Similar to MNIST, the Cifar datasets consist of uniformly sized low resolution (32×32 pixel) images, making them very economical for model training compared to higher resolution datasets like ImageNet. The datasets consist of colored real-world images. The classes are real-world objects. For these reasons, the Cifar-datasets can be considered more complex than MNIST. For the same reasons, these datasets can be used as more economical proxy problems for computationally demanding tasks such as ImageNet (Real et al. (2019); Zoph et al. (2018)).	8
4	The ImageNet dataset features RGB-images of various aspect ratios and image qualities belonging to a heterogeneous set of 1,000 classes. These properties render the ImageNet-dataset a very challenging classification problem, which makes it ideal for training well generalizing models. From the early days of deep learning until today improving the predictive performance of ImageNet-trained models can be considered an active field of research and central to innovations in deep convolutional neural network classifier design (Dosovitskiy et al. (2021); Khan et al. (2020); Krizhevsky et al. (2012); Tan, Le (2019)).	10
5	This example of a convolution operation using a 3×3 kernel illustrates how a convolutional layer integrates local context into a position in its output. The left matrix is the input feature map, the right matrix is the output feature map. The highlighted area illustrates the kernel being convolved over the input feature map. Every position of the output is computed from the region of the input feature map covered by the kernel.	11

6	A simple example of a convolution operation using a vertical Sobel filter (Kanopoulos et al. (1988)). The kernel (middle) is convolved over the input image (left), which results in a feature map (right) containing information on the presence of vertical edges.	12
7	Predictions of an ImageNet pretrained ResNet50. The texture of the figure 7(b) is transferred on figure 7(a) using the style transfer GAN by Ghiasi et al. (2017). While the cat is still recognizable the texture transfer results in the top 3 classes being closer to the texture image compared to the predictions of the original cat-image. This indicates that textures are weighted over the general silhouette of objects, suggesting a bottom-up pattern recognition process (Geirhos et al. (2019)).	12
8	A VGG13 convolutional neural network (Simonyan, Zisserman (2015)) was trained on MNIST without any data augmentation and is therefore not invariant towards rotations. The predictions made by this model vary for the same test-set image depending on the angle it was rotated.	16
9	To avoid scenarios like the one depicted in figure 8, where the model is not invariant towards certain properties of the input image, training images are randomly augmented. This example shows various combinations of data augmentation techniques applied to an image of Felix the horse. The model can be forced to be robust against these kinds of distortions, by training a model on randomly distorted versions of the original images. .	17
10	The optimizer influences the trajectory of the model on its error surface. In this example, a model with 2 parameters is trained using three different optimizers. The visualization shows, that properties of the optimizers influence the trajectory and as a result, whether a model may "get stuck" in local minima and saddle nodes (SGD, middle) a local optimum (RMSProb, right) or converge into good, in this case global, minimum (Adam, left).	18

11	Network families (connected via dotted lines) are composed of models that resemble different trade-offs between predictive performance and computational efficiency (measured in FLOPs to perform a forward pass on a single image). Computationally more demanding variants of the architecture achieve the best predictive performance than their more computationally efficient relatives.	22
12	The Graph shows the best performing architectures of the respective years (multiples if the benchmark was beaten multiple times during the year). Over 90% of the predictive performance can be explained by an exponential increase in parameters. This indicates that the increasing capacity of neural networks is the main driving factor behind improvements in cutting edge-models.	25
13	Networks in the same family (connected via dotted lines) that feature more parameters perform better, trading computational and parameter efficiency for predictive performance at diminishing returns.	27
14	Convolutional Neural Network classifiers can be divided into a feature extractor and a classifier part. The feature extractor is fully convolutional and extracts discriminatory features from the input image. The classifier consists of densely connected layers and produces a prediction based on the output of the feature extractor.	28
15	The feature extractor of VGG16 consists only of convolutional (yellow) and pooling-layers (orange). Note the pyramidal structure of the feature maps caused by the 4 downsampling layers (orange).	30
16	Two common ways convolutional layers are implemented. The variant depicted in (a) is commonly used in older architectures such as VGG16. The variant depicted in (b) used in more recent architectures such as MobileNet, EfficientNet and AmoebaNet. When processing convolutional layers, the feature map is padded (zero padding is the norm) to keep input and output shape independent of the kernel size. The output of the convolution-operation is fed into a batch-normalization to combat the vanishing-gradient problem and a non-linear activation function to make the layer non-linear.	33

17	To reduce parameters and the number of computations required to process the convolution, the convolution operation is sometimes decomposed into multiple operations. This can be achieved by separating the kernel into two 1-dimensional kernels of shape $k \times 1$ and $1 \times k$ (a). Another possibility (b) is to filter-wise and spatial processing by decomposing the convolution into a depth-wise separable convolution and a convolution with 1×1 kernel.	34
18	Stems are a sequence of layers at the input of a neural network that does not consist of the regular building blocks like the rest of the architecture. The main purpose of these layers is reducing the height and width of the feature map for consecutive layers. The first two layers of the depicted stem reduce the size of the feature map by a factor of 4, making the entire architecture more computationally efficient. Later architectures like ResNet and DenseNet adopt a simplified version of this stem, which only consists of the first two layers. This particular stem belongs to the GoogLeNet architecture by Szegedy et al. (2015).	35
19	The GoogLeNet architecture is one of the first architectures to rely on repeated building-blocks (Inception Modules) used throughout the network.	37
20	The original implementations of residual connections add the input of the building block to the pre-activation output of the same block, allowing signal and gradient to skip the encapsulated layers. The left version is used by ResNet18 and ResNet34. The right version is used by ResNet50, 101, 152 and 1001. . . .	38
21	A DenseBlock concatenates the output of every previous convolutional layer to the output of the current convolutional layer in an attempt to minimize the loss of information inside the DenseBlock.	39
22	MobileNetV2s inverted residual block follows a narrow-wide-narrow approach in contrast to ResNets wide-narrow-wide approach, allowing the feature extracting 3×3 convolution to detect more patterns by increasing the number of filters. The increase in computations is counteracted by using a depth-wise-separable convolution.	40

23	The stem of InceptionV4 uses 1×1 convolutions to compress the feature map in both pathways in order to reduce the number of filters after concatenation.	41
24	An illustration of a squeeze-and-excitation module. The filters of a feature map are weighted against each other. The weights are generated dynamically from the feature map by a quasi-autoencoder sub-network, that produces a single weight for every filter. The weight is applied by filter-wise multiplication of the weight-vector with the feature map tensor	43
25	This diagram illustrates the functionality of spatial attention. Spatial attention weights the positions of the feature map against each other, highlighting important regions on the image in the process. First, the filters are reduced to two filters by concatenating the result of 2 pooling layers with 1×1 -kernels. This is followed a 7×7 convolution with a softmax activation function. This layer reduces the feature map to a single filter. The resulting map of attention weights is then multiplied element-wise to each filter of the original feature map.	43
26	CBAM is a combination of spatial and filter wise attention. The above illustration depicts a residual block by He et al. (2016a) with CBAM added into the building block. CBAM combines spatial attention and filter attention (squeeze-and-excitation modules) for more fine-grained combined attention that considers all three tensor axis of the feature map.	44
27	Grad-CAM class activation on ImageNet images from three ResNet50 models. The first model is the baseline ResNet50, the second (ResNet50+SE) uses squeeze-and-excitation modules for filter attention and the third model using filter and spatial attention (CBAM). We see that attention mechanisms are sharpening the contours of the class activation's, bringing them closer to the object of interest.	45

28	It is also possible to create attention-based image classifiers based on transformers (a). The depicted ViT model achieves state-of-the-art performance on ImageNet without requiring convolutional layers. The transformer stack (b) is processing regional embeddings together with visual embeddings to obtain a prediction. This technology can be considered still in the early phases of development, since they require expensive pre-training with 300 times more data and are more expensive to train than convolutional neural networks with similar performance (Dosovitskiy et al. (2021)).	46
29	Rendering of the error surfaces from various models around the converged minimum can be used to gain insights into the effects of architectural properties. For instance, the surface-region of ResNet54 (a) is much rougher when the networks skip connections are disabled (b). The minimum of VGG16 (d) is also much sharper when no dropout for regularization is used (c) (sharp minima are hypothesized to generalize worse by authors like Keskar et al. (2017b)).	49
30	Class activation maps like the here depicted GradCAM++ algorithm by Chattopadhyay et al. (2018) provide insight into the inference process by highlighting the regions that activate specific classes.	50
31	By maximizing the activation value of a specific filter-unit in a convolutional neural network, we can visualize patterns specific filters are very sensitive towards. Based on these visualizations from an ImageNet-trained VGG19, we can see how the extracted patterns go from structurally small and simple (a) to increasingly complex patterns (b, c) depending on the location of the layer in the network.	51

32	Principal Component Analysis (PCA) on the outputs of neural network layers can be used for optimizing neural architectures for higher computational efficiency. The figures above illustrate the two steps of PCA-based pruning by Garg et al. (2020). Significant eigendirections of every layer’s output are computed up to a threshold of 99.99% explained variance (a), the obtained value will also be the number of filters of the layer in the optimized architecture. Layers that do not increase the dimensionality of the data from the previous layer are removed in the following pruning step (b). The reduced network is then retrained to obtain the more efficient model.	53
33	The probe performance at each layer of VGG16 trained on CIFAR10. Logistic Regression Probes allow the practitioner to observe the evolution of the intermediate solution quality from layer to layer. The performance is increasing layer by layer, indicating that the problem is solved incrementally and that the inference process is evenly distributed among layers.	55
34	The images show the basic neural architecture with a skip connection (a) and the probe performances of each layer (b). The setup is designed to provoke the network to "skip" the 128 layers by learning some identity-mapping analog. By observing the probe performances, this behavior can be observed. After the initial layer, the performance degrades until reaching chance level. The probe performances recover as soon as the skip connection is added to the layers again. We will observe similar behavior on convolutional neural networks over the course of this work.	56
35	A 1-dimensional example of receptive field expansion in a convolutional neural network. Each layer has a kernel size of 3, resulting in an expansion of the receptive field from layer to layer.	57
36	A 1-dimensional example of receptive field expansion with a convolutional downsampling layer (Layer 1). The increased step size from kernel position to kernel position results in less mutual inputs between two adjacent feature map positions in the output of Layer 1, resulting in an accelerated growth of the receptive field for consecutive layers and a reduction in feature map size.	57

37	The information in a given layer (blue) in a non-sequential neural architecture is based on multiple receptive fields with sizes. These receptive field sizes are bound in an interval $(r_{l,min}, r_{l,max})$. It is possible to compute $r_{l,min}$ and $r_{l,max}$, by calculating the receptive field size of the sequences (green) with the smallest and largest receptive field size leading from the input to the layer l (blue). These figures were previously published in Richter et al. (2021b).	59
38	The illustration depicts the general software architecture of Delve. From a user perspective, Delve is represented by an instance of a monolithic Tracker-object. This object can be used for calculating layer-based information about the PyTorch-model as well as logging experimental data. The strategy for recording and persisting the information is internally handled by an implementation of the Writer-Interface. Covariance-Approximation and extracting other information directly from the PyTorch-model is done using the forward-hook interface of PyTorch and is organized in a Key-Value mapping.	65
39	Typical program loop demonstrating the intended interaction between a model training and Delve experimental logging. The interaction between Delve and the training of the model is kept minimal to allow for easy integration and usability.	66
40	Delve can automatically generate plots for quick life analysis during training. The depicted example shows the intrinsic dimensionality of VGG19 after 29 epochs of training on Cifar10 using a batch size of 512 and an input resolution of 32×32 pixels.	71
41	PHD-Lab is configured over a simple json-file, which specifies parameter configurations and major components, i.e. the dataset, model, optimizer and metrics. These components are provided by the respective registries. For practical reasons the training of the model and the training of logistic regression probes is disentangled in the architecture to allow for easier distributed computing.	73
42	An example for the automatically generated folder structure used for saving experiment results in PHD-Lab. The logging structure is nested and the naming of files copied regularly are named based on the configuration to avoid confusion of experimental results.	77

43	Random orthonormal projections (right) with same dimensionality as E_l^k degrade the performance significantly faster than projections into E_l^k (left). This image was first published in Richter et al. (2021c).	89
44	When the same model is trained multiple times using the same setup, the same saturation values emerge from the layers with only minor fluctuations. These figures were previously published in Richter et al. (2021c).	95
45	Nearest-Interpolation downsampling to a single pixel induces heavy artifacts, thereby destroying the otherwise prevalent pattern of probe performances. Otherwise, probe performances maintain a visible pattern and improve with diminishing returns as the output resolution of the downsampling is increased. These figures were previously published in Richter et al. (2021c).	100
46	Average pooling is better able to maintain the structure of probe performances, thereby not showing the artifacts observed in figure 45. These figures were previously published in Richter et al. (2021c).	101
47	Predictive performance and average saturation form a logarithmic relationship when we train models on different filter sizes. The reduction in model capacity leads to the data "occupying" a higher percentage of the available feature space with a degrading effect on the predictive performance. This image was first published in Richter et al. (2022).	104
48	Overall patterns increase in contrast, and the overall saturation increases when the dataset is more difficult. This figure was previously published in Richter et al. (2022).	107
49	Examples of tail patterns from Conv8 to the output of the model. This pattern emerges when training a model, in this case VGG16, optimized for high resolution, in this case 224×224 pixels, on a low input resolution, which is 32×32 pixels in both scenarios. Layers that are part of the tail pattern are substantially lower saturated relative to the rest of the model, and the logistic regression probes stagnate at the performance level of the final model. The probe performance indicates that these layers act as pass-through layers and do not contribute to the quality of the solution, which can be regarded as a parameter-inefficiency since these layers are effectively underutilized.	108

50	Changing the input resolution changes how the inference is distributed among the layers. The resolution ResNet18 that was designed for (b) distributes the inference most evenly, while too small (a) and too large (c) resolution shift the bulk of the inference process to early and later layers respectively, resulting in worse predictive performance.	109
51	A tail of unproductive layers can be produced by placing the Cifar10 images on a 160×160 pixel canvas (b). This indicates that the locality of discriminatory features (essentially the size of the object) is responsible for the observed effect on the inference process. These figures were previously published in Richter et al. (2021a).	118
52	By analyzing the receptive field size, the start of the unproductive convolutional layers within the network architecture can be predicted. The <i>border layer</i> , marked with a black bar, separates the part of the network that contributes to the quality of the prediction from the part that does not. The <i>border layer</i> is the first layer with a $r_{l-1} > I$, where I is the maximum value of either the height or the width of the input image and r_{l-1} refers to the receptive field the layer's input is based on. Here, Cifar10 with the resolution of 32×32 pixels is used, and therefore $I = 32$. These figures were previously published in Richter et al. (2021a).	123
53	ResNet18 exhibits the same patterns observed in figure 52, when the skip-connections are removed (a). Increasing the receptive field by dilating convolutions (b) for VGG19 produces results consistent with figure 52. These figures were previously published in Richter et al. (2021a).	124

54	The heatmaps display the relative performance of probes trained on individual positions of the feature map to the performance of the model. The partial solutions contained in every feature map position improve from the image borders to the center, since the center can integrate the most information by expanding the receptive field (a). Once the border layer is reached (b), the partial solutions in the center reach the quality of the model prediction. In the following layer (c, d) the other partial solutions reach a similar quality. Effectively, the tail layers homogenize the partial solution quality of their feature maps. These figures were previously published in Richter et al. (2021a).	124
55	Residual connections allow networks to utilize layers past the border layer (marked with the horizontal bar). The zig-zag-pattern and drops of probe performances observed in both networks are artifacts previously observed by Richter et al. (2021c) and Alain, Bengio (2017). These indicate that the networks attempt to "skip" the convolutional layers in the low saturated tail. It is also worth noting that these skips are only present on later layers that are part of the tail. These figures were previously published in Richter et al. (2021a)	126
56	The Cifar10 optimized version ResNet18 and 34 has a roughly quartered receptive field size in every layer. These networks still display similar behavior to the ImageNet optimized version when it comes to the border layer. However, the reduction in the receptive field size has further removed the low saturated tail and in both cases leads to a steady increase in performance over the entire network's structure. These figures were previously published in Richter et al. (2021a)	128
57	The attention mechanisms Squeeze-and-Excitation modules (SE), spatial attention, and CBAM added to the ResNet 18 architecture with skip connectors (a, b), and without skip connectors (c, d). Attention mechanisms neither change the logistic regression probes' accuracy nor their saturation value. This indicates that attention mechanisms aid the feature extraction for the receptive field size present, but do not change the size of features extracted. These figures were previously published in Richter et al. (2021b).	134

58	The architecture used for the experiments in this section. Each stage consists of k blocks and has double the filters of the previous stage. The first layer of each stage is a downsampling layer with a stride size of 2. The building block for the shallow architecture MPNet18 is depicted in (a), and the building block of the deep architecture MPNet36 is depicted in (b). These figures were previously published in Richter et al. (2021b).	136
59	The <i>border layer</i> computed from the smallest receptive field size b_{min} can predict unproductive layers for ResNet architectures. The skip connections allow information based on smaller receptive field sizes to skip layers, resulting in a later <i>border layer</i> b_{min} compared to the same network with disabled skip connections. This allows networks with skip connections to involve more layers in the inference process than would be possible compared to a simple sequential architecture with a similar layout. For an example of this, compare Fig. 53 (a) to Fig. 60 (a). These figures were previously published in Richter et al. (2021b). . . .	139
60	The <i>border layer</i> computed from the smallest receptive field size b_{min} can predict unproductive layers for ResNet architectures. The skip connections allow information based on smaller receptive field sizes to skip layers, resulting in a later <i>border layer</i> b_{min} compared to the same network with disabled skip connections. This allows networks with skip connections to involve more layers in the inference process than would be possible compared to a simple sequential architecture with a similar layout. For an example of this, compare Fig. 53 (a) to Fig. 60 (a). These figures were previously published in Richter et al. (2021b). . . .	141
61	ResNet18 can be considered overparameterized for the ImageNet 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) is depicted in the order of the forward pass from input to output. This image was also published in Richter et al. (2021c).	145

62	The width of the VGG16 model 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. This image was also published in Richter et al. (2021c).	146
63	The width of the VGG16 model 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. This image was also published in Richter et al. (2021c).	148
64	A simple exemplary modification based on receptive field analysis. All layers with $r_{l-1,min} > I$ are replaced by a simple output-head, effectively removing the <i>border layer</i> and every layer after it. Removing these layers improves the efficiency by improving the performance and reducing the number of parameters and computations required. This image was also published in Richter et al. (2021b). © 2021 IEEE	149
65	Removing the tail layers of VGG19 and retraining the truncated model reduced the computational and memory footprint, while slightly improving the performance. Both models are trained on Cifar10 at native resolution. This image was also published in Richter et al. (2021c).	150
66	Saturation and loss converge at a similar pace. These results are also published in Shenk et al. (2019). © 2022 IEEE	171
67	When overfitting the saturation pattern keeps converging, indicating that overfitting is not drastically altering the lossless eigenspace. © 2022 IEEE	172
68	Saturations of convolutional neural networks show a converging behavior regarding saturation similar to previous observations in figure 66. This figure was also published in Shenk et al. (2019). © 2022 IEEE	173

69	Depending on the neural architecture, tail patterns may deviate in their appearance in probe performance. In sequential architectures (a) the layers maintain the quality of the intermediate solution. If shortcut connections exist in the architecture, layers may be <i>skipped</i> . Skipped layers are apparent by their decaying probe performance (Alain, Bengio (2017)). This is apparent on DenseNet18 (b) and ResNet34 (b) where a single DenseBlock and multiple ResidualBlocks are skipped respectively. All models are trained on Cifar10 at native resolution. © 2022 IEEE . . .	175
70	Reproduction of the experiment depicted in figure 50 using the VGG16 architecture. Because of the larger memory footprint of the model, the batch size of this experiment is reduced to 20. The observed pattern stays the same. Low-resolution results in a tail close to the output, while high resolution exhibits a tail at the input. The (medium-sized) resolution of 160×160 pixels performs best. © 2022 IEEE	177
71	Reproduction of the experiment depicted in figure 50 using the VGG16 architecture. Because of the larger memory footprint of the model, the batch size of this experiment is reduced to 20. The observed pattern stays the same. Low-resolution results in a tail close to the output, while high resolution exhibits a tail at the input. The (medium-sized) resolution of 160×160 pixels performs best.	178
72	Resolution Experiment reproduced on ResNet18 using the MNIST dataset	179
73	The higher native resolution combined with the residual connection removes the tail pattern even at 64×64 pixels for ResNet18. We see that a tail pattern can be produced in the input part of the network when the resolution is increased drastically to 1024×1024 , confirming observations from Cifar10.	180
74	Producing a tail pattern in the layers near the input and the output is also possible for very deep models like ResNet50 with the expected effects on the performance. The Bottleneck-Modules with 1×1 convolutions and varying filter sizes induce additional noise into saturation patterns.	181
75	Random Positioning Experiments conducted with ResNet18 on Cifar10 data.	182

76	Random Positioning Experiments conducted with VGG16 on MNIST data.	183
77	Random Positioning Experiments repeated on ResNet50.	184
78	ResNet18 trained on ImageNet.	185
79	ResNet18 trained on iNaturalist.	186
80	VGG16 trained on ImageNet.	187
81	VGG16 trained on iNaturalist.	188
82	Performance improvements past the border layer are miniscule, even though the capacity of each layer is reduces to $\frac{1}{8}$ of the original capacit. This indicates that the networks unable to shuft processing to otherwise unused layers even if the capacity is limited.	190
83	Repeating the experiment depicted in figure 52 on TinyImageNet yields consistent results regarding the border layer.	191
84	DenseNet18 - Cifar10 - 32×32 input resolution.	192
85	DenseNet65 - Cifar10 - 32×32 input resolution.	192
86	VGG16 trained on Cifar10 alongside the heatmaps generated from the relative accuracy of the partial solutions in each layer. Note that the border layer (Conv8) is the first layer to have partial solutions of equal quality to the model's solution (measured in accuracy).	193
87	ResNet18 trained on Cifar10 alongside the heatmaps generated from the relative accuracy of the partial solutions in each layer. Note that "skipped" layers (Block6-Conv2 and Block8-Con2) have a worse performance and worst partial solutions.	194
88	Cifar10 optimized ResNet18 trained on Cifar10 alongside the heatmaps generated from the relative accuracy of the partial solutions in each layer. Note that no layers are skipped and the solution quality of the partial solution develops up until the final residual block.	195

Abstract

The design and adjustment of convolutional neural network architectures is an opaque and mostly trial and error-driven process. The main reason for this is the lack of proper paradigms beyond general conventions for the development of neural networks architectures and lacking effective insights into the models that can be propagated back to design decision.

In order for the task-specific design of deep learning solutions to become more efficient and goal-oriented, novel design strategies need to be developed that are founded on an understanding of convolutional neural network models. This work develops tools for the analysis of the inference process in trained neural network models. Based on these tools, characteristics of convolutional neural network models are identified that can be linked to inefficiencies in predictive and computational performance. Based on these insights, this work presents methods for effectively diagnosing these design faults before and during training with little computational overhead. These findings are empirically tested and demonstrated on architectures with sequential and multi-pathway structures, covering all the common types of convolutional neural network architectures used for classification. Furthermore, this work proposes simple optimization strategies that allow for goal-oriented and informed adjustment of the neural architecture, opening the potential for a less trial-and-error-driven design process.

Zusammenfassung

Das Design und die Anpassung von Neuroarchitekturen für Convolutional Neural Networks ist gegenwärtig ein erratischer Entwicklungsprozess, der stark auf dem Entwicklungsprinzip von "Versuch und Irrtum" aufbaut. Es existieren zwei vorrangige Gründe für diesen Umstand: Zum einen, ein Mangel an Entwicklungsparadigmen für Neuroarchitekturen, die über allgemeine Konventionen hinaus gehen. Zum anderen, ein Mangel an Analysetechniken für neuronale Netze, die belastbare Rückschlüsse vom Verhalten des trainierten Netzwerkes auf Designentscheidungen in der Neuroarchitektur erlauben. Damit der Entwicklungsprozess von Convolutional Neural Networks aufgabenspezifischer, zielgerichteter und letztlich effizienter werden kann, bedarf es an neuartigen Designstrategien, die auf einem Verständnis der Informationsverarbeitung (Inferenz) innerhalb von Convolutional Neural Networks beruhen. In dieser Arbeit werden Analysewerkzeuge vorgestellt, mit denen die Inferenz von neuronalen Netzen analysiert werden kann. Auf Basis dieser Analysewerkzeuge werden Ineffizienzen in trainierten Convolutional Neural Network Modellen identifiziert und auf Designentscheidungen in der Neuroarchitektur zurückgeführt. Diese Einsichten erlauben die Entwicklung von Methoden, die eine schnelle, praxistaugliche Diagnose von Ineffizienzen während und vor dem Training eines neuronalen Netzes erlauben. Die Anwendung dieser Methoden wird empirisch auf sequenziellen und Multi-Pfad Architekturen getestet, um deren Zuverlässigkeit zu demonstrieren. Ferner werden auf Basis dieser Methoden einfache Optimierungsstrategien entwickelt, die es erlauben, die charakterisierten Ineffizienzen zielgerichtet zu entfernen und damit die Effizienz und Vorhersageleistung des Modells zuverlässig zu steigern. Mit diesen Strategien wird dargelegt, dass die Optimierung von Neuroarchitekturen auch ohne das "Versuch und Irrtum"-Prinzip möglich ist und infolgedessen ein informierter, effizienterer und zielgerichteter Designprozess ebenfalls prinzipiell möglich ist.

Abbreviations

Border Layer The first layer to receive input from a layer with $r_{l-1} > i$.

CNN Convolutional Neural Network

Classifier Refers to a CNN that solves classification tasks. In context of neural architectures, classifier refers to the layers after the the convolutional part of the network.

Feature Extractor The part of the convolutional neural network that consists of convolutional layers. Generally separates from the classifier by a "readout" (usually a Global Average Pooling layer)

Feature Map The input and output of a convolutional operation. Can be viewed as a stack of equally sizes matrices containing local pattern information.

Feature Space The feature space Z_l of a layer l is the space the output of this layer exists in.

Filter Sometimes also referred to as Channel (in case of color information). An output feature map of a convolution operation can be viewed as a stack of filters, where each filter is detecting a different pattern.

GAP Global Average Pooling. A Layer in a CNN architecture that pools a stack of filters into a vector by computing the average of each filter.

Latent Representations An intermediate state of the data between input and output. Usually the output of a layer. Also referred to as z_l .

LRP Logistic Regression Probes, a logistic regression trained on the same (classification) task as the model, using the output of hidden layer of this model as input.

Relevant Eigenspace A subspace of a layers feature space. When the data is projected into this space the output of the model does not change it's behavior.

Maximum Receptive Field The largest receptive field present in the output of a layer l . Can be seen as spatial upper bound regarding the locality of information integrated into a single feature map position

Minimum Receptive Field The smallest receptive field size present in the output of a layer l , can be seen as a spatial lower bound regarding the locality of information integrated into a single feature map position. For sequential architectures $r_l = r_{l,min} = r_{l,max}$, since only a single receptive field size is present in l

Multi-path Architecture A feed-forward neural network architecture where the information passes through more than one sequence of layers to reach the output. Examples: ResNet by He et al. (2016a) and GoogLeNet by Szegedy et al. (2015).

Probes Logistic Regression Probes (see LRP)

Probe Performance See p_l .

Saturation The percentage of dimensions of the lossless eigenspace in a layer relative to the available dimension.

Receptive field The area on an image that can influence the output of a single position on the feature map.

Sequential Neural Network A neural architecture is called sequential, when it can be described as a sequence of layers, where each layer has at most one predecessor and consecutive layer.

Skip Connection Architectural component in a convolutional neural network. Usually a parameter-less shortcut between layers that allows the information to "skip" layers.

Tail A unproductive subsequence of convolutional layers that does not contribute qualitatively to the prediction.

Tail Pattern See Tail.

VGG A family of sequential neural networks by Simonyan, Zisserman (2015), the number refers to the number of layers. (for instance VGG16 has 16 layers).

Symbols

- b The first layer to receive input from a layer with $r_{l-1} > I$.
- b_{min} The border layer in a multi-path architecture that is based on the minimal receptive field of the layers $r_{l-1,min} > I$. In sequential architectures equivalent to b_{max}
- b_{max} The border layer in a multi-path architecture that is based on the maximum receptive field of the layers. Equivalent to the border layer.
- E_l^k The approximation of the relevant eigenspace. A k -dimensional space spanned by the largest eigendirections of the covariance matrix of the output data in layer l .
- I The input resolution of a given training setup.
- l An arbitrary layer in a convolutional neural network architecture.
- p_l The test accuracy a logistic regression probe has achieved when trained on the output of the layer l . Sometimes also referred to as "probe performance".
- r_l The receptive field of layer l , equivalent to $r_{l,max}$
- $r_{l,max}$ The largest receptive field present in the output of a layer l . Can be seen as spatial upper bound regarding the locality of information integrated into a single feature map position
- $r_{l,min}$ The smallest receptive field size present in the output of a layer l , can be seen as a spatial lower bound regarding the locality of information integrated into a single feature map position. For sequential architectures $r_l = r_{l,min} = r_{l,max}$, since only a single receptive field size is present in l
- s_l The saturation value of layer l computed on the training data during the final epoch of training.
- Z_l The feature space of a layer l . This is the space where the output of l exists in.
- z_l The output data of layer l .

1 Introduction

1.1 Motivation

Convolutional Neural Networks (CNN) have dominated the field of deep learning and image classification in recent years. Ever since the publication of AlexNet by Krizhevsky et al. (2012), neural architectures were a key factor in improving the predictive performance and computational efficiency of the trained model (Khan et al. (2020)). From the multitude of published architectures certain design conventions have developed like the use of 3×3 convolutions, the overall pyramidal shape of the architecture and an input resolution of 224×224 pixels. However, these design decisions can be regarded as compromises, driven mostly by the desire to construct larger architectures while keeping the computational power required on a manageable level. Furthermore, the design process itself is inherently driven by trial and error, since the effect of architectural changes on the predictive performance can only be measured by comparative evaluation of trained models. This is exemplified when looking at state-of-the-art neural architecture search (NAS) solutions like NasNet by Zoph et al. (2018) and AmoebaNet by Real et al. (2019). These NAS solutions divert to using brute-force meta-heuristics with a search space limited only by aforementioned conventions in neural network design. Finally, on a larger scale, this mode of development has favored an increase in model parameters as the most reliable source of predictive performance improvement, making the resulting architectures ever more computationally expensive to train and develop (Huang et al. (2019)). In the long term, this mode of development is unsustainable and strongly limits the potential developers and application scenarios in which a modern CNN can be deployed.

To achieve widespread adoption of convolutional neural networks in a wide variety of applications, a more efficient design process is required. This design process needs to be driven by some degree of understanding regarding causes of inefficiencies and the general processing of information inside convolutional neural networks.

In this work, the foundation of a novel convolutional neural architecture design and optimization is proposed. This foundation is based on the understanding of the information processing inside convolutional neural networks on a layer-by-layer basis, allowing informed decision-making with a high probability of success.

1.2 On the Usage of the Third-Person Plural

While this monography is written by only one person, many results this work discusses were separately published as papers (Richter et al. (2021a, 2022, 2021b,c); Shenk et al. (2019)). The main reason this work is written as a monography instead of a cumulative style is for aesthetic reasons, since I am of the opinion that a book provides a better reading experience with a streamlined storyline and is thus better suited to summarize the research I have done in the last three years with the aid of my colleagues. To honor the co-authors of these papers, I will from here on stick to the first-person plural: *we*.

With few notable exceptions, namely the concept of the saturation metric (see section 8.1), the initial version of the framework *delve* (see section 6.1) these co-authors contributed to the publications as internal peer-reviewers and proofreaders for experimental results and drafts I produced over the years. The aforementioned exceptions are correctly noted in the text of the respective section as well. In any other case, it can be assumed that formulating hypothesis, conceptualizing experiments, conducting experiments as well as result analysis and writing drafts of the respective conclusions and publications was done by me.

1.3 Structure of this Work

We will first discuss the foundation of deep learning and convolutional neural networks in general in chapter 2. This part of the work covers the most relevant datasets, basic functionality and setup of convolutional neural network training and evaluation. Chapter 3 is focused on the design of convolutional neural architectures, summarizing the current state of research in this regard. Design criteria will be discussed, as well as structural conventions and components in neural networks for computer vision. The following chapter 4 proceeds to discuss the related work regarding the analysis of convolutional neural networks. In this chapter, we also establish important metrics like logistic regression probes and the receptive field, which are central to the empirical part of this work.

The empirical part of this work is headed by chapter 5, a brief discussion of the central research questions and goals. Before the work moves to the original research, we present our software in chapter 6. This chapter will primarily focus on the OpenSource-Libraries PHD-Lab and Delve for experiment control, reproducibility and evaluation. Both Libraries were developed

as an integral part of this work and serve as the backbone for all empirical studies on neural architectures. We then move on to the first experimental part of the work, which is dedicated to analyzing the information processing inside convolutional neural networks. By approximating the subspaces inside neural network layers that are critical for the quality of the prediction (chapter 7), we can derive a novel metric, "saturation" (chapter 8). By exploring the properties of saturation and logistic regression probes, we can demonstrate that both metrics can be used to find a pathological pattern (tail pattern) indicating inefficiencies in convolutional neural networks. Based on these findings, we investigate the cause of the inefficiencies in chapter 9. Finally, we demonstrate the ability to predict the occurrence and location of architectural inefficiencies in the network solely based on the evolution of the receptive field from layer to layer. This is demonstrated first for sequential neural architectures in chapter 9. In chapter 10 this capability is expanded to architectures with multiple pathways and attention-mechanisms. Based on these results, we can propose design guidelines in chapter 11 for optimizing convolutional neural architectures that can be applied without the need for comparative evaluation.

2 Basics

This section will briefly elaborate on the foundations and general concepts of classifiers and deep convolutional neural networks in computer vision. In section 2.1 we conceptually introduce classifiers as black-box models. The following section 2.2 briefly introduces the most relevant datasets that are used throughout this work and discusses their relevance in the field of computer vision. After that, section 2.3 introduces the convolution operation, and its properties that are most relevant for this work, before we elaborate on the general training and evaluation methodology for deep convolutional neural network classifiers in section 2.4.

2.1 Deep Convolutional Neural Network Classifiers

With the submission of Krizhevsky et al. (2012) to the ImageNet competition, deep computer vision and deep learning in general gained a surge of popularity. The historically central problem that these deep computer vision algorithms solve is the scene categorization problem also known by the more general term of classification problem (Krizhevsky et al. (2012)). The figure below illustrates a simplified structure of the inference process of a neural network classifier.

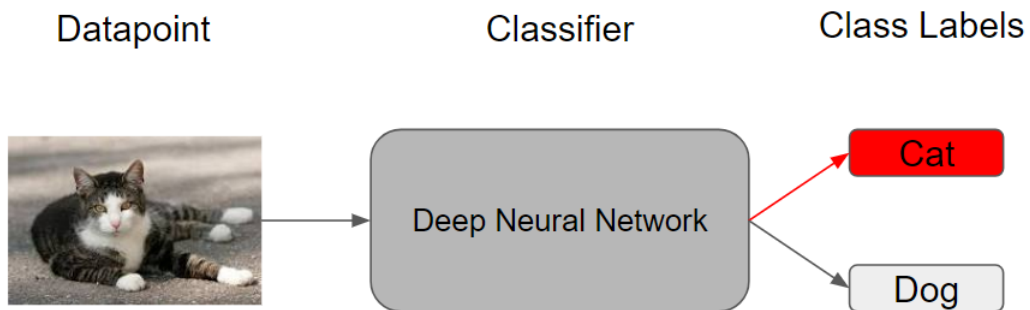


Figure 1: A simple hypothetical example of a cat-dog-classifier: The classifier receives an input image and predicts a label from a set of discrete symbols.

The problem requires the model to assign one discrete symbol (colloquially referred to as "label") to a given data point (image). Due to its simplicity, image classification has served as a benchmark task for many innovations in convolutional neural architecture design. For instance, the architectures and design concepts popularized by Simonyan, Zisserman (2015), Szegedy et al.

(2015) and He et al. (2016a) were first utilized and designed for the ImageNet-Classification challenge and later adopted to more complex tasks like object localization (Ren et al. (2015)), segmentation (He et al. (2017)) or similarity measurement (Koch et al. (2015)). For the same reason, this work will also focus on deep convolutional classifiers.

2.2 Benchmarks

In this work, we define a benchmark problem as a task consisting of a training and test set, on which a model can be trained and evaluated on. Depending on the size, visual-domain and the complexity of the classification problem a benchmark may be utilized for different purposes like serving as a proxy problem for a more complex task, proof-of-concept testing or pushing the state-of-the-art in predictive performance. Thus, benchmark problems are crucial in deep learning research for the comparison of convolutional neural architectures, since they provide standardized scenarios that greatly simplify comparative evaluation. Since benchmark problems are central for empirical research on convolutional neural networks, we will briefly present the classification benchmarks relevant to this work and elaborate their typical use-cases in the wider area of deep learning research and their utilization in this particular work.

2.2.1 MNIST

The MNIST dataset by LeCun, Cortes (2010) is a dataset for handwritten digit recognition. The dataset consists of 28×28 pixel binary images of handwritten digits ranging from 0 to 9 (10 class classification problem). The data is split into a training and a test set. The training set consists of 50,000 images (5,000 per class), the validation set consists of 10,000 images (1,000 per class).

Due to the low dimensionality and relatively low difficulty of the classification problem compared to other popular classification datasets like Cifar10 and ImageNet (Scheidegger et al. (2021)), MNIST is often considered the "Hello World" of deep learning. For this reason, MNIST is mostly used for proof of concepts like in the publication of Li et al. (2018a) and foundation work like the experiments of Zhang et al. (2017) on the properties of generalization with deep neural networks. This work uses MNIST primarily as a redundant testing ground to demonstrate that insights gained from experiments on other datasets generalize to datasets with different visual domains and difficulty.

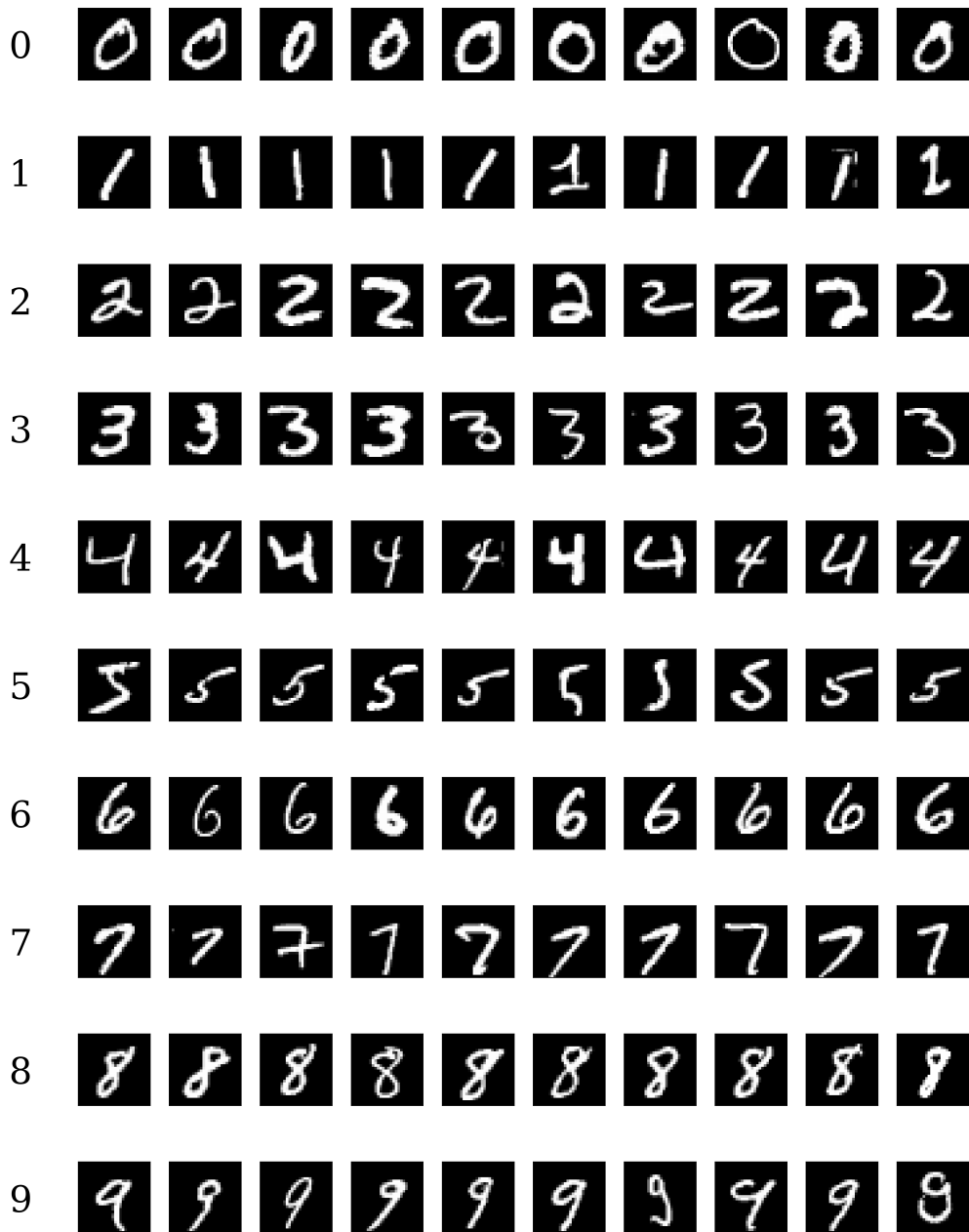


Figure 2: The dataset consists of 10 classes of hand-written digits, the images are 28×28 binary images, which can be considered a very low dimensional visual domain compared to other popular datasets like Cifar10 and ImageNet. Due to its simplicity, MNIST is mostly used for proof of concept work.

2.2.2 Cifar10 / Cifar100

Cifar10 and Cifar100 are datasets of small images belonging to 10 and 100 categories, respectively. The datasets were published by Krizhevsky et al. (2015) and consist of 32×32 pixel color images. Cifar10 and 100 are composed of a training and a validation subset each. The training subset consists of 50,000 images and the validation subset consists of 10,000 images for both datasets. The classes of both datasets are balanced, resulting in 5,000 training and 1,000 validation samples for Cifar10 and 500 training samples and 100 validation samples for Cifar100. The categories are composed of a wide range of different objects, animals and scenarios, like Frogs, Bears, Bridges, Rockets, Forests and Mountains. In figure 3 we can see a collage of images taken from all categories of Cifar10.

Due to the relatively low resolution of 32×32 pixels, both datasets can be used for quickly training and evaluating models. Compared to MNIST, their respective visual domains are also more realistic and can thus be considered more complex. The difficulty of the classification problem can be considered harder as well, based on the performances achieved by classifiers in recent years (Scheidegger et al. (2021)). This makes the Cifar datasets popular in proof of concept work for neural architecture innovations, since performance improvements can be observed more clearly compared to MNIST. Furthermore, improvements made in Cifar10 often generalize to more complex datasets of real-world imagery like ImageNet. For instance, Simonyan, Zisserman (2015) and He et al. (2016a) demonstrate the predictive performance of their neural architectures on Cifar10 as well as ImageNet. For the same reason, Cifar10 and Cifar100 are also used in automated neural architecture search as proxy problems for the more complex ImageNet dataset (Real et al. (2019); Zoph et al. (2018)).

Due to the aforementioned combination of a realistic visual domain, non-trivial classification problem and economical training compared to ImageNet, we use Cifar10 as our primary proofing ground for the empirical research presented in this paper.

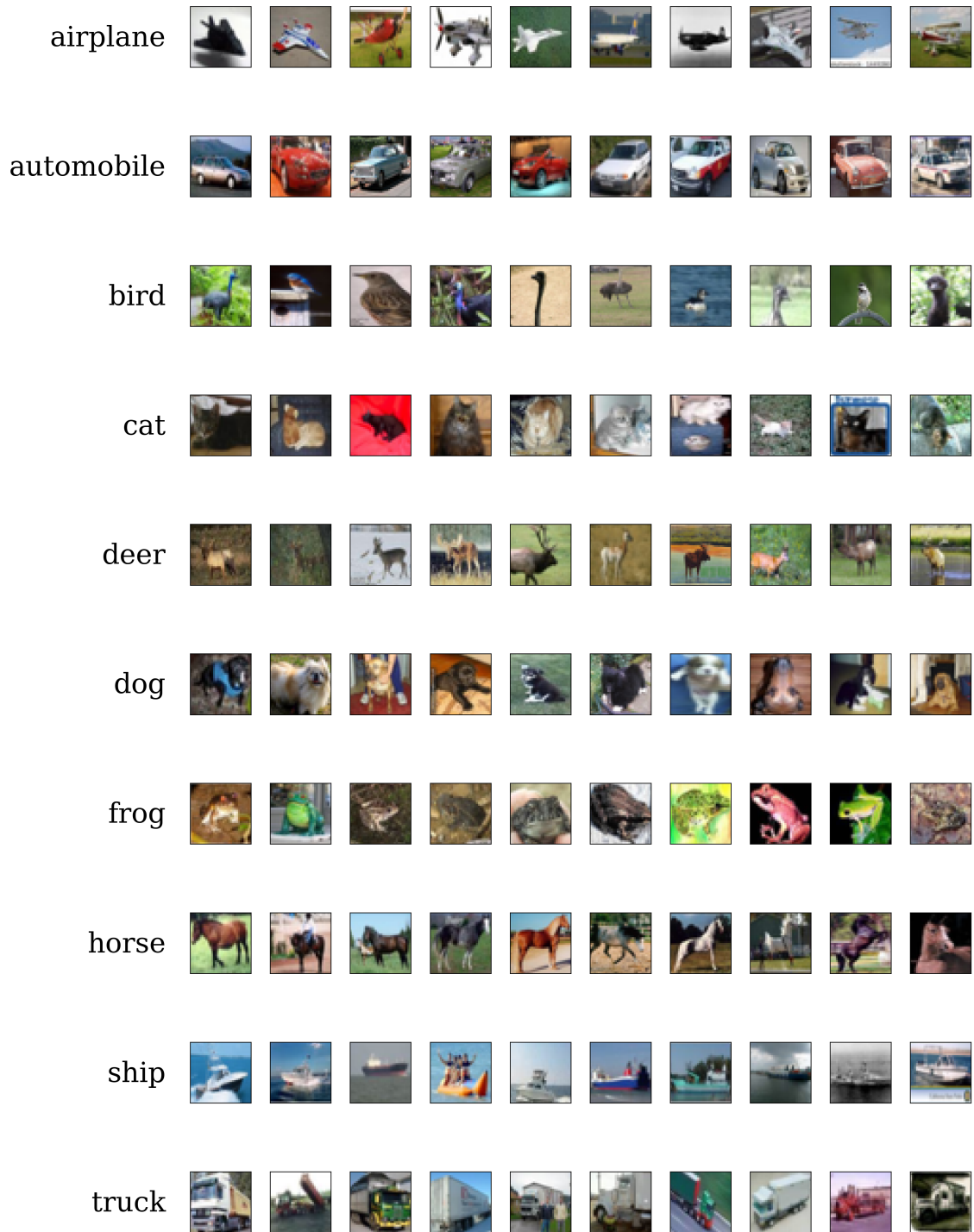


Figure 3: Similar to MNIST, the Cifar datasets consist of uniformly sized low resolution (32×32 pixel) images, making them very economical for model training compared to higher resolution datasets like ImageNet. The datasets consist of colored real-world images. The classes are real-world objects. For these reasons, the Cifar-datasets can be considered more complex than MNIST. For the same reasons, these datasets can be used as more economical proxy problems for computationally demanding tasks such as ImageNet (Real et al. (2019); Zoph et al. (2018)).

2.2.3 ImageNet

ILSVRC₁₂, colloquially referred to as "ImageNet-Dataset" or just "ImageNet" is a 1,000 category classification dataset by Deng et al. (2009). The name of the dataset is an acronym and stands for "ImageNet Large-Scale Visual Recognition Challenge", an annual competition held from 2010 to 2017. The dataset consists of roughly 1,280,000 training and 50,000 validation images. The categories are balanced in both subsets. The images have neither a uniform resolution nor aspect ratio. The average resolution of the dataset is 469×387 pixels and all images are RGB-colored photographs. The categories are composed of WordNet-IDs, which reference entries in the WordNet database. A WordNet-ID is therefore referring to such things as animals, objects and scenery. For instance, the category "meerkat" has the WordNet-ID n02138441 and the category of the WordNet-ID n04532670 contains images depicting Viaducts. While the classes are mostly clean and free of errors, they are not completely free of mislabeled and corrupted images.

The large number of classes with strongly varying inter- and intra-class homogeneity makes the ImageNet dataset very challenging, with 90% accuracy being reached by a model trained by Pham et al. (2021), 9 years after the publication of the dataset. For this reason, model performance is reported with two metrics: The regular Accuracy (in this context also referred to as "Top1-Accuracy") and the Top5-Accuracy. The latter counts a prediction as true positive if the true class is contained within the top 5 classes with the highest predicted probability.

The ImageNet dataset can be considered the historical starting point of modern deep learning research. The significance of the AlexNet convolutional neural network by Krizhevsky et al. (2012) was demonstrated on the ImageNet dataset, beating the competition by 10 percent points top5-accuracy in the year 2012. Since then, convolutional neural network classifiers are generally optimized for the ImageNet dataset and improvements on the predictive performance are generally used as validation in various works ranging from efficient parallelization techniques for large models (Huang et al. (2019)) over optimizers and preprocessing techniques (Cubuk et al. (2019); Kingma, Ba (2014); Liu et al. (2020)) to entire neural architectures (He et al. (2016a); Huang et al. (2017); Real et al. (2019); Simonyan, Zisserman (2015); Szegedy et al. (2015, 2016); Tan, Le (2019)). Furthermore, models trained on ImageNet are known to generalize well to other problems and are thus well suited for transfer-learning applications (Smith, Topin (2017a)).

Due to the great demand in computational resources, it is not possible to

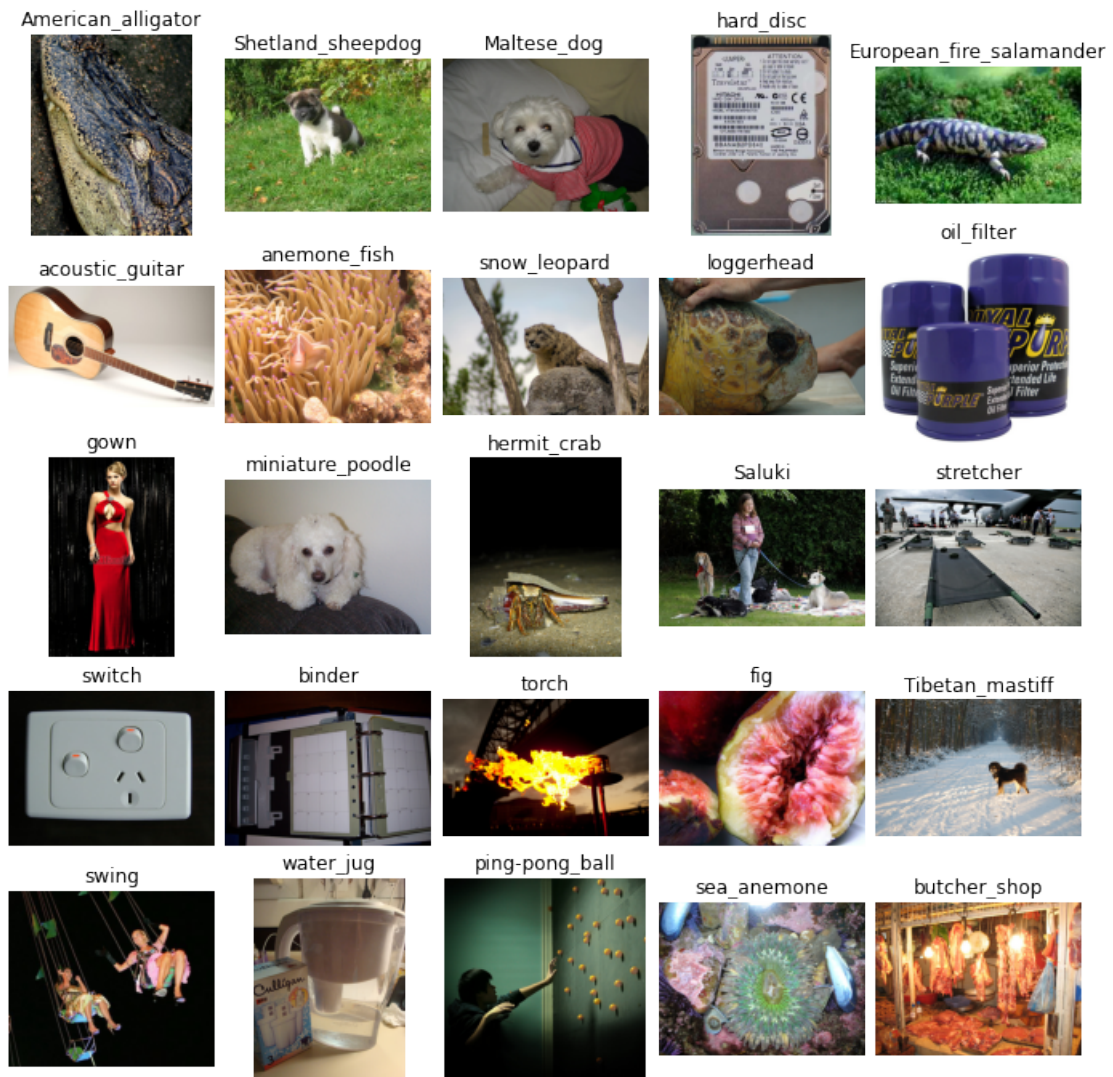


Figure 4: The ImageNet dataset features RGB-images of various aspect ratios and image qualities belonging to a heterogeneous set of 1,000 classes. These properties render the ImageNet-dataset a very challenging classification problem, which makes it ideal for training well generalizing models. From the early days of deep learning until today improving the predictive performance of ImageNet-trained models can be considered an active field of research and central to innovations in deep convolutional neural network classifier design (Dosovitskiy et al. (2021); Khan et al. (2020); Krizhevsky et al. (2012); Tan, Le (2019)).

conduct large numbers of experiments required for the research presented in this work. However, we train models on ImageNet in a selected number of experiments to demonstrate that certain key-observations made on Cifar10 and other datasets can be reproduced on a much more complex, high-resolution classification problem.

2.3 The Convolution Operation

While there are many properties of the convolution that can be discussed, we will focus for readability on the properties that are most significant for understanding this work. For further details, we refer to the work of Dumoulin, Visin (2016), which is a detailed compilation of the properties of convolution operations commonly used in deep learning.

The convolution operation is one of the key components in any convolutional neural network architecture. The operation itself has been derived from cross correlation and has many applications in computer vision, which pre-date the application in neural networks, like smoothing, blurring and edge detection. The basic principle is illustrated in figure 5.

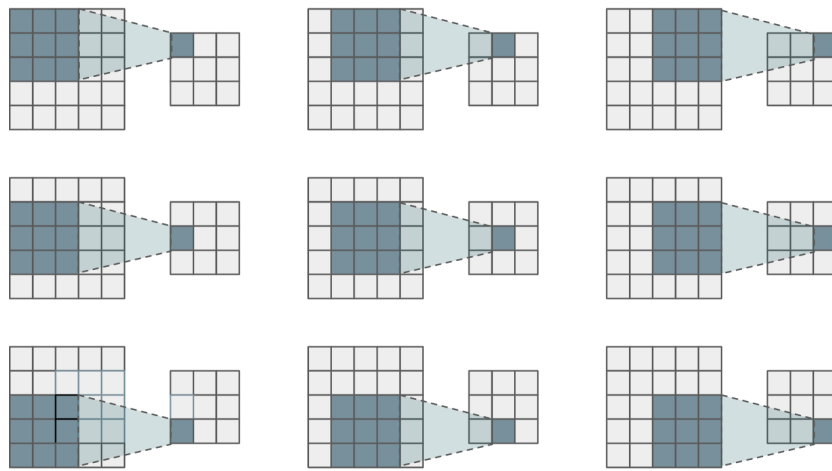


Figure 5: This example of a convolution operation using a 3×3 kernel illustrates how a convolutional layer integrates local context into a position in its output. The left matrix is the input feature map, the right matrix is the output feature map. The highlighted area illustrates the kernel being convolved over the input feature map. Every position of the output is computed from the region of the input feature map covered by the kernel.

A filter of a fixed size (kernel size) is moved (convolved) step by step over the input. Every position on the input produces a single scalar output, resulting in a grid-structure of outputs, which we will refer to as a "feature map" from now on. An example of an application of a convolutional kernel on an image can be seen in figure 6.

The (in this case hand-crafted) kernel by Kanopoulos et al. (1988) is convolved over a gray scale image. The depicted kernel is sensitive towards strong vertical color gradients and will thus produce high values when placed over vertical edges. The resulting feature map therefore contains information on the presence of vertical edges on the image. While the convolution operation and the depicted application is linear. Non-linearity is created by applying an activation function on the output of a convolution operation e.g.

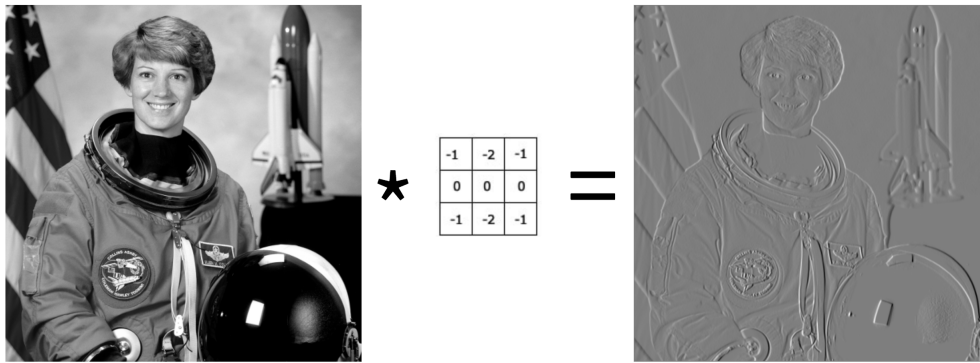


Figure 6: A simple example of a convolution operation using a vertical Sobel filter (Kanopoulos et al. (1988)). The kernel (middle) is convolved over the input image (left), which results in a feature map (right) containing information on the presence of vertical edges.

the ReLU-activation function (Krizhevsky et al. (2012)) or the SWISH-function by Ramachandran et al. (2018).

The application of a convolution operation inside a neural network structure has many advantages. Since each output depends only on the positions on the input feature map under the kernel, the computations are reduced drastically. Since the size of the kernel is limited, the extracted features are limited in their locality. This forces convolutional neural networks to extract local patterns like edges and gradually construct a more complex pattern when the data is passed from layer to layer, thereby expanding the receptive field.



<p>(a) Original: 54% persian cat 13% siamese cat 8 % tabby cat</p>	<p>(b) Texture: 53.74% ind. elephant 24.37% black swan 21.60% indri</p>	<p>(c) Stylized: 32.48% ind. elephant 18.57% black swan 14.35% indri</p>
--	---	--

Figure 7: Predictions of an ImageNet pretrained ResNet50. The texture of the figure 7(b) is transferred on figure 7(a) using the style transfer GAN by Ghiasi et al. (2017). While the cat is still recognizable the texture transfer results in the top 3 classes being closer to the texture image compared to the predictions of the original cat-image. This indicates that textures are weighted over the general silhouette of objects, suggesting a bottom-up pattern recognition process (Geirhos et al. (2019)).

This enforces a bottom-up process of pattern recognition in the neural architecture. While this inductive bias of convolutional layers is by its nature

more biased towards textures (see figure 7) compared to human perception, it is also strategically limiting the search space of detectable patterns beneficially. This has led to almost all state-of-the-art classifiers in computer vision being convolutional neural networks (Khan et al. (2020)). Transformers have recently shown to be able to produce similar predictive performance without this inductive bias, but require orders of magnitude more data and computational resources to do so. For example, Dosovitskiy et al. (2021) successfully trained transformers that achieved similar performance to the best contemporary convolutional neural networks on the ImageNet dataset. However, the authors required 300 million additional weakly labeled images to do so (ImageNet contains roughly 1.3 million images). The last property of convolution operations we want to discuss is that convolutions are agnostic to the dimensionality of the axis the kernel is convolved over. In terms of images, this generally means that a convolution operation is agnostic towards the height and the width of the image. This contrasts with the matrix multiplication used in densely connected layers, which requires a fixed dimensionality of input and output. Fully convolutional neural networks can thus process arbitrarily large input images, while neural networks containing dense layers are fixed to a specific resolution. This property can be exploited to avoid scale variance in the trained model. For example, Redmon, Farhadi (2017) devise a multi-scale training scheme with a randomized input resolution, which force the model to learn patterns invariant to scale. We exploit this property in chapter 9 to deepen our understanding of the relationship between neural architecture and input resolution.

2.4 Conventions in Training and Evaluation Methodology

Besides the neural architecture, the more general setup used for training can have a great influence on the performance of the model, as many publications show (Kingma, Ba (2014); Liu et al. (2020); Smith, Topin (2017b)). Before we can discuss neural architectures and their properties, it is also important to look into the general methodology used for training models that achieve state-of-the-art performance on ImageNet. This section will focus on supervised learning of classifiers in particular, since it is the simplest and commonly used for evaluating novel architectures and thus the most relevant for this work.

Similar to general neural architecture designs, there is currently no precise, agreed upon standard on how exactly training of a convolutional neural

network is conducted on ImageNet or any other dataset. However, the basic setup consists of an optimizer, and two preprocessing pipelines (one for training, one for evaluation), a loss-function and the (untrained) model instance based on a neural architecture. We will briefly discuss the aforementioned components and explain their most significant effects on the training process. We will also briefly describe the general training and evaluation methodology used for training convolutional neural network classifiers.

2.4.1 Loss

When thinking about a trained model as an answer, then the loss-function can be considered the question. The goal of any gradient descent algorithm is minimizing the loss during training. In effect, the minimization of the loss should result in a stable, well generalizing model, given a good context, which is provided by the dataset. In supervised classification tasks the cross entropy between the ground truth and the predictions is used as target function for minimization, since the mean squared error can lead to bad estimators if the probability distribution is not normal, which cannot be guaranteed for arbitrary classification problems (Kosheleva, Kreinovich (2017)). In more complex tasks like cancer mammography, classification and object detection, a combination of losses may be used to phrase the "question" more precisely and ensure that concepts like "objectness" and "locality" are learned properly (He et al. (2017); Rakhlin et al. (2018); Redmon et al. (2016); Ren et al. (2015); Shen et al. (2021); Wu et al. (2020)). However, since the main focus of this work are classifiers, we will primarily focus on cross-entropy loss which can be considered the standard loss for classification tasks. Cross Entropy can be considered a measure of difference between a probability distribution p and an approximation of the same distribution q .

The cross entropy is computed as follows:

$$H(p, q) = - \sum_{x \in X} p(x) \log(q(x)) \quad (1)$$

The variable x is a discrete instance value of the set of possible discrete values X of the probability function p and its approximation q .

If the probability distributions are equal $p = q$, then the cross entropy is equal to the entropy of the probability distribution $H(p, q) = H(p, p) = E(p)$. In any other case, $H(p, q)$ can be described as the sum of $E(p)$ and the

Kullback-Leibler-divergence $D_{KL}(p||q)$ (Bishop (2006)):

$$H(p, q) = E(p) + D_{KL}(p||q) \quad (2)$$

During training, the ground truth is one-hot encoded. In terms of probability distributions, this means that a single category for a given data point has a probability of 100.0% and all other categories have a probability of 0%. The prediction of a deep neural network classifier is encoded as a "softmax", which can be interpreted as a probability distribution over the categories of the classification problem. Since $p(x) = 0$ for all categories except one $E(p) = 0$, the only nonzero term of $H(p, q)$ is the term of the true category with $p(t) = 1$ with $t \in X$. The loss for a single data point can therefore be simplified:

$$loss(t; p, q) = -p(t)\log(q(t)) = -\log(q(t)) \quad (3)$$

The loss for a given data point will be zero if the softmax produced by the model is equal to the one-hot encoded prediction, since $\log(1) = 0$. In any other case, $-\log(q(t))$ will monotonically increase as the predicted probability $q(t)$ approaches zero. Since $q(t)$ is a probability and therefore a value in the interval, $[0, 1]$ it is impossible for the cross entropy loss to be negative. The only way to minimize the cross entropy is to maximize the predicted probability value of the true category $q(t)$. Hence, when used for training, the cross entropy forces the model to maximize the predicted class probability of the likeliest category.

2.4.2 Preprocessing and Data Augmentation

The raw images are preprocessed before being fed into the model. While training and validation often share preprocessing stages like channel wise color normalization and resizing, the pipelines differ at training and inference time (Cubuk et al. (2019); He et al. (2016a); Krizhevsky et al. (2012); Simonyan, Zisserman (2015)). The primary purpose of preprocessing is, to bring the image data into a uniform format, which can be optimally processed by the model. This primarily involves resizing the images to a fixed input resolution and normalizing the pixel values. The training pipeline is more complex and features also probabilistic preprocessing techniques, collectively referred to as "data augmentation".

The primary purpose of data augmentation is to optimize the predictive performance of the model. The features that a convolutional neural architec-

ture learns to detect in order to estimate the category of the image are not necessarily invariant towards transformations that would (judged from a human perspective) not change the category of the image. An example of this can be seen in figure 8. The image category does not change with the rotation angle, yet the model has not learned features that are invariant to this affine transformation, resulting in a changing prediction depending on the rotation angle. This can be considered an inductive bias of the dataset, allowing the model to overfit on features that do not generalize to other real-world instances.

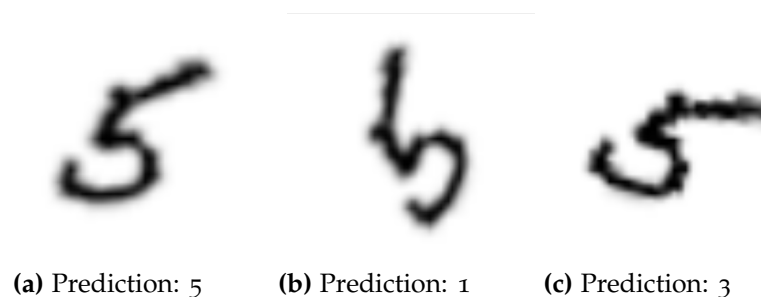


Figure 8: A VGG13 convolutional neural network (Simonyan, Zisserman (2015)) was trained on MNIST without any data augmentation and is therefore not invariant towards rotations. The predictions made by this model vary for the same test-set image depending on the angle it was rotated.

Data augmentation can be considered a set of probabilistic regularization techniques that take advantage of the properties of images in general. Distortions are randomly applied on data points. These distortions are known to not change the category. Depending on the dataset these transformations can for example be rotations by a random angle, horizontally mirroring an image (horizontal flipping), the application of Gaussian noise in the HSV-space and other random changes on image properties like contrast, color-pallet etc. By doing this, we can artificially enlarge the dataset (see figure 9 for examples) and force the model to detect patterns that are invariant towards the applied distortions.

Krizhevsky et al. (2012) used some of the aforementioned techniques to increase the size of the training dataset by a factor of 2048. In more recent application like Tan, Le (2019) and Real et al. (2019) the augmentation steps are applied randomly on every loaded batch of data, allowing theoretically for an unlimited amount of augmented images.



Figure 9: To avoid scenarios like the one depicted in figure 8, where the model is not invariant towards certain properties of the input image, training images are randomly augmented. This example shows various combinations of data augmentation techniques applied to an image of Felix the horse. The model can be forced to be robust against these kinds of distortions, by training a model on randomly distorted versions of the original images.

2.4.3 Optimizers

Optimizers are responsible for minimizing the loss of the model by updating its parameters during training by the means of back propagation and gradient descent (Rumelhart et al. (1986)). Similar to quasi-Newtonian optimization, the training process can be imagined as a ball (representing the current state of the model) rolling on a high dimensional surface into a minimum. The surface is called the "error surface" and is created by mapping every set of parameter values of the model to a loss value.

Convolutional Neural Network classifiers are designed for non-linear optimization problems, which makes the error surface non-convex. For this reason, gradient descent may lead to suboptimal performance. The reason for

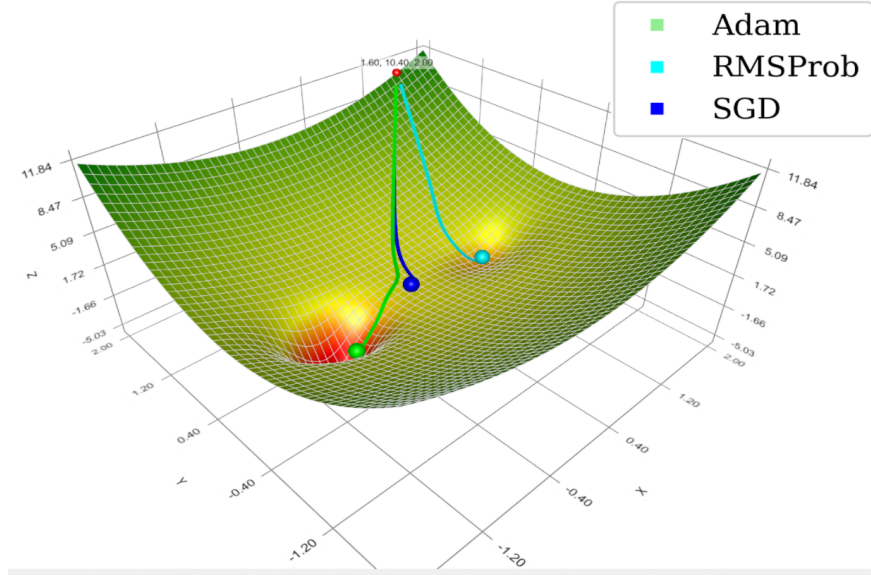


Figure 10: The optimizer influences the trajectory of the model on its error surface. In this example, a model with 2 parameters is trained using three different optimizers. The visualization shows, that properties of the optimizers influence the trajectory and as a result, whether a model may "get stuck" in local minima and saddle nodes (SGD, middle) a local optimum (RMSProb, right) or converge into good, in this case global, minimum (Adam, left).

this can be seen by looking at the simplest possible learning rule for gradient descent:

$$w_{t+1} = w_t - \gamma * \Delta w_t \quad (4)$$

w_t represents the weights of the model at time step t , Δw is the gradient of the weights computed by back propagation and γ is the learning rate. The update of the weights is based only on the gradient of the current optimization step. Since the error surface is non-linear, $\Delta w = 0$ can be caused by a saddle node or a relatively bad local optimum on the error surface. This can theoretically lead to a training being effectively "stuck", prematurely ending the optimization on a possibly inferior solution. An example of a simple model with two parameters can be seen in figure 10. Specifically, figure 10 shows stochastic gradient descent (SGD) getting stuck in a local minimum, while the other gradient descent optimizers RMSProb by Dauphin et al. (2015) and Adam by Kingma, Ba (2014) navigate their respective trajectories to better solutions. Optimizers like Adam, Adagrad, Adadelta and RMSProb use various techniques for dynamically adapting the learning rate during training. (Dauphin et al. (2015); Duchi et al. (2011); Kingma, Ba (2014); Zeiler (2012)). This introduces dynamic perturbations into the weight update, effectively allowing the optimizer to escape local minima. Another way to combat convergence

towards suboptimal solutions is introducing various kinds of perturbations in the gradient update to make these and similar situations less likely. For example, dividing the dataset into mini batches and updating the weights batch by batch is a de facto standard technique to induce random perturbations into the gradient. Goyal et al. (2017) demonstrate in their study on the effect of large batch sizes empirically that smaller batch sizes (around 256 images) are consistently performing better than models trained on batches composed of thousands of images on the ImageNet dataset. Other techniques based on the current and past state of the model, like momentum and weight decay, are also commonly used to influence the trajectory of the model on the error surface to further reduce the probability of convergence on a bad solution (Ruder (2016)). Finally, when training state-of-the-art models, the learning rate is reduced at discrete time steps by a multiplicative factor. This technique was originally pioneered by He et al. (2016a), who showed that a rapid decrease in the learning rate can "restart" the convergence process multiple times during training, improving the performance even further. Since then, this technique has been adopted by other significant publications like Howard et al. (2017); Huang et al. (2017); Sandler et al. (2018); Tan et al. (2020) and Bochkovski et al. (2020).

2.4.4 Models

In this paragraph, we will briefly discuss the external properties of convolutional neural network models used for training from a purely functional perspective. A more detailed analysis of commonly used architectural properties and the evolution of neural architecture design will be discussed in chapter 3. For standard classification problems, a convolutional neural network is generally built as a feed-forward structure with a single input and a single output.¹ The input is expected to be a third degree tensor with 3 color channels in one axis (the other two representing the height and the width of the image). Non fully convolutional architectures like AlexNet (Krizhevsky et al. (2012)) and the VGG-family of networks (Simonyan, Zisserman (2015)) furthermore require a fixed input resolution. By convention the standard resolution of classifiers has been 224×224 pixels since introduced by Krizhevsky et al. (2012), even if the fully convolutional nature of the model would allow other resolutions (He et al. (2016a); Szegedy et al. (2016)). Only recently, AmoebaNet and EfficientNet increased the resolution to up to 600×600 pix-

¹Deviations are rare but do exist. For instance GoogLeNet by Szegedy et al. (2015) features multiple output heads as a solution to the vanishing gradient problem.

els (Huang et al. (2019); Real et al. (2019); Tan, Le (2019)). In other fields of deep computer vision like object detection and segmentation, higher resolutions are more common (He et al. (2017); Redmon, Farhadi (2017); Tan et al. (2020)).

The output of the model is a single layer with a number of neurons equal to the number of categories in the classification problem. The activation function is a normalized exponential function called softmax:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad (5)$$

Where z_j refers to the output of the j th out of the K neurons in the softmax-layer. The softmax function projects every component z_j of the activation vector into the interval of $(0, 1]$ furthermore, the sum of all neuron activation values is always 1:

$$\sum_{j=1}^K \sigma(z)_j = 1 \quad (6)$$

For this reason, the output of a neural network classifier can be interpreted as a probability distribution over the categories.

2.4.5 General Methodology of Training and Evaluation

Training is split in training intervals referred to as "epochs". By convention, a single epoch is completed when the entire training dataset is processed. The processing itself is sub-divided into a sequence of mini-batches. The weights of the model are updated after a mini-batch is processed. There are two primary reasons for this methodology. First, as mentioned in the previous section, small mini-batches induce random perturbations in the gradient, resulting in a decreased probability of converging into saddle-nodes and sharp optima (Li et al. (2018b), Goyal et al. (2017)). Second, the memory limitations of modern GPU and TPU hardware makes processing the entire dataset at once impractical or physically impossible. Furthermore, the training dataset is shuffled at the beginning of each epoch to avoid overfitting on specific image combinations inside the batches. At regular intervals (usually between epochs) the model is tested on a validation set. During the test, the model's predictive performance is evaluated on a designated evaluation set, which contains only samples the model was not trained on and that are drawn from the same distribution as the training data. The metrics evaluated on the validation data allow assessing how well the model generalizes to unseen data and how strong the model is currently overfitting on the training data. Train-

ing is conducted for a set number of epochs or until a stopping criterion is reached (Li et al. (2020)).

2.4.6 Evaluation Metrics and Methodology

The evaluation of models is subdivided into the performance evaluation of the model, which is done in regular intervals (usually at the end of each epoch) during training and a post-hoc evaluation, which is generally more detailed and elaborate. The former is mainly focused on the evaluation of fast-to-compute metrics like accuracy, precision, recall or the confusion matrix to allow for quick intermediate assessments of the current performance. In the latter case, more detailed evaluation techniques like inspection for dissecting the trained model’s behavior are applied. Examples of such techniques are Lime by Ribeiro et al. (2016), error surface visualization by Li et al. (2018b) and logistic regression probes by Alain, Bengio (2017).

The primary goal of this analysis is to deepen the understanding of model quality and derive next steps for model improvement. Furthermore, evaluation after training may utilize more elaborate inference schemes like multi-crop evaluation (Simonyan, Zisserman (2015)) to stabilize and improve the performance of the trained model. This is commonly done for highly optimized models trained for competitions (He et al. (2016a); Simonyan, Zisserman (2015); Szegedy et al. (2015, 2016)).

2.5 Metrics

While there are many metrics for assessing the qualitative properties of classifiers, benchmarks are generally designed to allow the comparison based on a single primary metric. In case of Cifar10, Cifar100, MNIST and ImageNet this primary metric is the accuracy score:

$$acc = \frac{TP}{TP + FP} \quad (7)$$

The accuracy can be interpreted as the percentage of the prediction that are correct. In the equation, TP (true positives) references the number of all correctly classified images and FP the number of all incorrectly classified images (false positives). An image is counted as correct when the predicted labels are matching the true label. In some publications like He et al. (2016a)

and Simonyan, Zisserman (2015), the error-rate is reported instead:

$$err = \frac{FP}{TP + FP} = 1 - acc \quad (8)$$

Due to the high difficulty of the ImageNet dataset, additionally the Top5-Accuracy is reported, which is more error-tolerant and allows therefore for a better evaluation of very hard problems. This metric deviates from the aforementioned accuracy (in this context referred to as Top1-Accuracy) in their definition of true and false positives. While the accuracy only considers the highest predicted probability category, the Top5 version of this metric considers the 5 categories with the highest predicted probabilities. A prediction is regarded as a true positive if the true category is one of these five categories.

Beside the predictive performance of the trained models, the efficiency is sometimes reported as well (Howard et al. (2019, 2017); Sandler et al. (2018); Tan, Le (2019)). This is commonly done in two ways. First, the computational efficiency measured in FLOPs (Floating-Point Operations) as a measure of computational power required to conduct a single forward pass on a single image. The second measure of efficiency is measured in the number of parameters the model has. The number of parameters can be considered a crude way of expressing the "capacity" or complexity of the model.

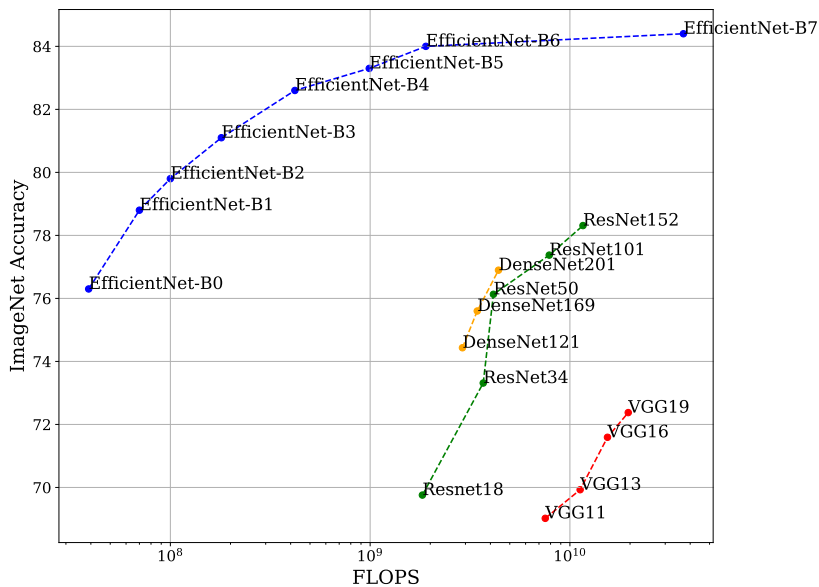


Figure 11: Network families (connected via dotted lines) are composed of models that resemble different trade-offs between predictive performance and computational efficiency (measured in FLOPs to perform a forward pass on a single image). Computationally more demanding variants of the architecture achieve the best predictive performance than their more computationally efficient relatives.

Both metrics are usually computed only once after training has concluded, since the computations required per forward pass and the number of model parameters are known based on the setup. The efficiency of models is typically not expressed in a single number, since most architecture families feature different variations of the same basic architecture design. These variations intentionally strike different balances between predictive performance and both efficiencies, with some emphasizing "high efficiency" and others "high performance". An example of such a visualization is figure 11, the predictive performance is plotted against the computations required. We can see that model families trade an increase in FLOPS for more predictive performance with decreasing returns as the models grow larger. Generally, this behavior is similar for parameter efficiency, as we can see from comparing figure 11 and figure 13.

3 The Current State of Convolutional Neural Architecture Design

The neural architecture of a deep neural network defines the characteristics of individual layers as well as the overall layout that connects the individual layers. Historically, the architecture can be considered the most significant component of a training setup for maximizing the predictive performance of CNN-based models. We can say this because the neural architecture directly defines the capacity of the model, the historically most critical factor for improving upon state-of-the-art architectures, as figure 12 illustrates. Since this work aims at improving the design process of convolutional neural networks, it is important to understand the basic design goals, conventions and building blocks of modern convolutional neural network architectures. As of the writing of this work, currently no clear rules or principles for the design of neural architectures exists in the literature. However, during the years, conventions and components have been popularized to deal with reoccurring challenges in the design of neural architectures. Section 3.1 discusses the basic design criteria that are taken into consideration when designing and optimizing a neural architecture. We then move on to discussing the state of neural architecture design on two different scales. The macroscopic scale is introduced in section 3.2, where general conventions in the design of convolutional neural network classifiers are introduced. The more detailed view is elaborated on in section 3.3 and is focused on common building blocks and their variations used in convolutional neural networks.

3.1 Design Criteria

In benchmarks scenarios, the predictive performance of the model is generally considered the primary optimization goal. However, depending on the application and the available resources, other criteria, like the memory footprint, need to be recognized during development. In this section, we will briefly discuss different criteria in the design of convolutional neural networks and elaborate on how they are reflected in design decisions.

3.1.1 Predictive Performance

As was previously mentioned, the quality of the prediction can be considered the primary goal in the development of a deep learning solution.

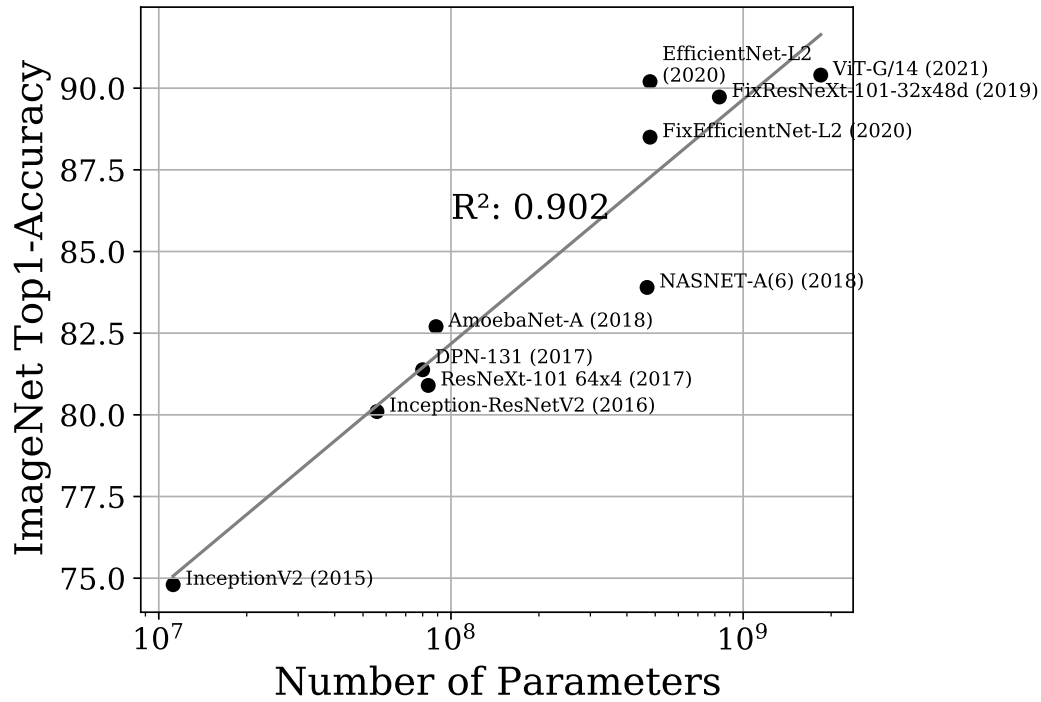


Figure 12: The Graph shows the best performing architectures of the respective years (multiple if the benchmark was beaten multiple times during the year). Over 90% of the predictive performance can be explained by an exponential increase in parameters. This indicates that the increasing capacity of neural networks is the main driving factor behind improvements in cutting edge-models.

Historically, improvements regarding the predictive performance are achieved by increasing the model's capacity, as figure 12 illustrates. The capacity of a model is measured in the amount of trainable parameters (Huang et al. (2019)). In fact, 90.2% of the ImageNet test accuracy's variance in figure 12 can be explained by the number of parameters. Improving performance by increasing the number of parameters is also done within families of CNN architectures. One strategy for increasing the capacity of models is to increase their "depth". This is achieved by the addition of layers or building blocks. Examples for this practice are the VGG (Simonyan, Zisserman (2015)), MobileNet (Howard et al. (2017)), DenseNet (Huang et al. (2017)) and EfficientNet (Tan, Le (2019)) -families. Besides building deeper architectures, it is also possible to increase the number of filters per layer and thus the number of patterns that can be extracted. This is colloquially referred to as making the architecture "wider". Tan, Le (2019) show that both strategies work reliably, but also yield diminishing returns and need to be compound-scaled together with the input resolution to be most effective.

In figure 12 we can also see that the predictive performance of the de-

picted models increased incrementally from 75 to 90% accuracy on ImageNet. In the same time, the parameters in the models have increased from millions to trillions. Generally, an increase in parameters coincides with an increase in the computational resources and memory required for training and inference. High-capacity models are therefore becoming less efficient, necessitating compromises in the design to accommodate resource limitations like the GPU-memory and available computational resources.

3.1.2 Memory Footprint

The memory footprint is especially important for non-distributed training, since GPU and TPU memory is currently not expandable and therefore putting hard restrictions on the size of the model. While data parallelism effectively scales linear (Goyal et al. (2017)), distributing a model over multiple compute devices (model parallelism) suffers from inefficiencies and increasing idle times as a result of the dependencies between the network partitions (Huang et al. (2019)). This effectively limits the size of models that can be trained with given resource and time constraints. The memory footprint is primarily determined by the size of the feature maps, the number of filters in the feature maps, as well as the number of parameters per layer. Therefore, design choices around limiting the memory footprint revolve primarily around limiting the sizes of the feature map and number of filters. Examples for such design choices are stems, discussed in section 3.3.2, bottleneck layers discussed in section 3.3.5 and the overall pyramidal shape of architectures described in section 3.2.

3.1.3 Computational Efficiency

The computational efficiency is considered the number of computations (measured in FLOPs) required to process a single input image and was discussed earlier in section 2.5 (Dosovitskiy et al. (2021); Real et al. (2019); Tan, Le (2019); Tan et al. (2020)). Since the computations required limit the data throughput of a trained model with given hardware, computational efficiency can be viewed as the most direct measure of how economical a model is in a deployment scenario. Computational efficiency is strongly related to the memory footprint, which are both heavily influenced by the size of feature maps. Therefore, components like stems (see section 3.3.2) or the use of 1×1 convolutions to reduce the number of filters (see figure 3.3.1) are also used to gain computational efficiency. Techniques that have no positive effect on the

memory footprint but reduce the number of parameters involves alternatives to regular convolutions. Depth-wise separable convolutions introduced by Howard et al. (2017) and decomposing convolutions into multiple operations (Szegedy et al. (2016)) are examples of such techniques. We discuss these in section 3.3.1 in greater detail.

3.1.4 Parameter Efficiency

Parameter efficiency measures how well the model utilizes the available capacity. While all convolutional neural networks can be considered highly over parameterized (Zhang et al. (2017)), some models utilize the available parameters better than others. This is especially true inside a network family, where deeper architectures strongly increase the number of parameters while gaining diminishing returns in predictive performance (see figure 13 for examples). While there are known strategies for increasing the computational efficiency, memory footprint and predictive performance, the design choices can be considered trade-offs, that either reduce the predictive performance or decrease memory and computational efficiency. Getting the most use out of a given number of parameters has no real strategies that map to neural architecture design directly and reliably.

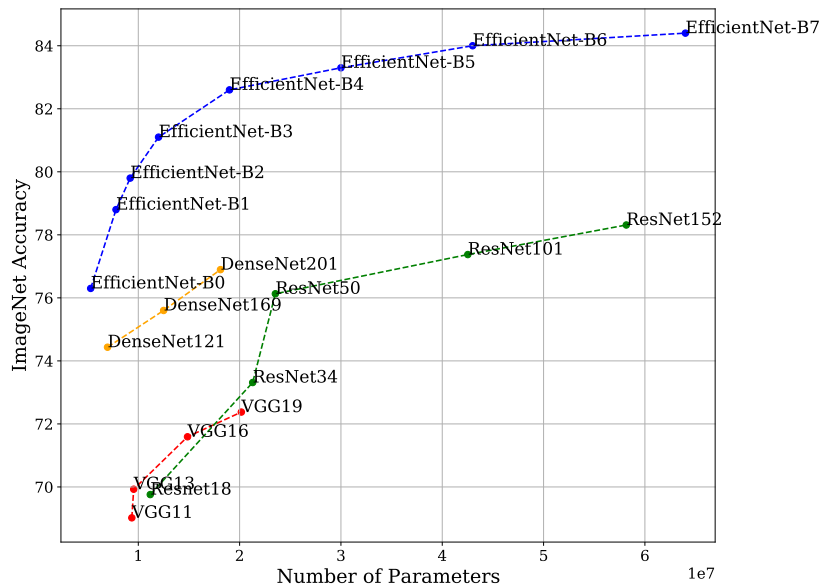


Figure 13: Networks in the same family (connected via dotted lines) that feature more parameters perform better, trading computational and parameter efficiency for predictive performance at diminishing returns.

The only experimentally validated heuristic was presented by Tan, Le (2019), who showed that compound-scaling input size, number of filters per

layer and number of layers yields effective results on multiple architectures for classification and object detection. However, this heuristic is designed for scaling architectures and does not involve designing a novel architecture for efficiency. The compound scaling strategy by Tan, Le (2019) is also parameterized, requiring an expensive grid-search to obtain the optimal compound-scaling hyperparameters.

As a general guideline: Parameter efficiency is lower for larger architectures. Increasing the number of parameter yields diminishing returns, as we can see on various network families in figure 13. We see this also a core motivation to this work, which is in large part dedicated to identifying and resolving parameter inefficiencies in convolutional neural architectures.

3.2 Basic Structural Conventions in Convolutional Neural Network Architectures

In this section, we will discuss the basic high-level structure used in convolutional neural networks. In recent years, many architectures for convolutional neural network classifiers were proposed. While these architectures strongly differ in the number, size and types of layers, all architectures achieving high predictive performance ² on ImageNet follow some basic convention in high-level architectural design (He et al. (2016a); Howard et al. (2019); Hu et al. (2018); Khan et al. (2020); Krizhevsky et al. (2012); Real et al. (2019); Simonyan, Zisserman (2015); Szegedy et al. (2015); Tan et al. (2020)).

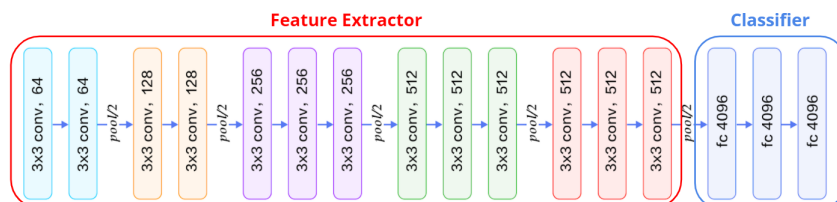


Figure 14: Convolutional Neural Network classifiers can be divided into a feature extractor and a classifier part. The feature extractor is fully convolutional and extracts discriminatory features from the input image. The classifier consists of densely connected layers and produces a prediction based on the output of the feature extractor.

To demonstrate what these high-level architecture design conventions are, we illustrated the basic structure of a convolutional neural network classifier on a simple example architecture in figure 15. The overall feed forward structure can be divided into two main parts. The first component is the feature extractor and is fully convolutional. The second part, referred to as

²we consider 70% Top1-Accuracy as high performance

"classifier", is composed of densely connected layers and is separated from the feature extractor by a transition layer also referred to as "readout-layer", which transforms the feature map of the convolutional layers into a vector representation that can be consumed by the classifier.

3.2.1 Classifier

We refer to the final sequence of layers as "classifier". This includes the output layer of the network. The classifier is separated from the feature extractor by the read-out layer. The primary responsibility of the classifier is to transform the feature maps of the feature extractor into a prediction. In older architectures like LeNet by Lecun et al. (1998) and the VGG-family of networks by Simonyan, Zisserman (2015) the classifier is a multi layer perceptron and is thus capable of non-linear transformations on the entire image. Modern architectures like the Inception-network family by Szegedy et al. (2015), the ResNet-network family by He et al. (2016a) and derivative architectures like SENet by Hu et al. (2018) and Wide-ResNet by Zagoruyko, Komodakis (2016) etc. moved to a linear classifier design. These networks feature a single classifier-layer, which is also the softmax-output of the network. This classifier design was pioneered by Lin et al. (2014) and later popularized by Szegedy et al. (2015) as a simple yet effective way to avoid over-fitting and reduce required memory and computational resources. From a conceptual level, a single layer classifier also reduced the core responsibilities of the classifier to that of a linear model, relying on the feature extractor to transform the data into a linear separable representation.

3.2.2 Feature Extractor

We define the feature extractor as the sequence of layers before the readout-layer. The feature extractor is fully convolutional, in the sense that all layers are kernel-based ³, which makes the feature extractor technically agnostic towards the input resolution.

Traditionally, the feature extractor contains the most layers in the neural architecture (He et al. (2016a); Krizhevsky et al. (2012); Simonyan, Zisserman (2015); Szegedy et al. (2015)). Early designs like the VGG family of networks by Simonyan, Zisserman (2015) can be described as a sequence of consecutive layers, where every layer receives input from a single previous layer.

³we regard, in this context, non-parametric layers like pooling-layers as convolutional layers as well.

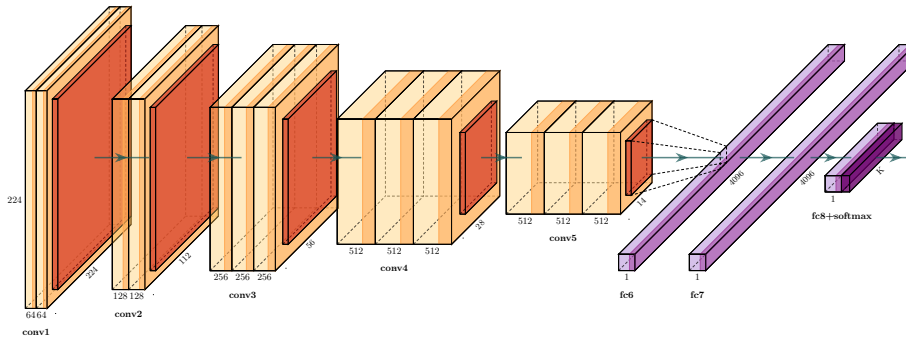


Figure 15: The feature extractor of VGG16 consists only of convolutional (yellow) and pooling-layers (orange). Note the pyramidal structure of the feature maps caused by the 4 downsampling layers (orange).

We refer to models with these feature extractors as sequential architectures (see figure 15). The strictly sequential architectures have since fallen out of favor, since they are inherently limited in their depth by the vanishing gradient problem (He et al. (2016a), Szegedy et al. (2015)). For enabling training of deeper architectures, non-sequential architectures like ResNet (He et al. (2016a)), DenseNet (Huang et al. (2017)) and MobileNetV3 (Howard et al. (2017)) were developed. These architectures feature multiple pathways in the feed-forward structure, allowing the signal to effectively "skip" certain layers, counteracting the vanishing gradient problem.

The primary purpose of the feature extractor is detecting discriminatory patterns from the image, that allow the classifier to make a prediction. When considering that classifiers in modern architectures are linear, the feature extractor of a trained model can be also seen as a trainable data preprocessor that produces a representation of the data that is as linear separable as possible.

3.2.3 Pyramidal Architectures

Another observation that can be made from the VGG16 feature extractor in figure 15 is the pyramidal shape of the feature maps. By convention, the pyramidal structure separates the feature extractor into a number of roughly evenly sized stages (a common number of stages is 4).⁴ The number of layers

⁴The following architectures used 4 stages: The VGG-family by Simonyan, Zisserman (2015), the inception family by Szegedy et al. (2015) and Szegedy et al. (2016), the ResNet-family by He et al. (2016a), the MobileNet-family by Howard et al. (2019, 2017); Sandler et al. (2018), the NASNet-Networks by Zoph et al. (2018), AmoebaNet by Real et al. (2019), EfficientNet by Tan, Le (2019), SqueezeNet by Iandola et al. (2016), RexNext by Xie et al. (2017), ResNetV2 by He et al. (2016b) and Xception by Chollet (2017). This list may be incomplete, but it contains all architectures the authors considers significant since 2012 in the

and filters in a stage relative to the other stages is commonly held consistent within a architecture family (He et al. (2016a); Howard et al. (2019); Real et al. (2019); Sandler et al. (2018); Simonyan, Zisserman (2015); Tan, Le (2019)). By transitioning from one stage to the next, the feature map is downsampled, halving the height and width of the feature map in the process.⁵ Also part of this convention is the increase in the number of filters in the feature map, which is doubled with each stage compared to the previous. There are multiple reasons for why this pyramidal structure is still commonly used. First, the pyramidal structure makes the training of deeper networks more efficient. By downsampling feature maps, the memory consumption and computations of all following convolutional layers are reduced by a factor 4 compared to the current feature map size. Another important point is that the reduction in height and width of the feature maps allows for the addition of more filters in later layers. The intuition for doing so brought forward by Simonyan, Zisserman (2015) is, that later layers combine the patterns of previous layers to create more abstract patterns. Increasing the number of filters in later layers effectively increases the capacity of later layers for detecting additional patterns by combining the existing patterns.

3.2.4 Fully Convolutional Neural Networks

Convolutional Neural Networks are referred to as "fully convolutional" when every layer in the network functionally behaves like a convolution. This is conventionally done by using a global pooling layer like global average pooling (GAP) as the readout layer. These global pooling layers reduce the output of the feature extractor down to a single vector with a dimensionality equal to the number of filters in the input feature map. This is done by applying a parameter-less operation like computing the maximum or average activation value of each filter. Since the output of the global pooling layer can be considered a feature map with height and width of 1, every following densely connected layer is therefore functionally indistinguishable from a convolution with a kernel size of 1×1 .

There are two advantages compared to a conventional reshaping (flattening) as a readout-technique. The first advantage is dimension reduction from a vector of $(H * W * C)$ to a vector of size (C) , where C is the number of filters and H and W represent height and width of the feature maps. This

field of classification.

⁵the only exception to this rule is the final stage, where the stack of filters is transformed into a vector

is especially important in the readout layer, since the number of parameters in the first layer after the readout grows quadratic with height and width of the feature map, which in turn risks the danger of overfitting and increases computation time. Another advantage of the fully convolutional network is, that the network becomes agnostic towards the input resolution, as long as the downsampling operation is not reducing the resolution of a feature map below 1×1 during the forward pass. This allows different resolutions to be processed without altering the architecture. While this property is not actively used in many publications on deep convolutional image classifiers, there are exceptions like the works of Tan, Le (2019) and Real et al. (2019) that utilize this property to build computationally efficient network families. We will also utilize this property in chapter 9 to demonstrate the effect of the input resolution on the inference process in convolutional neural networks.

3.3 Conventions in Neural Architecture Design

In this section, we will talk about components commonly used in convolutional neural networks. We will briefly present the basic components that have seen applications in notable publications in recent years. Furthermore, we discuss alterations of these components and elaborate on the problems the architectural components are designed to solve.

3.3.1 Convolutional Layers

What is often referred to as a convolutional layer in the literature is actually composed of multiple components that are executed in sequence and could be technically viewed as separate steps in the feed forward architecture.

The sequence is composed of 3 steps in older architectures like VGG16 and AlexNet and 4 steps in more modern architectures such as DenseNet, EfficientNet and InceptionV4 (see figure 16). Padding is conducted before the convolution is applied. By convention, zero padding is applied such that the kernel of the convolution can be placed with the center pixel on any position of the input feature map. When the stride size of the kernel is 1 in all directions, the input feature map of this sequence will have the same shape as the output feature map, for this reason, this padding strategy is referred to as "same padding". Convolutions in convolutional neural networks primarily utilize 3×3 and 5×5 kernels. In very deep convolutional neural architectures like ResNet, DenseNet and EfficientNet, the bias of the convolution is omitted. Deviations from this do exist. AlexNet, for instance, uses a 11×11

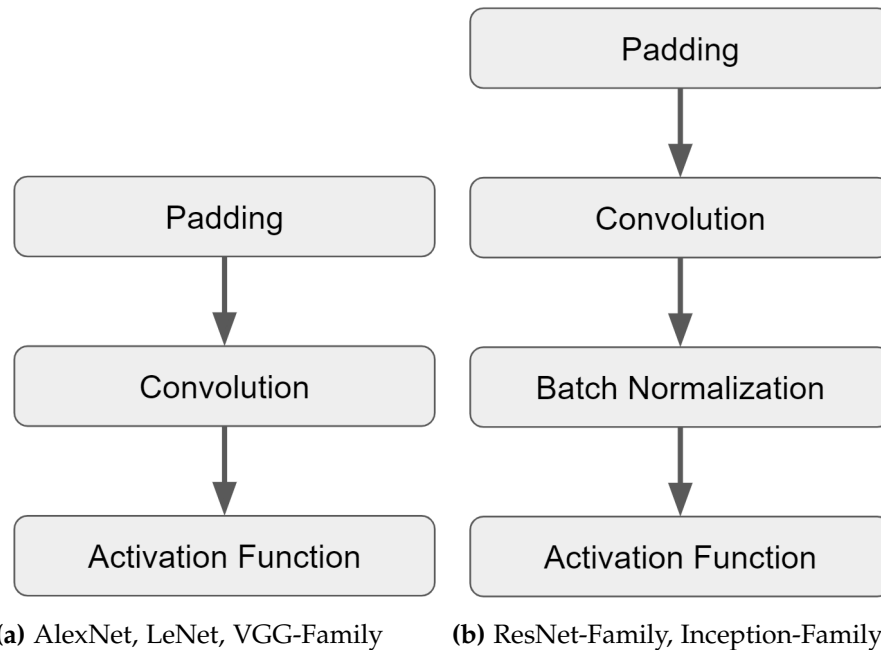


Figure 16: Two common ways convolutional layers are implemented. The variant depicted in (a) is commonly used in older architectures such as VGG16. The variant depicted in (b) used in more recent architectures such as MobileNet, EfficientNet and AmoebaNet. When processing convolutional layers, the feature map is padded (zero padding is the norm) to keep input and output shape independent of the kernel size. The output of the convolution-operation is fed into a batch-normalization to combat the vanishing-gradient problem and a non-linear activation function to make the layer non-linear.

kernel in the first layer. Convolutional neural networks with a stem (see section 3.3.2) sometimes use a single convolutional layer with a large kernel e.g. 7×7 in case of ResNet (He et al. (2016a)).

As a way to save computational resources and reduce parameters, the regular convolution is sometimes replaced with a depthwise-seperable convolution, where every feature map is processed independently. EfficientNet, MobileNetV_{1, 2} and 3 are examples for this. Other economical conventions involve the decomposition of the convolutional operation into multiple operations. InceptionV₄, for instance, emulates a $k \times k$ convolution by applying a $1 \times k$ and a $k \times 1$ convolutional operation in direct sequence. This reduces the number of parameters from $infilter \times k \times k \times \#outfilters$ to $2 \times \#infilters \times k \times \#outfilters$ (Szegedy et al. (2017)). A similar approach is used by MobileNetV_{1,2} and 3 as well as EfficientNet, which decompose $k \times k$ convolutions into a $k \times k$ depthwise separable convolution followed directly by a 1×1 convolution operation, effectively separating the processing of the individual filters and the combination of these into separate steps (Howard et al. (2019, 2017); Sandler et al. (2018); Tan, Le (2019)).

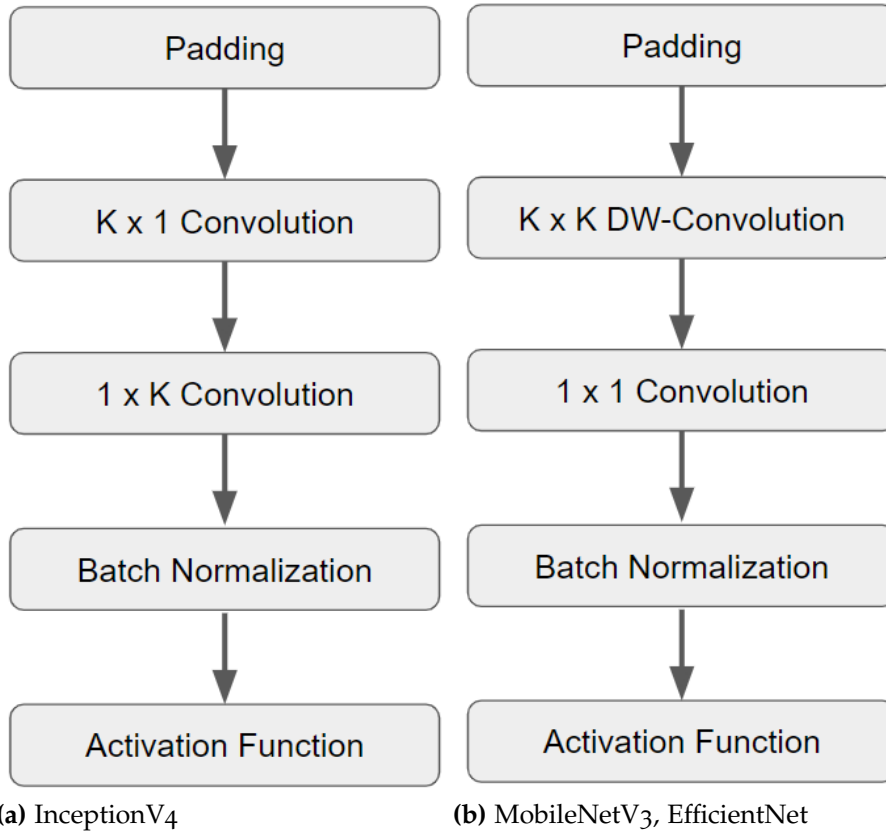


Figure 17: To reduce parameters and the number of computations required to process the convolution, the convolution operation is sometimes decomposed into multiple operations. This can be achieved by separating the kernel into two 1-dimensional kernels of shape $k \times 1$ and $1 \times k$ (a). Another possibility (b) is to filter-wise and spatial processing by decomposing the convolution into a depth-wise separable convolution and a convolution with 1×1 kernel.

Batch Normalization (originally proposed by Ioffe, Szegedy (2015)) is used by many architecture-families such as DenseNet, EfficientNet and ResNet to combat the vanishing gradient problem and improve predictive performance. Based on the widespread use of batch normalization, it can be argued that the use of Batch Normalization has become the de facto standard in modern architectures (He et al. (2016a); Howard et al. (2019); Real et al. (2019); Szegedy et al. (2016)).

The activation function induces the non-linearity into the convolutional layer. While the ReLU function can be considered one of the more popular choices (used by AlexNet, VGG, ResNet, InceptionV1-3). Attempts to replace the ReLU to avoid the dying ReLU problem⁶ were made multiple times by authors like Klambauer et al. (2017) and Ramachandran et al. (2018). Recent architectures like the EfficientNet, AmoebaNet and MobileNetV3 have switched

⁶in certain weight-configurations a ReLU could "die" by being effectively unable to produce any other output than 0, functionally removing the underlying neuron from the network. This state cannot be recovered from.

to other non-linearities like the swish activation function by Ramachandran et al. (2018) and ReLU6 in case of MobileNetV1.

3.3.2 Stem

One of the key advantages of downsampling is the reduction of computations and memory required to process any consecutive layer after the downsampling layer. This property becomes increasingly important for very deep architectures like ResNet, which was trained with up to 1001 layers by He et al. (2016a). For this reason, many architectures use additional downsampling layers directly in front of the input to boost the efficiency of the entire model. By doing so, information from the larger resolution can still be integrated by the first layers, while the network operates on feature map size that would otherwise require a smaller input resolution.

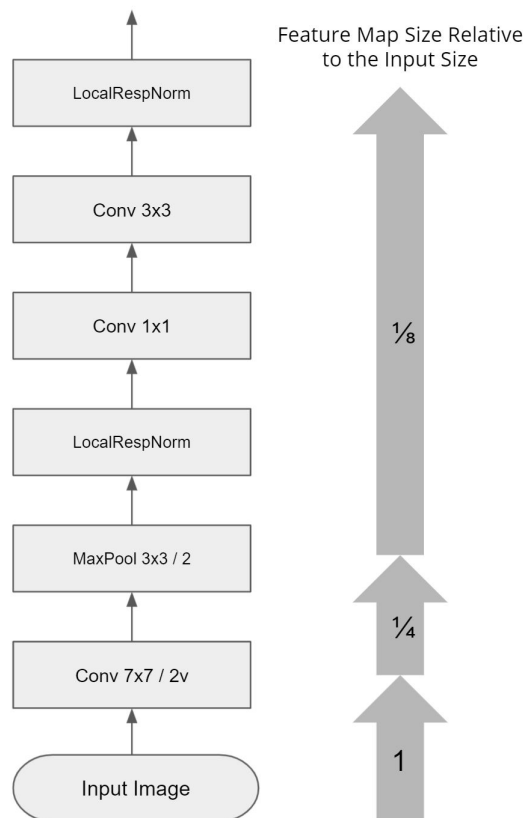


Figure 18: Stems are a sequence of layers at the input of a neural network that does not consist of the regular building blocks like the rest of the architecture. The main purpose of these layers is reducing the height and width of the feature map for consecutive layers. The first two layers of the depicted stem reduce the size of the feature map by a factor of 4, making the entire architecture more computationally efficient. Later architectures like ResNet and DenseNet adopt a simplified version of this stem, which only consists of the first two layers. This particular stem belongs to the GoogLeNet architecture by Szegedy et al. (2015).

This short sequence of layers is part of the feature extractor, but is not in-

cluded into the otherwise repetitive building-block structure. (Chollet (2017); Huang et al. (2017); Xie et al. (2017)). For this reason, we refer to this sub-sequence as "stem", based on terminology coined by Szegedy et al. (2017). The first architecture to utilize a stem is GoogLeNet by Szegedy et al. (2015), depicted in figure 18. A more aggressively downsampling stem is used by InceptionV4 by Szegedy et al. (2017), reducing the size of the input resolution from 299×299 pixels to 35×35 in the output of the last stem layer. A more common version of stem, used by multiple architectures such as AmoebaNet, ResNet, EfficientNet, NASNet, MobileNetV3 and other ResNet-derivatives, only consists of the initial 7×7 convolution followed by the 3×3 max-pooling layer (both with stride size 2).

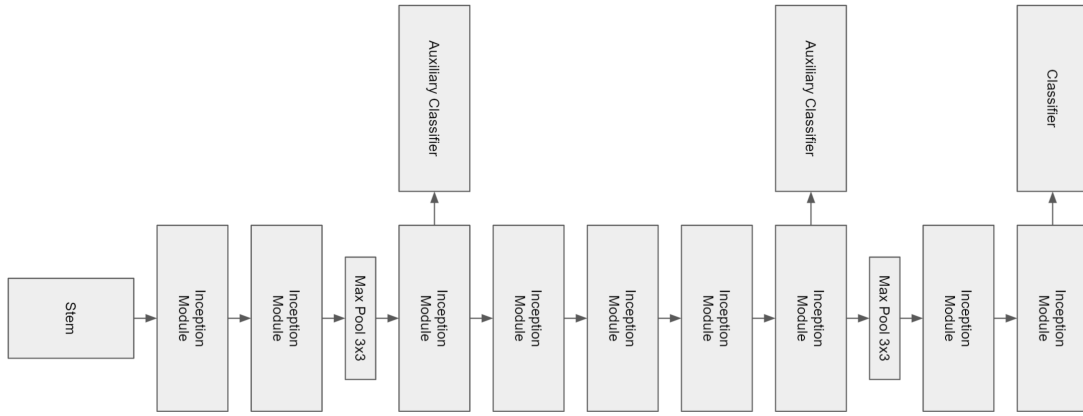
3.3.3 Building Block-Style Feature Extractors

With increasingly deeper architecture, the design of convolutional neural architectures were further structured and modularized. The idea of modularization is to have a common, shared interface of building blocks, that allows for easy construction and modification of the architecture. Typically, the building blocks are sequentially stacked into a feed forward architecture. The number and heterogeneity of building blocks varies strongly from architecture to architecture and from architecture family to architecture family. GoogLeNet is the first architecture that is mostly composed of building-block like structures (see figure 19) based on a single "Inception-module". Later iterations of the Inception architecture like Inception-ResNet, InceptionV3 and InceptionV4 used multiple different building blocks (Szegedy et al. (2017, 2016)).

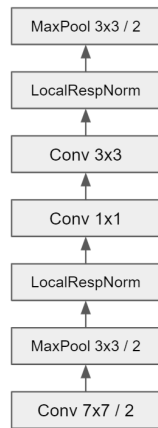
It can be argued that recent architectures are mainly iterating on the design of building blocks. The higher level design (number of building blocks per stage, number of downsampling layer etc.) has become increasingly static even across families of architectures like EfficientNet, AmoebaNet and MobileNetV3, ResNet etc. This is further emphasized by state of the art Neural Architecture Search algorithms like the ones used for creating AmoebaNet and NASNet, which kept the higher level structure static while only altering the internal structure of the building blocks to optimize the models (Real et al. (2019); Zoph et al. (2018)).

3.3.4 Skip Connections

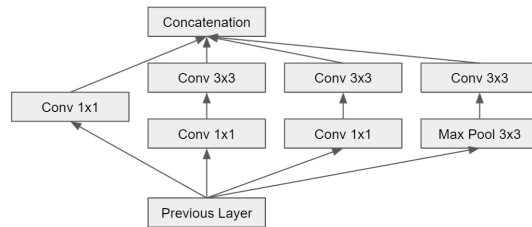
While deeper networks tend to deliver better predictive performance, the vanishing gradient problem effectively limits how deep simple sequential con-



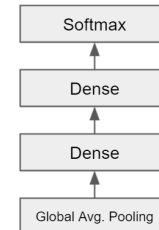
(a) Overall Architecture



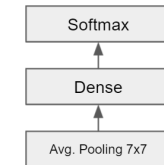
(b) Stem



(c) Inception Module



(d) Aux. Classifier



(e) Classifier

Figure 19: The GoogLeNet architecture is one of the first architectures to rely on repeated building-blocks (Inception Modules) used throughout the network.

volutional neural networks can be trained without negative impact on the predictive performance. Over the years, multiple solutions to the vanishing gradient problem were proposed, like auxiliary classifier output by Szegedy et al. (2015) and Batch Normalization by Ioffe, Szegedy (2015) to name only two examples. The skip connections can be considered the most effective solution to the vanishing gradient problem. The use of skip connections ultimately enabled He et al. (2016a) and He et al. (2016b) to train models with more than 1,000 layers. Skip connections are a special kind of non-sequential architecture component used in architectures like ResNet, DenseNet, NAS-Net, EfficientNet, AmoebaNet, MobileNetV2, MobileNetV3 and InceptionV3 as well as other derivative architectures such as WideResNet or Inception-ResNet. The basic principle of skip connections is to allow the network to "skip" parameterized layers by using a secondary path. By doing so, the information in the forward pass and the gradient in the backward pass can

effectively jump over subsequences of layers, resolving the vanishing gradient problem in the process. Another view on skip connections is that they allow the building blocks to add "deltas" to the existing representation of the data (He et al. (2016a); Huang et al. (2017)) instead of transforming it entirely.

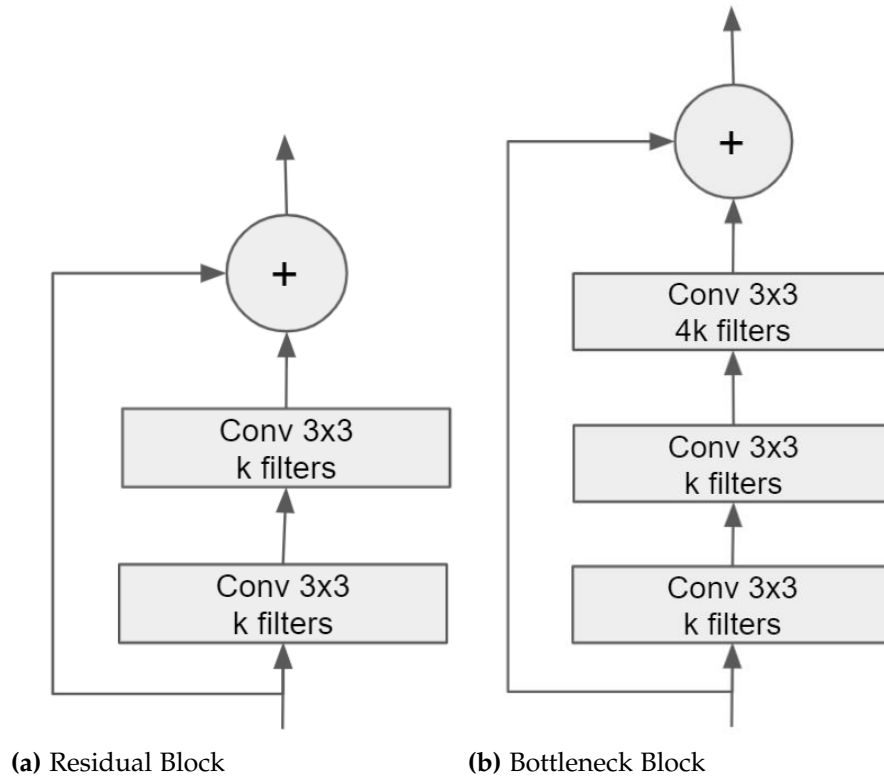


Figure 20: The original implementations of residual connections add the input of the building block to the pre-activation output of the same block, allowing signal and gradient to skip the encapsulated layers. The left version is used by ResNet18 and ResNet34. The right version is used by ResNet50, 101, 152 and 1001.

Over the years, multiple variants of the skip connection have been proposed. One of the earliest designs is the residual connection by He et al. (2016a). Building blocks using the residual connection are depicted in figure 28. The residual connection adds the input to the pre-activation output of the final layer of the building block by an element wise addition. A parallel development to the residual connection is the Highway-Connection by Zilly et al. (2017), which features 1×1 convolutions in the skip-connection that act as "gates" for the information.

A more substantial change to the skip-connection concept is made by Huang et al. (2017), with the introduction of the "DenseBlock" (figure 21). A DenseBlock building block consists of a sequence of convolutional layers. The output of all previous convolutional layers in the DenseBlock is concatenated and fed into the current convolutional layer. The idea of the DenseBlock

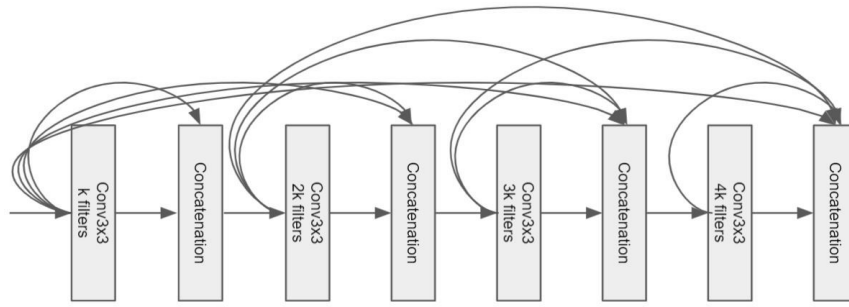


Figure 21: A DenseBlock concatenates the output of every previous convolutional layer to the output of the current convolutional layer in an attempt to minimize the loss of information inside the DenseBlock.

is to minimize the loss of information that could occur by passing information through more layers than necessary. While less popular in classification, this concept is highly successful in the related field of object detection, where the variance of object sizes and regions of interest are generally much higher (Redmon, Farhadi (2017), Redmon, Farhadi (2018), Lin et al. (2017), Bochkovski et al. (2020), Tan et al. (2020)).

3.3.5 Bottlenecks and Inverted Bottlenecks

The number of filters in a convolutional layer controls how many patterns a single feature map can extract. For this reason, it makes sense to have as many filters as possible in each convolutional layer. However, by doing so the layer’s memory footprints, parameters and required computation are increased. This makes deeper networks increasingly uneconomical and thus necessitates dimension reduction.

The pyramidal shape of ResNet, VGG, and other network families discussed in section 3.2 as well as the usage of a stem described in section 3.3.2 are ways to reduce the overall resource requirements of a convolutional neural network architecture. A third technique is the use of 1×1 convolutions, which can be used to reduce the number of filters. This commonly referred to as a bottleneck, and we can see an example of this in figure 28 (b), the first convolutional layer in the building block is reducing the dimensionality by a factor 4, reducing the computation and memory footprint of the building block. A second 1×1 convolution at the end of the bottleneck is expanding the feature map again. By doing so, the computationally expensive 3×3 convolution operates with fewer filters, making the building block more economical, while the input and output number of filters stays the same.

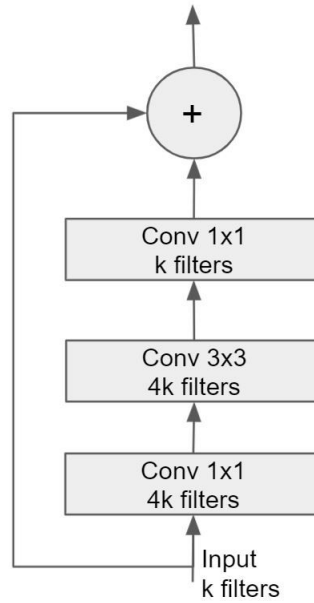


Figure 22: MobileNetV2s inverted residual block follows a narrow-wide-narrow approach in contrast to ResNets wide-narrow-wide approach, allowing the feature extracting 3×3 convolution to detect more patterns by increasing the number of filters. The increase in computations is counteracted by using a depth-wise-separable convolution.

For this reason, the bottleneck building block was used for deeper architectures like ResNet50, 101, 152, 200 and 1001, while the less economical residual block in figure 28 (a) is used for the shallower ResNet18 and 34 architectures (He et al. (2016a), He et al. (2016b), Zagoruyko, Komodakis (2016)). Designs similar to the bottleneck were used in DenseNet and SqueezeNet (Huang et al. (2017); Iandola et al. (2016)).

The inverted bottleneck used in MobileNetV2 (see figure 22), MobileNetV3 and EfficientNet (Howard et al. (2019); Sandler et al. (2018); Tan, Le (2019)) inversely trades computational performance for predictive performance by expanding the number of filters inside a building block and compressing it in the end. By doing so, the number of patterns the 3×3 convolution inside the block can detect is increased without increasing the number of output filters. By using a depth-wise separable 3×3 convolution, the increase in computational resource requirements is mitigated.

Another common point where bottlenecks are applied is before and after junctions of a building block with multiple pathways. Because pathways are unified by a concatenation operation, the number of filters resulting from the concatenation of various pathways can become too large, increasing the memory and computational footprint of all consecutive layers. This can be counteracted by a 1×1 convolution, which reduces the number of filters to a lower, more manageable number. Examples of this can be seen in figure

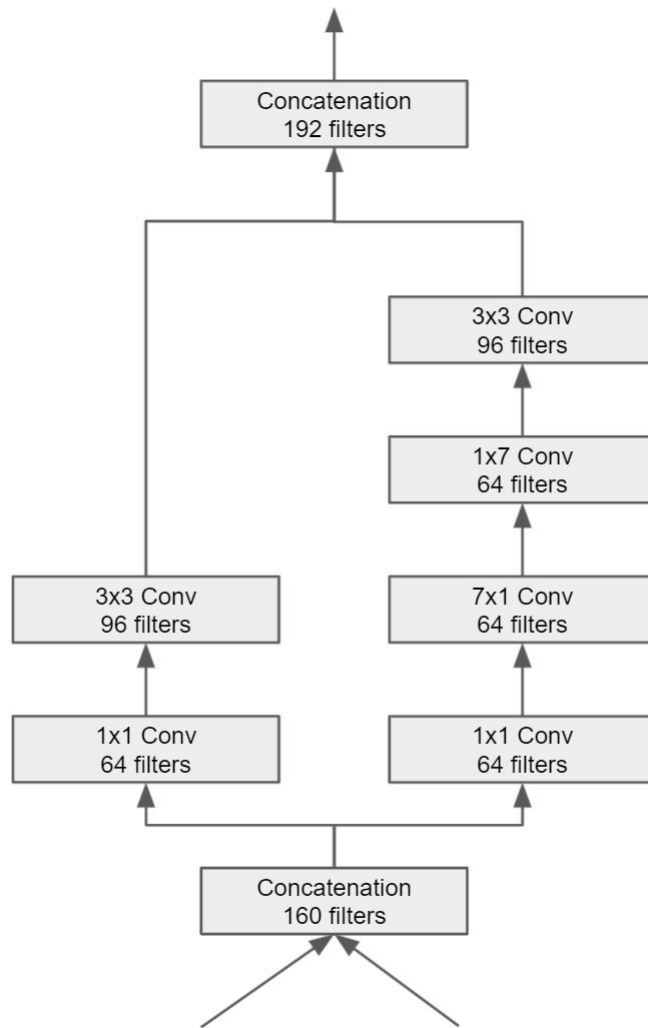


Figure 23: The stem of InceptionV4 uses 1×1 convolutions to compress the feature map in both pathways in order to reduce the number of filters after concatenation.

21 and figure 23. InceptionV3, Inception-ResNet and SqueezeNet are other noteworthy architectures that make use of this design.

3.3.6 Attention Mechanisms

Attention mechanisms are an additional weighting mechanism that are applied on the output of a layer in some form of multiplication operation. Therefore, attention mechanisms can be interpreted as layers with weights that are dynamically computed from the input of the model. The implementations of attention mechanisms can be subdivided into two subgroups. The first being the add-on-type attention mechanisms, the second type the transformer-based attention mechanism. Add-on-type attention mechanisms such as squeeze-and-excitation-modules by Iandola et al. (2016), spatial-attention by Woo et al. (2018) and CBAM by Woo et al. (2018) are added on to preexisting build-

ing blocks. Thus, addon-type attention modules are not considered stand-alone components. Addon-type attention mechanisms have seen application in state-of-the-art models like pr existing and SENet and EfficientNet (Howard et al. (2019); Hu et al. (2018); Tan, Le (2019)). According to Hu et al. (2018), Tan, Le (2019), Howard et al. (2019) and Sandler et al. (2018) the performance of the model is reliably increased while the efficiency is slightly degraded (> 3% more computation required according to Hu et al. (2018)). These attention mechanisms are commonly applied after the final layer of a building block, where they enhance the feature map by multiplying the attention weights to the features.

Transformer-type attention was originally proposed by Vaswani et al. (2017) and can be seen as full building blocks that are able to replace convolutional layers within a neural network structure as a core component of the architecture. The latter has been attempted successfully first by Dosovitskiy et al. (2021). However, as of the writing of this work transformer-based systems require a larger model and substantially more training data to outperform convolutional neural networks (Dosovitskiy et al. (2021); Heo et al. (2021); Touvron et al. (2021b); Zhou et al. (2021)). The training methodology also diverges from conventional classifier-systems by requiring a transformer-specific pre-training strategy followed by a fine-tuning step (Caron et al. (2021); Dosovitskiy et al. (2021)). Transformer-based types of attention for computer vision are currently very active field of research with currently no established standard or convention of implementation in the field of computer vision (Chu et al. (2021); Graham et al. (2021); Touvron et al. (2021a,b); Zhou et al. (2021)). For these reasons and since this work discusses convolutional neural networks, we focus in this work on the addon-type attention mechanisms like filter (also known as squeeze-and-excitation modules and spatial-attention), which have been used in multiple established classification architectures in recent years like MobileNetV2, MobileNetV3, SENet, AmoebaNet and EfficientNet.

The filter-attention mechanism is depicted in figure 24 and effectively multiplies a single weight scalar to each feature map, that was generated by a quasi-autoencoder from the same feature map. A quasi-autoencoder is effectively composed of two 1×1 convolutions, that process a globally pooled version of the feature map. The first convolution is reducing the number of filters (squeeze), while the second one is expanding the number of filters again the original amount. By doing so, the information is put through a bottleneck, forcing the squeeze-and-excitation module to compress the information

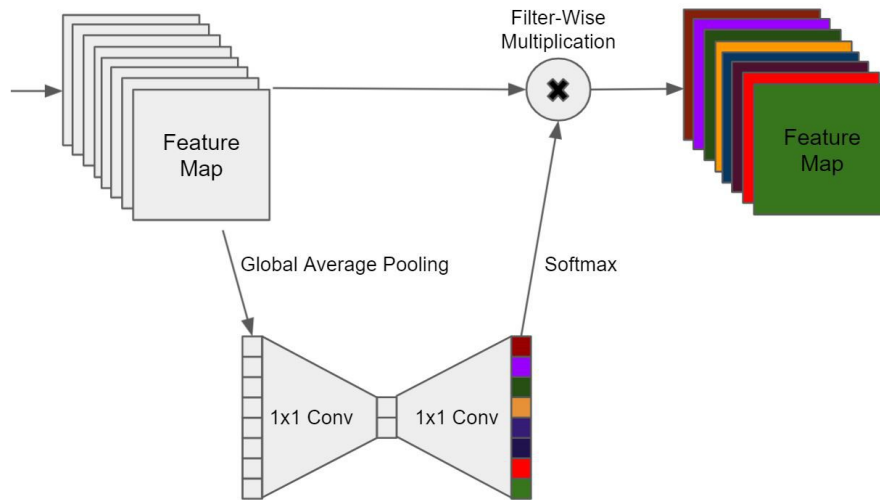


Figure 24: An illustration of a squeeze-and-excitation module. The filters of a feature map are weighted against each other. The weights are generated dynamically from the feature map by a quasi-autoencoder sub-network, that produces a single weight for every filter. The weight is applied by filter-wise multiplication of the weight-vector with the feature map tensor

contained in the pooled vector. The resulting weights for the filters are scaled by a softmax operation and multiplied to their respective filters (excitation).

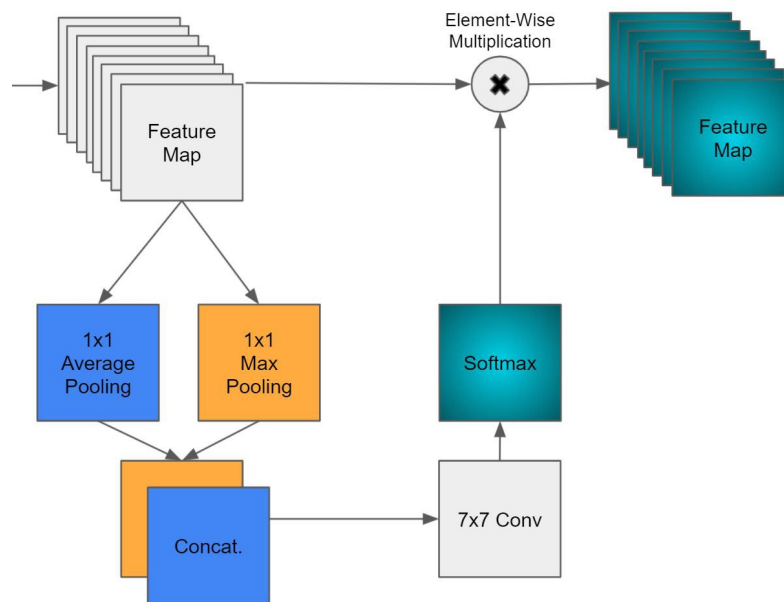


Figure 25: This diagram illustrates the functionality of spatial attention. Spatial attention weights the positions of the feature map against each other, highlighting important regions on the image in the process. First, the filters are reduced to two filters by concatenating the result of 2 pooling layers with 1×1 -kernels. This is followed a 7×7 convolution with a softmax activation function. This layer reduces the feature map to a single filter. The resulting map of attention weights is then multiplied element-wise to each filter of the original feature map.

A very similar form of attention used in convolutional neural networks is

spatial attention, introduced by Woo et al. (2018). A spatial attention module uses 1×1 pooling layers and a convolution of the same kernel size to reduce the stack of filters to a single feature map (see figure 25). This weight-feature-map is then multiplied element-wise to each of the original filters. The softmax function in both attention-implementations is crucial for forcing the model to distribute the attention effectively, since the attention-weights have to sum up to 1. Depending on the attention mechanism this induces an implicit sense of locality for the object of interest (spatial attention) or which combination of extracted features is most significant (squeeze-and-excitation) for categorization.

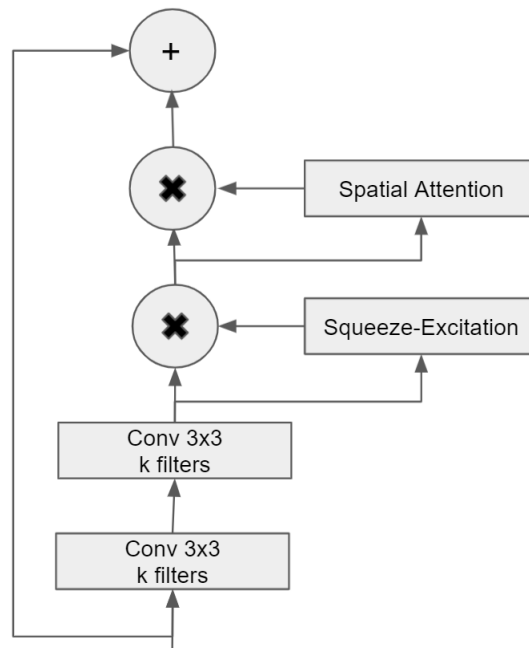


Figure 26: CBAM is a combination of spatial and filter wise attention. The above illustration depicts a residual block by He et al. (2016a) with CBAM added into the building block. CBAM combines spatial attention and filter attention (squeeze-and-excitation modules) for more fine-grained combined attention that considers all three tensor axis of the feature map.

It is also possible to directly link spatial and squeeze-and-excitation modules to combine a dynamical weighting for features and location (see figure 26), this combined attention is also referred to as CBAM and was introduced by Woo et al. (2018). While not actively used by any high scoring ImageNet-classifier as of the writing of this paper, CBAM is used in object detection system like YoloV4 (Bochkovskiy et al. (2020)), where the spatial attention due to greater variance in object size is more important. The effect of these different attention mechanisms is visualized in figure 27 using the Grad-CAM class activation map by Selvaraju et al. (2017). Based on this visualization, we

can see that the class activations are sharper, when the attention mechanisms are used inside the networks structure.

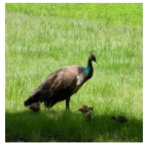




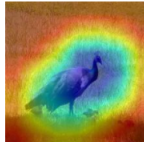
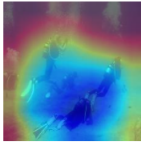

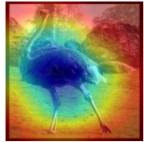


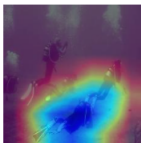



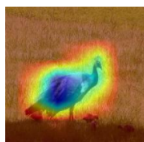
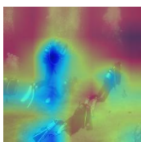



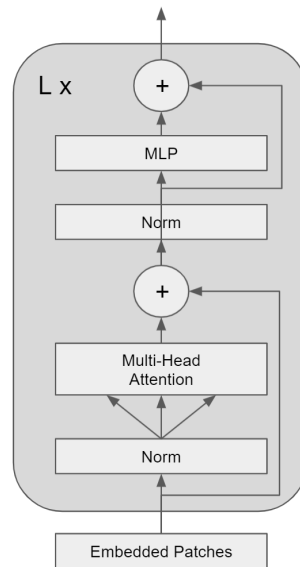
	Peacock	Scuba Diver	Moving Van	Ostrich	Chainsaw
Original					
ResNet50					
ResNet50+SE					
ResNet50+CBAM					

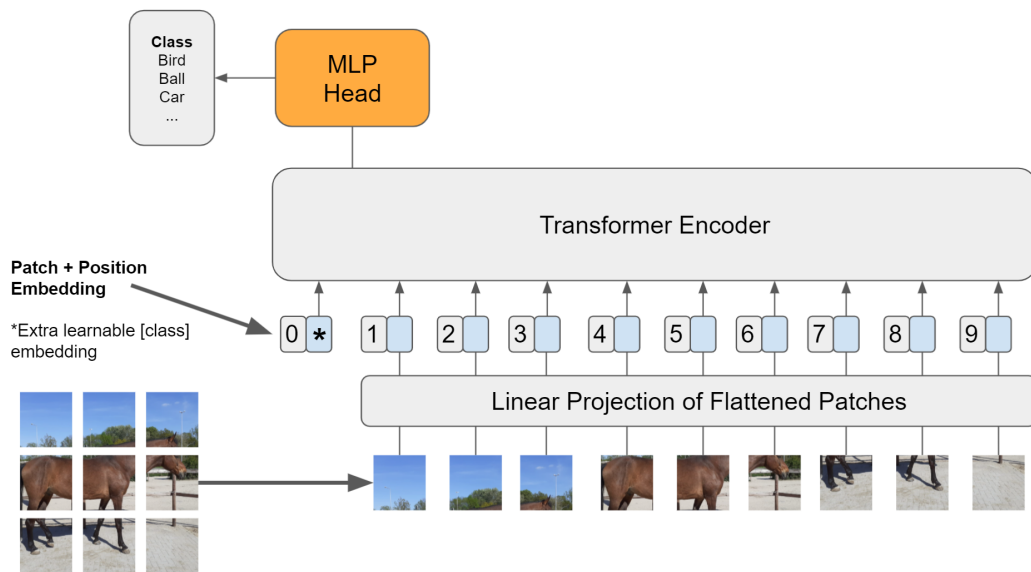
Figure 27: Grad-CAM class activation on ImageNet images from three ResNet50 models. The first model is the baseline ResNet50, the second (ResNet50+SE) uses squeeze-and-excitation modules for filter attention and the third model using filter and spatial attention (CBAM). We see that attention mechanisms are sharpening the contours of the class activation’s, bringing them closer to the object of interest.

Self-attention mechanisms more closely related to the attention mechanism presented by Vaswani et al. (2017) are also transferred to computer vision tasks. In implementations like the self-attention convolutional layer presented by Ramachandran et al. (2019) are hybridized convolutional add-ins that also use a self attention component. Purely attention-based image classification has been successfully attempted by Dosovitskiy et al. (2021) too, who build a fully attention-based image classifier. Ramachandran et al. (2019), Dosovitskiy et al. (2021), Touvron et al. (2021b) and Zhou et al. (2021) demonstrate that purely transformer-based models can achieve state-of-the-art performance on ImageNet. However, there are currently serious drawbacks to these approaches. For instance, ViT by Dosovitskiy et al. (2021) requires 300 million additional weakly labeled training images to outperform state-of-the-art convolutional classifiers (ImageNet has 1 million training images) and the attention-based ResNet derivatives of Ramachandran et al. (2019) are only

outperforming ResNet models when all kernel-sizes of ResNet are increased to 7×7 , which is an architectural choice that is not beneficial in any way to the predictive performance or efficiency of ResNet.



(a) Transformer Encoder



(b) The overall ViT architecture

Figure 28: It is also possible to create attention-based image classifiers based on transformers (a). The depicted ViT model achieves state-of-the-art performance on ImageNet without requiring convolutional layers. The transformer stack (b) is processing regional embeddings together with visual embeddings to obtain a prediction. This technology can be considered still in the early phases of development, since they require expensive pre-training with 300 times more data and are more expensive to train than convolutional neural networks with similar performance (Dosovitskiy et al. (2021)).

For this reason, these models can be currently considered mostly in the proof of concept stage for classification tasks in computer vision and are therefore not further considered in this work. However, we think it is important to mention these novel approaches, since they demonstrate that purely convolutional neural networks are not necessarily the only possible solution for achieving the state-of-the-art performance on benchmark datasets in computer vision tasks.

4 Analyzing Convolutional Neural Networks

The primary goal of neural architecture design is to maximize the predictive performance and efficiency for a given dataset. Chapter 3 established that the current design of neural architectures is driven by conventions and the maximization of abstract metrics like predictive performance and various metrics of efficiency. In effect, this leads to a comparative and thus trial and error-driven mode of development. In order to move to a more informed mode of development additional insights and thus a more elaborate analysis of the model is required. For this reason, we will discuss related work on analyzing convolutional neural architectures in this chapter.

We define an analysis technique as a method that provides insights into the trained model. Analysis techniques are different from metrics, which were discussed in section 2.5 in multiple ways: For one, metrics are expected to be agnostic towards the implementation of the model. In other words, metrics like accuracy always treat a model as a black box system. Analysis techniques can be model specific and can thus provide more detailed insights. Additionally, these techniques are not bound to produce scalar values. Instead, a tool for analyzing a trained model can create more complex output that may require extensive evaluation on its own e.g. GradCam by Selvaraju et al. (2017), which produces a class activation map for every combination of images in the dataset and thus requires through inspection. An example of this can be seen in figure 27.

The surrounding literature on this topic that is not directly involved in our work is discussed in section 4.1 to provide additional context. We will then move on to introduce the concept of the feature space and how it is analyzed in the literature. Next, we will elaborate logistic regression probes in section 4.3, which is a concrete tool for analyzing the inference process, that we use in this work. Finally, we will discuss the concept of receptive fields and introduce the arithmetic used to compute the receptive fields of convolutional neural network layers.

4.1 Related Work

Neural Networks in general are non-linear models with millions of parameters (Dosovitskiy et al. (2021), Huang et al. (2019)). This makes an exhaustive yet easily understandable explanation of the process leading from input to prediction (inference process) impossible. To allow for an explanation of the inference process to be useful, methods need to produce a simplified view

that can still provide interesting insights. Generally, techniques for analyzing neural networks can have different levels of abstraction on model and dataset. For instance, the error surface visualization by Li et al. (2018b) can be considered very abstract, since the model is seen primarily as a "bag of parameters" with a state interpreted as a discrete position on an error surface created from the test data. Such an abstract view can still provide interesting insights into characteristics of the trained model.

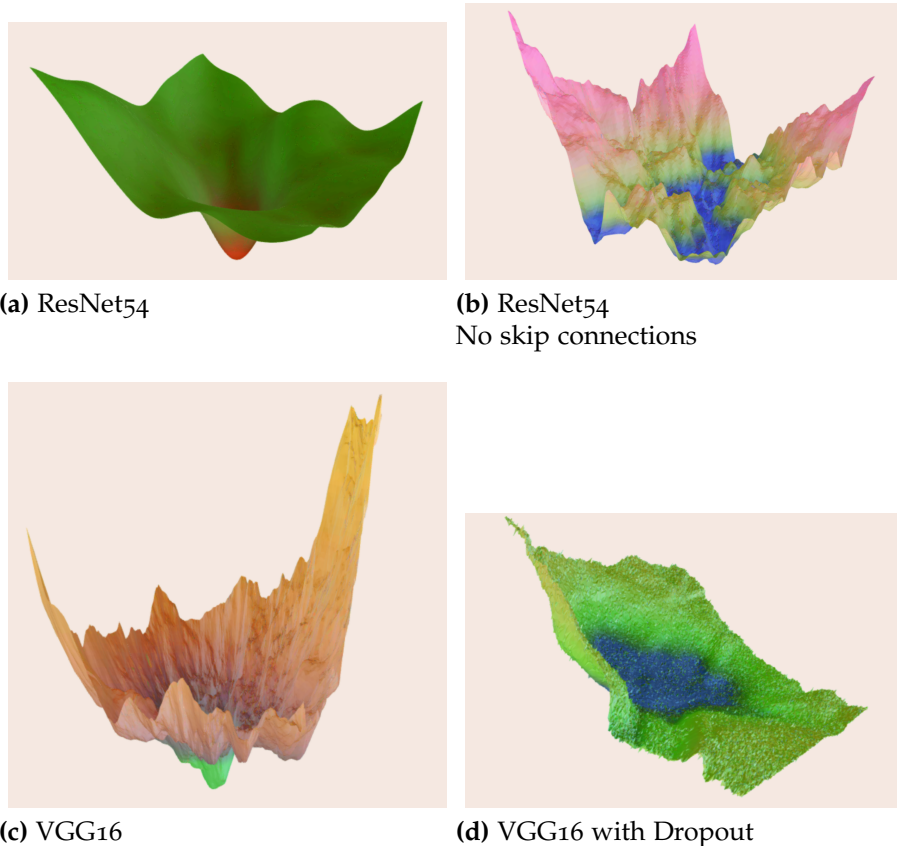


Figure 29: Rendering of the error surfaces from various models around the converged minimum can be used to gain insights into the effects of architectural properties. For instance, the surface-region of ResNet54 (a) is much rougher when the networks skip connections are disabled (b). The minimum of VGG16 (d) is also much sharper when no dropout for regularization is used (c) (sharp minima are hypothesized to generalize worse by authors like Keskar et al. (2017b)).

For example, in figure 29 we can see how certain architectural components like Dropout and Skip-Connections (see section 3.3.4) influence the error surface and thus the quality of the model. Other high-level approaches for analyzing the error surface proposed by Li et al. (2018c), Wang et al. (2018), Keskar et al. (2017b) and Keskar et al. (2017a). Keskar et al. (2017b) also demonstrate that the smoothness of converged optima play a significant role in the generalization, since they are hypothesized to be more resistant towards

perturbations compared to sharp optima. To estimate the smoothness of converged optima Novak et al. (2018) proposed a metric for estimating the local roughness of the error surface.

On the other end of the abstraction-spectrum information can be extracted in a very fine-grained and detailed manner. Class-activation maps are such a fine-grained technique, since they allow attributing activation of certain classes of the model to regions of individual input images (see figure 30 for an example). This effectively allows the practitioner to understand which features the model has learned to be discriminatory for certain classes and thus get a more profound understanding of the model quality. Various techniques to achieve this have been proposed over the years. The occlusion-based technique by Zeiler, Fergus (2014a) can be seen as one of the earliest successful attempts of this type of analysis tool. The Cam-Family class-activation-visualization techniques by Zhou et al. (2015), Selvaraju et al. (2017) and Chattopadhyay et al. (2018) can be considered more advanced back-propagation based variants of the same principle.

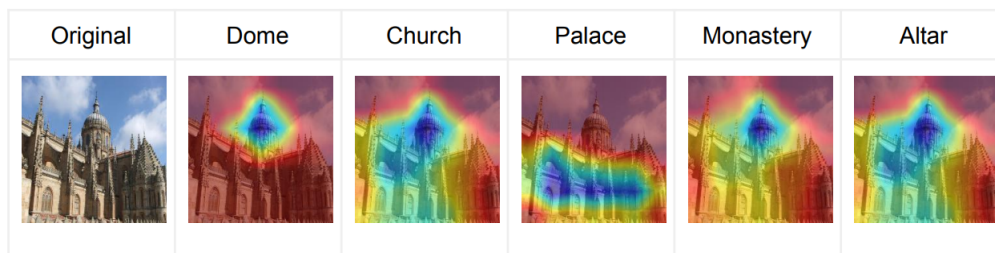


Figure 30: Class activation maps like the here depicted GradCAM++ algorithm by Chattopadhyay et al. (2018) provide insight into the inference process by highlighting the regions that activate specific classes.

Analysis techniques also exist on a spectrum between model centric and data-centric. The aforementioned heatmap-visualizations can be considered data-centric, since the model is reduced to a heatmap highlighting the significant regions on a specific image. Therefore, the level of detail in the analysis is high regarding the properties of the data and low regarding the properties of the model. A model centric counterpoint to this is activation-maximization, a technique proposed by Erhan et al. (2009). Activation maximization visualizes the features extracted by a specific filter of a feature map. This is achieved by maximizing the output of the unit producing the filter using an artificial input image generated by gradient ascent (see figure 31). While more abstract in its visualization than class activation maps, activation maximization is purely dependent on the trained model and allows the analysis of every individual unit of the model.

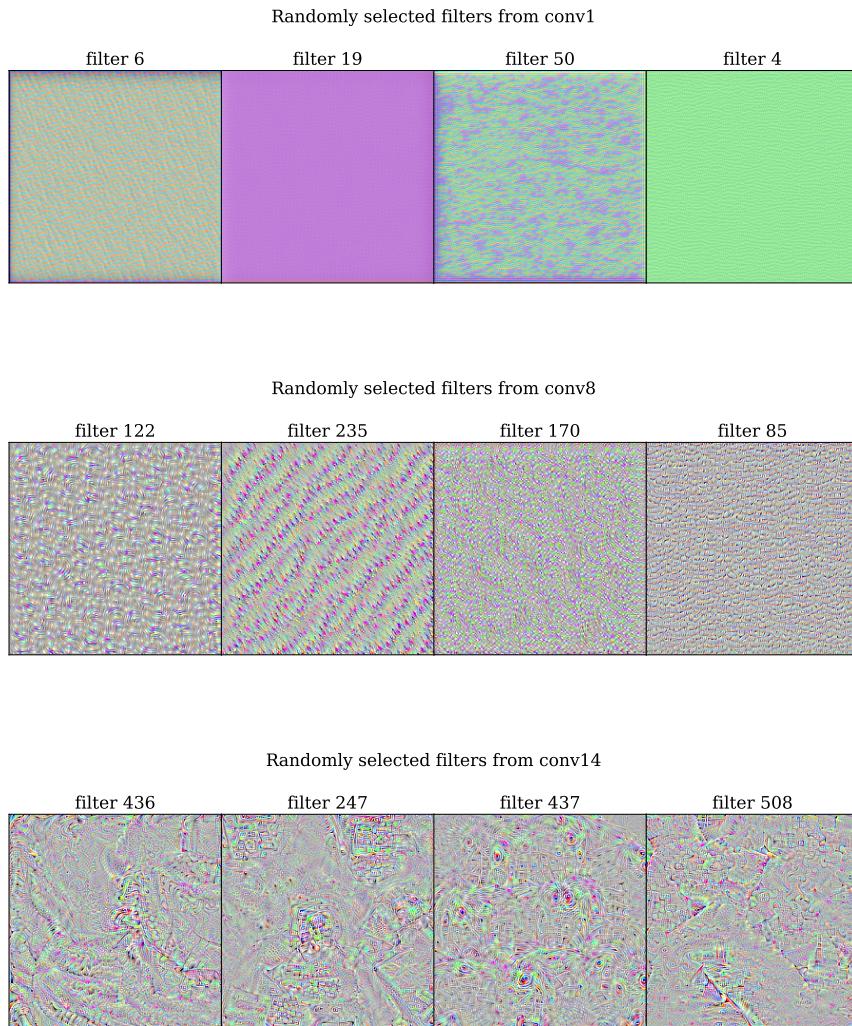


Figure 31: By maximizing the activation value of a specific filter-unit in a convolutional neural network, we can visualize patterns specific filters are very sensitive towards. Based on these visualizations from an ImageNet-trained VGG19, we can see how the extracted patterns go from structurally small and simple (a) to increasingly complex patterns (b, c) depending on the location of the layer in the network.

By doing so, it is possible to observe redundancies in the learned features. This was used by authors like Garg et al. (2020) and Chakraborty et al. (2019) to motivate and develop pruning techniques to reduce the number of redundant filters in neural architectures.

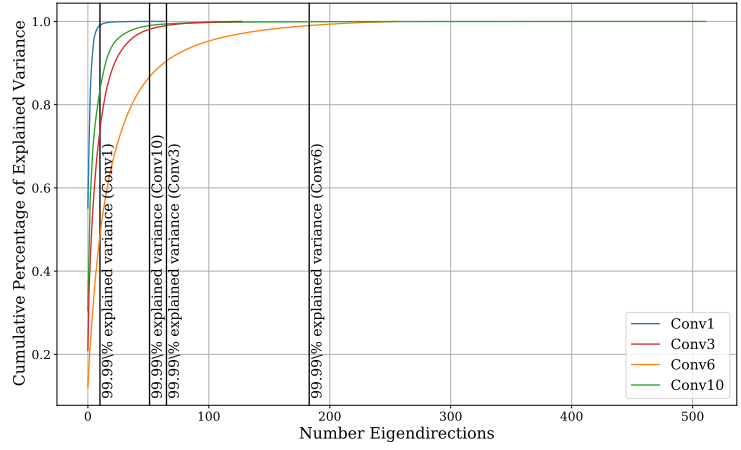
4.2 Feature Space Analysis

In this work, we are interested in analysis techniques that can be used to diagnose inefficiencies and other shortcomings in convolutional neural network architectures in a realistic application scenarios. Realistic application scenarios may involve visual domains and tasks that the model was not orig-

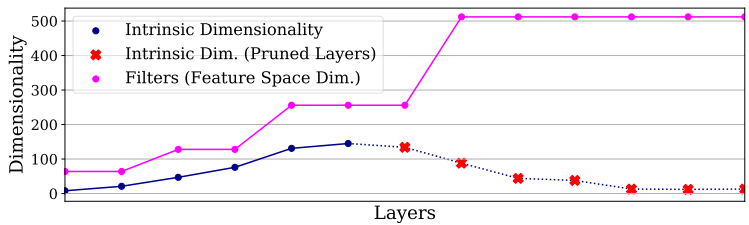
inally developed for. The mamography-based tumor-classifiers by Rakhlin et al. (2018), Wu et al. (2020) and Shen et al. (2021) exemplify that substantial changes to architectures are sometimes required to optimize predictive performance and efficiency if the visual domain and task diverges strongly from the task the model was originally developed for. Therefore, we want to avoid relying on strongly data or model-centric analysis techniques, since we suspect that these are unlikely to reflect the relationship of architecture and dataset in a way that allows the practitioner to make informed decisions on neural architecture design. Concerning the level of abstraction, we are interested in techniques that can reflect the inner processes of the model with sufficiently high-resolution to attribute inefficiencies on specific components in the structure of the neural network. Therefore, we decide to investigate the feature spaces of convolutional neural network layers. The feature space can be considered the vector space of a layer's output. By analyzing the feature spaces, we can analyze the network on a layer by layer basis. Layers of neural networks are often regarded as the atomic component of convolutional neural architecture design, we therefore assume this to be an ideal resolution regarding the analysis of the model structure for our purposes. Since we are analyzing the output space of neural network layers and therefore data in an intermediate state of processing (latent representation), techniques that analyze the feature space can provide insights about the neural architecture, the data and their interaction.

Thus, a significant part of this work involves the analysis of feature spaces in hidden layers and developing a layer-based metric for this purpose. However, this idea and the methods used in this work are not entirely new. For instance, Montavon et al. (2010) uses principal component analysis based on RBF-Kernels to analyze how the solutions evolve during training. By doing this, the authors find two important properties that will be expanded on in this paper. First, the data inside the feature space exists in small subspaces, and these subspaces change in dimensionality from layer to layer. Second, CNN build their solution bottom-up when trained on a task, which is different from MLPs that lack this inductive bias caused by the convolution operation. Similar observations were made by Garg et al. (2020) that were later expanded on by Chakraborty et al. (2019), which make use of this property to prune and fine tune convolutional neural network architectures.

Unnecessary layers are identified by a PCA-based heuristic. The number of eigendirections that span the sub-space in which the data resides are estimated. Layers that break the monotony of increasing number of eigendirec-



(a) The Intrinsic Dimensionality is computed by number of eigendirections required for explaining 99.99% of the total variance. The filters in each layer will be reduced to the intrinsic dimensionality.



(b) Layers that break the monotonic increase of intrinsic dimensionality are removed. The network is then retrained

Figure 32: Principal Component Analysis (PCA) on the outputs of neural network layers can be used for optimizing neural architectures for higher computational efficiency. The figures above illustrate the two steps of PCA-based pruning by Garg et al. (2020). Significant eigendirections of every layer’s output are computed up to a threshold of 99.99% explained variance (a), the obtained value will also be the number of filters of the layer in the optimized architecture. Layers that do not increase the dimensionality of the data from the previous layer are removed in the following pruning step (b). The reduced network is then retrained to obtain the more efficient model.

tions from layer by layer are removed. The pruned network is then retrained, resulting in increased efficiency by only minor losses in performance according to Garg et al. (2020).

An example of this can be seen in figure 32. This method is only applicable on sequential architectures without any kind of skip-connections. This method was later expanded by Chakraborty et al. (2019) to also be usable in embedded scenarios and networks with skip-connection. These techniques are very significant to our work, since also one of our primary analysis techniques (saturation), is heavily based on PCA and uses similar computational techniques for computing eigendirections. We also gain more insight on how

the observed patterns of eigendirections like in figure 32 (b) emerge.

Another noteworthy technique is SVCCA by Raghu et al. (2017). In contrast to previously mentioned techniques, SVCCA, using singular vector decomposition and canonical correlation to analyze the similarities in feature activation between different networks and different sets of data. By doing so, the authors find that early layers share more similarities than later layers, even if the neural architectures are very different.

4.3 Logistic Regression Probes

Analyzing a network layer by layer was also done by Alain, Bengio (2017) using "logistic regression probes". We elaborate on their work in greater detail, since we modify and heavily utilize logistic regression probes in this work. The idea behind logistic regression probes is that the output of any intermediate layer between input and output of the network can be viewed as an intermediate solution. We established in section 3.2 that the primary purpose of the convolutional layers in the feature extractor is to transform the data from layer to layer into a linear separable representation, so that the (linear) softmax layer can solve the classification problem by minimizing cross-entropy. This also means that the output of every intermediate layer should become increasingly linear separable if the layer actively contributes to the quality of the solution. To measure this, the authors train logistic regression using the output of a specific layer for each probe as input and using the labels of the task as ground truth. Since logistic regressions are linear (which makes their training a convex optimization problem) and the variants used by Alain, Bengio (2017) minimize the cross entropy loss, we can compare the predictive performance of the probes to the predictive performance of the model and other probes to visualize the evolution of the intermediate solutions during the forward pass.

In figure 33 we can see a visualization of this evolution. A VGG16 model is trained on the Cifar10 dataset, and probes are trained on the output of every layer. We observe that the validation accuracy of the probes is monotonically increasing from layer to layer, indicating that the qualitative inference process is distributed among the entire network.

As a proof of concept that probe performance can be used to detect certain pathological behaviors, the authors create a simple MNIST-experiment. An overparameterized 128-layer multilayer perceptron with 128 units per layer and ReLU activation-functions is trained on MNIST. The model has a sin-

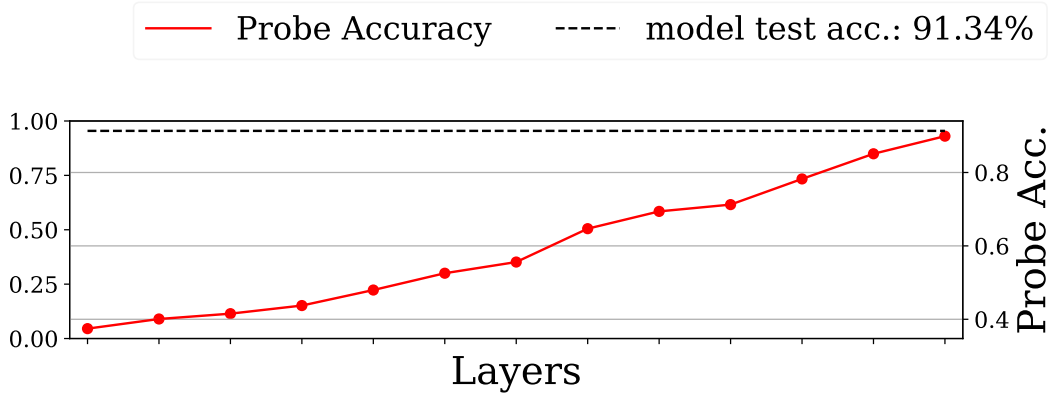
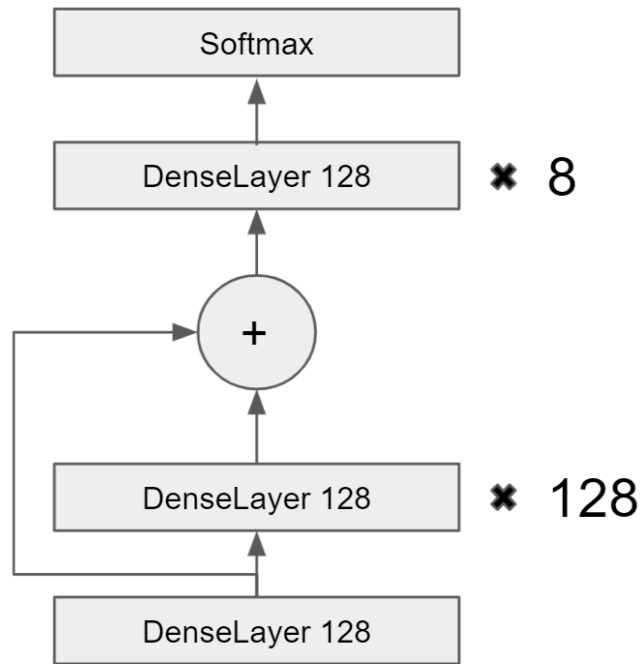
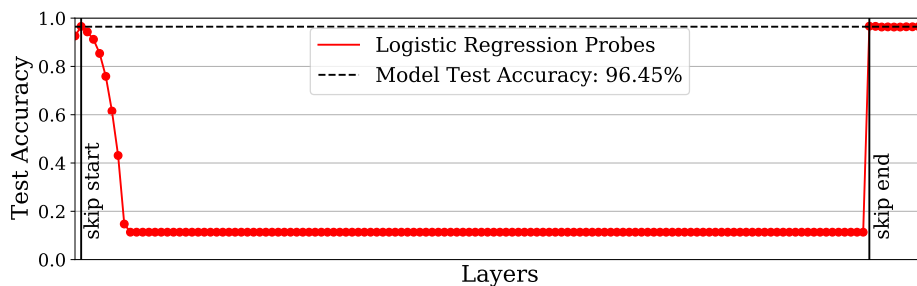


Figure 33: The probe performance at each layer of VGG16 trained on Cifar10. Logistic Regression Probes allow the practitioner to observe the evolution of the intermediate solution quality from layer to layer. The performance is increasing layer by layer, indicating that the problem is solved incrementally and that the inference process is evenly distributed among layers.

gle skip connection that connect the first and the 64th layer of the network, effectively allowing the signal to skip 50% of the network. The key idea behind this setup is to provoke an identity mapping on the layers encapsulated by the skip connection. This behavior was first hypothesized by He et al. (2016a). In theory, networks with skip-connections should be able to skip layers encapsulated by skip connections by producing an output o , effectively learning an identity mapping. While this exact behavior was not observed by the authors, they could still observe that skip connections are utilized by the model to skip unneeded layers. To illustrate, we reproduce this experiment using a slightly modified architecture that connects the first and the 129th layers instead with a sequence of 8 layers after the skip connection has been closed (see figure 34 (a)). This minor alteration makes the architecture easier to train due to a reduced vanishing gradient problem, while achieving similar results to the original experiment by Alain, Bengio (2017). In figure 34 (b) the probe performances in order of the forward pass are depicted. We can see that the section encapsulated by the skip connection is decaying in predictive performance until reaching chance-level. When the skip connection is added back, the predictive performance recovers instantaneously, indicating that the previous layers were "skipped". While the network did not learn an identity mapping, the network learned to ignore layers by making them produce a latent representation that can be added to the feature vector of the skip connections without obscuring or destroying information. This is important for our work, since we will observe the behavior on multiple occasions when



(a) The multi-layer perceptron architecture used for the reproduction of the experiment on skip connections by Alain, Bengio (2017). The architecture deviates slightly from the original by Alain, Bengio (2017) to make the model easier to train.



(b) The performance of the logistic regression probes in the order of the forward pass. Note the stark difference in behavior between the first part (encapsulated by the skip connection) and the second part of the network.

Figure 34: The images show the basic neural architecture with a skip connection (a) and the probe performances of each layer (b). The setup is designed to provoke the network to "skip" the 128 layers by learning some identity-mapping analog. By observing the probe performances, this behavior can be observed. After the initial layer, the performance degrades until reaching chance level. The probe performances recover as soon as the skip connection is added to the layers again. We will observe similar behavior on convolutional neural networks over the course of this work.

evaluating probes on ResNet-like models. This proof of concept is also interesting from a conceptual level, since a large part of this thesis is dedicated to finding pathological patterns that allow to detect inefficiencies in neural networks without the direct comparison to other models. The behavior of the

probe performances in figure 34 (b) indicate that pathological inefficiencies do exist under certain circumstances and can be detected with the right analysis technique. In fact, we observe two types of inefficiencies in the probes of this experiment. First, the skipping of the encapsulated layers discussed previously. Second, the final layers after the skip connection are not improving the probe performance and thus not enhance the quality of the intermediate solution. Thus, these layers can be considered unproductive as well, since their positive impact on the quality of the prediction is negligible.

4.4 The Receptive Field

The receptive field is the area on an image that influences the output of a convolution operation.

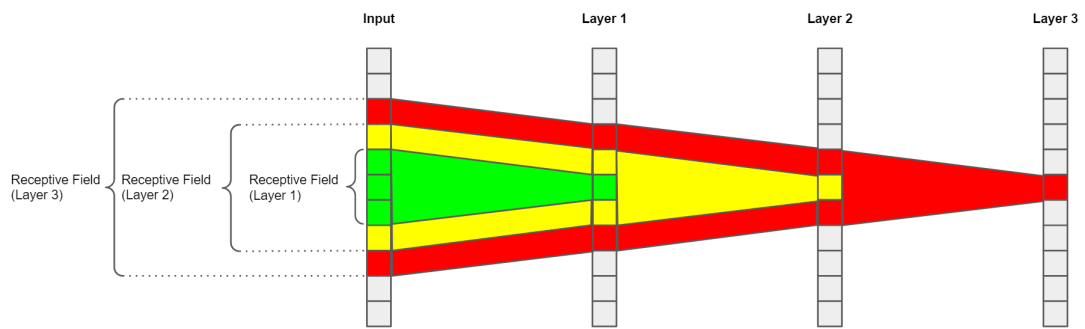


Figure 35: A 1-dimensional example of receptive field expansion in a convolutional neural network. Each layer has a kernel size of 3, resulting in an expansion of the receptive field from layer to layer.

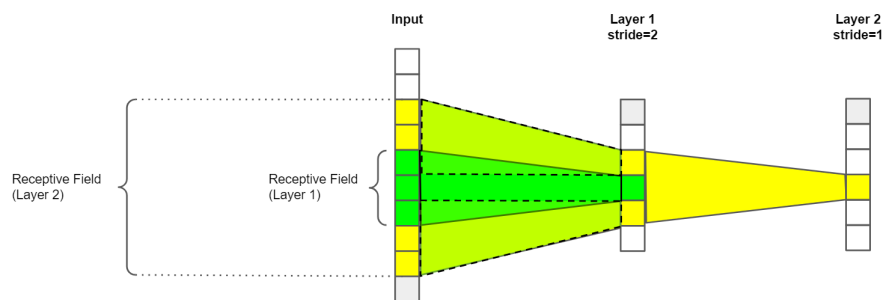


Figure 36: A 1-dimensional example of receptive field expansion with a convolutional down-sampling layer (Layer 1). The increased step size from kernel position to kernel position results in less mutual inputs between two adjacent feature map positions in the output of Layer 1, resulting in an accelerated growth of the receptive field for consecutive layers and a reduction in feature map size.

For this work, the size of the receptive field (described as a scalar⁷) is important, since it reflects a spatial upper bound of visual patterns detectable by the respective layer.

For sequential convolutional neural networks (no multiple pathways during the forward pass) the receptive field size can be computed analytically. We refer to the receptive field r of the l th layer of the sequential network structure as r_l (with $r_0 = 1$, which is the "receptive field" of the input). For all layers $l > 0$ in the convolutional part of a sequential network, the receptive field can be computed with the following formula:

$$r_l = r_{l-1} + ((k_l - 1) \prod_{i=0}^{l-1} s_i) \quad (9)$$

where r_{l-1} is the receptive field of the previous layer, k_l refers to the kernel size of the layer l (with potential dilation already accounted for) and s_i the stride size of the layer i . The receptive field increases with every convolutional layer with a stride or kernel size > 1 (see figure 35). Strides also have a multiplicative effect the growth rate of consecutive layers r_{l+n} , since the feature map is downsampled (see figures 36).

For networks with multiple pathways and skip-connections, it is not possible to precisely compute the receptive field r_l , since one layer may receive input from multiple layers with different receptive field sizes $r_{l-1,i}$ existing in the input of the layer l . However, in most cases we are not interested in the precise receptive field sizes present, rather we are interested in the receptive field size as an upper bound. In essence, we view the receptive field size r_l as the largest possible spatial extension of a feature that is still detectable by the respective layer. In this situation, we can simplify the receptive field computation by only considering the sequence of layers from the input to the layer l with the largest receptive field⁸:

$$r_{l-1} = \max(r_{l-1,0}, r_{l-1,1}, \dots, r_{l-1,n})$$

This allows us to compute the receptive field sizes for architectures like ResNet-models by simply ignoring the skip connections.

⁷Technically a 2-tuple, however since square kernels are the norm we can make this simplification.

⁸This computation also makes the assumption that the growth rate of the receptive field in all pathways is equal (essentially the same downsampling occur on each pathway). While it is possible to build an architecture where this is not the case, no classifier known to the authors actually has a property like this.

4.4.1 Minimum and Maximum Receptive Field

In our later experiments, we will see that the receptive field as an upper bound is insufficient for predicting unproductive layers. In this case, we have to view the information present in an image as being composed of multiple receptive field sizes in an interval $(r_{l,min}, r_{l,max})$, where $r_{l,min}$ is the smallest possible receptive field size present in the layer l and $r_{l,max}$ the largest.

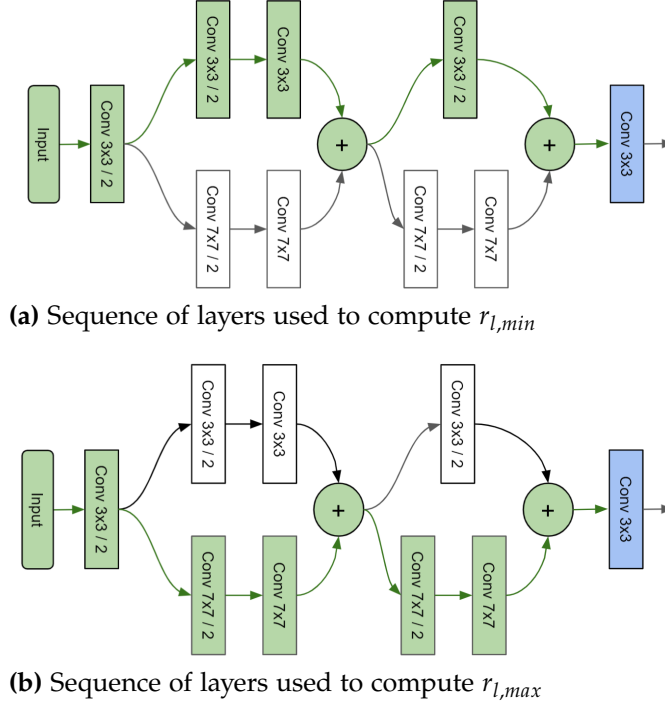


Figure 37: The information in a given layer (blue) in a non-sequential neural architecture is based on multiple receptive fields with sizes. These receptive field sizes are bound in an interval $(r_{l,min}, r_{l,max})$. It is possible to compute $r_{l,min}$ and $r_{l,max}$, by calculating the receptive field size of the sequences (green) with the smallest and largest receptive field size leading from the input to the layer l (blue). These figures were previously published in Richter et al. (2021b).

Simply speaking, every possible path from the input to the layer l has a receptive field size $r_{l,i}$. Since feed-forward neural networks are non-recursive, the number of paths in this kind of network is always finite. Therefore, in a general case, $r_{l,min}$ and $r_{l,max}$ can be obtained by simply computing the receptive field sizes $r_{l,i}$ of every possible path from the input to layer l .

The upper and lower bound can then simply be obtained by picking the minimum and maximum values of these paths respectively:

$$r_{l,min} = \min(r_{l,0}, r_{l,1}, \dots, r_{l,i})$$

$$r_{l,max} = \max(r_{l,0}, r_{l,1}, \dots, r_{l,i})$$

For a more visual example, we refer to a hypothetical multi-path architecture, which is depicted in figure 37. To compute the minimum and maximum receptive field of the final 3×3 -convolutional layer, we compute the receptive field of the sequence of layers highlighted in green. We basically "cut" all layers from the network, except for a single sequence of layers. Since we can compute the receptive field for a sequence of layers analytically, we can compute the receptive field for this particular path. By picking the path with the largest possible receptive field size (figure 37 (a)) and the smallest possible path (figure 37 (b)) we can obtain $r_{l,min}$ and $r_{l,max}$ respectively, with l being the layer highlighted in blue. Since the most popular CNN architectures follow a building block style pattern similar to the depicted hypothetical architecture, it is generally not necessary to compute the receptive field sizes of all paths.

These computations are still consistent with the computation of r_l discussed earlier. In a sequential architecture, always exactly one path exists leading from the input to a layer l . In this case $r_l = r_{l,min} = r_{l,max}$.

Networks with skip connections are a special case of the multi-path architectures, since they can be considered multi-path-architectures even when technically only containing a single sequence of convolutional layers like ResNet-style networks (He et al. (2016a)). The skip-connections of these architectures open (layer-less) paths between otherwise non-consecutive layers. These paths do not expand the receptive field. Therefore, the information reaching a layer after a skip-connection can originate from multiple pathways. Effectively, information based on lower receptive field sizes can "skip" layers, making the architecture non-sequential as a result. Thus, $r_{l,max} \geq r_{l,min}$ for the layers in architectures with skip-connections.

5 Problem Description

Before we move on to the empirical part of this work, we briefly elaborate on the goals we want to achieve through the following experiments. The main contributions can be summarized under two primary goals. First, developing methods that allow analyzing neural networks that can be used to guide neural architecture design. These methods need to be computationally lightweight to allow for quantitative studies on large sets of models and a variety of datasets. We elaborate further on the design goal for this analysis tool in section 5.1. The second goal is presented in section 5.2 of this work. This second objective is to use these aforementioned methods to build an understanding of architectural components and their influence on the inference process. This understanding should be able to reliably aid developers and scientists to diagnose and resolve inefficiencies in neural network architectures during development.

5.1 Developing a Practical Technique for Analyzing Convolutional Neural Networks

The first major part of this work focuses on developing a methodology for analyzing convolutional neural networks that is useable by practitioners and researchers alike. Techniques like SVCCA by Raghu et al. (2017) and logistic regression probes by Alain, Bengio (2017) have demonstrated that interesting insights can be gained from analyzing the output of intermediate layers. However, these and many other techniques like the metrics for error surface estimation by Keskar et al. (2017b) and Novak et al. (2018) are computationally very demanding. This is particularly problematic since the techniques are generally more CPU-intensive tasks, for they do not involve retraining the model, resulting in different hardware setups being optimal for model training and analysis. An expensive analysis after training also at least partially negates the benefits of analyzing the model and making informed decisions, since trial-and-error can be more practical in such circumstances. A practical tool for analysis should therefore allow gaining insights quickly, ideally life during the model training, so that decisions can be made as early as possible and with little computational overhead.

Another important property of a practically feasible method for analyzing the properties of trained models are the type of insights gained from the analysis. Methods like GradCAM by Selvaraju et al. (2017) allow to fairly quickly

visualize the inference decision by highlighting the responsible regions on a heatmap, but knowledge about the individual decision made by the classifier does not easily trace back directly to potential improvements or solutions to resolve a diagnosed problem.

In short, a practical methodology should fulfill the following requirements:

- The analysis method should be quick to compute with little overhead (ideally life during training) as an automatized process.
- The results should be easy to interpret and visualize by themselves, not requiring comparative evaluation and complex post-processing.
- The analysis should provide insights that allow the diagnosis of inefficiencies and performance bottlenecks within the neural architecture.
- The results of the analysis should enable the user to resolve those inefficiencies in an informed manner without requiring trial and error.

5.2 Diagnosing and Resolving Architectural Inefficiencies in Convolutional Neural Networks

In order for the methodology described in the previous section to be useful, experimental evidence regarding inefficiencies and performance bottlenecks needs to be gathered systematically and evaluated. Furthermore, it is necessary to acquire an understanding on how properties of the architecture like the input resolution, number of layers, the use of multiple pathways and skip connections (to name only a few) affect how the model processes information. Empirical studies of this kind were done on smaller scales by multiple authors. Li et al. (2018c) for instance showed the effects of DenseBlocks and skip-connections on the error surface. In other cases like the works of Howard et al. (2019), Tan, Le (2019) and Woo et al. (2018) smaller studies on the presented architecture innovations are conducted. However, these experiments on the impact of an architectural component act primarily as demonstrators to show the functionality of the proposed architecture or analysis technique. There is currently no publication known to the author that attempts to empirically analyze and characterize the properties of convolutional neural architectures, such that design decisions with a high chance of success can be deducted. In short our goals are:

- Acquire an understanding for the relation of neuro-architectures and information processing during the forward pass.

- Derive reliable diagnostics for inefficiencies in convolutional neural networks.
- Propose reliable solution strategies for resolving the inefficiencies found in the neuro-architectures.

6 Open-Source Software and Technical Foundation

In this section, we will briefly discuss the technical foundation and basic methodology that was used to conduct all experiments of this work. This thesis is based on two open-source frameworks, that were developed alongside this project. We will first present Delve, an experiment control system that allows fast and easy analysis and logging during model training. We will briefly introduce the basic functionality, summarize the feature set of Delve, and provide a link to the repository.

The second open-source framework is PHD-Lab, a framework for training and evaluating deep neural networks. PHD-Lab is primarily used to automate the experiments and basic analysis as well as execute them reproducibly, hardware independent and crash resistant. We will introduce the software architecture of PHD-Lab, elaborate on the basic functionality and the logging behavior. We will further discuss how experiments are logged, reproducibly and resistant towards hardware and software-based crashes.

6.1 Delve: Framework for Experiment Control

Delve contains most of the basic analysis tools developed, presented and used in this work. The module also contains an experiment control system that allows logging and persisting metrics and more general information gathered during an experiment. The basic software-architecture of Delve and the initially releases were developed as part of the Master Thesis by Shenk (2018). Since then, the architecture and functionality of Delve have been heavily modified. The feature set presented in this work was developed as part of the works of Shenk et al. (2019) Richter et al. (2021c), Richter et al. (2021a) and Richter et al. (2021b). The module is compatible with Python 3.6 and higher. It is currently published on PyPi ⁹ and is open sourced under <https://github.com/delve-team/delve>. The system is compatible with PyTorch releases 0.8.0 and later. The open-source library is also separately published in Shenk et al. (2021).

6.1.1 Overview

Delve has three primary features:

1. Computing layer-specific information based on the covariance matrix of the layer's output.

⁹<https://pypi.org/project/delve/>

2. Basic logging functionality for persisting the aforementioned information as well as additional externally computed information regarding the experiment like metrics, training time etc.
3. Automated basic analysis to enable life insights during training.

The software architecture is depicted in figure 38. All the aforementioned features are bundled in a monolithic object called the "Tracker", which functions as a logger after initialization. It is the only instantiated object of the Delve-framework that the rest of the program interacts with during the experiment. Delve considers an experiment, a program that conducts at least a single training or inference step using a single neural architecture.

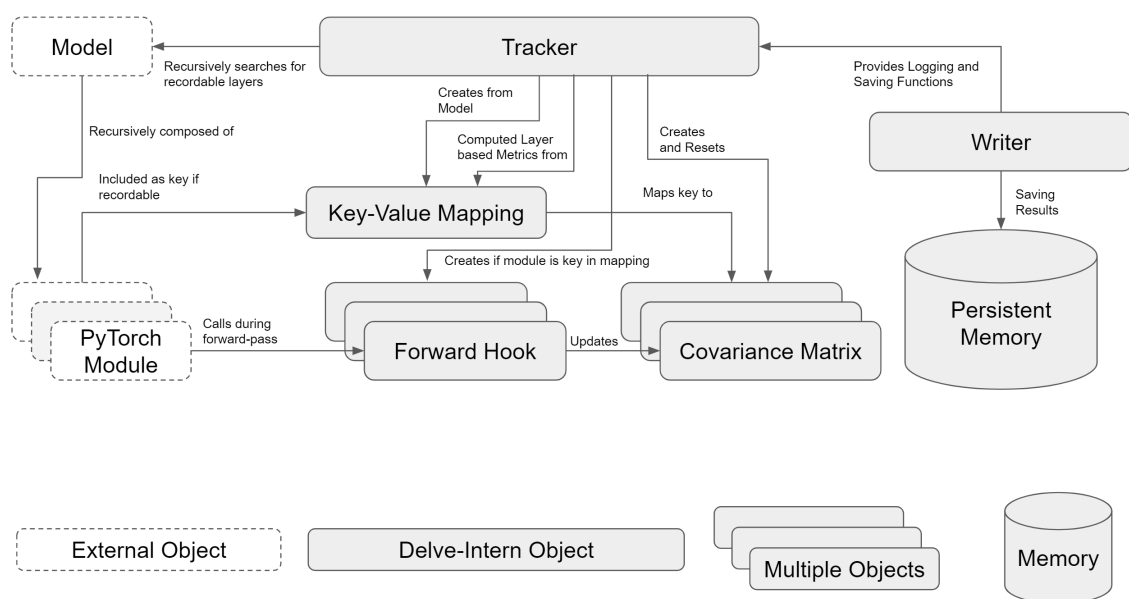


Figure 38: The illustration depicts the general software architecture of Delve. From a user perspective, Delve is represented by an instance of a monolithic Tracker-object. This object can be used for calculating layer-based information about the PyTorch-model as well as logging experimental data. The strategy for recording and persisting the information is internally handled by an implementation of the Writer-Interface. Covariance-Approximation and extracting other information directly from the PyTorch-model is done using the forward-hook interface of PyTorch and is organized in a Key-Value mapping.

Internally, the Tracker-Object fulfills three different purposes. First, finding and registering layers that can be recorded. For these layers, the covariance matrix will be approximated using the forward-hook interface of PyTorch. A key-value Hash-Map is used for the registration process, mapping an automatically generated and human-readable name of the layer to the approximated covariance matrix. Second, managing covariance matrices and updating them as soon as novel data passes through the layers, this involves also preprocessing the data if the type of the layer requires it. Third, general

logging, analysis and persistence, summarized as experiment control. This is handled by the Writer-object, which acts as an abstract interface to the Tracker to allow variability regarding the format and type of logged information as well as the exact nature of the persistence strategy (databases, hard-drive folders with files, TensorBoard etc.).

6.1.2 Experiment Control and Integration in the Training Loop

We will briefly discuss the intended usage of Delve and how it is meant to be integrated into the training process. The flow-diagram illustrating this can be seen in figure 39.

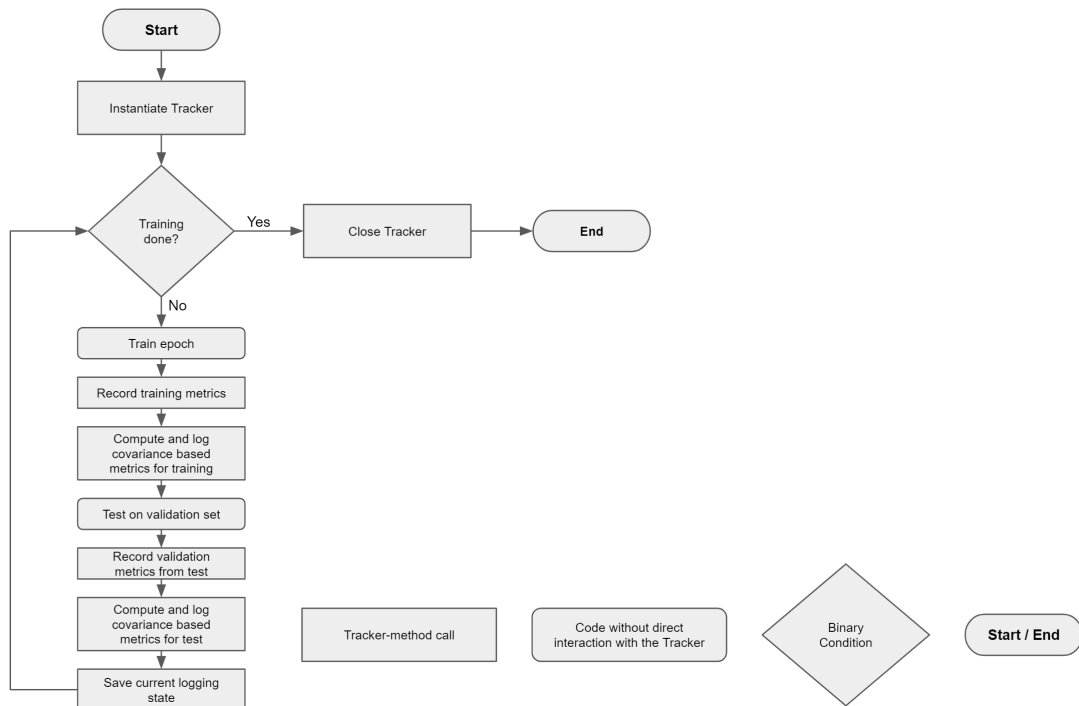


Figure 39: Typical program loop demonstrating the intended interaction between a model training and Delve experimental logging. The interaction between Delve and the training of the model is kept minimal to allow for easy integration and usability.

First, the Tracker is instantiated. This will configure the logging process. By doing this, the model-to-be-trained is registered and hooks are attached to recordable layers. Whether a layer is recordable or not is defined by an internal whitelist. As of the writing of this thesis, every convolutional layer, LSTM-cell and densely connected layers are defined as being recordable. This list is restrictive because non-parameterized layers like pooling and dropout-layers are generally not adding interesting information to the metrics computed from the covariance matrices. Therefore, to reduce the number of com-

puted covariance matrices and the overall overhead, the list is restricted to the most common parameterized layers in neural networks.

During the training, the hooks attached to the layers are recording the data and update the covariance matrices automatically. This process is explained in detail in section 6.1.3 and is fully automatic, requiring no interaction with the Tracker-instance.

After the training epoch has concluded, the metrics computed during training (for example accuracy, training time in seconds etc.) are logged. The covariance-matrices are used to compute the layer-wise metrics like saturation (see section 8.1). These metrics are automatically logged, and the covariance matrices are automatically reset to allow recomputing the covariance matrices on the validation set. The entire process is repeated on the validation set, concluded by another reset of the covariance matrices anticipating the next epoch of training. Finally, all logged information is saved and the cycle repeats for the next epoch until training has concluded. At this stage, automatic analysis is conducted by the Writer-Instance as part of the saving process. After training has concluded, the Tracker-Instance is closed. This is necessary to signal the Writer-object to close all potentially open connections and file pointers, as well as doing other cleanup operations.

6.1.3 Covariance Approximation

One of the key features of Delve is the capability to compute information on layers based on the covariance matrices of the output of the respective layer. To achieve this, the covariance matrix of the output of every layer is computed during training and validation. However, keeping the entire output during training and validation in memory for every recordable layer is only feasible for small, low-resolution datasets like MNIST and small neural architectures. To process datasets of arbitrary size during training and testing, an algorithm for approximating the covariance matrix is required that has constant memory usage and can be updated batch wise, whenever data is passed through the network.

Let A_l be the output of layer l generated of the entire training or validation set. We assume that A_l is a matrix with shape $(n \times \#features)$. Each row resembles one of the n data points in the dataset. Every column refers to the output of a neuron (in case of an LSTM or dense-layer) or a filter (in case of a convolutional layer). To compute the covariance matrix $Q(A_l, A_l)$, we need to compute the covariance of every combination of feature dimensions. We treat each column of A_l as a random variable and each row as an observa-

tion. To compute the covariance $\sigma_{X,Y}$ between two random variables X and Y iteratively with constant memory usage, the following formula is utilized:

$$\sigma_{X,Y} = \frac{\sum_{i=1}^n x_i y_i}{n} - \frac{(\sum_{i=1}^n x_i)(\sum_{i=1}^n y_i)}{n^2} \quad (10)$$

We make this computation more efficient by exploiting the shape of the layer’s output matrix A_l : We assume that the matrix A_l is subdivided into batches of data points, which we refer to as $A_{l,b}$ where $b \in \{0, \dots, B - 1\}$ for B batches. We can compute $\sum_{i=1}^n x_i y_i$ for all feature combinations simultaneously in the layer l by calculating the running sum of squares $\sum_{b=0}^B A_{l,b}^T A_{l,b}$ of the batch output matrices, $A_{l,b}$. We can do the same for $\frac{(\sum_{i=1}^n x_i)(\sum_{i=1}^n y_i)}{n^2}$ by computing the outer product $\bar{A}_l \otimes \bar{A}_l$ of the sample mean \bar{A}_l . The sample mean \bar{A}_l is a vector with a dimensionality equal to the column-rank of A_l and can be considered the column-wise mean of A_l . This value is computed iteratively by aggregating the running sum of all outputs $a_{l,k}$, where $k \in \{0, \dots, n\}$, at training time and divide it by the total number of training samples n . The final formula for covariance approximation is then:

$$Q(A_l, A_l) = \frac{\sum_{b=0}^B A_{l,b}^T A_{l,b}}{n} - (\bar{A}_l \otimes \bar{A}_l) \quad (11)$$

Since we only store the running sum of squares, the running sum and the number of observed samples, we do not need to keep A_l in memory. Instead, the current approximation of $Q(A_l, A_l)$ is updated with every batch that is processed by the layer, while the aforementioned variables require a constant amount of memory. The algorithm requires roughly the same number of computations as the processing of a forward pass of the respective layer does. Our algorithm uses a thread-safe 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. The algorithm is therefore compatible with model and data parallelism (Huang et al. (2019)).

In convolutional layers, we treat every kernel position as an individual data point. 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). The advantage of this strategy is that no information is lost, while keeping Q at a manageable size. This strategy was proposed independently by Raghu et al. (2017), after their initial publication of SVCCA, and Garg et al. (2020), who use it in their PCA-based pruning strategy for CNNs. The entire process operates purely in double precision, regardless of the floating-point precision the

model is trained on. The primary reason for this is to avoid numeric instabilities caused by rounding errors, which are more likely on 32-bit floating-point operations (standard for convolutional neural network training). This algorithm was previously published in Richter et al. (2021c).

Further Optimizations: To further boost the performance of the covariance approximation additional, technical optimization steps were undertaken. First, Delve conducts the covariance approximation in torch, by default, on the same compute device as the model. This allows for a highly efficient execution of tensor operations using CUDA and avoids unnecessary memcopy-operations between nodes, devices, torch and the python interpreter, since the model is part of the same dynamic compute graph on the same compute device as the covariance matrices. Since this may result in unwanted memory limitations on the respective compute device, the compute device may be changed by the user.

Other optional optimization steps involve downsampling the feature maps to a lower resolution using different downsampling strategies like linear, bi-linear and nearest interpolation, as well as limiting the maximum number of batches that are recorded or recording only every n th batch for covariance approximation. All of these optional optimization steps are implemented with the practical applications on large networks and datasets in mind. While these techniques were tested, we omit the use of the potentially biasing techniques for the experiments in this work, to eliminate the possibility of errors induced by these techniques.

6.1.4 Supported Layer-Based Information

Delve can extract various information from the covariance metrics. The following information can be automatically extracted from the recordable layers using Delve:

- saturation for a threshold δ (see section 8.1)
- intrinsic dimensionality (eigendirections required to explain a percentage δ of the variance)
- trace of the covariance matrix
- trace of the diagonal matrix of the covariance matrix
- determinant of the covariance matrix

- the embedded data (the data is projected onto the 2 largest eigendirections)
- the covariance matrix

From the list, it is apparent that it is also possible to extract information that does not qualify as a layer-based metric. With the term "metric" we describe a single scalar value that carries well-defined information about the evaluated subject (in this case a layer of a neural network). This additional information (covariance matrix and embedded data) are treated as second-class-citizen in terms of the software architecture. This means that, for instance, an implementation of the Writer-Interface does not have to guarantee compatibility with non-scalar information extracted from a layer. This means that they should be looked at as "freebies" or debugging tools rather than part of the core functionality of Delve.

6.1.5 Automated Analysis, Logging and Saving

As we previously showed in figure 38, the logging, saving and the automated analysis is handled by an implementation of the Writer-Interface. Therefore, the Writer-instance in the setup contains the only part of Delve that is allowed to have side effects. Logging functionality is directly passed from the Tracker to the Writer object without further processing. The logged information is cached in memory by the Writer until the saving-function is called explicitly on the Tracker-Instance. Automated analysis is considered part of the saving-procedure and is always executed when the current state of the experiment is persisted. Automatic analysis involves primarily the visualization of the aforementioned results. An example can be seen in figure 40.

The system can also handle multiple Writers at the same time. In this case, the individual Writer-instances are automatically wrapped into a single CompositeWriter-Instance, that redundantly uses all Writer-Instances in sequence whenever called. This allows for a better modularization and separation of different saving and analysis functionalities.

6.1.6 PCA-Layers

PCA-Layers are a special kind of PyTorch-layers that can be used for some experiments regarding the latent space and are based around the covariance approximation algorithm described in section 6.1.3. These layers are not part

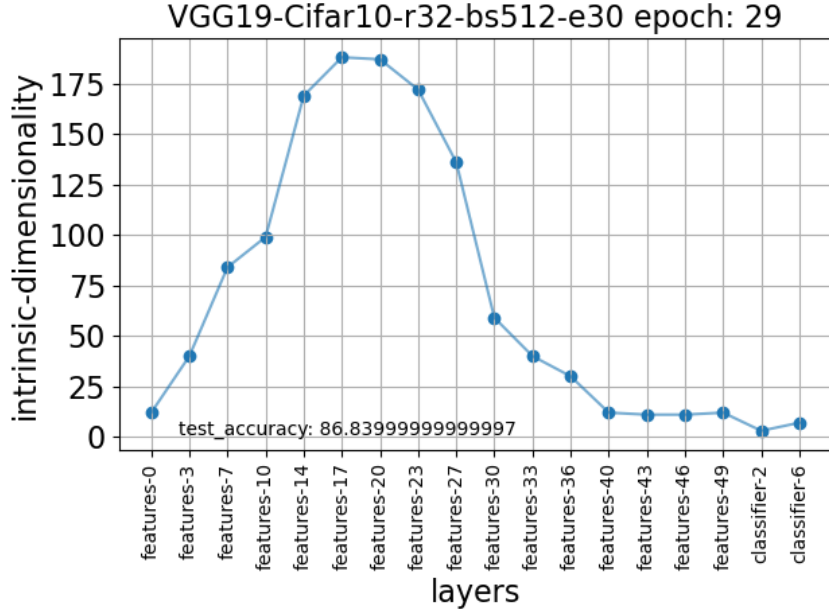


Figure 40: Delve can automatically generate plots for quick life analysis during training. The depicted example shows the intrinsic dimensionality of VGG19 after 29 epochs of training on Cifar10 using a batch size of 512 and an input resolution of 32×32 pixels.

of the core functionality of Delve depicted in figure 38, they are merely a utility tool for conducting experiments that require restricting the data to the subspace of the feature space. Delve provides a total of two PCA-Layers, a densely connected and a convolutional PCA-Layer. Both layers produce output with the same shape as the input. During training time, PCA-Layers behave like pass-through layers and do not manipulate the data passed to them. However, while training, they use the covariance approximation algorithm described in section 6.1.3 to compute a covariance matrix. When the model is switched into evaluation mode, the PCA layers compute a projection matrix P_δ into a subspace that explains the variance of the input data up to a threshold δ using the largest eigendirections. The densely connected PCA Layer projects all input vectors x_{in} into this eigenspace to obtain the output x_{out} :

$$x_{out} = x_{in}(P_\delta^T P_\delta) \quad (12)$$

The matrix multiplication of P_δ with its transposed version is necessary to keep the shape of the output intact while restricting the data to the low dimensional subspace of P_δ . The convolutional version of the PCA-Layers convolves this operation as a 1×1 kernel over the feature map, since every position of the feature map is treated as an individual observation for com-

puting the covariance matrix. The covariance matrix is reset every time when the model is switched into training mode, avoiding biases in the covariance matrix due to strongly changing latent representations of the data over the course of an epoch of training.

6.2 PHD-Lab: Experiment Automation Framework

The experiments conducted on this thesis require large quantities of trained models using various configurations of hyperparameters. To conduct and reproduce the experiments as well as manage the data and results produced by these, a framework was implemented. All experiments conducted in this work were conducted with the PHD-Lab environment. The framework is open sourced under <https://github.com/MLRichter/phd-lab>. This repository furthermore contains configuration-files for all experiments in this work, which allows the reproduction of results for any experiment conducted in the following chapters.

6.2.1 Overview

The main design goal of PHD-Lab is to build a setup that allows conducting experiments in a reproducible manner with a high level of automation and recoverability in case of a crash with minimal loss of time and information.¹⁰

To achieve this, the following core functionalities are required:

- Automatized assembly of experiment setups (see section 6.2.2).
- Automatized execution of experiments, consisting of model training and analysis (see 6.2.2).
- Standardized way of configuring arbitrary experiments without changes to the source code (see section 6.2.3).
- Automated, structured logging and information extraction during and after training (see section 6.2.4).
- Automatized training of logistic regression probes (see section 6.2.5).

¹⁰Within PHD-Lab context we consider an experiment an arbitrary amount of training procedures on arbitrary architectures, datasets and hyperparameter configurations with common postprocessing and analysis steps as well as optional training of logistic regression probes.

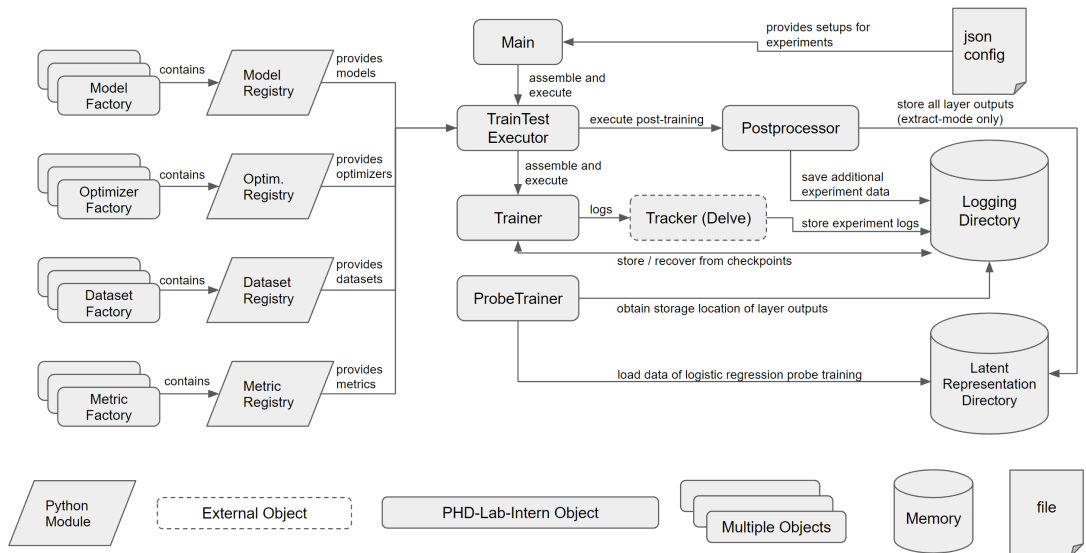


Figure 41: PHD-Lab is configured over a simple json-file, which specifies parameter configurations and major components, i.e. the dataset, model, optimizer and metrics. These components are provided by the respective registries. For practical reasons the training of the model and the training of logistic regression probes is disentangled in the architecture to allow for easier distributed computing.

To achieve these goals, a highly modular setup is required that allows for easy assembly in a way that is similar to dependency injection during runtime. The software architecture can be seen in figure 41. The core executable is always an instance of the callable Main-class. The primary purpose of the Main-class is to ingest a configuration file, decode it and then assemble a single TrainTestExecutor per model training. The TrainTestExecutor conducts training and post-processing on a single training setup. Training is conducted with a Trainer object, which executes the model training as well as information extraction and analysis during training using Delve (see section 6.1 for details). The Postprocessor on the other hand implements additional model specific information, like computing the FLOPs required for training, computing the receptive field size of every layer or extracting the outputs of every layer (latent representations) and saving them for later logistic regression probe training. Computing and logging logistic regression probe performances can be intentionally detached from this main path of execution and is executed after the model training. The main reason for this is the different requirements of logistic regression probe training and convolutional neural network training. Logistic regression probes require large quantities of CPU-Memory and CPU-Cores, while training a model requires GPU-resources. Having matching hardware for both training types is vital, since some experiments would otherwise require weeks to conclude in scenar-

ios where models have many layers (for example ResNet50) and the dataset is large (for example ILSVRC12). Being able to detach probe training and the rest of the experiment allows us to run logistic regression probes and model training distributed on separate computation nodes that are better optimized for the specific tasks.

6.2.2 Automated Assembly and Execution of Experiment-Setups

A big advantage of this experimental setup is that certain dependencies are known to likely change in between experiments, while others are likely to be static. This allows us to build a simple register-based architecture, where the TrainTestExecutor simply fetches the required model, dataset, optimizer and metrics from factory-functions that are in the scope of a python-module that acts as a registry. This allows for quick and easy expansion of the framework with new models, datasets, optimizer and metrics without changing core components of the application. This is important for making experiments reproducible, since the business-logic that conducts the experiments, handles logging, analysis and persistence is factually frozen after it was initially implemented and tested. Since this business logic is also setting seeds for random number generators and handles the standardized setup of other random or potentially biasing parts of the training. Ruling these random components out as a potential source of deviance between experiments is vital for guaranteeing that results are reliable and reproducible.

6.2.3 Configuration

Another advantage of this register-based architecture is that the experimental setup can be changed without requiring changes to the source code. This is important, since the exact state of the code repository when the experiment was conducted is not necessarily known. This can make experimental results harder to reproduce. To avoid this and allow quick and easy repetition of experiments, we decided to use a json-file-based configuration. A single experiment is thus represented as a single json-file containing a dictionary of key-value pairs.

This configuration can be semantically separated into an experimental, technical and evaluation part (see code box 1). The experiment configuration defines the basic components of the training setup, and it's corresponding hyperparameters like the neural architecture, the datasets, the number of trained epochs and input resolution. The evaluation configuration sets the primary

metrics computed on the training and validation dataset. The technical configuration on the other hand contains parameters concerning the business logic, like the location of the logging-directory for saving results or the compute-device used for training. It is also possible to pass lists of different values. In this case, a single training and post-processing will be conducted for every value in this list. If more than one configuration key-value pair is a list, every possible combination of these values will be used for a model training and post-processing. The configuration-file shown in code box 1 in particular contains two model trainings on Cifar10 per model. One is conducted on VGG16, the other one on ResNet18. Both are trained for 120 epochs using a batch size of 64, an input resolution of 32×32 pixels and the ADAM optimizer (Kingma, Ba (2014)). The evaluation configuration contains the metric used for evaluation during training and inference. The technical configuration is responsible for the components of the setup that are not part of the experiment and not influencing the results directly. Such things are (in the same order as in the config file) the directory where results are logged, the device where Delve stores the covariance matrices and finally the bool-flag to distribute the training of the convolutional neural network on all available compute devices by using data parallelism.

There are also additional (optional) technical parameters that can be added to a configuration file like this. However, since this thesis is not primarily about the software architecture, and we want to avoid confusing the reader with implementation details we deliberately decide to omit these details and advice to study the readme-file of the PHD-Lab repository instead.

6.2.4 Logging

Standardizing and structuring logging is crucial in managing numerous experiments. To achieve this, we utilize Delve to implement a file-system based saving strategy. Logging in PHD-Lab is directly handled by the components that produce the respective data. This means that the Trainer instance is responsible for logging all training related data, while the Postprocessor also saves the information it extracted.

The target folder in which all results are stored is sorted and organized based in model, dataset, resolution and a user-specified run-id (this ID allows redoing the same experiment multiple times). An exemplary folder structure is depicted in figure 42.

```

1  {
2      // experiment configuration
3      "model": ["vgg16", "resnet18"],
4      "epoch": 120,
5      "batch_size": 64,
6      "dataset": "Cifar10",
7      "resolution": 32,
8      "optimizer": "adam",
9
10     // evaluation configuration
11     "metrics": ["Accuracy", "Top5Accuracy"],
12
13     // technical configuration
14     "logs_dir": "./logs/",
15     "device": 'cuda:0',
16     "data_parallel": True,
17 }

```

Listing 1: An example configuration-file, demonstrating the way PHD-Lab is set up. This file trains VGG16 and ResNet18 on Cifar10. The training is conducted for 120 epochs, using a batch-size of 64 images. The images have a resolution of 32×32 pixels. Optimization is done using the Adam-Optimizer by Kingma, Ba (2014). Accuracy and top5-accuracy are used for evaluation. The results are stored in a local folder called "logs". The training and analysis is conducted using cuda:0 (the first GPU) as primary compute device. Data parallelism is furthermore enabled, which means that the training will be distributed on all available GPUs.



Figure 42: An example for the automatically generated folder structure used for saving experiment results in PHD-Lab. The logging structure is nested and the naming of files copied regularly are named based on the configuration to avoid confusion of experimental results.

All data relevant to an individual training is saved in a single folder. This includes multiple tables of tabular data that contain such things as the performance of logistic regression probes, receptive field sizes of every layer, predictive performance and saturation-values of the individual layers. Files unique to this particular run are named after critical parameters of this run, to avoid confusion of different files, if these are ever extracted from the folder

structure for analysis.

The folder also contains information that is not strictly part of the results, but is required for reproducing a specific experiment. For instance, model checkpoints and other files that allow for a swift recovery after a system crash. We will talk about the crash-recovery in greater detail in section 6.2.6.

6.2.5 Logistic Regression Probes

As previously mentioned, the training and evaluation of logistic regression probes is separated from the main program flow for performance reasons. During the post-processing of a trained model, the latent representations of all recordable layers are extracted and stored on the hard drive alongside the matching labels of the ground truth. In the logs of the respective model, a file is created which acts as a pointer leading to the folder containing the previously extracted latent representations. The system assumes that each file in the folder contains the latent representation of a layer of the model and that the filename is corresponding to the name of the layer. When training of logistic regression probes is started, each file in the latent representation folder is loaded and a logistic regression is trained on its content for 100 epochs. The SAGA optimizer by Defazio et al. (2014) is used for training, enabling fast training while limiting the memory consumption. The probes are then evaluated on the extracted latent representations of the evaluation data. The accuracy scores of training and evaluation datasets is recorded in a csv-file in the log-folder, the model itself is discarded. By default, this part of PHD-Lab uses multiprocessing to train and evaluate multiple models in parallel, speeding up computations substantially. Optionally, the training can be conducted single threaded as well for debugging purposes.

6.2.6 Recovery and Reproducibility

Being able to reproduce and recover the results is critical, since the experiments are very resource intensive and may take multiple days or even weeks to conclude. For reproducing an individual experiment, PHD-Lab automatically creates a json-file in the log-folder of every individual experiment (see figure 42) before training starts. This file serves as a config-file and allows to exactly recreate the singular experiment, that was saved in this folder. By doing so, it is possible to recreate every analysis and model training solely from the logs, even when the original configuration-file was lost, altered or contained additional experiments that do not need to be reproduced.

For conducting numerous experiments, a sound recovery strategy is needed to avoid the potential loss of longer periods of processing time. For this reason, neural network and logistic regression probe training are implemented in a way that allows for quick recovery in case of a crash. As mentioned in section 6.2.4 besides experimental results also model checkpoints are saved. The recovery procedure is executed whenever a new Trainer is instantiated. If the Trainer finds preexisting results, it checks the saved experimental results to see how far the training has proceeded. If a training has been concluded previously, the training will not be executed and the experiment immediately jumps to the post-processing. If the training was interrupted, the model will recover with the state of the last concluded epoch, which is in most scenarios a loss of time of less than 20 Minutes. Since the Delve-Tracker saves all information at once, the possibility of a potential corrupted state or loss of information is limited to a crash occurring in the final phases of saving the model, when the file for storing the model-state is created but left incomplete. This dangerous phase usually takes less than 300ms to execute and is thus only a relatively small part of each epoch of training, which can take from 1 minute to 3 hours to complete, depending on the setup. Theoretically, this can result in the loss of the entire training progress for this particular model. However, this scenario is very unlikely due to the relative quickness of the saving-procedure and never occurred unintentionally during the experiments described in this work. Of equal importance is the recovery of the logistic regression probe training. This is especially true for large models like ResNet50, since training of logistic regression probes may exceed the training time of the model itself, depending on the hardware and dataset. For this reason, we apply a distributed caching logic that can recognize whether a logistic regression has been already trained on a specific combination of layer and dataset, in which case the cached result is loaded from memory. This is also the reason why the program-components used for logistic regression probe training are composed only of pure functions, which are by definition stateless and thus allow for a more reliable caching. This contrasts with the rest of PHD-Lab, which is coded in an object-oriented style.

7 Analyzing Neural Network Layers using Principal Component Analysis

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, Fergus (2014b), Zhang et al. (2017) and Yosinski et al. (2014) are examples of experimental work that show the boundaries and limits of generalization and transferability of features. Recent works by Raghu et al. (2017) and Alain, Bengio (2017) propose techniques that allow for a more profound 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 and in quantitative studies (Alain, Bengio (2017); Raghu et al. (2017); Zhou et al. (2016)).

This section shows, that a simple on-line computable property, like the covariance matrix of the layer outputs, can 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 that we can use 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*. The experiments presented in this section were published separately in Richter et al. (2021c) and Richter et al. (2022).

7.1 General Concept and Methodology of the Experiments

The experiments conducted in this section all involve PCA being applied on the output of neural network layers, thus a lot of technical details in the experimental setups are shared. To avoid bloating the following experiment descriptions with repetitive technical details that are necessary for reproducing the experiments but not critical for understanding the experiments, we decide to bundle this information here.

The core idea behind these experiments is to approximate the subspace of each layer's output space that is relevant for further information processing using PCA. We can define a good approximation of these subspaces of every layer as a set of subspaces that have a minimal impact on the predictive per-

formance of the model if the data is restricted to these approximations. In this case, the projection has not negatively impacted the predictive quality of the model, which means no critical information was lost due to the projections during the forward pass. However, PCA itself is not a parameter-free operation, since the dimensionality of the eigenspaces need to be determined. Since the number of filters and units in each layer and therefore the dimensionality of its output space varies greatly in neural architectures, setting this value to a fixed number is not feasible. Instead, we decide to use an explained variance threshold δ as a regulator to determine the number of dimensions in the eigenspace. In effect, we accumulate the k eigendirections with the highest variance, which explain combined a percentage δ of the total variance of the layer’s output. Since lower values of δ result in lower dimensional eigenspaces, we expect a decreasing predictive performance as δ decreases. A main purpose of the following experiments will be to understand the effects of different values for δ on the quality of the approximation.

All experiments conducted in this section are done in the PHD-Lab experiment environment described in section 6.2. To approximate the relevant eigenspaces, PCA needs to be applied on the output of any given layer without changing the shape of the output. This is done using the PCA-Layers presented in section 6.1.6 and the covariance approximation algorithm discussed in section 6.1.3. Whenever PCA-Layers are used in the experiments, they are placed after every weighted layer in the network. The exact type of PCA-Layer depends on the type of layer. Layers that produce feature maps as output will be followed by a convolutional PCA-Layer, and layers that produce an output vector for a given data-point will be followed by a linear PCA-Layer instead. The number of eigendirections k used for the PCA-projection is computed for each layer and resembles the number of eigendirections that explain $\delta\%$ of the variance. The percentage of explained variance δ for all PCA-Layers is set globally for the entire network. We refer to a network that is modified in such a way as a *projected network*, since all layers effectively operate on data that is projected into their reduced eigenspace E_l^k in all layers during inference time. If not mentioned explicitly otherwise, the training of any model is conducted over 30 epochs using a batch size of 64 and the ADAM optimizer by Kingma, Ba (2014). No further hyperparameter optimization was undertaken, since it is not the goal of the experiments to achieve state-of-the-art performance. Instead, we choose a training setup that resembles a standard configuration found in frameworks like PyTorch, Keras and Tensorflow (Abadi et al. (2015); Chollet (2015); Paszke et al. (2019)). The intention behind this is to bring the

tested scenario closer to an early development scenario in a real-world application. The data is augmented during inference time by channel-wise normalization using means and standard deviations computed from the ImageNet-training set. During training the data is randomly horizontally flipped with a probability of 50% as well as randomly cropped.

7.2 Demonstrating That Projected Networks can Maintain Some Predictive Quality.

The manifold hypothesis suggests that data in a high dimensional space occupies a lower dimensional subspace within this higher dimensional space. The output space of a neural network layer can be considered such a high-dimensional space. The dimensionality of a layer’s output is determined for this purpose by the number of filters ¹¹ (convolutional layer) or the number of output units (fully connected layer). We refer to this space as *feature space*, for it effectively consists of information about the presence and absence of learned features in the data. If the manifold hypothesis is applicable for neural network layers, it should be possible to find a subspace in the feature space, that contains the latent representations of the data. When the data is projected into this subspace during the forward pass of the model, the information critical for the quality of the prediction should be maintained, since the data remains unchanged by the projection.

This experiment is a proof of concept, testing whether it is possible to approximate this subspace, which we refer to as the *latent space*. Since we can batch-wise approximate covariance matrices based on the training data during the forward pass (see section 6.1.3) we decide to use PCA as a candidate method for the approximation of the latent space. PCA was initially proposed by Jolliffe (1986) and is a popular technique for dimension reduction. PCA is a linear projection of the data into a subspace spanned by k eigenvectors of the data’s covariance matrix. However, since convolutional neural networks perform non-linear transformations in each layer, we cannot say exactly how much information will be lost by linearly projecting the data inside a feature space of a layer.

In this section, we test if PCA can maintain at least some predictive quality

¹¹Since the processing of every feature map position on an image is independent of any other position on the feature map, we can consider each feature map position an individual data point in terms of information processing. This simplification is not unprecedented and was previously made by other authors that study the output of neural network layers such as Raghu et al. (2017) and Garg et al. (2020).

of our model, when the output of every layer in the model is projected into its respective eigenspace. In this experiment, we are not (yet) interested in quantifiable results (a more quantitative study on classifiers will be conducted in the next section). Instead, we want to obtain an intuitive understanding of what the effects are, when each layer is projected into their approximated latent space E_l^k . For this reason, we will conduct the experiments on a convolutional autoencoder. The reconstruction of input images by the autoencoder allows for an intuitive analysis of the ablation in prediction quality caused by the projection. We also directly observe the dimensionality of $E_{encoding}^k$, which is the dimensionality of the reduced eigenspace in the encoding layer, the lowest dimensional layer in the model. By doing so, we can obtain a grasp on the upper bound of the size of the latent space, relative to the surrounding feature space in the given setup. If the manifold hypothesis is applicable in this scenario, we expect $E_{encoding}^k$ to be substantially lower dimensional than the feature space, when no visible differences in the reconstruction loss occurs.

7.2.1 Methodology

To test whether it is possible to approximate the latent space using PCA, we train a neural network and project the output of each layer l into a k -dimensional, PCA-based approximation of the latent space E_l^k . E_l^k is composed of the k highest variance eigendirections of the data. In combination, these eigendirections explain a percentage δ of the data's variance. The technical details of this process are described in section 6.1.6. We refer to these networks as *projected networks*, since the latent representations are projected into our approximation of the latent space. As a proof of concept, we want to see whether a projected network can maintain some predictive quality for any amount of explained variance $\delta < 100\%$. Furthermore, we are interested in visualizing the ablative effect that will likely occur for decreasing values of δ . To get a basic idea of the ablative effects, we train a simple convolutional autoencoder on the Food101 dataset on a reconstruction loss. Reconstructing an input image is a simple task that allows for an easy visualization and interpretation of the ablative effect caused by the projections. The convolutional autoencoder architecture is depicted in table 1. A convolutional autoencoder is a special type of neuro-architecture consisting of two components. The first part is referred to as an "encoder". Its structure similar to a convolutional neural network feature extractor in the sense that it is fully convolutional. It also has a pyramidal shape like a feature extractor described in section 3.2. The flattened output of the encoder is referred to as "encoding" or "code".

This encoding is strongly reduced in dimensionality compared to the input, forcing the autoencoder to compress the information contained in the input image, which can be considered the main purpose of the encoder. The second part of the model is referred to as "decoder" and resembles a mirrored version of the structure of the encoder. Its purpose is to "decode" the low-dimensional encoded image back to the original input image. The so-called reconstruction loss is the mean-squared-error of the pixel wise distance between the input image and the prediction.

Table 1: Convolutional Autoencoder Architecture.

Encoder	Decoder
$224 \times 224 \times 3$ Input	(3×3) conv, 8 ReLU
(3×3) conv, 16 filters, ReLU	upsampling, nearest, scale-factor 2
(2×2) max pooling, strides 2	(3×3) conv, 8 filters, ReLU
(3×3) conv, 8 filters, ReLU	upsampling, nearest, scale-factor 2
(2×2) max pooling, strides 2	(3×3) conv, 16 filters, ReLU
(3×3) conv, 8 filters, ReLU	upsampling, nearest, scale-factor 2
(2×2) max pooling, strides 2	(3×3) conv, 3 filters, ReLU

Since this is a proof of concept, we are interested in using a dataset that is challenging for the model to solve while not being too resource intensive. The Food101 dataset by Bossard et al. (2014) features real world, high-resolution imagery of 101 different food items (75.750 train, 25.2500 test images), making it a sufficiently heterogeneous and high dimensional dataset, while still being manageable in terms of size and required computational resources for training.

Table 2: Hyperparameters for the convolutional autoencoder.

Parameter	Values
Input Resolution	(224×224)
Epoch	50
Batch size	128
Optimizer	ADAM
ADAM: beta1	0.9
ADAM: beta2	0.999
ADAM: epsilon	$1e-8$
ADAM: learning rate	0.0001




The remaining setup of the training can be read from table 2. After the training has concluded, we evaluate the reconstruction loss on the test set on for different values of δ . This is necessary, since δ is a hyperparameter of the

projection, and thus influences the quality of the latent space approximation by influencing the dimensionality of E_l^k . A lower δ -value will result in a lower dimensional E_l^k , which in turn will likely have a negative effect on the prediction, i.e. the reconstruction of the input image. We test the following values for δ : 90%, 95%, 99.5%, 99.9% as well as 100%, the latter being the unprojected network acting as a baseline reference. In preparation for this experiment, we also tested lower values of δ . However, $E_{encoding}^k$ is already 1-dimensional for $\delta = 90\%$. For this reason, we decide to omit these additional results, since they do not add any additional insights.

7.2.2 Results

From the aggregated results in table 3 we can make multiple interesting observations. We can see from the example that the reconstruction image is still recognizable for $\delta \geq 95\%$, demonstrating that for these values of δ the inference process still produces recognizable output. While the loss of $\delta = 99.9\%$ is still almost double the loss of the unprojected network, the model clearly did not collapse as a result of the projections and perceived loss of detail on the image is minimal. Also interesting is that the dimensionality of this subspace is only 597 dimensional, which is 7.29% of the dimensionality of the feature space. Even at $\delta = 99.9\%$ explained variance $E_{encoding}^k$ is still only 4374 dimensional, which is 53.34% of the feature space dimensionality. Apparently, relatively few (compared to the feature space dimensionality) high-variance eigendirections are responsible for a large part of the qualitative inference process - even in the bottleneck of an autoencoder. Based on these results, we proceed to investigate PCA as a candidate method for latent space approximation further.

Table 3: When projecting all layers l of an autoencoder into their reduced eigenspaces E_l^k , the reconstructed examples are still recognizable, even when only a fraction of the dimensionality of the feature space is used in every layer. It is also worth noting that an eigenspace with only 53.3% of the feature space dimensionality is needed in the bottleneck of the autoencoder, where the feature space has the lowest dimensionality, to explain 99.9% of its variance. The images depicted here can also be found in Richter et al. (2021c).

δ	$\dim E_{encoding}^k$	loss	Reconstruction Example
Input Image	-	-	
100%	8192	0.033	
99.9%	4374	0.065	
99.5%	1332	0.089	
99%	597	0.120	
95%	17	0.216	
90%	1	0.234	

7.3 Investigating the Information Content of Projected Networks

The previous experiment demonstrated that an autoencoder can still reconstruct recognizable images, while the processing of the network is restricted to an PCA-based approximation of the latent space E_l^k in each layer. However, we do not know how good the E_l^k -projections conserve the performance relative to other projections of the same dimensionality. Therefore, the conservation of predictive quality observed in the previous section could still be a product of chance. Neither do we know whether the observed behavior is transferable to cross-entropy minimizing classifiers, which this work focuses on primarily. In the following experiment, we aim to answer these two questions in a quantifiable manner. We do this by training networks of various depth (number of layers) and width (number of filters) on a dataset. We observe the ablation in predictive performance of the projected networks for various values of δ and compare this to the performance-ablation of projected networks that utilize a k -dimensional randomly chosen orthonormal subspace instead of E_l^k . With ablation, we refer to the relative loss in performance of a projected network compared to the unprojected model. If the PCA-projections are reliably maintaining information critical for the inference process, the performance ablation relative to the unprojected model will be smaller for the E_l^k -projected network than for the projected networks using random orthonormal basis of the same sizes.

7.3.1 Methodology

To make the experiments more quantifiable, we train in this experiment a total of 20 architectures 3 times, to obtain a larger size of samples. The architectures are based on the VGG-network family by Simonyan, Zisserman (2015), since they can be considered very conventional from a design perspective without significant design quirks that could bias the results. The VGG family consists of 4 architectures in total with 11, 13, 16 and 19 layers (named VGG11, VGG13, VGG16 and VGG19 respectively). We use all of these architectures to include variations in network depth. We also train a total of 4 additional variants of each of the aforementioned architectures. These variants have their "width" (number of filters per layer) reduced by a factor of $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$ and $\frac{1}{16}$, to also account for varying degrees of overparamterization in each layer. By doing so, we cover the two-major axis that are commonly used for scaling architectures, the width and depth of a network (He et al.

(2016a); Huang et al. (2017); Real et al. (2019); Tan, Le (2019); Zagoruyko, Komodakis (2016)). The models are trained on Cifar10 using the setup described in section 7.1.

The trained models are evaluated a total of 41 times each. For reference, the model is evaluated once with disabled PCA layers. Twenty times, with enabled projections into a reduced eigenspace E_l^k in every layer for various values of δ . Based on previous experiments, we test $\delta \in \{90\%, 91\%, \dots, 99\%\}$ as well as $\delta \in \{99.1\%, 99.2\%, \dots, 99.9\%\}$. The first set of values for δ is used to observe the ablation of the model performance until chance-level¹² is reached, which can be considered the catastrophic collapse of the inference process. The second set of values is for a closer observation on the ablation, close to 100% explained variance. The latter is necessary to limit the search space to find the relevant eigenspace in later experiments. All models are evaluated one more time for each value of δ . However, instead of projecting the output of every layer into the respective version of E_l^k , the models are instead projected into subspaces spanned by k randomly chosen orthonormal vectors. We do this to see how well the reduced eigenspace E_l^k can maintain the predictive performance compared to a random orthonormal basis of the same dimensionality. If E_l^k reliably contains information critical for the inference process, the observed ablation of predictive performance when reducing δ will be more severe for the random orthonormal basis than for E_l^k .

The predictive performance for this experiment is measured in relative accuracy. The relative accuracy is computed by dividing the accuracy of the projected model by the accuracy of the unprojected model of the same architecture. This is done because of the high variance in accuracy between the different neuro-architectures. This makes the effects observed in this experiment harder to visualize in absolute terms.

Since we are interested in the ablation of the predictive performance as a function of δ we normalize the accuracy value by computing the accuracy relative to the accuracy of the unprojected model.

Since this experiment requires large quantities of model trainings and evaluations, we decide to use the Cifar10 dataset, which can be trained and evaluated fast due to its small native resolution of 32×32 pixels. See section 2.2.2 for a more detailed description of the Cifar10 dataset.

¹²With "chance level" we refer to a model with an accuracy equal to a model that makes random predictions based on a uniform probability distribution over all classes. Since this model can be considered the least informed decision maker, the performance of a chance level predictor can be considered the worst possible performance realistically achievable by a model. On Cifar10, the random predictor will achieve $\approx 10\%$ accuracy

7.3.2 Results

From the evaluation results, visualized in figure 43, we can see a stronger ablation, when the data is projected on the k -dimensional random orthonormal basis instead of E_l^k in every layer. The performance stabilizes at chance level at $\delta = 96\%$, which is significantly earlier than models with layers projected into E_l^k , where even at $\delta = 90\%$ some models still outperformed a chance level predictor. The degradation when projecting the networks layers in the eigenspaces is also slower, decaying over the entire range of tested δ values, while the orthonormal basis decay very rapidly for $\delta > 99\%$.

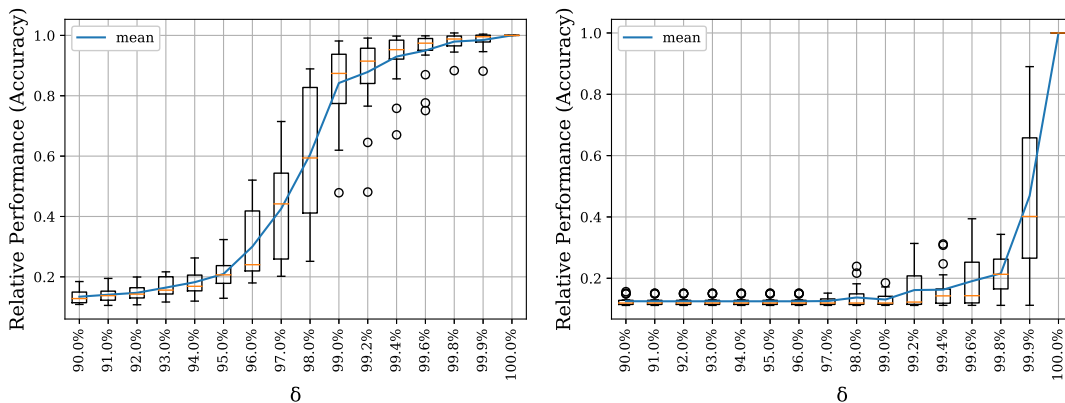


Figure 43: Random orthonormal projections (right) with same dimensionality as E_l^k degrade the performance significantly faster than projections into E_l^k (left). This image was first published in Richter et al. (2021c).

From these observations, we can conclude that projecting the output of each layer l into the reduced eigenspace E_l^k can maintain the original performance significantly better than projections into k dimensional random orthonormal subspaces of the feature space. This means that we can view E_l^k as an approximation of the "active subspace" of l 's feature space, where the information critical for inference is processed.

7.4 Finding Relevant Eigenspaces

So far, we demonstrated that projections of every output layer into its reduced eigenspace E_l^k can maintain parts of the original predictive performance, depending on the explained variance percentage threshold δ . We now attempt to find *relevant eigenspaces* for various convolutional neural network architectures. The *relevant eigenspace* is a reduced Eigenspace of a convolutional neural network layer l 's output with $k < \dim(Z_l)$, where k is the dimensionality of E_l^k and $\dim(Z_l)$ is the dimensionality of the feature space.

Additionally, when the network is projected into the relevant eigenspaces at every layer, the test error is statistically indistinguishable from the original network. We refer to this eigenspace as relevant, since the predictive performance and by extension the behavior of the model is not significantly influenced by the projections, the relevant eigenspace thus contains all *relevant* information for the qualitative inference process. If such a space can be reliably found for multiple different architectures, we have demonstrated that the processing in a convolutional neural network is utilizing only a portion of the available feature space dimensions.

7.4.1 Methodology

In section 7.1 we describe that the dimensionality of the reduced eigenspaces is controlled by a value $\delta \in (0\%, 100\%)$, which acts as a percentage threshold for the explained variance. Finding a relevant eigenspace can therefore be reduced to finding a value for δ , such that performance of the model is statistically indistinguishable from the unprojected network. Thereby, we can guarantee that the data is processed exclusively in our approximation of the eigenspace and that this projection is not affecting the predictive performance. We train the models using the same setup for the previous experiment. We evaluate the projected networks with δ values in the range from 99% to 99.99% in steps of 0.0001.

We train each architecture using the same configuration multiple times to obtain multiple samples and conduct the aforementioned evaluations on every instance (the sample size n is provided in the caption of the respective result table). We will provide the results of ResNet18 in this section. This model was chosen to demonstrate that projecting networks is still possible, when the architecture is more complex and features modern architecture components like stems (see section 3.3.2) and skip connections (see section 3.3.4).

To test whether the predictive performance of the projected network is indistinguishable from the unprojected network, we use student's two-tailed t-test to check whether the unprojected accuracies and the projected accuracies from the same network are drawn from the same distribution. The test is repeated for every value of δ . We also provide additional information, like the average difference in performance between the unprojected and the projected network μ_{diff} as well as the standard deviation in predictive performance over the projected networks σ_{sample} . We also compute the average total number of eigendirections used in a single network \pm the standard deviation of the same value.

7.4.2 Results

Based on the results in table 5 we can see that we indeed were able to find a relevant eigenspace at $\delta = 0.9998$. Even more interesting is that from 3904 dimensions in all feature spaces of the network, only 2153 ± 67 are used, which is only $55.15\% \pm 1.7\%$ of the total eigendirections. We interpret this as a confirmation of the manifold hypothesis in deep convolutional neural networks. This experiment also demonstrates that it is possible to approximate the latent space inside the feature space by using PCA. The results are also consistent with the lottery ticket hypothesis, which postulates that only small subnetworks are responsible for the quality of the inference process (Frankle, Carbin (2019)).

However, judging from the overall results in table 5 it is apparent that the evaluation is not flawless. The basic assumption of the experimental setup is that performance is either unaffected or degrading for any value $\delta < 100\%$. In some situations, however, we observe small, consistent improvements in the predictive performance. We were able to reproduce this phenomenon on VGG13 (see table 4). This indicates that PCA-projections may have a very slight denoising effect on the signal. However, since the effect size is so small, this is unlikely to be useful in practice.

Table 4: VGG13 t -statistic, $\mu \neq 0$, selected δ at $\alpha = 0.01$ ($n=26$). $\mu \neq 0$ in *italics*. Note that projections for some values of δ improve performance.

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

7.5 Conclusion

By applying PCA-projections to the data while passing through the network, we were able to demonstrate that processing of the data is occurring in low dimensional subspaces of each layer. We demonstrated that we can project data into approximations of this space with minimal performance losses, depending on the percentage total of variance δ this subspace explains.

Table 5: Sum of projections in ResNet18 (n=15). $\mu_{diff} \neq 0$ (p=.95) in **bold**. μ_{diff} refers to the average difference in performance between the projected network at threshold value δ and the unprojected network. We also provide the standard-deviation of the accuracy σ_{sample} over all samples. $\mu(\sum dim E_l^k)$ refers to the average number of eigendirections in the entire network \pm the $\sigma(\sum dim E_l^k)$.

δ	μ_{diff}	σ_{sample}	t-stat	p-value	$\mu(\sum dim E_l^k)$
1.0	0.0000	0.0000	nan	nan	3904 \pm 0
0.9999	-0.0002	0.0012	-0.52	0.611	2338 \pm 87
0.9998	0.0000	0.0013	0.0796	0.983	2153 \pm 74
0.9997	-0.0002	0.0016	-0.521	0.610	2043 \pm 67
0.9996	-0.0009	0.0020	-1.66	0.119	1963 \pm 63
0.9995	-0.0005	0.0022	-0.813	0.430	1900 \pm 61
0.9994	-0.0006	0.0019	-1.18	0.256	1847 \pm 57
0.9993	-0.0007	0.0019	-1.48	0.162	1802 \pm 55
0.9992	-0.0007	0.0022	-1.29	0.217	1763 \pm 54
0.9991	-0.0006	0.0022	-1.13	0.279	1728 \pm 52
0.998	0.0031	0.0046	2.63	0.020	1493 \pm 40
0.996	0.0213	0.0285	2.9	0.012	1294 \pm 32
0.994	0.0389	0.0454	3.32	0.005	1181 \pm 28
0.992	0.0579	0.0596	3.76	0.002	1100 \pm 27
0.99	0.0812	0.0782	4.02	0.001	1038 \pm 26
0.98	0.1899	0.1042	7.06	0.000	841 \pm 26
0.97	0.2918	0.1057	10.7	0.000	731 \pm 26
0.96	0.3649	0.0834	16.9	0.000	654 \pm 25
0.95	0.4333	0.0757	22.2	0.000	595 \pm 24
0.94	0.4544	0.0667	26.4	0.000	548 \pm 24
0.93	0.4787	0.0668	27.7	0.000	508 \pm 24
0.92	0.4896	0.0638	29.7	0.000	475 \pm 23
0.91	0.5119	0.0582	34	0.000	446 \pm 22
0.9	0.5296	0.0574	35.8	0.000	421 \pm 21

We were further able to show that we can approximate relevant eigenspaces using PCA, which do not alter the predictive performance of the networks significantly, when the data is projected into them in every layer. Based on these findings, we have shown that eigenspaces of latent representations inside a convolutional neural network bear information about the processing inside the network. In the following chapters, we investigate the sequence of these eigenspaces closer to derive an analysis tool for understanding the forward pass of convolutional neural networks.

8 Understanding the Behavior of Layer-Based Analysis Tools

The experiments of Tan, Le (2019), Zhang et al. (2017), and Frankle, Carbin (2019) show that overparameterization is to some degree necessary for neural networks to generalize well but eventually leads to diminishing returns. Therefore, finding a "sweet spot" of overparameterization could allow for the development of a model with good efficiency. In chapter 7 we demonstrate that the feature space of a convolutional neural network layer is relatively oversized compared to the subspace the data actually occupies in common architectures by approximating the latent space using PCA. The dimensionality of the feature space in a layer is strongly related to the number of parameters of the respective layer. Based on this relation, we hypothesize that our proposed approximation of the latent space can be used to better understand the overparameterization in the network by gaining a more profound understanding of the processing during the forward pass. We refer to the processing of data during the forward pass as an inference process.

For analyzing the inference process, we first propose a metric, *saturation*, which is based on the PCA-based latent space approximation described in chapter 7. In the experiments of this section, we demonstrate the reliability of the saturation (section 8.2) and logistic regression probes (section 8.3). We further investigate how saturation responds to changing degrees of overparameterization (section 8.4) and problem difficulty (section 8.5) and whether the observations agree with the hypothesis that saturation is indicative of the overparameterization level. Finally, we investigate the patterns of saturation-levels and logistic regression probe performances to investigate how the inference process is distributed in the neural network and how this is reflected in saturation. The experiments in this section were separately published in Richter et al. (2021c) and Richter et al. (2022).

8.1 Saturation

When looking at the dimensionality of all approximated latent spaces (see table 5 in the previous chapter 7) inside a convolutional neural network, it is apparent that their combined dimensionality is much lower than the sum of feature space dimensions. However, we did not yet study how the dimensionality of the approximated latent spaces changes in between layers. Simply looking at $\dim(E_l^k)$ on a layer-by-layer basis is possible but hard to interpret,

since the number of filters and thus the dimensionality of the feature space changes between layers. To normalize the measurements for all layers, we instead compute the dimensionality of the latent space $\dim(E_l^k)$ relative to the dimensionality of the feature space $\dim(Z_l)$ of the same layer l :

$$s_l = \frac{\dim(E_l^k)}{\dim(Z_l)}$$

We refer to the resulting value s_l as the layer saturation of the layer l . Since $\dim(E_l^k) \leq \dim(Z_l)$ this value is bound between 0 and 1 for all layers l and thus allows for a simple comparison between different layers and networks. We chose the name "saturation" since this value effectively describes how much the feature space is "saturated" with data. The intuition here is that high saturation corresponds to a low degree of overparameterization, and vice versa for low saturated layers.

In all subsequent experiments, we standardize the value of δ to 99% for computing E_l^k , which is akin to the threshold chosen by similar techniques such as SVCCA by Raghu et al. (2017) and the pruning techniques by Garg et al. (2020) and Chakraborty et al. (2019). From a practical perspective, we find that while this value is unlikely to produce *relevant eigenspaces* for the layers of complex networks, the resulting saturation values are easy to interpret, which is more relevant for the empirical part of this work.

Conceptually, this metric was initially proposed by Shenk (2018) and the computation methodology and analysis presented here were proposed and developed and utilized by Shenk et al. (2019), Richter et al. (2021c), Richter et al. (2022), Richter et al. (2021a) and Richter et al. (2021b).

8.2 Evolution and Stability of Saturation Patterns

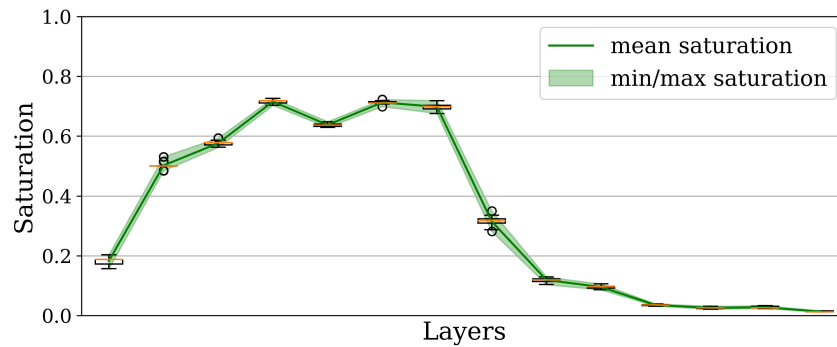
Since we use saturation as a primary analysis tool for this work, it is of vital importance to check the stability of the resulting saturation values given the typical random components of model training, which are data augmentation, weight initialization, and shuffling of the training dataset. Quantifying the influence of random variables in the training setup further allows us to estimate the reliability of saturation values and the emerging patterns of saturation values. Since the performance of the trained model also fluctuates due to these components, we also expect slight fluctuation in the saturation, which should be less or equal in proportion to the fluctuation in test accuracy.

8.2.1 Methodology

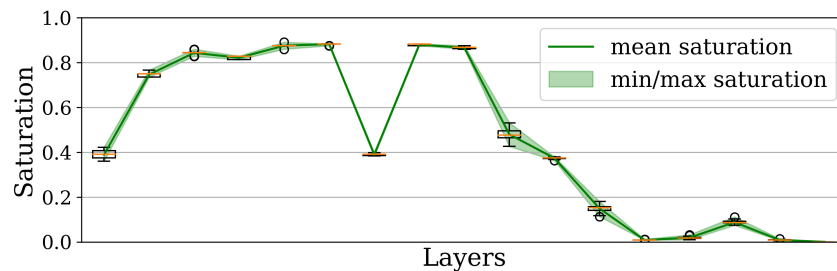
We test the stability of models on ResNet18 and VGG16. We test a model with skip connections and a model without skip connections. This is especially important in this scenario, since a network with residual connections can theoretically skip any layer in the network, which may result in different saturation patterns. We train each model 50 times on Cifar10 to obtain a reasonable sample size. The hyperparameters and preprocessing are equivalent to the setup described in section 8.4.1. The saturation values of every layer are visualized in order of the forward pass.

8.2.2 Results

The result in figure 44 suggests that ResNet18 and VGG16 have very stable saturation values in all layers. The minimum and maximum deviations are small and do not influence the overall pattern of saturation levels within the architecture.



(a) Saturation values of 50 VGG16 models trained on Cifar10.



(b) Saturation values of 50 ResNet18 models trained on Cifar10.

Figure 44: When the same model is trained multiple times using the same setup, the same saturation values emerge from the layers with only minor fluctuations. These figures were previously published in Richter et al. (2021c).

In fact, the standard deviation σ_s of VGG16's saturation is 0.281 whereas 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 the saturation is sufficiently stable to allow for the analysis of convolutional neural networks.

8.3 Logistic Regression Probe Ablation Study

To understand the semantics of high and low saturated subsequences of layers it is essential to connect saturation with another measurement with a known semantic. The logistic regression probes by Alain, Bengio (2017) is such a measurement, and it is described in greater detail in section 4.3 (theoretical) and section 6.2.5 (technical).

However, before we proceed to compare these two analysis tools, we need to ensure that the logistic regression probes produce reliable, artifact-free results. One of the key concerns with the original implementation of logistic regression probes is the way high dimensionality in convolutional neural networks is reduced for logistic regression probe training. The dimension reduction on the extracted data from the layers is a necessary concession to practicality as the size of the datasets and the dimensionality of the data vectors would otherwise make the application of logistic regression probes resource-intensive to the point of impracticality (Alain, Bengio (2017)). To address this issue, Alain, Bengio (2017) use two distinct techniques for dimension reduction. In one experiment, the authors downsample the feature maps using global average pooling (GAP), while in another experiment a random selection of feature map positions is used instead. While both experiments show consistent results, they also both induce unquantified biases into the analysis.

Since we want to extensively use logistic regression probes in this work, we need to investigate the biases induced by dimension reduction techniques. We decide against using a random selection of feature map positions, since the selection adds a random component to the analysis. On the other hand, GAP is commonly used in convolutional neural architectures as a readout-layer between feature extractor and classifier. However, we suspect that reducing a feature map of arbitrary resolution down to a 1×1 feature map is likely to induce biases at high resolutions. We also suspect that this effect is especially severe for early layers, where the information contained in individual feature map positions will be more local and, as a consequence, more heterogeneous. However, common implementations of GAP also allow for the reduction of feature maps to other feature map shapes, thereby effectively

averaging evenly sized sectors of the feature map to produce an output of shape $h \times w$ with h and w being arbitrary integers smaller than the original height and width of the input feature map. We also test nearest interpolation as an alternative downsampling strategy.

8.3.1 Methodology

Since the main purpose of this experiment is to support the reliability of the experiments of the following sections we choose a dataset and models that are heavily used throughout this work. For the dataset, we choose Cifar10 and 2 ResNet18 variants for the models. Both variants were originally proposed by He et al. (2016a), where one variant is optimized for 32×32 pixel input resolution and the other variant is optimized for 224×224 pixel input resolution. The difference between the two versions is that the low-resolution variant has the stem replaced by a 3×3 convolution with stride size 1. The primary reason we choose these architectures is that both do not exhibit a tail pattern (see section 8.5) at their respective design resolution. Tail patterns are recognizable in logistic regression probes by stagnating probe performances, which is not useful for an ablative study. To observe the degenerative effect of downsampling, we require models that distribute the inference process as evenly as possible across the network’s structure. In section 9.4, we demonstrate that networks with residual connections are especially good at distributing the inference across many layers. By training both models on their respective design resolution we can furthermore observe potential differences in ablation between high and low-resolution feature maps on very similar architectures. The training is conducted on Cifar10 using the following hyperparameters:

Table 6: Hyperparameters for the ablation study.

Parameter	Values
Input Resolution	(32×32) and (224×224)
Epoch	90
Batch size	64
Optimizer	SGD
SGD: learn rate	0.1 (decayed every 30 epochs)
SGD: decay-factor	0.1
SGD: momentum	0.9
SGD: weights decay	disabled

Since we want to maximize the presence of potential artifacts, we decide to train the model for 90 epochs, which gives both models enough time to

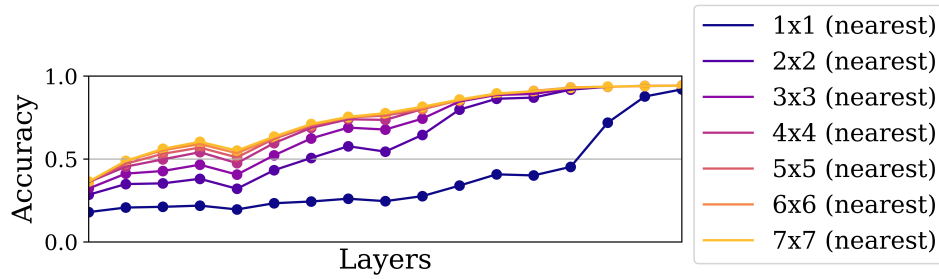
overfit on the data. Zhang et al. (2017) demonstrated that both models have enough capacity to overfit on the Cifar10 dataset and thus be sensitive to arbitrary details. If this overfitting is structurally represented on the feature map, downsampling could potentially destroy information required for classification, which could create artifacts in the test accuracy of the logistic regression probes that are dependent on the resolution of the feature map. We use the preprocessing and data augmentation strategies that are also used in most other experiments of this work. The images are channel-wise normalized with $\mu = (0.4914, 0.4822, 0.4465)$ and $\sigma = (0.2023, 0.1994, 0.2010)$, which is a procedure proposed by Krizhevsky et al. (2012). At training time, the images are first cropped randomly with a 4 pixel zero-padding on all edges. The size of the crop is 32×32 pixels. The crops are then randomly horizontally flipped with a probability of 50%. Since the augmentation strategies contain random components, they also have the potential to induce artifacts and inconsistencies. Finally, the images are resized to the input resolution and the images of the training set are reshuffled after each epoch. The models' weights are initialized using the Kaiming-He initialization proposed by He et al. (2015), which is used for all models throughout this work.

All logistic regression probes are trained on the layer outputs of the trained models. Unfortunately, due to resource limitations, it is not possible to directly measure the bias caused by the loss of information by feeding the entire feature map into a logistic regression probe for all layers. However, we can approximate the bias by feeding different feature map sizes into the logistic probes and observing the difference in performance. If this approximation is sufficient, we will see diminishing differences in the logistic regression probe performances with increasing feature map resolution. We feed (1×1) , (2×2) , (3×3) , (4×4) , (5×5) , (6×6) and (7×7) feature maps into the logistic regression probes using either GAP or nearest interpolation. The aforementioned sizes are maximum feature map sizes, and the feature maps are not up-sampled if they are naturally lower resolution. The logistic regression probes are trained using the SAGA-solver by Defazio et al. (2014), which is necessary to allow training within our resource limitation. No preprocessing of the data of any kind is conducted while training the probes, except for the mandatory flattening the downsampled feature map into a vector for processing.

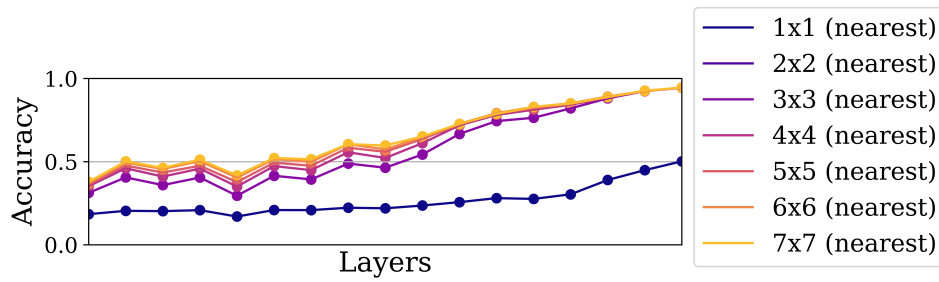
8.3.2 Results

We first explore the results from logistic regression probes trained on downsampled feature maps using the nearest interpolation algorithm. Downsampling to 1×1 feature maps results in strong ablation on multiple levels (see figure 45). First, the performance is strongly degraded on all layers for both resolutions. Reducing the resolution to 1×1 furthermore induces heavy artifacts in the patterns of logistic regression probes when visualized in the sequence of the forward pass. For any other feature map resolution, the overall pattern of the probe performances is similar. The overall predictive performance of the probes decreases with smaller feature map sizes. Earlier probes ablate more in terms of predictive performance than those trained on the output of later layers. The ablation shrinks with increased resolution, indicating convergence towards the "true" probe performance when trained on the entire feature map. Besides the anomalous behavior of the 1×1 downsampling, this behavior was expected, since earlier layers feature more local information and thus suffer the most from downsampling. An increase in probe performance with increased feature map resolution after downsampling is also expected in this regard, since more information is added to the feature vector.

When using GAP as a downsampling strategy, the ablation in terms of performance is stronger for low resolutions such as 1×1 and 2×2 (see figure 46). Similar to nearest-interpolation, the ablation is strongest for early layers and decreases in a converging manner with increased resolution. The anomalous behavior of the 1×1 downsampling when using nearest-interpolation is not present when using GAP. Thus, we consider the observed behavior as being an artifact of the downsampling strategy. The probe performance patterns look slightly different on 32×32 and 224×224 pixel resolution models. However, since these changes are consistent for the respective model over different levels of downsampling, we attributed this to the changes in the architecture. Interestingly, the ablation is not visibly stronger at higher resolution.

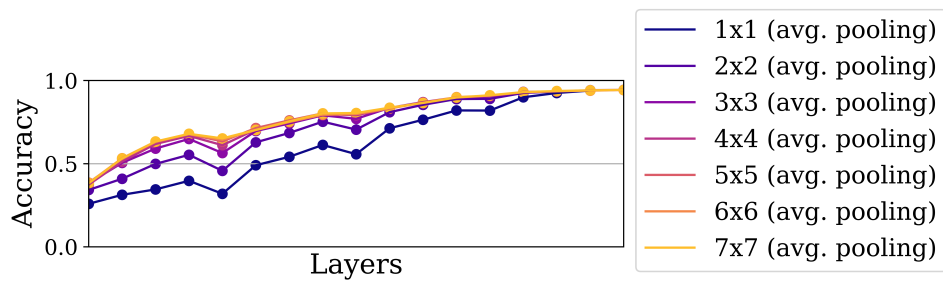


(a) Probes trained on downsampled feature maps. The model was trained on a 32×32 input resolution.

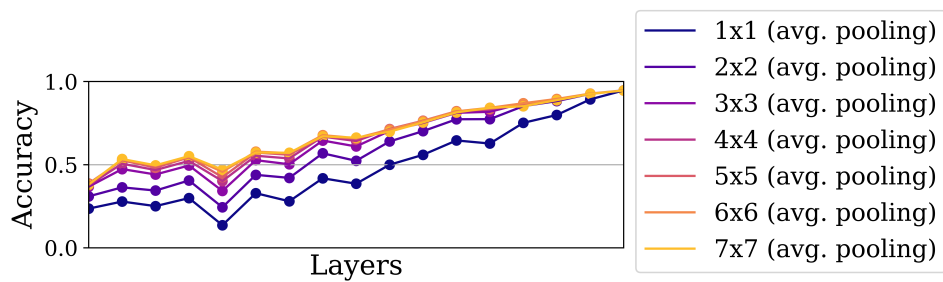


(b) Probes trained on downsampled feature maps. The model was trained on a 224×224 input resolution.

Figure 45: Nearest-Interpolation downsampling to a single pixel induces heavy artifacts, thereby destroying the otherwise prevalent pattern of probe performances. Otherwise, probe performances maintain a visible pattern and improve with diminishing returns as the output resolution of the downsampling is increased. These figures were previously published in Richter et al. (2021c).



(a) Probes trained on adaptive average pooled feature maps. The model was trained on a 32×32 input resolution.



(b) Probes trained on adaptive average pooled feature maps. The model was trained on a 224×224 input resolution.

Figure 46: Average pooling is better able to maintain the structure of probe performances, thereby not showing the artifacts observed in figure 45. These figures were previously published in Richter et al. (2021c).

8.3.3 Implications for Experimental Setups Using Logistic Regression Probes

The results of this small ablation study show that artifacts are present and visible when strongly downsampling the feature maps. For this reason, we do not adopt the GAP strategy originally used by Alain, Bengio (2017). We reject nearest-interpolation as an alternative, since the artifacts on 1×1 downsampling are even more severe. Instead, as a compromise between accurate probe performance measurement and practicality, we divert to using GAP for downsampling to a 4×4 feature map. A 4×4 feature map is still 16 times more computationally expensive to train than a globally pooled variant. However, in our experiment, logistic regression probes trained on 4×4 average pooled feature maps still performed close to probes trained on higher resolution. Since we are most interested in patterns in the sequence of probe performances, the ablation caused by downsampling to 4×4 is still tolerable as it maintains the structure of the probe performance sequence depicted in figures 46 and 45.

8.4 Studying the Average Saturation of Convolutional Neural Architectures

The underlying hypothesis in this and subsequent experiments is that more complex data requires more model capacity in each layer to be processed, which in turn is expressed in a higher dimensionality of the latent representation, i.e., higher saturation values. We first test this hypothesis by studying how saturation is affected by evenly increasing the number of parameters in the network. This is achieved by increasing the number of filters in each convolutional and the number of units in each dense layer. This is also colloquially referred to as changing the *width* of the network and it is a common scaling strategy for increasing the capacity of convolutional neural networks (Zagoruyko, Komodakis (2016), Tan, Le (2019), Tan, Le (2021), Tan et al. (2020)). If saturation is indicative of the level of overparameterization, we expect the average saturation s_μ of the networks to increase proportionally with a decrease in network width. By extension, we expect the predictive performance of the models to change anti-proportionally to s_μ . This anti-proportional relationship is also suggested by the lottery-ticket hypothesis, which postulates that overparameterization is necessary for a neural network to generalize well during training, even though only a small portion (the "winning ticket") of the network will be responsible for the quality of the prediction (Frankle, Carbin (2019)).

8.4.1 Methodology

We test this hypothesis on Cifar10 on a set of 20 different architectures. We train VGG11, 13, 16, and 19 for 30 epochs on the respective dataset. To observe changes in predictive performance and average saturation, we reduce the dimensionality of each layer by reducing the number of filters per layer by a factor of $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, and $\frac{1}{16}$ respectively. We use the standard input resolution of Cifar10 of 32×32 pixels. During training time, the images are randomly horizontally flipped, cropped, and finally channel-wise normalized using channel-wise means and standard deviations taken from Krizhevsky et al. (2012). The exact hyperparameter configuration can be seen in table 7.

Table 7: Hyperparameters of training setups for all models trained on Cifar10 in this experiment.

Parameter	Values
Input Resolution	(32×32)
Epoch	30
Batch size	128
Optimizer	ADAM
ADAM: beta1	0.9
ADAM: beta2	0.999
ADAM: epsilon	1e-8
ADAM: learning rate	0.0001

We use the accuracy metric as a measure of predictive performance. For measuring the average saturation s_μ of the entire model we compute:

$$s_\mu = \frac{1}{|L|} \sum_l^L s_l$$

where L is the set of all layers l with trainable parameters (in essence all convolutional and dense layers). We omit computing the saturation for non-parameterized layers such as pooling, flattening, or dropout layers to avoid biases of s_μ . These non-parameterized layers are unlikely to drastically change the intrinsic dimensionality and would thus closely resemble the feature space from the previous convolutional or dense layer, resulting in oversampling of specific feature spaces right before such a non-parameterized layer. All model trainings are repeated 3 times, and the measurements are subsequently averaged.

8.4.2 Results

We visualize the results in figure 47. The average saturation decays when the number of dimensions in the layers feature spaces is reduced and the predictive performance decays alongside it. The relationship between a model's s_μ and predictive performance on Cifar10 is logarithmic. Since the number of filters (and therefore the feature space dimensionality) were reduced in exponential steps, this pattern can be expected.

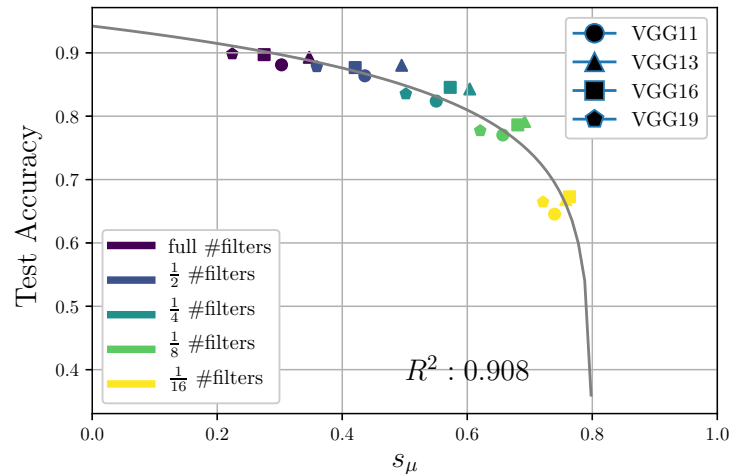


Figure 47: Predictive performance and average saturation form a logarithmic relationship when we train models on different filter sizes. The reduction in model capacity leads to the data "occupying" a higher percentage of the available feature space with a degrading effect on the predictive performance. This image was first published in Richter et al. (2022).

The relation of s_μ to the predictive performance also indicates that some degree of overparameterization is required for these architectures to achieve high predictive performance, which aligns with the lottery ticket hypothesis of Frankle, Carbin (2019). In the experiment-scenario, highly saturated networks generally achieve lower performance, while low saturated networks improve performance with diminishing returns.

In summary, s_μ behaved as expected, increasing with a decrease in network capacity and predictive performance. The diminishing decrease in saturation resulting from increasing the capacity also implies that it should be possible to find a sweet spot concerning the width of the network that can be determined using a saturation interval. This is investigated further in section 11.3.

8.5 Understanding Saturation Patterns

From section 8.4 we learn that the average saturation of a neural network increases with decreasing capacity. However, this insight does not provide any information regarding the distribution of the saturation within the network. In this section, we aim to build a basic understanding of the semantics of saturation based on comparing logistic regression probe performance with saturation values. To achieve this, we view the saturation and probe performance values as a sequence in the order of the layers during the forward pass, similar to the visualization of section 8.2. By visualizing these sequences, we aim to build an understanding of how saturation is distributed among the layers and what the implications of this distribution are. Furthermore, we explore how these patterns are affected by the input resolution and difficulty of the dataset. The goal of these experiments is to understand how saturation is affected by different circumstances on a basic level, and whether there are states of trained models that can be detected by saturation and are undesirable from an efficiency and/or performance perspective.

8.5.1 Methodology

We first test how stable a saturation pattern is when the same model is trained across multiple datasets of varying difficulty. We train ResNet18 on MNIST, Cifar10, TinyImageNet, and ImageNet. We chose these datasets since they can be considered as being of ascending difficulty, based on the heterogeneity of the input data, the number of classes, and the typical accuracy achieved by models on these datasets (Scheidegger et al. (2021)). For all datasets, we train the model using the same hyperparameter configuration (see Table 8).

Table 8: Hyperparameters for training ResNet18 on datasets of varying difficulty.

Parameter	Values
Input Resolution	(224 × 224)
Epoch	30
Batch size	64
Optimizer	ADAM
ADAM: beta1	0.9
ADAM: beta2	0.999
ADAM: epsilon	1e-8
ADAM: learning rate	0.0001

The data augmentation and preprocessing are the same as the experimen-

tal setup described in section 8.4.1. Due to resource limitations, we do not train logistic regression probes for this particular experiment.

To investigate the influence of the input resolution, we train ResNet18 on Cifar10 on three different resolutions using the hyperparameter settings and preprocessing described in table 8. The first model is trained on the native resolution of Cifar10 of 32×32 pixels, whereas the second model is trained on 224×224 pixel resolution, and the final model is trained on 1024×1024 pixel resolution. The tested architectures were initially designed and optimized for a 224×224 pixel input resolution. The largest and the smallest input resolution are intentionally chosen to be very over and undersized to observe the effect on probe performances and accuracy as clearly as possible. The Cifar10 dataset was chosen since the native resolution of Cifar10 of 32×32 pixels is rather small, thereby ruling out potential side effects by the addition of details from an increased input resolution. We will analyze the distinct effects of additional details and the input resolution in chapter 9.

To test the observed results for consistency, we reproduce these results on MNIST and TinyImageNet. We also test this experimental setup and slight variations of it on VGG16, ResNet34, and ResNet50. For the sake of readability, this section does not depict these additional reproductions, since these experiments only showcase that the observed behavior can be reproduced on different datasets and models. The results of these additional experiments can be found in appendix D.

8.5.2 Results

The results concerning the difference in dataset complexity can be seen in figure 48. The depiction shows that the overall saturation increases with the problem complexity. ImageNet causes the highest overall amount of saturation per layer and is also the most complex dataset with 1,000 classes and a high heterogeneity in the resulting images (Scheidegger et al. (2021)). MNIST, on the other hand, can be regarded the simplest dataset, only consisting of 28×28 pixel binary images belonging to 10 classes. The model trained on MNIST is also the lowest saturated. Another observation made during this experiment is that the overall pattern of how the saturation is distributed does not substantially change in shape. From these observations, we can conclude that the dataset has an influence on saturation. An increase in difficulty primarily causes an increase in saturation, possibly due to an increase in processing required to advance the solution quality. This also implies that problem difficulty and the capacity in each layer are related, since the lower

capacity networks tend to have a higher average saturation (see section 8.4). In later experiments (section 11.3) we will demonstrate that this property can indeed be leveraged to optimize convolutional neural network architectures by matching the model capacity and problem difficulty

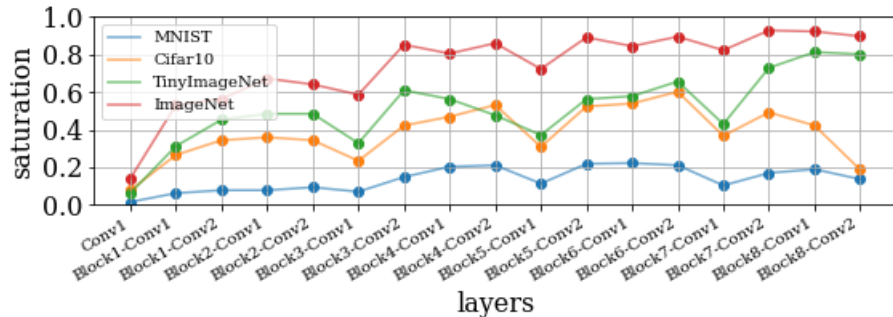


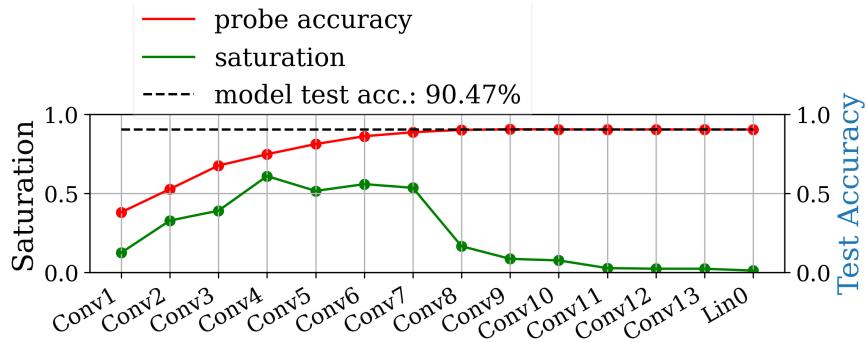
Figure 48: Overall patterns increase in contrast, and the overall saturation increases when the dataset is more difficult. This figure was previously published in Richter et al. (2022).

While the previous results demonstrate that the interaction of dataset and neural architecture is reflected in the sequence of saturation values, we have yet to identify specific patterns that are indicative of inefficiencies in the analyzed model. To find such a pathological pattern, we proceed to study the sequence of saturation values in a convolutional neural network and compare these to the logistic regression probe performances obtained from the same layers. In figure 49 we can see that an interesting pattern emerges when a model is trained on a resolution of 32×32 pixels, which is lower than the input resolution the architecture is designed for, which is 224×224 pixels in the case of all architectures used for this experiment. Starting from layer "Conv8" the predictive performance of the logistic regression probes has reached the predictive performance of the softmax output of the trained model on both datasets. The following layers thus no longer improve the quality of the intermediate solutions. We can consider this an inefficiency in the trained model, since the parameters of all layers past Conv8 are effectively unused as they do not change the predictive quality of the model. This is also reflected by the saturation of these layers, which is substantially lower in the unproductive part between Conv8 and the output of the model compared to the layers in the rest of the network. We refer to this saturation pattern as a *tail* or *tail pattern*, due to its visual appearance when plotting the saturation values in the order of the forward pass. Based on this observation and other tail patterns created on the different dataset and neural architecture combinations (see appendix D), we define a tail as follows:

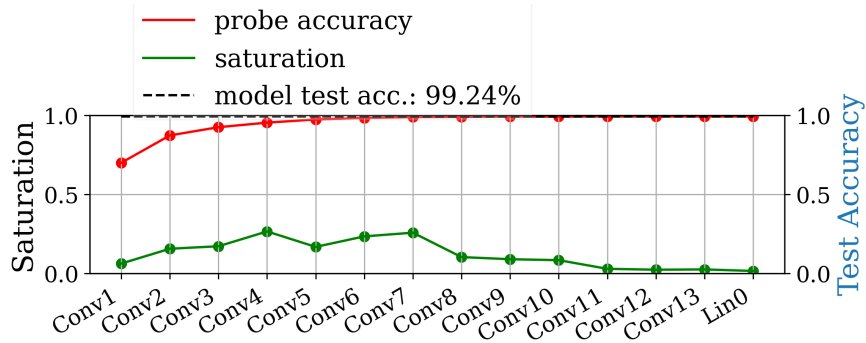
A tail is a subsequence of at least 3 consecutive layers in a feed-forward neural

architecture with an average saturation that is 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 being 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.



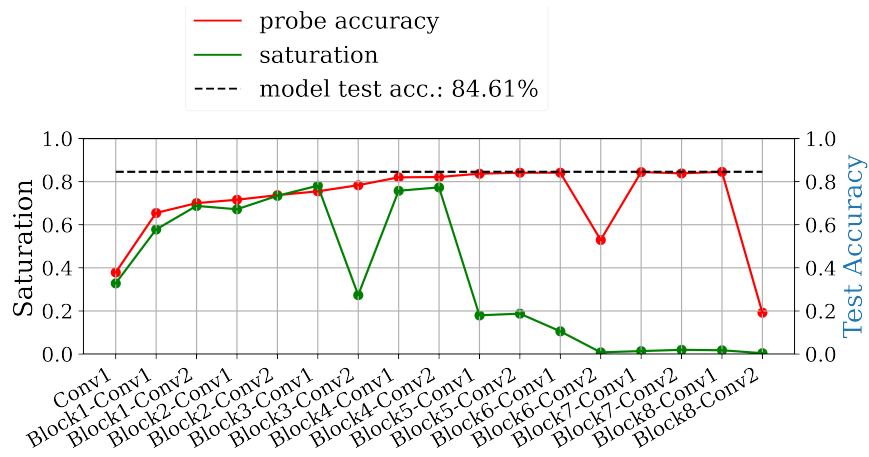
(a) VGG16 trained on Cifar10



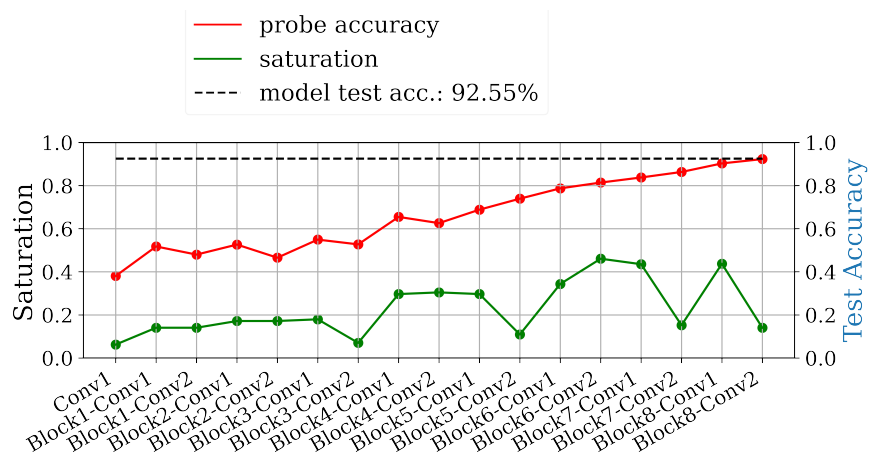
(b) VGG16 trained on MNIST

Figure 49: Examples of tail patterns from Conv8 to the output of the model. This pattern emerges when training a model, in this case VGG16, optimized for high resolution, in this case 224×224 pixels, on a low input resolution, which is 32×32 pixels in both scenarios. Layers that are part of the tail pattern are substantially lower saturated relative to the rest of the model, and the logistic regression probes stagnate at the performance level of the final model. The probe performance indicates that these layers act as pass-through layers and do not contribute to the quality of the solution, which can be regarded as a parameter-inefficiency since these layers are effectively underutilized.

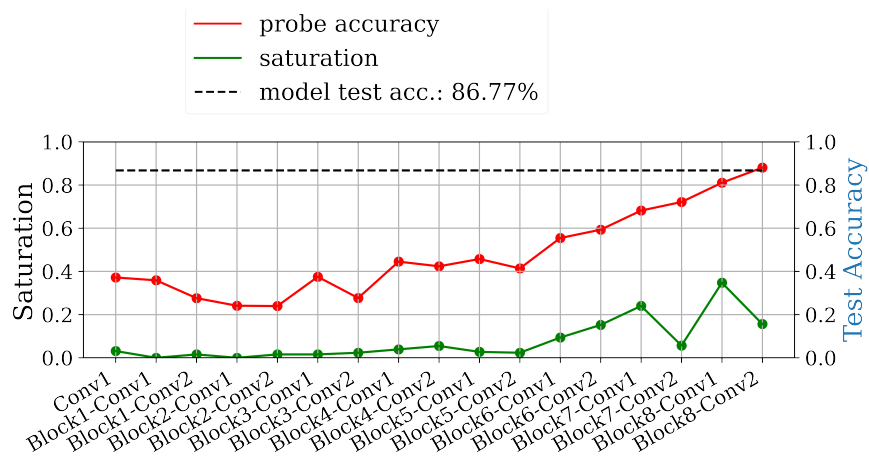
By studying the result in figure 50, we further observe that it is possible to shift and remove the tail pattern by changing the input resolution. ResNet18 in figure 50a shows an unproductive tail similar to VGG16 in figure 49. When oversizing the resolution to 1024×1024 (see figure 50c), the tail is shifted to the input of the model, with the result that these layers do not contribute qualitatively to the inference according to the logistic regression probes. In both cases, the predictive performance of the model is also substantially lower than the same model trained on the design-resolution of 224×224 pixels.



(a) 32×32 (Cifar10 native resolution)



(b) 224×224 (ResNet standard)



(c) 1024×1024

Figure 50: Changing the input resolution changes how the inference is distributed among the layers. The resolution ResNet18 that was designed for (b) distributes the inference most evenly, while too small (a) and too large (c) resolution shift the bulk of the inference process to early and later layers respectively, resulting in worse predictive performance.

The drop is from 92.55% to 84.61% in the case of 32×32 pixel images and to 86.77%, when trained on 1024×1024 pixel images.

In summary, the results provide further evidence that saturation contributes interesting insights to the inference process. We can see that more complex problems, which could be argued to require more processing to be solved as a consequence, increase the overall saturation of the model. Furthermore, we can see that mismatches between input resolution and neural architecture result in unproductive layers that consequently have substantially lower saturation than the rest of the network (tail pattern). The latter provides evidence that not only the absolute saturation value may be important, but also the saturation values in a network relative to each other. The causes for this resolution-dependent distribution of the inference process will be investigated in greater detail in chapter 9, where we will also provide further evidence (section 9.2) that only higher saturated sequences of layers tend to actively contribute to the inference process when saturation is distributed unevenly. In chapter 11, we will also show that removing the layers of the tail pattern is an effective way to reduce overparameterization and increase computational and parameter efficiency.

8.6 Conclusion

In this section, we investigated whether our PCA-based approximation of the latent space can provide interesting insights on the inference process of convolutional neural networks for the entire network and on a layer-by-layer basis. We proposed the saturation metric and studied its properties experimentally. We were able to show that the average saturation of models is influenced by the problem difficulty and model capacity.

Furthermore, we identified a pathological pattern in the saturation values, which is indicative of underutilized layers in the model. This pattern can be described as a sequence of layers with substantially lower saturation values than the rest of the model. Due to its appearance in the visualization, we refer to this pattern as a "tail pattern". Based on the comparison with logistic regression probes, it is possible to show that layers that are part of the "tail" do not advance the quality of the intermediate solution and can thus be considered as being underutilized and unproductive, thereby providing further evidence that saturation can provide insights into the inference process. We further demonstrated that the presence and location of tail patterns are caused by a mismatch between neural architecture and input resolution.

While saturation has shown promising results, further investigation is needed to gain a more profound understanding of saturation and the relationship of model and input resolution, both of which are explored in chapter 9 and chapter 10. We will proceed to use saturation with logistic regression probes as redundant metrics in future experiments to demonstrate the consistency of the results. This is also relevant for demonstrating the usability of saturation for practical applications, since saturation is computed live during training with little overhead, while the training of logistic regression probes often exceeded the computation time of the models themselves, thereby making the former a substantially more practical tool to use in real-world applications than the latter.

9 Exploring the Relationship Between Input Resolution and Neural Architecture

The superior performance of convolutional neural networks in computer vision can be attributed to the way they extract information from image data. The information processing follows a bottom-up approach, whereby smaller, less complex features extracted by earlier layers are successively combined to larger and more complex features in later layers (see section 2.3). The receptive field of a convolutional layer can be seen as an upper bound of the size of features it can extract¹³. This property results in deeper layers being able to detect larger patterns than earlier layers, since they can "see" an increasingly wider area on the input image. This also means that the size of the input image controls – to some degree – how the inference process is distributed inside the network's structure. In section 8.5, we demonstrate this using logistic regression probes and saturation. For any conventional CNN classifier, the input images are resized to a standardized, square (often 224×224 pixels) input resolution, which means that there is also an upper limit to the usefulness of implementing increasingly larger receptive field sizes. The relevance of this is demonstrated by Tan, Le (2019), who show that a network needs to be scaled together with the input resolution to achieve good efficiency. Independently, we also demonstrate experimentally in section 8.5 that convolutional neural networks have a preferred input resolution, where over and undersized images would perform suboptimally due to unproductive layers that do not contribute qualitatively to the inference.

In this section, we investigate the relation of input resolution and neural architecture further by answering the following questions:

- Does the size of the input image have an effect on the predictive performance of CNN classifiers? Answer: yes, altering the resolution and adding details improves performance independently of each other (section 9.1).
- Does the size of discriminatory features¹⁴ influence how the information is processed in the network? Answer: yes, we can show that processing

¹³In this work, we refer to the height and width measured in pixels (absolute size) as "size".

¹⁴We refer to any pattern depicted on images in the dataset that yields useful information for solving the classification task as a discriminatory feature. We find this term less problematic than "object of interest", since a classifier does not have a concept of "objectness", "object", or "object of interest" and will thus opportunistically detect any feature if it helps to minimize the cross-entropy loss. Furthermore, a classifier could be used to detect textures, scenes, or other data that has no well-defined object of interest.

significantly differs depending on input size (section 8.5) and the size of the depicted objects in a very similar way (section 9.2).

- Can we know in advance which layers will contribute to the quality of the prediction based on the input size? Answer: For strictly sequential architectures the receptive field size allows for the identification of unproductive layers (section 9.3).
- Do residual connections influence the observed behavior? Answer: Yes, residual connections can help to involve more layers in the inference process (section 9.4).
- Do the results have implications on neural architecture design? Answer: Yes, we propose methods to optimize architecture before and after training (section 9.5).

The results of these sections were previously published in Richter et al. (2021a).

9.1 The Effect of Input Resolution and Details on the Predictive Performance

A simple explanation for the positive effects of larger input resolutions in modern architectures like EfficientNet and GPipe on predictive performance is the addition of more detailed information contained in the data, allowing for better decision-making. This argumentation was brought forward by multiple authors over the years, including Szegedy et al. (2016); Tan, Le (2019) and Szegedy et al. (2017). However, this explanation contradicts the results in section 8.5, where we show that models perform substantially differently on Cifar10 when the resolution is altered, even though the input resolution in all scenarios is greater or equal to the native resolution of Cifar10. This means that the addition of detail cannot explain the effect on the predictive performance in this scenario. Based on these results, we hypothesize that the influence on predictive performance by input resolution and by the addition of information (details) on the image are separate effects. We investigate this by training models multiple times on high-native-resolution datasets using small and large input resolutions. The third training setup will use the downsized low-resolution input images upscaled to the large input resolution. This third setup therefore maintains the large input resolution, while having no more

details than the low-resolution images. As our working hypothesis, we assume that the addition of details is the only influencing factor when it comes to predictive performance. If this hypothesis is true, the model trained on a large input resolution will outperform the other two training setups, since it has the most information content. The other two setups will perform equally, since every image effectively contains the same information for both datasets.

9.1.1 Methodology

We train multiple models on the ImageNet-dataset and iNaturalist in three different settings A, B, and C. Models of set A are trained on images with a size of 224×224 pixels, thereby providing the performance baseline. Set B models are trained on images of size 32×32 . Based on our hypothesis, we expect a drop in performance relative to A. Set C is trained on the images used in set B up-sampled to 224×224 pixels, thereby effectively keeping a low amount of detail on higher resolution input images. According to the working hypothesis, performance should not increase relative to group B. In any other regard, the training setup is identical for all runs, using the hyperparameter settings in table 9.

Table 9: Hyperparameters for the resolution experiments.

Parameter	Values
Input Resolution	(32×32) and (224×224)
Epoch	60
Batch size	64
Optimizer	SGD
SGD: learn rate	0.1 (decayed every 20 epochs)
SGD: decay-factor	0.1
SGD: momentum	0.9
SGD: weights decay	disabled

The preprocessing pipeline consists of a channel-wise normalization using the same parameters for μ and σ as Krizhevsky et al. (2012). During training, the data is augmented by randomly cropping the image using a square containing between 5% and 100% of the original image, in addition to horizontal flipping. The same pipeline was used by He et al. (2016a), Szegedy et al. (2016), and Simonyan, Zisserman (2015) and can be considered typical for high-resolution datasets such as ImageNet and iNaturalist.

Table 10: Relative Top1-Accuracy: The table shows the predictive performance of trained models relative to a baseline model. The baseline uses the same model and architecture and is trained on images resized to 224×224 pixels. Training the models on 32×32 pixel images results in less than half the predictive performance in all tested scenarios relative to the baseline. Resizing the 32×32 images back to 224×224 results in a significant recovery of lost performance in all tested scenarios, despite the fact that no information is added by upscaling. These results were previously published in Richter et al. (2021a).

Model	Dataset	downscaled upscaled	
		32×32	224×224
VGG16	ImageNet	15.35%	66.08%
VGG16	iNaturalist	36.83%	58.38%
ResNet18	ImageNet	45.74%	67.39%
ResNet18	iNaturalist	30.4%	52.68%
ResNet50	ImageNet	28.21%	71.8%
ResNet50	iNaturalist	33.87%	54.38%
EfficientNet-Bo	ImageNet	19.32%	64.45%
EfficientNet-Bo	iNaturalist	15.34%	63.79%

9.1.2 Results

From the results in table 10 we can conclude that decreasing the resolution has a negative effect on performance. Using up-sampled versions of these low-detail images for training partially regains the lost performance. Based on these results, we conclude that the size of the input images is an additional factor to the amount of information contained in the image, thereby influencing the accuracy of the model.

9.2 The Role of the Size of Discriminatory Features in the Relation of Model and Input Resolution

In the introduction of chapter 9, we briefly elaborated on how the receptive field size influences the distribution of the inference among the layers by limiting the size of features that can be recognized by a layer. Since the input resolution implicitly changes the size of discriminatory features (measured in pixels), we can deduct that the expansion of the receptive field could be responsible for the saturation patterns observed in section 8.5.

We test how the size of discriminatory features influences the formation of saturation patterns by restricting them to a local area on the image. We do this by training three distinct models using similar setups. The first model is trained on resized 160×160 resolution images of Cifar10, while a second model is trained on the native Cifar10 resolution of 32×32 pixels. These models serve as references for high and low-resolution saturation patterns. A

third model is trained on Cifar10 images in their native resolution, but placed randomly on a 160×160 canvas. While the images that this model is trained on are of high resolution, the discriminatory features are restricted to an area of 32×32 pixels, whereas the remaining image carries no information. If the size of the discriminatory features influence how saturation patterns form, we would expect a similar tail pattern from the model trained on 32×32 pixel images and the model trained on the Cifar10 images on the 160×160 pixel canvas.

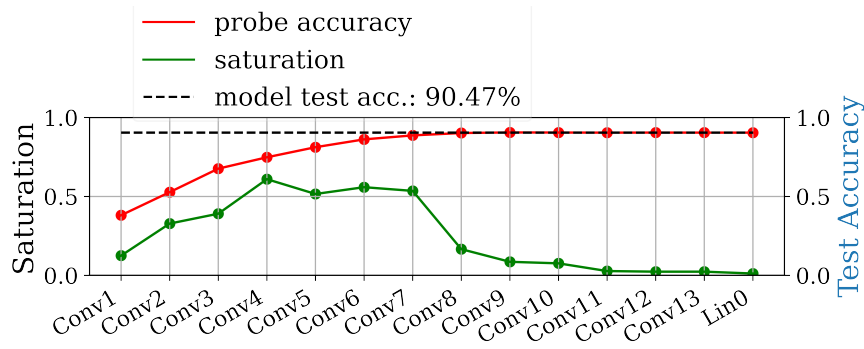
9.2.1 Methodology

We train the models on the same setup as in section 8.5, differing only in the resolution in two out of three experiments. As previously mentioned, the first two experiments act as baselines and differ only in the input size. The first model is trained on 160×160 pixel input size, while the second model is trained on 160×160 pixel inputs. The third model is trained on input images consisting of a 32×32 black canvas that contain a single Cifar10 image at a position with coordinates uniformly drawn from $[0, 128]$ for the top left pixel that serves as a placement-anchor. By doing so, the Cifar10 image on the canvas is never truncated, thus guaranteeing that the *object of interest* is always restricted to the same area as in Cifar10. The random position was chosen to avoid biases that could be caused by always placing the object of interest in the center (or any other) fixed position on the image. This position is chosen as the first step of the preprocessing pipeline during training and inference. The random position likely makes the problem more difficult and will thus affect the predictive performance. However, we can neglect this for this particular setup since we are primarily interested in the emerging saturation patterns and not in any changes concerning the predictive performance. We chose the Cifar10 dataset for its small input resolution and repeat the experiment on MNIST to demonstrate that the results are reproducible on other datasets (see Appendix E). The experiments are conducted on ResNet50 (Cifar10), VGG16 (MNIST, Cifar10), and ResNet18 (Cifar10) to reproduce the observed phenomena on models of different structures and depths. The results are analyzed using saturation (computed during the final epoch of training) in addition to probe performances, which serve as a redundant measurement to saturation. We expect the behavior of these two measurements to yield similar results and primarily chose to use both techniques for redundancy and to demonstrate the consistency of the observed relations between probe performance and saturation patterns initially presented in section 8.5.

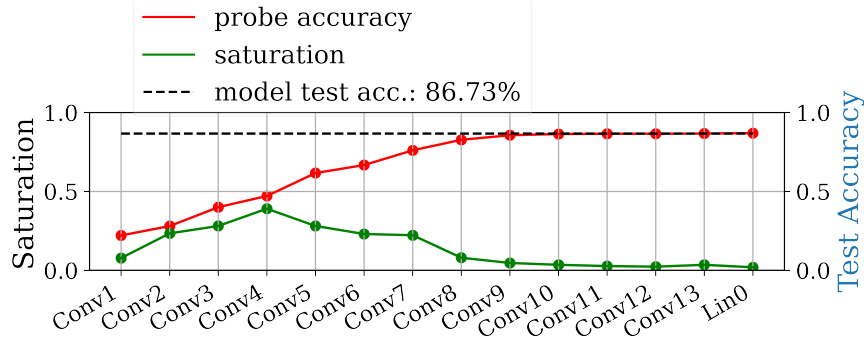
9.2.2 Results

As expected, the saturation patterns of the two baseline models (figure 51c and figure 51a) are very differently distributed, which is also reflected in the logistic regression probes. The model trained on the smaller resolution exhibits a tail pattern at the beginning of the network, while the model trained on the higher resolution does not, as is expected based on the results of section 8.5. The saturation of the model trained on the canvas-images in figure 51b is similar in shape to the 32×32 -pixel baseline model (figure 51a). The low saturated tail starts on convolutional layer 8 (Baseline) and 9 (Canvas) respectively.

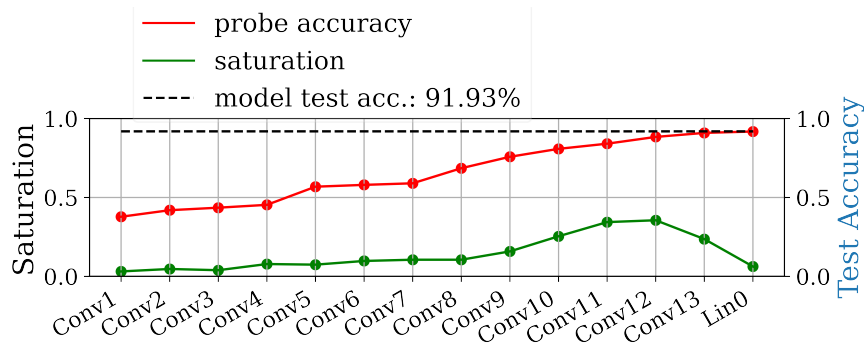
However, there are also some deviations between the saturation patterns in figure 51a and figure 51b that need to be addressed. First, the overall lower saturation in absolute values needs to be considered. We suspect that this can be attributed to the fact that the black canvas that the image was placed in drastically reduces the size of the gradient. We hypothesize that this is also the reason for the sub-optimal performance of the model, combined with the added difficulty of the object of interest being randomly placed on the image. The latter is also the suspected cause for the slightly shorter tail of the model since the information is ingested slower when it is not centered on the image, as the receptive field expands out of bounds on some edges earlier than others. This does not detract from the key observation made in this experiment, namely that the formation of a tail pattern is not caused by the resolution of the image but rather by the size of the discriminatory features. Differently sized discriminatory features are processed by different layers, and this does not change even if the input resolution changes. However, we proceed to use the input resolution as an approximation of the size of the largest discriminatory features in future experiments. We decide to do so, since it is impossible to primitively compute the size and shape of discriminatory features that the models may learn to detect. Using an object size like a bounding box as an estimation would also be possible, but it would further complicate the computation, since such object sizes would be different for every image. Furthermore, a classifier has no concept of "object" and therefore will simply detect any pattern regardless of any context that a human would perceive as an object.



(a) 32 × 32 without canvas



(b) 160 × 160 with canvas



(c) 160 × 160 upsampled

Figure 51: A tail of unproductive layers can be produced by placing the Cifar10 images on a 160×160 pixel canvas (b). This indicates that the locality of discriminatory features (essentially the size of the object) is responsible for the observed effect on the inference process. These figures were previously published in Richter et al. (2021a).

9.3 Receptive Field Size and Tail Patterns

From the previous section, we can derive that features of different sizes are recognized by different layers in the model. In this section, we investigate the causes of this phenomenon from an architectural point of view. From the perspective of the architecture, the receptive field of a layer can be considered as an upper bound for the size of recognizable features. Since the receptive field expands with every layer that has a stride and/or kernel size > 1 , increasingly

larger features can be recognized. We hypothesize that for simple, sequential architectures like the VGG-family of networks ¹⁵ the receptive field is the dominating factor that influences the presence and position of unproductive subsequences of layers. We test this hypothesis by studying the relationship of the receptive field, saturation, and probe performance, with the goal of finding a regularity that allows us to predict which subsequence of layers will be unproductive. For this experiment, we assume that the expansion of the receptive field promotes a greedy strategy of information integration. This means that a layer will always try to integrate all novel information available to it and will only use this mechanism to improve the quality of the solution. Based on this assumption, we can define a clear border $r_{l-1} > I$, where r_{l-1} is the receptive field size that the input of the layer l is based on and I is the input resolution. We refer to the first layer that fulfills this property as the *border layer*. In a sequential architecture, this layer would be the first layer to be unproductive (dropping saturation, no probe performance improvement relative to the previous layer), since it – and any following layer – cannot integrate additional information by expanding the receptive field.

9.3.1 Methodology

We investigate the above aspects by conducting two distinct sets of experiments. The first set of experiments is conducted on Cifar10 using the entire VGG-family of networks, and thus consisting of VGG11, 13, 16, and 19. The training setup utilizes the following hyperparameters:

Table 11: Hyperparameters for the experiments conducted on the VGG-family of networks and the modified VGG and ResNet models.

Parameter	Values
Input Resolution	(32×32)
Epoch	60
Batch size	64
Optimizer	SGD
SGD: learn rate	0.1 (decayed every 20 epochs)
SGD: decay-factor	0.1
SGD: momentum	0.9
SGD: weights decay	disabled

We compute the receptive field, probe performances, and saturation val-

¹⁵We define a simple architecture as a sequential architecture only consisting of convolutional and (global) pooling layers. Convolutional layers may contain dropout, batch normalization, and non-linear activation functions.

ues for each layer. We choose the VGG family of networks for their architectural simplicity, which, we believe, will yield more clearly visible results. We train the model on the Cifar10 dataset using the dataset’s native resolution of 32×32 pixels to guarantee the emergence of a tail pattern on all four architectures. To further study the behavior of the differences between the layers of the tail and rest of the network, we train logistic regression probes on every position of every feature map of VGG16. By plotting these probe performances as heatmaps, we can observe how much information each position of a feature map contains. To make these heatmaps easier to read, we depict the relative accuracy, which is the probe test accuracy relative to the model test accuracy. For readability, we only provide a few selected layers in the results of these experiments. All heatmaps can be found in the appendix G, alongside a repetition of the same experiment on ResNet18. To confirm the observations made during these experiments, we repeat the experiments and analysis conducted on the VGG family of networks on a modified version of VGG19 with dilated convolutions instead of regular convolutions. The convolution’s dilation rate is 3 for all layers and the pooling layers are not dilated. Another repetition is conducted on ResNet18 with removed skip connections, which makes it a sequential architecture similar to VGG19. We do this to observe whether the observed regularities can be reproduced when applied to other architectures that use components that affect the receptive field and that are different from the ones used in the VGG-network family. In the case of the modified VGG19, the component is the dilated convolution operation, which effectively increases the kernel size in each layer to a 9×9 kernel. The ResNet18 variant features several such components. First, convolutions with stride = 2 are used instead of max-pooling layers for downsampling, a stem consisting of two layers (see section 3.3.2) and a different (more even) spacing of downsampling layers compared to the VGG-network family. We add the receptive field as an additional graph to our plot and mark the *border layer* with a black vertical bar. We define the border layer as the first layer to receive an input based on a receptive field size r_{l-1} bigger than the input resolution I : $r_{l-1} > I$. Since the size of the receptive field grows monotonically over the sequential structure of the network, every layer after the border will fulfill the same property and will thus be unproductive.

As a final analysis, we also train logistic regression probes on every individual feature map position to obtain a heatmap showing how far the partial solutions have progressed. The goal of this experiment is to get another view of the processing inside a layer that can show us the differences between the

productive and unproductive layers.

9.3.2 Results

For the tested sequential neural architectures of the VGG-family, the border layer predicts the start of unproductive layers precisely, as we can see in figure 52, which confirms our suspicions regarding the greedy integration of novel information and its relevance in advancing the intermediate solution quality during the forward pass.

We further investigate this observation by testing additional sequential architectures with properties that alter the receptive field size in ways different from previously tested VGG-models. These architectures are the previously described VGG19 variant with dilated convolutions and the ResNet18 variant with removed skip connections.

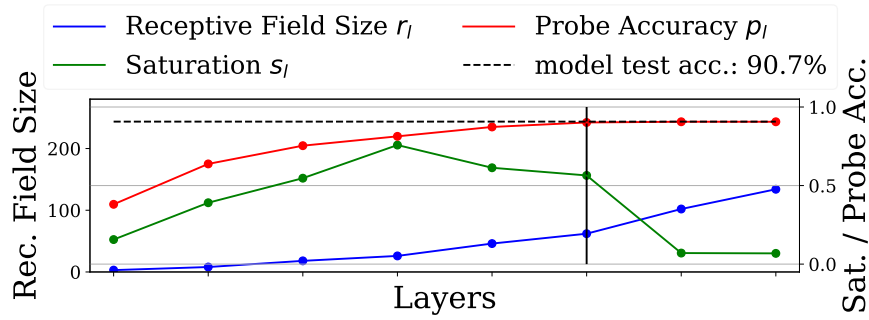
In figure 53, the results for these variants show that it is still possible to predict the border layer accurately, even when stems, a different spacing in downsampling layers, strided convolutions (ResNet18 depicted in figure 53 (a)) and dilated convolution (VGG19 depicted in figure 53 (b)) are used to influence the size of the receptive field. We see this as a further confirmation of our hypothesis.

Finally, we investigate how the solution develops inside the feature maps of different parts of the network. For this, we inspect the logistic regression heatmaps in figure 54 that are generated from training logistic regression probes on every individual feature map position and plotting their test performance relative to the model performance. Effectively, the plot visualizes the quality of the partial solutions contained in every position of the feature map based on their position. The central positions on the feature map generally perform best in early layers, while outer positions perform increasingly worse, with the corner positions generally being the worst. We suspect that the receptive field is at least partially responsible for this, since outer positions on the feature map will receive more black padding as input and thus less information with the receptive field expansion as a center pixel.

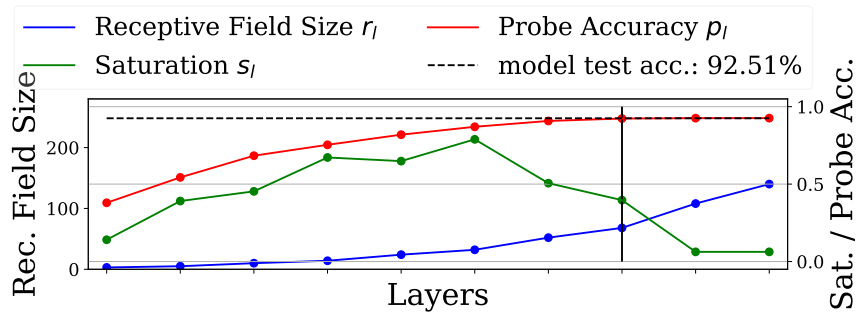
Another interesting observation in figure 54 (b) is that the center-most positions on the feature map contain partial solutions roughly equivalent to the performance of the entire model. As the saturation drops and layers become part of the low saturated tail, the partial solution quality becomes increasingly homogeneous across the feature map positions. In the final convolution layer depicted in figure 54 (d), the probe performance of all feature map positions is approximately equal to the predictive performance of the model. Based on

these measurements, we conclude that this homogenization of partial solution quality is also responsible for the drop in saturation.

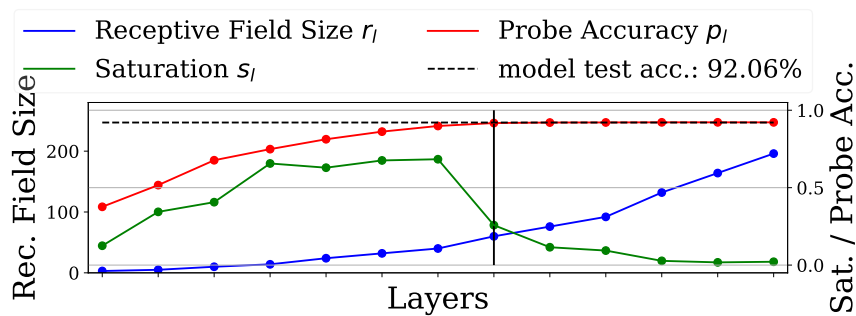
We can thus conclude, based on these observations, that simple, sequential neural networks may develop two stages of inference when the image is smaller than the receptive field size of the model. The first of these is the solving stage, where the data is processed incrementally to achieve loss minimization. The second stage, starting from the border layer, is the compressing stage. This stage compresses the latent space by homogenization of the partial solutions for every position in the feature map.



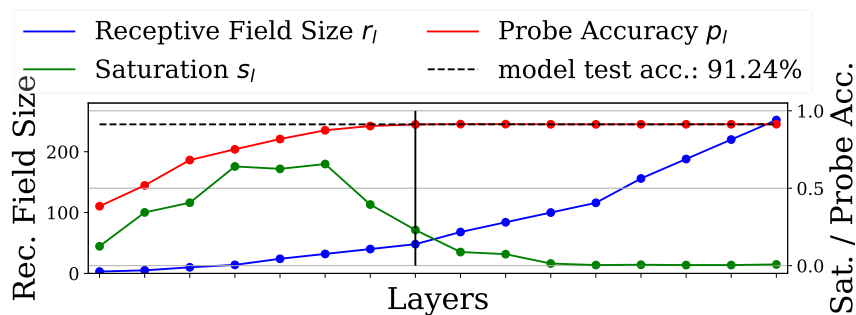
(a) VGG11—border layer at layer 6



(b) VGG13—border layer at layer 7

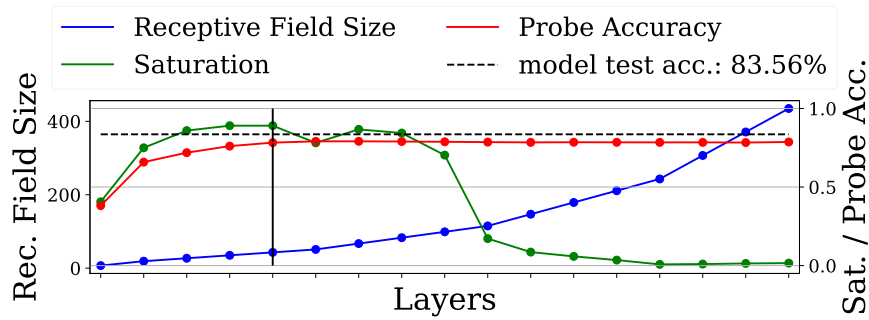


(c) VGG16—border layer at layer 8

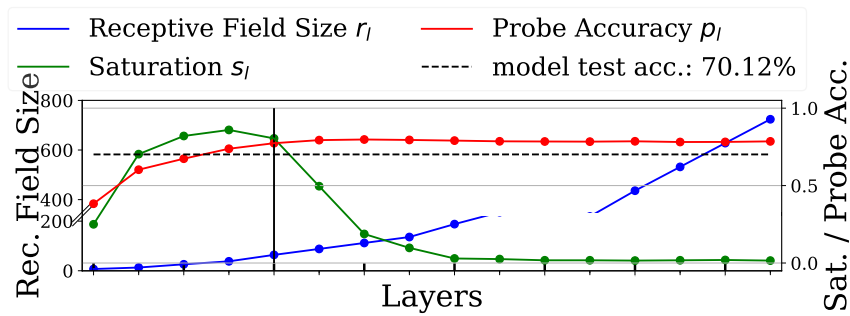


(d) VGG19—border layer at layer 8

Figure 52: By analyzing the receptive field size, the start of the unproductive convolutional layers within the network architecture can be predicted. The *border layer*, marked with a black bar, separates the part of the network that contributes to the quality of the prediction from the part that does not. The *border layer* is the first layer with $r_{l-1} > I$, where I is the maximum value of either the height or the width of the input image and r_{l-1} refers to the receptive field the layer's input is based on. Here, *Cifar10* with the resolution of 32×32 pixels is used, and therefore $I = 32$. These figures were previously published in Richter et al. (2021a).



(a) ResNet18 (no res. connections)—border layer at layer 5



(b) VGG19 (dilation=3)—border layer at layer 5

Figure 53: ResNet18 exhibits the same patterns observed in figure 52, when the skip-connections are removed (a). Increasing the receptive field by dilating convolutions (b) for VGG19 produces results consistent with figure 52. These figures were previously published in Richter et al. (2021a).

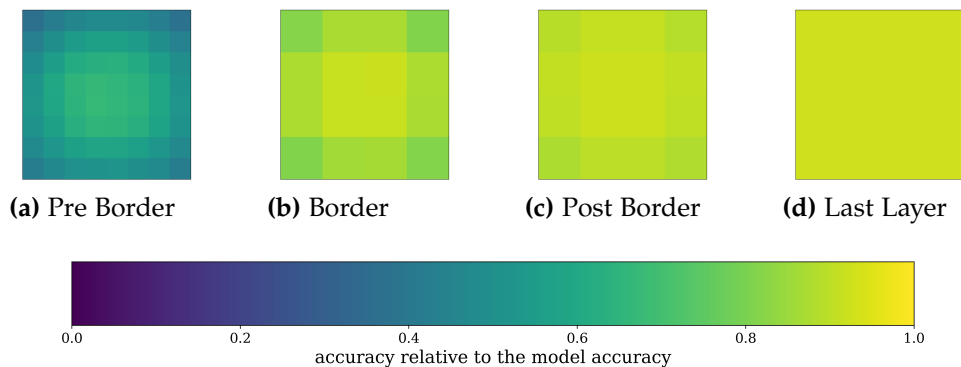


Figure 54: The heatmaps display the relative performance of probes trained on individual positions of the feature map to the performance of the model. The partial solutions contained in every feature map position improve from the image borders to the center, since the center can integrate the most information by expanding the receptive field (a). Once the border layer is reached (b), the partial solutions in the center reach the quality of the model prediction. In the following layer (c, d) the other partial solutions reach a similar quality. Effectively, the tail layers homogenize the partial solution quality of their feature maps. These figures were previously published in Richter et al. (2021a).

9.4 The Impact of Residual Connections on the Relation of Receptive Field Size and Tail Patterns

Residual connections are a popular component used in many neural architectures (He et al. (2016a); Huang et al. (2017); Szegedy et al. (2016); Tan, Le (2019)). According to He et al. (2016a), the networks with residual connections can add "deltas" to the existing representation of the data rather than transforming it entirely, thereby leading to a more distributed inference process. Explaining this property from a receptive field point of view, the residual connection itself does not expand or change the receptive field. However, after the residual connection is added to the output of a convolutional layer, information based on multiple receptive field sizes is present in the feature map. This theoretically allows features based on lower receptive field sizes to "skip" layers and to be processed later in the network, resulting in the network being able to distribute the inference on more layers.

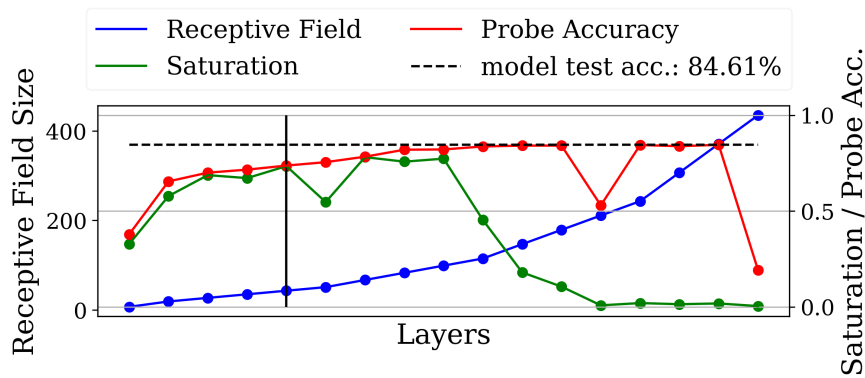
The aforementioned claims about the effects of residual connections suggest that models utilizing residual connections can utilize layers for improving the prediction after the border layer. This would also suggest that skip connections allow the neural network to integrate information in a less greedy manner, which is different from the previously tested sequential architectures in section 9.3. We investigate this by analyzing ResNet-style models in the same manner as the VGG-style models in section 9.3.

9.4.1 Methodology

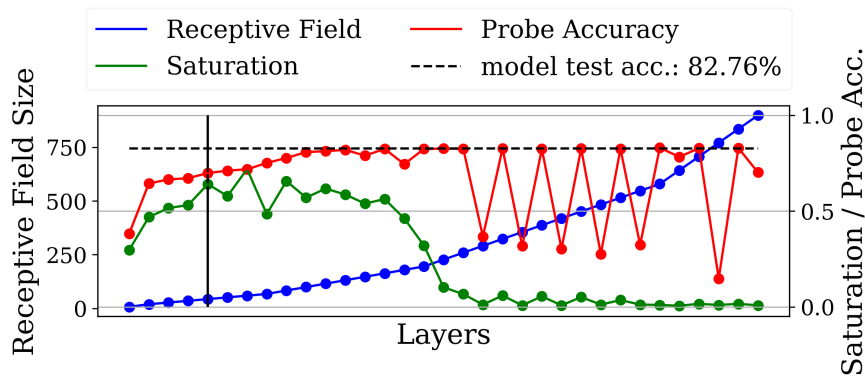
We train two variants of ResNet₁₈ and ₃₄ on the same setup, using the same analysis techniques that were previously described in section 9.3.1. We also repeat the experiments concerning the probe-heatmaps for every layer. However, we omit these when discussing the results, since they only provide further evidence for the consistency of the findings from previous observations but do not add any interesting additional insights beyond this. The heatmaps can be found in appendix G. The first version of the ResNet models is the ImageNet optimized version of the architectures as proposed by He et al. (2016a). The second variants are Cifar₁₀ optimized versions of the same architectures, also proposed by He et al. (2016a). The Cifar₁₀-variants only differ from the original architecture in the stem, which is replaced by a single 3×3 convolution with 64 filters and a stride size of 1, thereby effectively reducing the receptive field by approximately a factor of 8 in every subsequent layer. We deliberately chose the smallest version of the ResNet models

for two primary reasons. First, ResNet50 and beyond use the "bottleneck" blocks instead of the residual block, which features 1×1 convolutional layers for dimension reduction and expansion. We do not want dimension reduction and expansion inside the building blocks to influence these experiments, since we focus strictly on the residual connection. The residual connections in ResNet18 and 34 are therefore the only major deviation in the architectures that was not previously tested to be consistent with the observations in section 9.3. Secondly, the large number of layers for ResNet50, 101, 152, and 1000 makes the analysis much more computationally intensive and the visualizations harder to interpret.

9.4.2 Results



(a) ResNet18)—border layer at layer 5



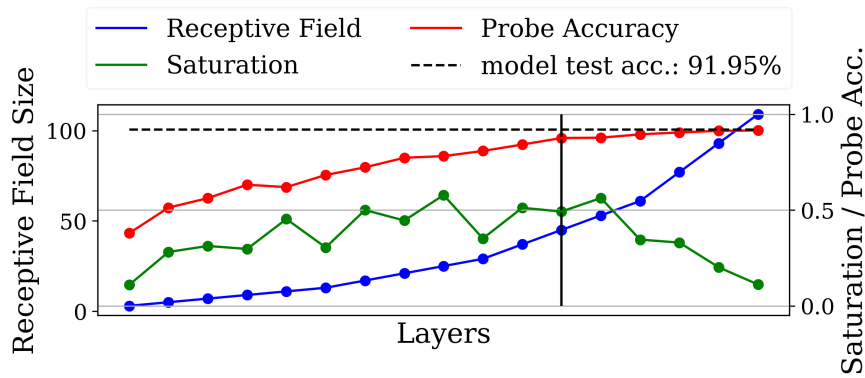
(b) ResNet34)—border layer at layer 5

Figure 55: Residual connections allow networks to utilize layers past the border layer (marked with the horizontal bar). The zig-zag-pattern and drops of probe performances observed in both networks are artifacts previously observed by Richter et al. (2021c) and Alain, Bengio (2017). These indicate that the networks attempt to "skip" the convolutional layers in the low saturated tail. It is also worth noting that these skips are only present on later layers that are part of the tail. These figures were previously published in Richter et al. (2021a)

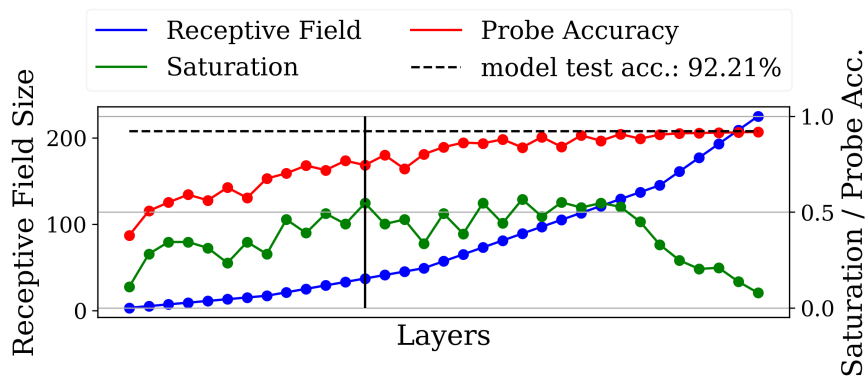
We can confirm our hypothesis by looking at the results in figure 55. Both networks improve the probe performances and stay highly saturated long after the border layer has processed the data. We can attribute this behavior to the residual connections, since we tested ResNet18 without them in the previous section (see figure 53), which resulted in behavior consistent with other sequential architectures (and lower performance).

While the presence of residual connections has a positive effect on the predictive performance of ResNet18 (84.61% accuracy with and 79.05% accuracy without residual connections), the performance still remains worse than the performance of the previously tested VGG-style models (see section 9.3). We attribute this to the lower overall receptive field size of these models. VGG19 (the VGG-model with the largest receptive field size) has a receptive field of 252 pixels in the final convolutional layer of the feature extractor. ResNet models utilize a stem that drastically increases the receptive field growth from layer to layer. For this reason, the receptive field size of the final layer of ResNet18's feature extractor has a receptive field size of 413, despite having roughly the same number of layers as VGG19. ResNet34 has a receptive field size of over 800 pixels. All in all, we deduct from this that while residual connections distribute the inference process better among the network, they do not result in tail-pattern resistant models. Thus, it is still relevant for ResNet-style models to have an input resolution close to the receptive field size of the final layer of the feature extractor to achieve good predictive performance at a high parameter-efficiency—despite apparently having a higher tolerance for mismatches than sequential architectures.

To demonstrate this, we improve the predictive performance and parameter-efficiency of the (ImageNet optimized) ResNet18 and 34 by reducing the size of the receptive field. This was implicitly done by He et al. (2016a), who proposed a Cifar10 optimized variant in addition to the ImageNet optimized versions of ResNet. The Cifar10 variant of ResNet has no max-pooling layer, and the first layer has its stride size in kernel size halved. By doing so, the receptive field of ResNet18, for example, was reduced from 413 (ImageNet optimized) to 109 pixels (Cifar10 optimized). The effect of these reductions can be seen in figure 56: the proportion of low saturated layers is drastically reduced for both models, and the inference process is now well/more evenly distributed. Thus, the performance in both cases increases compared to the ImageNet optimized models. The accuracy of ResNet18 improves from 84.61% (ImageNet optimized) to 91.95% (Cifar10 optimized) and ResNet34 improves from 82.76% to 92.21%.



(a) Cifar10 optimized ResNet18—border layer at layer 13



(b) Cifar10 optimized ResNet36—border layer at layer 13

Figure 56: The Cifar10 optimized version ResNet18 and 34 has a roughly quartered receptive field size in every layer. These networks still display similar behavior to the ImageNet optimized version when it comes to the border layer. However, the reduction in the receptive field size has further removed the low saturated tail and in both cases leads to a steady increase in performance over the entire network’s structure. These figures were previously published in Richter et al. (2021a)

9.5 Conclusion and Implications for Neural Architecture Design

9.5.1 A Priori

From the observations made in this work, we can derive some basic guidelines regarding the design of neural architectures. In section 9.3, we show that sequential models stop improving the intermediate solution qualitatively at the border layer. The border layer is the first layer to receive input from a layer with a receptive field size greater than the input resolution. Since the receptive field size is known beforehand, we can adjust the architecture before training, such that the receptive field matches the input resolution, and thereby avoiding the emergence of unproductive layers a priori. We will re-

fine and expand on this implication in chapter 10 and derive guidelines for neural architecture design in chapter 11.

9.5.2 Post Hoc

In section 9.2, we also show that the size of the discriminatory features is the underlying reason for the observed behavior. Therefore, an implicit assumption is that the size of the largest discriminatory features is (almost) as large as the image. Since the object of interest is usually depicted very prominently in classification tasks, this assumption can be regarded as true for popular classification datasets such as `Cifar10`, `MNIST`, `iNaturalist`, `ImageNet`, and `Food101` but not for classification tasks in general. If the assumption is not true for a given dataset, more layers may be part of the unproductive tail. However, since saturation can be computed live during training (see section 6.1.3), it requires little overhead to analyze the network for tail patterns during and after training, thereby allowing more informed decision-making based only on a single trained model. When a tail is detected, the architecture can be adjusted and retrained accordingly (see examples in figure 56). Alternatively, the unproductive layers can be replaced by a new classifier composed of a GAP and softmax layer, which is subsequently fine-tuned on the trained model stump. The latter is effectively equivalent to training a logistic regression probe on this layer. The effectiveness of this simple optimization technique is demonstrated in chapter 11.

10 Predicting Architectural Inefficiencies in Convolutional Neural Networks

Logistic regression probes (Alain, Bengio (2017)) and saturation (see chapter 7) were shown to be useful techniques for analyzing convolutional neural networks for inefficiencies (see chapter 9). In section 8.5, we demonstrate that tail patterns¹⁶ are an indicator for unproductive subsequences of layers in the network that do not contribute qualitatively to the inference result. In section 9.3, we also show experimentally that the receptive field expansion in sequential convolutional neural networks¹⁷ can predict these unproductive sequences of layers when analyzed together with the input resolution of the model. However, we also provide evidence that these predictions are only accurate for strictly sequential architectures (see section 9.4). This limitation is problematic, since architectures that feature multiple sequential paths from input to output like ResNet by He et al. (2016a), InceptionV3 by Szegedy et al. (2016), AmoebaNet by Real et al. (2019), and EfficientNet by Tan, Le (2019) have emerged as the norm in state of the art convolutional neural networks. Furthermore, attention-mechanisms (see section 3.3.6) such as squeeze and excitation modules (Hu et al. (2018)) are a popular add-on to these architectures to improve their predictive performance (Sandler et al. (2018), Howard et al. (2019), Tan, Le (2019)). The effects of these add-ons on the distribution of the inference process are unknown at this point.

In this section, we expand the predictability of unproductive layers to multipath-networks and to models that feature attention mechanisms. This allows us to not only predict tail-patterns for ResNet-like architectures with skip connections, but also for arbitrary feed-forward convolutional neural network architectures.

We first investigate attention mechanisms that technically induce global information into the feature map by multiplying dynamically generated weights on the output of certain layers.

We then move on to multipath architectures. We can expand the predictability of border layers to multipath architectures by using the novel concept of the minimum receptive field $r_{l,min}$ to predict the border layer b_{min} . This refined border layer provides accurate predictions for tail patterns on sequential and multipath architectures. The details on the computation of $r_{l,min}$ are

¹⁶Low saturated subsequences of neural network layers

¹⁷Sequential architectures can be described as a sequence of layers leading from the input to the output

described in section 4.4.

The experiments and results of this section were published separately in Richter et al. (2021b).

10.1 Attention Mechanisms and Their Effect on the Inference Process

Attention mechanisms such as Squeeze-and-Excitation-Modules (SE-Modules) and Spatial Attention Modules (SA-Modules) are added to existing neural architectures to boost their predictive performance (Howard et al. (2019); Hu et al. (2018); Tan, Le (2019)) by applying a multiplicative weight to the feature map. These weights are computed from the same feature map by a secondary pathway (for further details, see section 3.3.6). Attention mechanisms heavily make use of down and upsampling, thereby encoding global information in the dynamically generated attention-weights. Since we have demonstrated in chapter 9 that the receptive field and therefore the locality of information encoded in a feature map has a strong influence on how the inference is distributed among the network's layers, the use of attention mechanisms could hypothetically have an influence on the inference dynamic as well. For this reason, we investigate the influence of these add-on type attention mechanisms on the inference process of neural architectures. Our working hypothesis is that the addition of SA-Modules, SE-Modules, or both (also referred to as CBAM) to each building block has an impact on the distribution of the inference process. Since these modules deliver (theoretically) a more global context to the individual feature map positions, for our working hypothesis we expect that the tail patterns increase in size, meaning that additional layers will be part of the tail according to probes and saturation.

10.1.1 Methodology

For testing this hypothesis, the attention mechanisms SE-Modules by Hu et al. (2018), SA-Modules, and CBAM by Woo et al. (2018) are added to a ResNet18 architecture. To ensure that there are no unexpected interactions between skip connections and attention mechanisms, additional ResNet18 architectures with attention mechanisms but with disabled skip connections were trained. We refer to the ResNet-variant with disabled skip connections as ResNet18NoSkip. As in previous experiments, the training is conducted on Cifar10. For comparing the results, the logistic regression probes accuracy p_l and the saturation s_l are computed. It is important to note that additional lay-

ers added by the attention mechanism used are omitted to make the sequence of values comparable.

All models are trained on Cifar10 using the native resolution to promote the emergence of tail patterns during training.

Table 12: Hyperparameters used for training the ResNet variants with different attention mechanisms.

Parameter	Values
Input Resolution	(32×32)
Epoch	60
Batch size	128
Optimizer	ADAM
ADAM: beta1	0.9
ADAM: beta2	0.999
ADAM: epsilon	1e-8
ADAM: learning rate	0.0001

The preprocessing and data augmentation are conducted as follows: The images are channel-wise normalized with $\mu = (0.4914, 0.4822, 0.4465)$ and $\sigma = (0.2023, 0.1994, 0.2010)$, which is a procedure proposed by Krizhevsky et al. (2012). At training time, the images are first cropped randomly with a 4 pixel zero-padding on all edges. The size of the crop is 32×32 pixels. Subsequently, the crops are randomly horizontally flipped with a probability of 50%. Finally, the images are resized to the input resolution. The images of the training set are reshuffled after each epoch. The models' weights are initialized using the Kaiming-He initialization proposed by He et al. (2015), which is used for all models throughout this work.

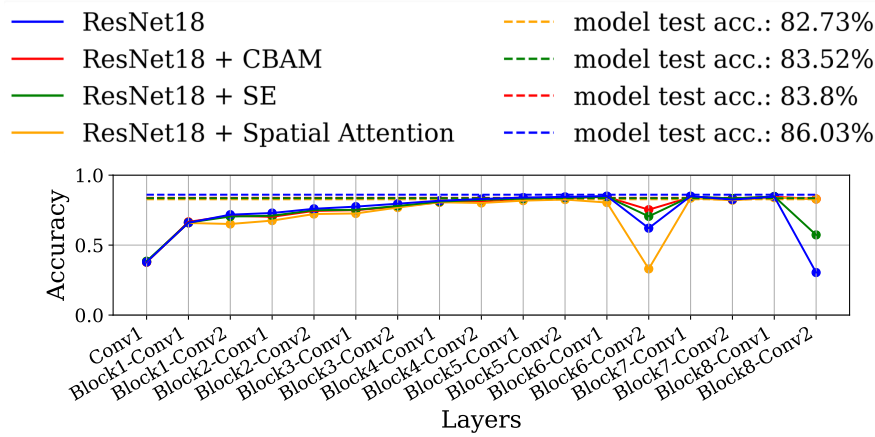
Since we are interested in seeing how the inference has shifted, the saturation and probe performances are only computed on the main path of the network and not on the added attention modules. Furthermore, the attention mechanism is multiplied with the weights and thus acts as a dynamically generated weight, which makes the probe performance less meaningful since the attention modules are not meant to directly make the data more linearly separable.

10.1.2 Results

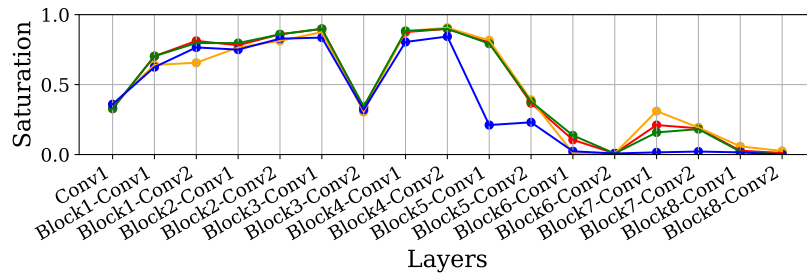
Based on the result in figure 57, we can see that the observed patterns in logistic regression probe performances and saturation are very similar for ResNet18NoSkip. Most importantly, the low saturated tail starts at the same

point for the baseline architecture and for all tested variants with attention mechanisms. This indicates that while attention modules technically enrich the data with more global information, they do not similarly expand the receptive field, which would result in an observable shift of the low saturated tail since information from a wider receptive field is available earlier. We can see that the same can be said for ResNet with enabled skip connections (see figure 57). It is interesting to see, however, that the layer skipping observable in Block6-Conv2 and Block8-Conv2 is apparently reduced by spatial attention and CBAM (which partially consists of spatial attention). However, the phenomenon cannot be directly linked to any improvements in the predictive performance. Furthermore, the skipped layers still do not contribute qualitatively to the inference process.

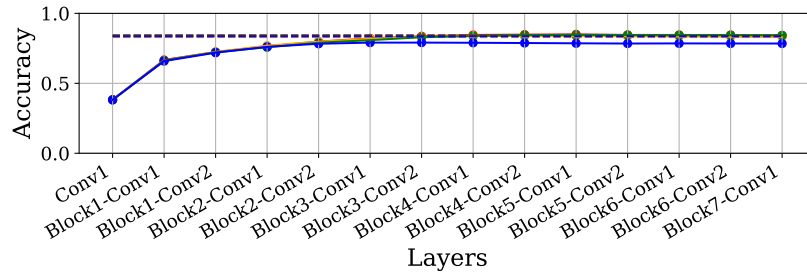
Based on these results, we can conclude that while attention can have a positive impact on the predictive performance, it does not affect where the information is processed in the network. This also indicates that the layers process features of similar sizes regardless of the use of attention mechanisms. However, it could be argued that this is somewhat expected, since attention mechanisms are meant as a dynamic weighting mechanism that does not directly contribute information to the intermediate solution. Attention mechanisms rather prioritize information by multiplying dynamically generated weights to the feature map, thereby improving the feature representation in the process.



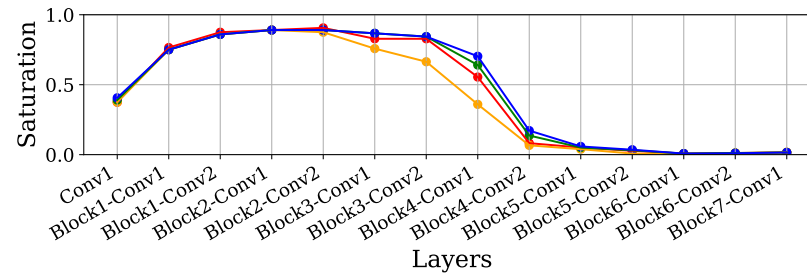
(a) Logistic Regression Probe accuracy p_l for ResNet18



(b) Saturation values s_l for ResNet18



(c) Logistic Regression Probe accuracy p_l for ResNet18 with disabled skip connections



(d) Saturation values s_l for ResNet18 with disabled skip connections

Figure 57: The attention mechanisms Squeeze-and-Excitation modules (SE), spatial attention, and CBAM added to the ResNet 18 architecture with skip connectors (a, b), and without skip connectors (c, d). Attention mechanisms neither change the logistic regression probes' accuracy nor their saturation value. This indicates that attention mechanisms aid the feature extraction for the receptive field size present, but do not change the size of features extracted. These figures were previously published in Richter et al. (2021b).

10.2 Characteristics of Multipath Architectures

In section 9.3, we demonstrated that it is possible to predict unproductive sequences of layers for sequential neural networks. However, with the definition of a receptive field used in chapter 9, we were unable to expand the predictive power of the *border layer* to non-sequential architectures, which are currently the most prevalent architecture in convolutional neural networks (Khan et al. (2020)). In this section, we expand the prediction of accurate border layers to multipath architectures by introducing the border layer b_{min} based on the minimum receptive field size $r_{l,min}$.

We have previously shown in section 9.4 that simply using the receptive field as the spatial upper bound is insufficient to predict the border layer for ResNet18 and 34. We hypothesize that this is caused by the presence of information based on multiple different receptive field sizes in the feature maps. In turn, this is caused by the multitude of pathways information can take from the input to the layer in question. In essence, the possible receptive field sizes present in a layer l exist within an interval $(r_{l,min}, r_{l,max})$, where $r_{l,min}$ is the smallest and $r_{l,max}$ the largest possible receptive field size present in the layer l . Both the upper and the lower bound of the receptive field in a given layer can be obtained by calculating the receptive field sizes of all possible sequences of layers leading from the input to the layer l and picking the minimum and maximum value respectively. For further details regarding the minimum receptive field size $r_{l,min}$ and the maximum receptive field size $r_{l,max}$ we refer to section 4.4.1, where this topic is discussed in greater detail.

We investigate whether a multipath architecture behaves similarly to ResNet in the sense that the border layer based on $r_{l,max}$ and the way it was computed in section 9.3, cannot predict the start of tail patterns. We refer to the border layer as it was computed in section 9.3 as b_{max} . Second, we are interested in learning more about the relationship of the tail pattern and $r_{l,min}$. In section 9.3, we established that the tail pattern effectively starts at the first layer, which is unable to enrich feature map positions by adding additional context. We hypothesize that this border is as accurate as b_{max} for sequential architectures, since information in any layer of a sequential architecture is always based on a single receptive field size. Hence, effectively $b_{min} = b_{max}$ in the case of sequential architectures. However, in a multipath architecture this is not the case, since $r_{l,min} = r_{l,max}$ cannot be guaranteed for all layers l . Theoretically, the feature map can still be enriched with novel information until there is no information based on a receptive field size present, which is smaller than the input resolution. However, this is only the case if $r_{l-1,min} < I$. For

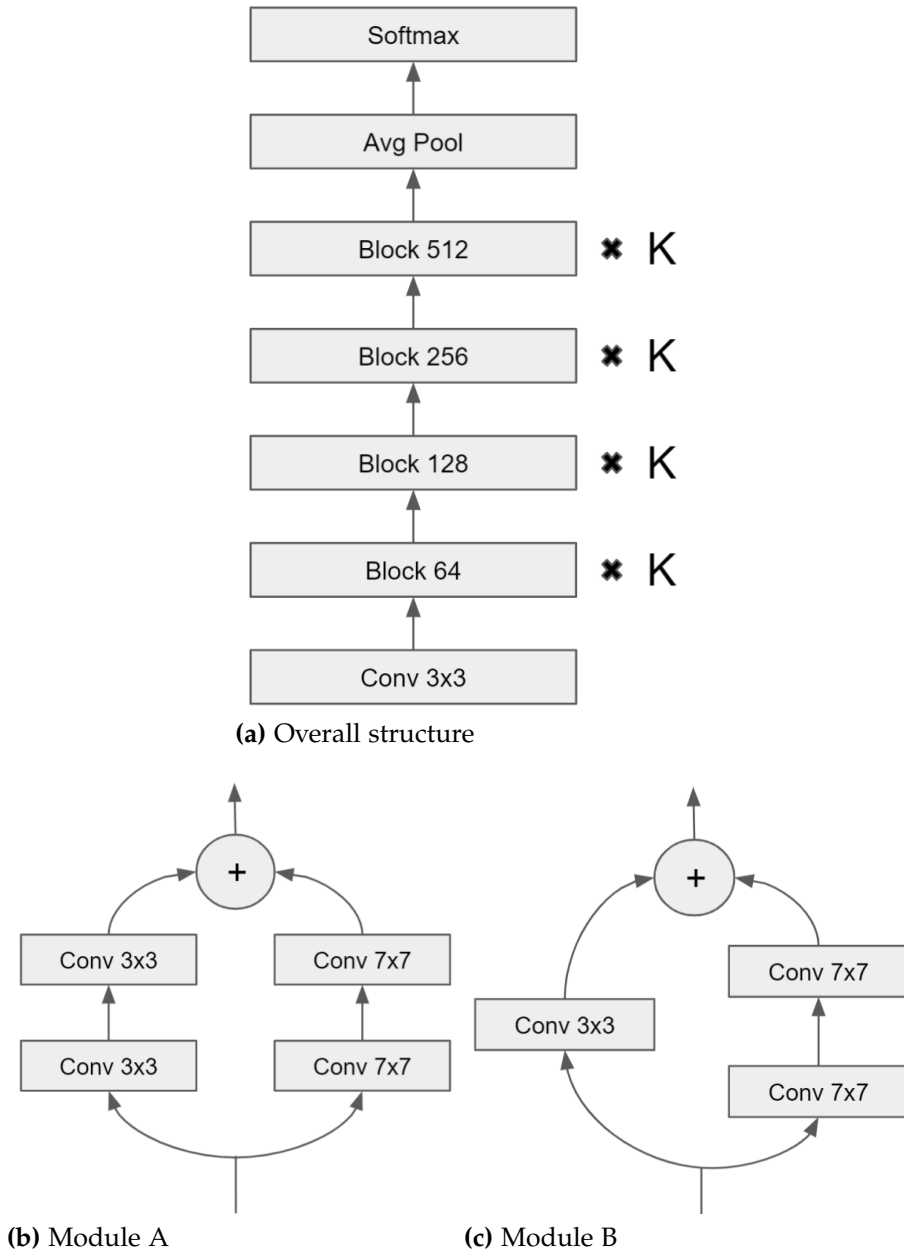


Figure 58: The architecture used for the experiments in this section. Each stage consists of k blocks and has double the filters of the previous stage. The first layer of each stage is a downsampling layer with a stride size of 2. The building block for the shallow architecture MPNet18 is depicted in (a), and the building block of the deep architecture MPNet36 is depicted in (b). These figures were previously published in Richter et al. (2021b).

this reason, we also compute the border layer b_{min} based on the lower bound of the receptive field size $r_{l,min}$. We define b_{min} as the first layer in a sequence with $r_{l-1,min} > I$ and with all layers in this sequence leading from b_{min} to the output fulfilling the same condition. Essentially, we define the border layer as the first layer in a sequence of layers leading to the output that can no longer enrich information on the input feature map by expanding the receptive field. The central hypothesis we investigate is that the tail pattern will start at b_{min} ,

since b_{min} will accurately predict the tail pattern for multipath architectures.

10.2.1 Methodology

To properly visualize and analyze the result, a simple multipath architecture is required that can serve as a "model organism" for non-sequential architectures, similar to the VGG family of networks in previous experiments. This basic architecture is depicted in figure 58. It follows the design conventions concerning downsampling and the general structure utilized in various architectures such as ResNet, VGG, Inception-ResNet, EfficientNet, and many others (He et al. (2016a); Simonyan, Zisserman (2015); Szegedy et al. (2017); Tan, Le (2019)). The networks have 4 stages consisting of building blocks with similar filter sizes (see figure 58 (a)). The first layers in each stage are downsampling layers that reduce the size of the feature maps by having a stride size of 2. We use two distinct building blocks for two architecture variants. The first, shallow variant uses the small building block depicted in figure 58 (b). It consists of a 3×3 convolutional path and a 7×7 convolutional path, which are combined again by an element-wise addition. The second, deeper variant uses the building block depicted in figure 58 (c), which features a different number of layers in each pathway and thus a larger difference between $r_{l,min}$ and $r_{l,max}$ for most layers. The convolutional layers utilize same-padding, batch normalization, and ReLU-activation functions, which can be considered standard for many common architectures. We choose element wise addition, since it does not increase the number of filters like concatenation and therefore does not require the use of 1×1 convolutions for dimension reduction, which could induce noisy artifacts in the analysis. The two distinct pathways with distinct kernel sizes make it easy to compute the path of the largest receptive field p_{max} and the path of the shortest receptive field p_{min} , which can easily be visualized without leaving out layers that are part of neither pathway, which would happen in more complex architectures such as GoogLeNet and InceptionV3 (Szegedy et al. (2015, 2016)). The shallow variant uses two of these building blocks for each of the stages and is therefore 18 layers deep with a total of 34 layers and utilizes the module depicted in figure 58(b). The deeper architecture uses double the amount of building blocks per stage and utilized the module depicted in figure 58(c). We chose the layout to observe whether changes to the number of layers and a stronger deviation in the receptive field sizes within a module affect the distribution of the inference process in unexpected ways.

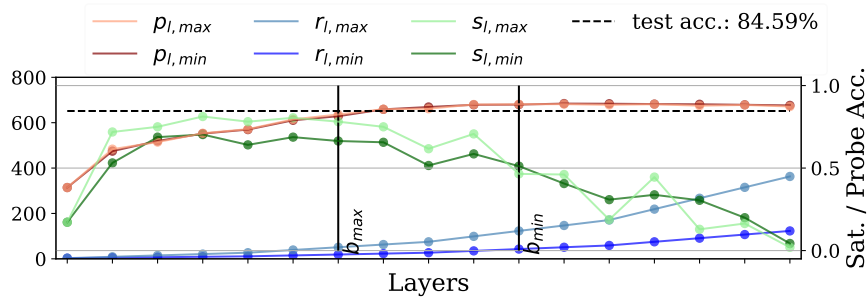
We reuse the same experimental setup from section 10.1 regarding training

and evaluation. However, we visualize the receptive field r_l , saturation s_l and probe performance p_l for the path of the largest receptive field and the path of the smallest receptive field. These paths are sequential in nature and only share the first and last layer of the model. We refer to the sequence of probe performances and layer saturation as $p_{l,min} / p_{l,max}$ and $s_{l,min} / s_{l,max}$ respectively, analog to the designation of the receptive fields in both paths through the network.

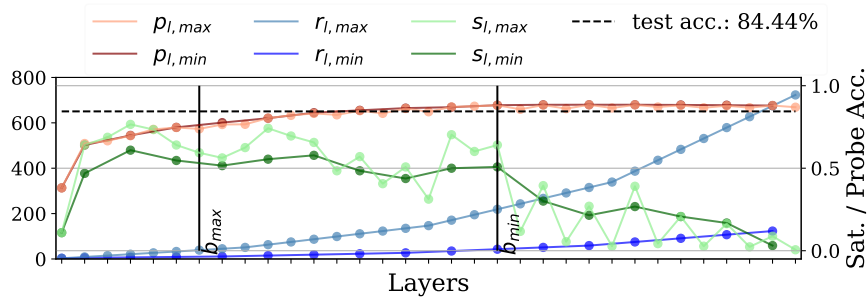
10.2.2 Results

From the results in figure 59 we can see that in both architectures the probes of both pathways perform in a very similar way. Interestingly, the border layer b_{max} based on the largest receptive field has no apparent effect in the development of the solution quality. This behavior was predicted by our working hypothesis, where we suspected that a multipath architecture would behave similarly to ResNet18 and ResNet34 in the experiments of section 9.4 regarding the border layer. However, the border layer b_{min} of the smallest receptive field exhibits the same behavior that was observed for the border layer in sequential architectures. This supports the notion that the integration of novel information is critical for the improvement of the solution. However, it also shows that layers still greedily integrate all available information as soon as possible into a single position on the feature map. Multipath architectures can better distribute the inference process, primarily because they intentionally provide information with $r_{l-1,min} < I$, which allows layers after b_{max} to expand the receptive field that some information in the feature maps is based on, even though information with a larger receptive field size may already be present in the layer.

These results are also consistent with the results of sequential neural architectures, since $b_{min} = b_{max}$ for sequential neural architectures. This is the case because information based on only one receptive field size is present in any given layer $r_{l,min} = r_{l,max} = r_l$ for sequential neural architectures.



(a) Small Multipath Architecture



(b) Large Multipath Architecture

Figure 59: The *border layer* computed from the smallest receptive field size b_{min} can predict unproductive layers for ResNet architectures. The skip connections allow information based on smaller receptive field sizes to skip layers, resulting in a later *border layer* b_{min} compared to the same network with disabled skip connections. This allows networks with skip connections to involve more layers in the inference process than would be possible compared to a simple sequential architecture with a similar layout. For an example of this, compare Fig. 53 (a) to Fig. 60 (a). These figures were previously published in Richter et al. (2021b).

10.3 b_{min} can Predict Tail Patterns in Networks With Residual Connections

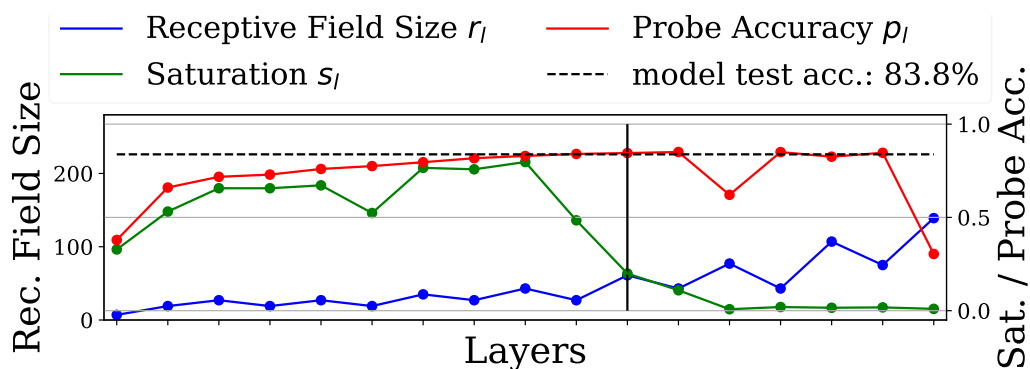
Skip connections are a special case of non-sequential architectures, since they effectively feature pathways that do not expand the size of the receptive field and often contain no layer. Multiple versions of skip connections have been proposed over the years, which generally deviate in the way the pathways are reunited and whether the skip connection itself is parameter-less (He et al. (2016a); Huang et al. (2017); Zilly et al. (2017)). In this work, we primarily focus on the residual connection proposed by He et al. (2016a), which uses an element-wise addition for reuniting the pathways and only features parameters in the skip connection if the residual block is downsampling the feature map with the first layer. We decided to use this approach as it is the most commonly used type of skip connection in popular architectures such as ResNet, and it has many derivatives including AmoebaNet, MobileNetV2, MobileNetV3, EfficientNet, and EfficientNetV2, to only name a few (Howard et al. (2019); Real et al. (2019); Sandler et al. (2018); Tan, Le (2019)).

If the receptive field expansion behavior is consistent with the behavior of multipath architectures discussed in the previous section, b_{min} should predict the unproductive tail of layers. We compute $r_{l,min}$ for the convolutional layers and compute the border layer b_{min} based on this sequence of receptive field sizes.

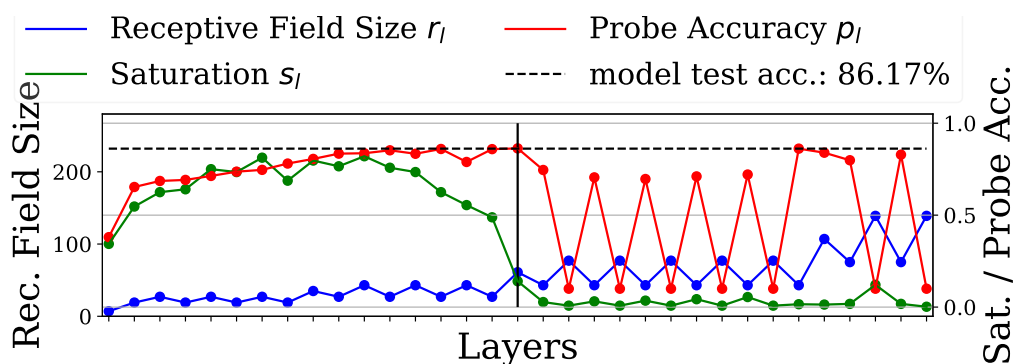
10.3.1 Methodology

For our analysis, we choose ResNet18 and ResNet34 as model organisms for their architectural simplicity. Modern networks with skip connections like EfficientNetV2 only deviate in nuances from this original architecture regarding the spacing of the downsampling layers between stages and the exact sequence of layers within each building block (Howard et al. (2019); Sandler et al. (2018); Tan, Le (2019, 2021); Zagoruyko, Komodakis (2016)). The training uses the same setup as in section 10.1 and is conducted on Cifar10. We only compute and visualize $r_{l,min}$ for each layer, since we are primarily interested in knowing whether the observations made in section 10.2 translate to networks with residual connections. Furthermore, we already know from section 9.4 that residual connections enable the network to improve the intermediate solution quality past the b_{max} and, therefore, we only need to investigate whether b_{min} predicts the start of a tail pattern.

10.3.2 Results



(a) ResNet18—*border layer* at layer 11



(b) ResNet34—*border layer* at layer 17

Figure 60: The *border layer* computed from the smallest receptive field size b_{min} can predict unproductive layers for ResNet architectures. The skip connections allow information based on smaller receptive field sizes to skip layers, resulting in a later *border layer* b_{min} compared to the same network with disabled skip connections. This allows networks with skip connections to involve more layers in the inference process than would be possible compared to a simple sequential architecture with a similar layout. For an example of this, compare Fig. 53 (a) to Fig. 60 (a). These figures were previously published in Richter et al. (2021b).

From the results in figure 60 we can clearly see that ResNet-networks behave similarly to the multipath architectures in section 10.2. In both scenarios, the qualitative improvement of the predictive performance stops when reaching the border layer b_{min} . The zig-zag-pattern of the minimal receptive field size derives from the fact that residual connections effectively allow the network to skip all layers except the stem, while still downsampling the layers at certain positions in the network. This has the result of a residual block with the same input and output feature map sizes having the same sequence of receptive field sizes. The downsampling increases the receptive field growth, which can be observed by the higher receptive field size and amplitude difference between layers in the same residual block. There are also additional anomalies present in the probe performances of ResNet18 and ResNet34 in the form of sudden drops in the predictive performance, occurring exclu-

sively after the border layer. We find that these are the result of "skipped" layers, where the model does not utilize layers and "bypasses" them using the skip connection. This phenomenon was first observed by the authors of Alain, Bengio (2017). The same effect was reproduced in this work on a multilayer perceptron in section 4.3 and on a DenseNet-architecture in appendix C. Effectively, the skipped layers learn a representation of the data that does not contribute qualitatively, but also does not deteriorate the quality of the state of the information when added back to the layers' input. Based on these experimental results, we can conclude that it is possible to predict tail patterns in multipath architectures (including networks with skip-connections) reliably using b_{min} . Concerning the results in section 10.2, these results are consistent with the results observed in sequential architectures, since $b_{min} = b_{max}$ and $r_{l,min} = r_{l,max}$ for sequential architectures.

10.4 Conclusion

In this section, we expanded the predictability of tail patterns to networks with parallel pathways in section 10.2 and residual connections in section 10.3. We can also show that attention mechanisms do not influence the way the inference is distributed (see section 10.1).

By doing so, we can effectively make the saturation and logistic regression probes obsolete for detecting unproductive sequences of layers for most common classification architectures. Implicitly, this means that training the model itself is no longer necessary to locate and resolve tail patterns, since the minimum receptive field size $r_{l,min}$ for a layer l and the input resolution I are independent of the model's state and can thus be obtained before training.

From a broader perspective, the results of this section and chapter 9 imply that the interaction of receptive field and input resolution is the dominating factor in the distribution of the inference process. While this assumption is likely to be an oversimplification, it has been demonstrated to be a useful heuristic for detecting potential inefficiencies. In the following section, we will demonstrate how this heuristic can be leveraged to improve the predictive performance and efficiency of convolutional neural network architectures.

11 Implications for Neural Architecture Design

Until now, in this work we established analysis techniques and used these to identify parameter-inefficiencies in neural architectures (see section 8). In section 8.4, we show that changes in the width of the network (number of filters in all layers) affects the average saturation of the model, which in turn is strongly correlated with the predictive performance of the model. In chapter 9 and chapter 10, we explored the causes of unproductive "tail"-layers (see section 8.5), which we can now predict precisely before training the model.

From these insights, design strategies can be derived that can improve neural architectures in an informed manner. We refer to this as neural architecture optimization, since we cannot provide a holistic design guideline that covers all aspects of creating neural architectures. Instead, we will use the collective insights gained from this work and derive strategies for optimizing existing designs of convolutional neural network architectures, with the goal of adapting these to a task that deviates from the task the architecture was originally designed for. The reasoning behind these strategies and why they are not implemented in the form of a neural architecture search algorithm or other forms of automation are elaborated in section 11.1. Next, the optimization of the network width using saturation is discussed in section 11.3. We then move on to discuss the optimization of the neural network depth and illustrate the proposed strategies with selected examples in section 11.4.

We consider this section as the culmination of this work, since the strategies proposed here are based on saturation and the insights on the distribution of the inference process in convolutional neural networks obtained from the many experiments conducted throughout this work. Furthermore, the strategies proposed here demonstrate that an informed and goal-oriented optimization of neural architectures is possible without the necessity of excessive trial-and-error based on the comparative evaluation of trained models. The results, strategies, and illustrations were also separately published in Richter et al. (2021c) and Richter et al. (2021b).

11.1 On the Goals of Architecture Optimization

Before we move on to discuss concrete strategies of optimizing convolutional neural network architectures, we have to elaborate on the goal of neural architecture optimization. In section 3.1, we have established that any convolutional neural network classifier can be reduced to 4 basic characteristics: the predictive performance, number of parameters, computations required

per forward pass, and memory footprint. An optimal model will maximize the predictive performance while minimizing the other three characteristics, which are strongly related to each other. However, in practice, achieving the highest possible predictive performance requires compromising efficiency, by for example increasing the parameters of the model and thereby reducing the computational, memory, and parameter efficiency. This circumstance has the effect that optimizing an architecture for predictive performance is not necessarily useful for the task at hand. Application scenarios may have requirements that necessitate high efficiency, for example in robotics applications where computational resources are scarce (Chakraborty et al. (2019)). Therefore, there is no algorithmic optimization strategy that can be applied with equal success in every application scenario, which is the main reason why we do not provide an algorithm or even an automated neural architecture search, as the latter commonly requires a singular, well formulated optimization goal. Instead, we provide basic guidelines and examples that demonstrate how the insights gathered in this work can be leveraged to increase the efficiency and predictive performance of models, thereby opening the possibility to a more guided and informed neural architecture design process.

11.2 Methodology

In the following sections, the proposed guidelines for optimizing the width and depth of convolutional neural network architectures will be illustrated with examples from various architectures and datasets. We will now briefly discuss the setups used to train and evaluate the models. We utilize the datasets of `Cifar10`, `ImageNet`, and `ImageWoof`, whereby the latter two are 10-class subsets of the `ImageNet` dataset. The `ImageNet` and `ImageWoof` datasets were chosen to demonstrate the proposed strategies on high-resolution datasets. Due to the number of experiments, we decide to use these subsets instead of the full `ImageNet` dataset or similar-sized datasets such as `iNaturalist`. Furthermore, it allows us to train logistic regression probes on the models, which would otherwise be impossible due to resource limitations. All models are trained using the setup outlined below.

The preprocessing and data augmentation is conducted as follows: The images are channel-wise normalized with $\mu = (0.4914, 0.4822, 0.4465)$ and $\sigma = (0.2023, 0.1994, 0.2010)$, which is a procedure proposed by Krizhevsky et al. (2012). At training time, the images are first cropped randomly with a 4 pixel zero-padding on all edges. The size of the crop is 32×32 pixels.

Table 13: Hyperparameters used for training the ResNet variants with different attention mechanisms.

Parameter	Values
Input Resolution	(32×32)
Epoch	60
Batch size	128
Optimizer	ADAM
ADAM: beta1	0.9
ADAM: beta2	0.999
ADAM: epsilon	1e-8
ADAM: learning rate	0.0001

Subsequently, the crops are randomly horizontally flipped with a probability of 50%. Finally, the images are resized to the input resolution. The images of the training set are reshuffled after each epoch.

11.3 Optimizing the Width of Neural Architectures

In section 8.4, we find that models with lower average saturation s_μ tend to perform better than architecturally similar models with higher s_μ (see figure 63). There are two main factors influencing s_μ : Problem difficulty, increasing s_μ , (see section 8.5) and the width of the network, decreasing s_μ (see section 8.4). 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.

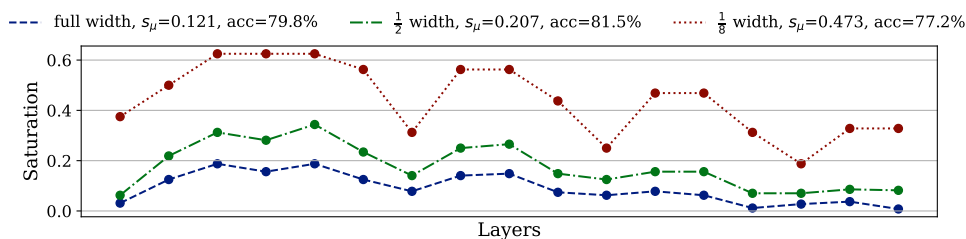
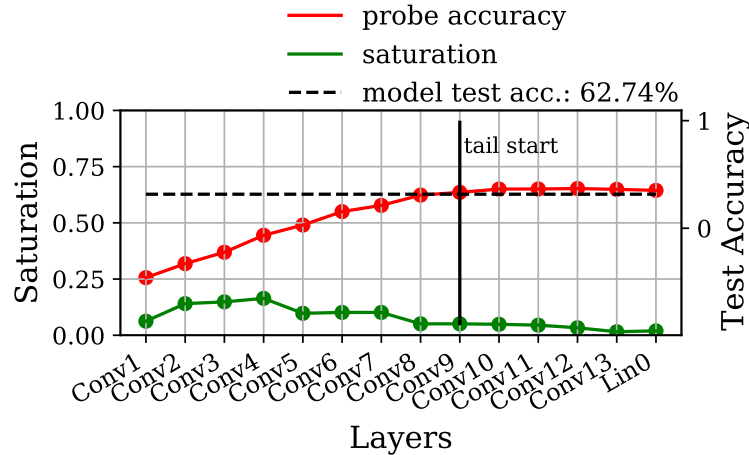


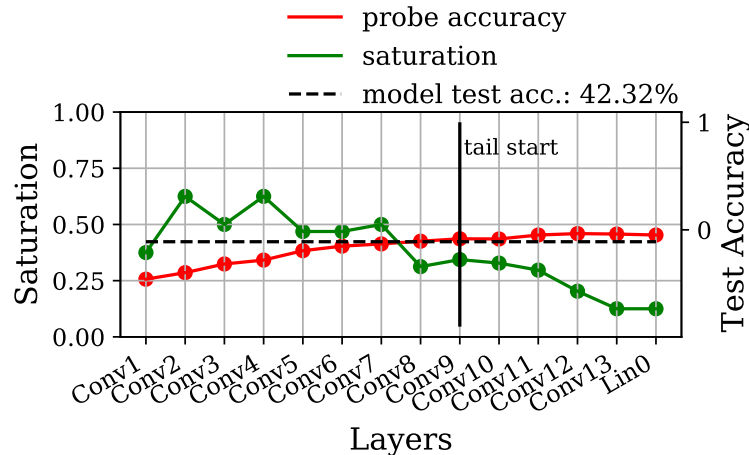
Figure 61: ResNet18 can be considered overparameterized for the ImageNette-dataset, which is apparent from the low saturation level (blue line). Halving the number of filters also halves the computational and memory footprint while slightly improving performance (green line). Reducing the width of the network too much results in a poor performing, underparameterized model (red line). The saturation of all layers (dots) is depicted in the order of the forward pass from input to output. This image was also published in Richter et al. (2021c).

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 figure 61, 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. Similar effects were also observed by Tan, Le (2019).



(a) VGG16 full width



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

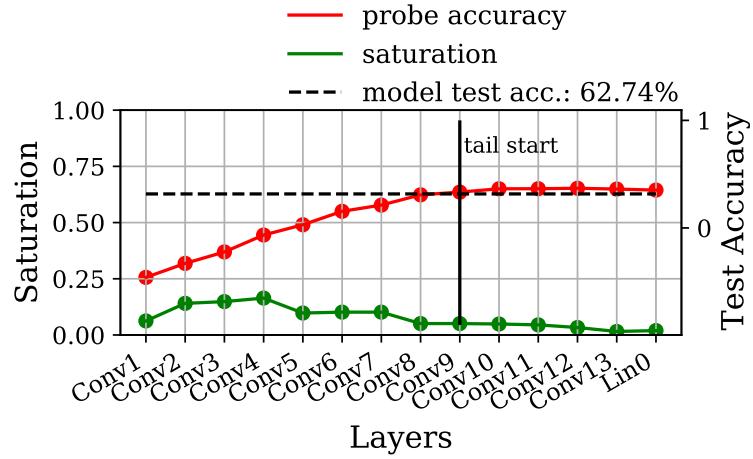
Figure 62: The width of the VGG16 model trained on the ImageWoof dataset at 32×32 pixel input resolution does not change the size of the tail, meaning that inefficiencies caused by low-saturated tails and under-parameterized layers are independent. This image was also published in Richter et al. (2021c).

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 (He et al. (2016a); Simonyan, Zisser-

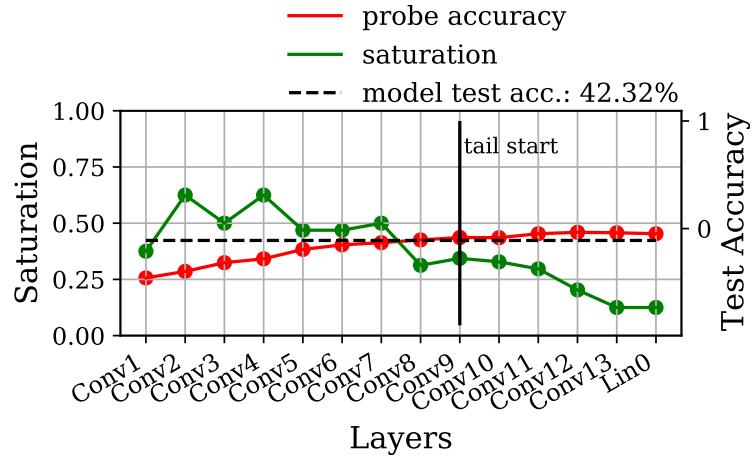
man (2015); Tan, Le (2019, 2021)). Models with s_μ below this interval provide approximately similar performance at reduced efficiency, while models with higher s_μ will degrade in predictive performance, as figure 61 demonstrates.

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 reasons. 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 such as tail-patterns. 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.

Generally, the inefficiencies of *width* (too high/low s_μ) and *depth* (tail pattern) can be considered independently, as shown in figure 63. 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 in 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 as the width can then be scaled to put s_μ into the sweet-spot of $s_\mu \in (20\%, 30\%)$.



(a) VGG16 full width



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

Figure 63: The width of the VGG16 model 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. This image was also published in Richter et al. (2021c).

11.4 Optimizing the Depth of Neural Architectures

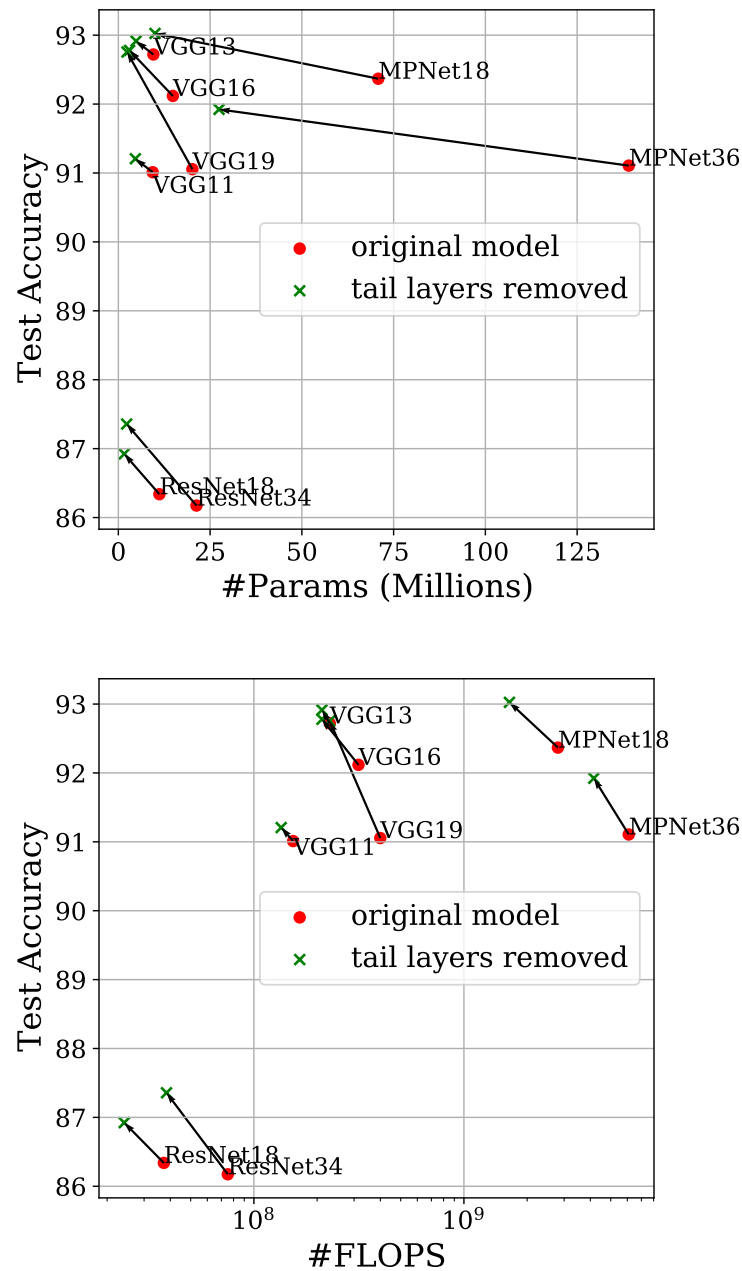


Figure 64: A simple exemplary modification based on receptive field analysis. All layers with $r_{l-1, \min} > l$ are replaced by a simple output-head, effectively removing the *border layer* and every layer after it. Removing these layers improves the efficiency by improving the performance and reducing the number of parameters and computations required. This image was also published in Richter et al. (2021b). © 2021 IEEE

By means of saturation and probe performances, this work has demonstrated experimentally that unproductive parts of the network can be identified and predicted using only the receptive field. Since we predict unproductive sequences of layers given only the architectures of receptive fields in each layer and input resolution, we can leverage this knowledge to improve

architectures. The biggest advantage of this approach is that all necessary properties are known before the start of training and thus allow for a very efficient design process, since the architecture can be optimized without requiring multiple training steps for comparative evaluation.

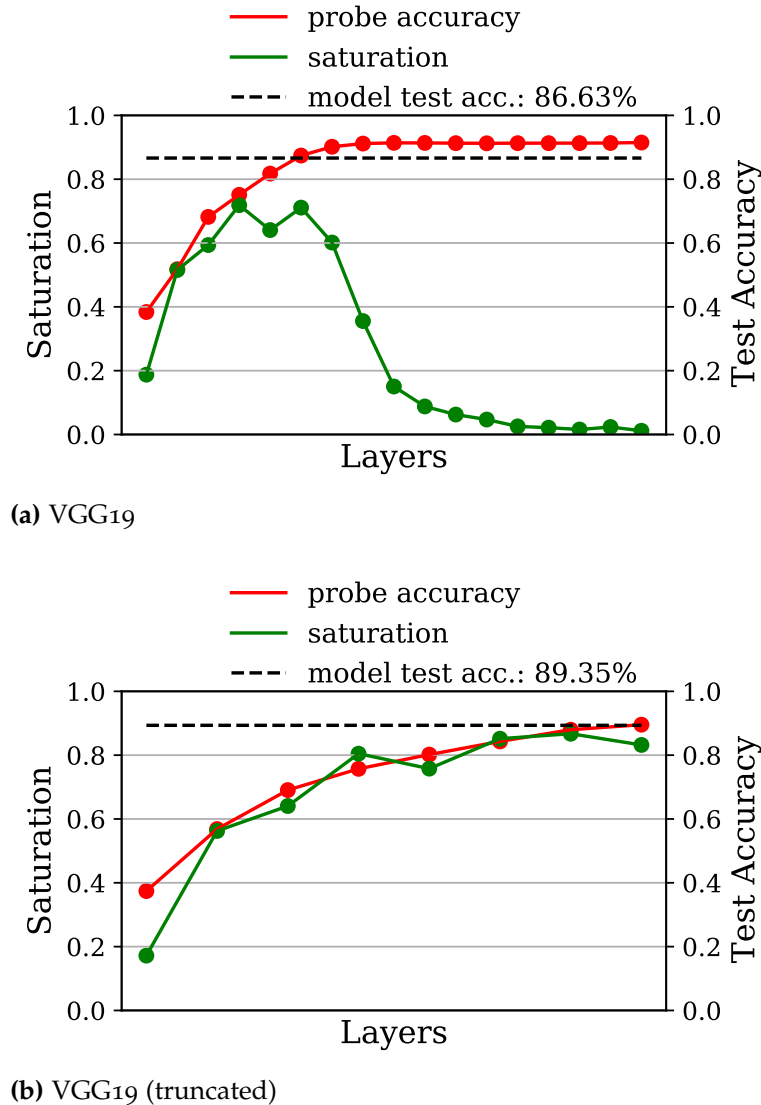


Figure 65: Removing the tail layers of VGG19 and retraining the truncated model reduced the computational and memory footprint, while slightly improving the performance. Both models are trained on Cifar10 at native resolution. This image was also published in Richter et al. (2021c).

We exemplify, by employing a simple optimization strategy on VGG11, 13, 16, 19, ResNet18, ResNet34, and the multipath architectures used in section 10.2 (abbreviated as MPNet18 and 36). The models are trained on Cifar10 using one setup from section 10.1. We then train truncated variants of these architectures, where all layers of the tail (including the border layer) are cut and replaced by a simple classifier consisting of a global average pooling

layer followed by a softmax layer. This can be seen as the simplest possible change to the architecture that removes the tail pattern. We train all models 10 times and average the test accuracy to mitigate the random fluctuations in the predictive performance. We further compute the number of parameters and FLOPS required for a forward pass of a single image. While it is unsurprising that the removal of layers leads to a reduction of parameters and computations required per forward pass, the performance of the truncated models also improves in all tested scenarios (see figure 64). Thus, all tested architectures became more computationally and parameter efficient compared to their truncated counterparts trained on the same setup.

We can further see from the example of VGG19 in figure 65 that this simple modification has also removed the tail from the saturation pattern of the model. However, the demonstrated technique can be considered crude and is not necessarily the most effective way of improving the predictive performance.

Another possibility to remove unproductive layers is to influence the growth rate of the receptive field in the neural architecture. This is done most effectively by adding, removing, or re-positioning pooling layers in the network. For instance, by removing the first two downsampling layers (colloquially referred to as "stem") in ResNet18 and ResNet34 the growth of the receptive fields in the entire network is reduced by a factor of 4. Removing the stem in ResNet34 improves the performance from 82.76% to 92.21%, while for ResNet18 the improved performance is 91.95% from the previous 84.61%. In both cases, the tail pattern is also removed in the process. Counterintuitively, the removal of the stem layers will also increase the computations required per image. In the case of ResNet34 the computation per forward pass increases from 0.76 GFLOPs to 1.16 GFLOPs, whereas for ResNet18 the computations increase from 0.04 GFLOPs to 0.56 GFLOPs. This is the case because the removal of a downsampling layer not only reduces the growth of the receptive field for consecutive layers but also increases the size of the feature maps for these layers. This in turn increases the computations required to process these layers, since more positions of the convolutional kernel need to be evaluated.

The changes to the architecture presented here for removing the tail also demonstrate that there is no singular optimal path to a well performing convolutional neural architecture, which is the main reason why we do not provide an optimization algorithm. Instead, changes to the architecture often resemble a trade off between the predictive performance and computations required. The best decision for a given scenario thus depends on the scenario

itself and the requirements for the task concerning the model. However, by analyzing the receptive field, the trade-off decision can be made more quickly and in a more informed manner.

In summary, the analysis of the receptive field allows for more discrete decision-making regarding the predictive performance without requiring model training. We explicitly do not provide an algorithm, since the decision generally involves a trade-off between predictive and computational performance, which is highly dependent on the specific problem and application.

12 Summary and Future Work

This work provides the foundation and basics for a high level guided design for convolutional neural architectures. With PHD-Lab and Delve, in our experiments we developed a basis for a reproducible workflow. By approximating the latent space of convolutional neural network layers with PCA, we establish that the space where the data is processed within a convolutional layer is a subspace of the available feature space. We further explore the properties of these subspaces and show that model capacity, problem difficulty, and the input resolution have an effect on the size of these subspaces and, by extension, on how the inference process is distributed. We also show that the subspaces are stable over many model training-procedures given similar setups. Based on these insights, we derive saturation as a metric for analyzing the information processing in convolutional neural networks. Together with logistic regression probe performances and the computation of the receptive field sizes of layers, we identify inefficiencies referred to as "tail patterns". We experimentally demonstrate – using logistic regression probes and saturation – that these tail patterns are pathological symptoms of inefficiencies caused by a mismatch between input resolution and neural architecture. We show experimentally that the absolute size of discriminatory features (measured in pixels) in the model extracts is the leading cause of these mismatches. Based on these insights, we can predict the start of these unproductive tails of layers by predicting a "border layer" based on the receptive field sizes present in every layer. In multiple experiments, we can demonstrate and expand this predictability of unproductive layers from sequential to multipath architectures, and architectures with residual connections and attention mechanisms. Finally, we show that simple optimization strategies based on the analysis of the receptive field and saturation consistently lead to improvements in predictive performance and efficiency. The proposed optimization and design of neural networks is strictly based on the analysis of the receptive field and thus does not require training, which allows for a guided design process that is less trial and error heavy. Furthermore, the experiments on saturation demonstrate that the metric delivers interesting insights into the utilization of the network's inference process on a layer by layer basis with little computational overhead.

From here on, there are two major points that need to be addressed next. The first is a shift in the investigation towards more automation. While we deliberately chose not to provide a neural architecture search algorithm in

this work, it is clear that these insights, especially those concerning the tail-pattern inefficiency, invite researchers to build some automatized system to search and/or optimize convolutional neural network architectures. Ideally, saturation and the proposed analysis of the receptive field can be used to drastically limit the search spaces of potential architecture by ruling out candidates with predictable inefficiencies.

Secondly, these analysis techniques should be expanded to other tasks. Currently, the clear focus of this work is on classification problems. However, many related fields such as cancer mammography classification (Rakhlin et al. (2018); Shen et al. (2021); Wu et al. (2020)), object detection (Bochkovskiy et al. (2020); He et al. (2017); Ren et al. (2015)), and face recognition (Jiang, Xiang (2021)) (to only name few) are using multitask systems that solve additional and more complex tasks. Some of these tasks break with assumptions that were made throughout this work. For example, object detection systems usually feature large numbers of small objects of interest, that cannot be assumed to be as large as the image, which is an assumption made for predicting the tail pattern (see chapter 9). Knowing whether – and to which degree – the observed behavior translates to these other tasks and whether other pathological patterns are observable is crucial for practical use.

References

- Abadi Martín, Agarwal Ashish, Barham Paul, Brevdo Eugene, Chen Zhifeng, Citro Craig, Corrado Greg S., Davis Andy, Dean Jeffrey, Devin Matthieu, Ghemawat Sanjay, Goodfellow Ian, Harp Andrew, Irving Geoffrey, Isard Michael, Jia Yangqing, Jozefowicz Rafal, Kaiser Lukasz, Kudlur Manjunath, Levenberg Josh, Mané Dandelion, Monga Rajat, Moore Sherry, Murray Derek, Olah Chris, Schuster Mike, Shlens Jonathon, Steiner Benoit, Sutskever Ilya, Talwar Kunal, Tucker Paul, Vanhoucke Vincent, Vasudevan Vijay, Viégas Fernanda, Vinyals Oriol, Warden Pete, Wattenberg Martin, Wicke Martin, Yu Yuan, Zheng Xiaoqiang.* TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Software available from [tensorflow.org](https://www.tensorflow.org).
- Alain Guillaume, Bengio Yoshua.* Understanding intermediate layers using linear classifier probes // 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings. 2017.
- Bishop Christopher M.* Pattern Recognition and Machine Learning (Information Science and Statistics). Berlin, Heidelberg: Springer-Verlag, 2006.
- Bochkovskiy Alexey, Wang Chien-Yao, Liao Hong-Yuan Mark.* YOLOv4: Optimal Speed and Accuracy of Object Detection. 2020.
- Bossard Lukas, Guillaumin Matthieu, Van Gool Luc.* Food-101 – Mining Discriminative Components with Random Forests // Computer Vision – ECCV 2014. Cham: Springer International Publishing, 2014. 446–461.
- Caron Mathilde, Touvron Hugo, Misra Ishan, Jégou Hervé, Mairal Julien, Bojanowski Piotr, Joulin Armand.* Emerging Properties in Self-Supervised Vision Transformers. 2021.
- Chakraborty Indranil, Roy Deboleena, Garg Isha, Ankit Aayush, Roy Kaushik.* PCA-driven Hybrid network design for enabling Intelligence at the Edge // ArXiv. 2019. [abs/1906.01493](https://arxiv.org/abs/1906.01493).
- Chattopadhyay Aditya, Sarkar Anirban, Howlader Prantik, Balasubramanian Vineeth N.* Grad-CAM++: Generalized Gradient-Based Visual Explanations for Deep Convolutional Networks // 2018 IEEE Winter Conference on Applications of Computer Vision (WACV). 2018. 839–847.
- Chollet François.* keras. 2015.

- Chollet François*. Xception: Deep Learning with Depthwise Separable Convolutions // 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017. 1800–1807.
- Chu Xiangxiang, Tian Zhi, Wang Yuqing, Zhang Bo, Ren Haibing, Wei Xiaolin, Xia Huaxia, Shen Chunhua*. Twins: Revisiting the Design of Spatial Attention in Vision Transformers. 2021.
- Cubuk Ekin D., Zoph Barret, Mané Dandelion, Vasudevan Vijay, Le Quoc V*. AutoAugment: Learning Augmentation Strategies From Data // IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019. 2019. 113–123.
- Dauphin Yann N., Vries Harm de, Chung Junyoung, Bengio Yoshua*. RMSProp and equilibrated adaptive learning rates for non-convex optimization. // CoRR. 2015. abs/1502.04390.
- Defazio Aaron, Bach Francis, Lacoste-Julien Simon*. SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives // Advances in Neural Information Processing Systems. 27. 2014.
- Deng J., Dong W., Socher R., Li L.-J., Li K., Fei-Fei L*. ImageNet: A Large-Scale Hierarchical Image Database // CVPR09. 2009.
- Dosovitskiy Alexey, Beyer Lucas, Kolesnikov Alexander, Weissenborn Dirk, Zhai Xiaohua, Unterthiner Thomas, Dehghani Mostafa, Minderer Matthias, Heigold Georg, Gelly Sylvain, Uszkoreit Jakob, Houtsby Neil*. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale // 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. 2021.
- Duchi John, Hazan Elad, Singer Yoram*. Adaptive subgradient methods for online learning and stochastic optimization // Journal of Machine Learning Research. 2011. 12, Jul. 2121–2159.
- Dumoulin Vincent, Visin Francesco*. A guide to convolution arithmetic for deep learning // CoRR. 2016. abs/1603.07285.
- Erhan Dumitru, Bengio Yoshua, Courville Aaron, Vincent Pascal*. Visualizing Higher-Layer Features of a Deep Network. VI 2009. Also presented at the ICML 2009 Workshop on Learning Feature Hierarchies, Montréal, Canada.

- Frankle Jonathan, Carbin Michael.* The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. // ICLR. 2019.
- Garg Isha, Panda Priyadarshini, Roy Kaushik.* A Low Effort Approach to Structured CNN Design Using PCA // IEEE Access. 2020. 8. 1347–1360.
- Geirhos Robert, Rubisch Patricia, Michaelis Claudio, Bethge Matthias, Wichmann Felix A., Brendel Wieland.* ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. // ICLR. 2019.
- Ghiasi Golnaz, Lee Honglak, Kudlur Manjunath, Dumoulin Vincent, Shlens Jonathon.* Exploring the structure of a real-time, arbitrary neural artistic stylization network // British Machine Vision Conference 2017, BMVC 2017, London, UK, September 4-7, 2017. 2017.
- Goyal Priya, Dollár Piotr, Girshick Ross B., Noordhuis Pieter, Wesolowski Lukasz, Kyrola Aapo, Tulloch Andrew, Jia Yangqing, He Kaiming.* Accurate, Large Mini-batch SGD: Training ImageNet in 1 Hour // CoRR. 2017. abs/1706.02677.
- Graham Ben, El-Nouby Alaaeldin, Touvron Hugo, Stock Pierre, Joulin Armand, Jégou Hervé, Douze Matthijs.* LeViT: a Vision Transformer in ConvNet’s Clothing for Faster Inference. 2021.
- He Kaiming, Gkioxari Georgia, Dollár Piotr, Girshick Ross.* Mask R-CNN // 2017 IEEE International Conference on Computer Vision (ICCV). 2017. 2980–2988.
- He Kaiming, Zhang Xiangyu, Ren Shaoqing, Sun Jian.* Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification // Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV). USA: IEEE Computer Society, 2015. 1026–1034. (ICCV ’15).
- He Kaiming, Zhang Xiangyu, Ren Shaoqing, Sun Jian.* Deep Residual Learning for Image Recognition // 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016a. 770–778.
- Identity Mappings in Deep Residual Networks. // . 9908. 10 2016b. 630–645.
- Heo Byeongho, Yun Sangdo, Han Dongyoon, Chun Sanghyuk, Choe Junsuk, Oh Seong Joon.* Rethinking Spatial Dimensions of Vision Transformers // CoRR. 2021. abs/2103.16302.

Howard Andrew, Pang Ruoming, Adam Hartwig, Le Quoc V., Sandler Mark, Chen Bo, Wang Weijun, Chen Liang-Chieh, Tan Mingxing, Chu Grace, Vasudevan Vijay, Zhu Yukun. Searching for MobileNetV3 // 2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019. 2019. 1314–1324.

Howard Andrew G., Zhu Menglong, Chen Bo, Kalenichenko Dmitry, Wang Weijun, Weyand Tobias, Andreetto Marco, Adam Hartwig. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications // CoRR. 2017. abs/1704.04861.

Hu Jie, Shen Li, Sun Gang. Squeeze-and-Excitation Networks // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). June 2018.

Huang Gao, Liu Zhuang, Van Der Maaten Laurens, Weinberger Kilian Q. Densely Connected Convolutional Networks // 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017. 2261–2269.

Huang Yanping, Cheng Youlong, Bapna Ankur, Firat Orhan, Chen Dehao, Chen Mia, Lee HyoukJoong, Ngiam Jiquan, Le Quoc V, Wu Yonghui, Chen zhifeng. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism // Advances in Neural Information Processing Systems. 32. 2019.

Iandola Forrest N., Han Song, Moskewicz Matthew W., Ashraf Khalid, Dally William J., Keutzer Kurt. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. 2016. cite arxiv:1602.07360Comment: In ICLR Format.

Ioffe Sergey, Szegedy Christian. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift // Proceedings of the 32nd International Conference on Machine Learning. 37. Lille, France: PMLR, 07–09 Jul 2015. 448–456. (Proceedings of Machine Learning Research).

Jiang Xiaolin, Xiang Yu. Face Detection Using Improved Multi-task Cascaded Convolutional Networks // Security, Privacy, and Anonymity in Computation, Communication, and Storage. Cham: Springer International Publishing, 2021. 321–333.

Jolliffe I.T. Principal Component Analysis. 1986.

Kanopoulos Nick, Vasanthavada Nagesh, Baker Robert L. Design of an image edge detection filter using the Sobel operator // IEEE Journal of solid-state circuits. 1988. 23, 2. 358–367.

Keskar Nitish Shirish, Mudigere Dheevatsa, Nocedal Jorge, Smelyanskiy Mikhail, Tang Ping Tak Peter. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima // 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. 2017a.

On large-batch training for deep learning: Generalization gap and sharp minima. // . 2017b. 5th International Conference on Learning Representations, ICLR 2017 ; Conference date: 24-04-2017 Through 26-04-2017.

Khan Asifullah, Sohail Anabia, Zahoora Umme, Saeed Aqsa. A Survey of the Recent Architectures of Deep Convolutional Neural Networks // Artificial Intelligence Review. 12 2020. 53.

Kingma Diederik P., Ba Jimmy. Adam: A Method for Stochastic Optimization. 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Klambauer Günter, Unterthiner Thomas, Mayr Andreas, Hochreiter Sepp. Self-Normalizing Neural Networks // Advances in Neural Information Processing Systems. 30. 2017.

Siamese Neural Networks for One-shot Image Recognition. // . 2015.

Why Deep Learning Methods Use KL Divergence Instead of Least Squares: A Possible Pedagogical Explanation. // . 2017.

CIFAR-10 (Canadian Institute for Advanced Research). // . 2015.

Krizhevsky Alex, Sutskever Ilya, Hinton Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks // Advances in Neural Information Processing Systems 25. 2012. 1097–1105.

MNIST handwritten digit database. // . 2010.

Lecun Y., Bottou L., Bengio Y., Haffner P. Gradient-based learning applied to document recognition // Proceedings of the IEEE. 1998. 86, 11. 2278–2324.

- Li Chunyuan, Farkhoor Heerad, Liu Rosanne, Yosinski Jason.* Measuring the Intrinsic Dimension of Objective Landscapes // 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. 2018a.
- Li Hao, Xu Zheng, Taylor Gavin, Studer Christoph, Goldstein Tom.* Visualizing the Loss Landscape of Neural Nets // Advances in Neural Information Processing Systems. 31. 2018b.
- Li Hao, Xu Zheng, Taylor Gavin, Studer Christoph, Goldstein Tom.* Visualizing the Loss Landscape of Neural Nets // Advances in Neural Information Processing Systems. 31. 2018c.
- Li Mingchen, Soltanolkotabi Mahdi, Oymak Samet.* Gradient Descent with Early Stopping is Provably Robust to Label Noise for Overparameterized Neural Networks // Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics. 108. 26–28 Aug 2020. 4313–4324. (Proceedings of Machine Learning Research).
- Lin Min, Chen Qiang, Yan Shuicheng.* Network In Network // 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings. 2014.
- Lin Tsung-Yi, Dollár Piotr, Girshick Ross, He Kaiming, Hariharan Bharath, Belongie Serge.* Feature Pyramid Networks for Object Detection // 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017. 936–944.
- Liu Liyuan, Jiang Haoming, He Pengcheng, Chen Weizhu, Liu Xiaodong, Gao Jianfeng, Han Jiawei.* On the Variance of the Adaptive Learning Rate and Beyond // Proceedings of the Eighth International Conference on Learning Representations (ICLR 2020). April 2020.
- Montavon Grégoire, Müller Klaus-Robert, Braun Mikio.* Layer-wise analysis of deep networks with Gaussian kernels // Advances in Neural Information Processing Systems. 23. 2010.
- Novak Roman, Bahri Yasaman, Abolafia Daniel A., Pennington Jeffrey, Sohl-Dickstein Jascha.* Sensitivity and Generalization in Neural Networks: an Empirical Study // 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. 2018.

- Paszke Adam, Gross Sam, Massa Francisco, Lerer Adam, Bradbury James, Chanan Gregory, Killeen Trevor, Lin Zeming, Gimelshein Natalia, Antiga Luca, Desmaison Alban, Kopf Andreas, Yang Edward, DeVito Zachary, Raison Martin, Tejani Alykhan, Chilamkurthy Sasank, Steiner Benoit, Fang Lu, Bai Junjie, Chintala Soumith.* PyTorch: An Imperative Style, High-Performance Deep Learning Library // Advances in Neural Information Processing Systems 32. 2019. 8024–8035.
- Pham Hieu, Dai Zihang, Xie Qizhe, Le Quoc V.* Meta Pseudo Labels // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). June 2021. 11557–11568.
- Raghu Maithra, Gilmer Justin, Yosinski Jason, Sohl-Dickstein Jascha.* SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability // Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA. 2017. 6076–6085.
- Deep Convolutional Neural Networks for Breast Cancer Histology Image Analysis. // . 06 2018. 737–744.
- Ramachandran Prajit, Parmar Niki, Vaswani Ashish, Bello Irwan, Levskaya Anselm, Shlens Jon.* Stand-Alone Self-Attention in Vision Models // Advances in Neural Information Processing Systems. 32. 2019.
- Ramachandran Prajit, Zoph Barret, Le Quoc V.* Searching for Activation Functions // 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings. 2018.
- Real Esteban, Aggarwal Alok, Huang Yanping, Le Quoc V.* Regularized Evolution for Image Classifier Architecture Search // Proceedings of the AAAI Conference on Artificial Intelligence. Jul. 2019. 33, 01. 4780–4789.
- Redmon Joseph, Divvala Santosh Kumar, Girshick Ross B., Farhadi Ali.* You Only Look Once: Unified, Real-Time Object Detection // 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. 2016. 779–788.
- Redmon Joseph, Farhadi Ali.* YOLO9000: Better, Faster, Stronger // 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017. 6517–6525.

- Redmon Joseph, Farhadi Ali.* YOLOv3: An Incremental Improvement. 2018.
- Ren Shaoqing, He Kaiming, Girshick Ross, Sun Jian.* Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks // Advances in Neural Information Processing Systems. 28. 2015.
- Ribeiro Marco Tulio, Singh Sameer, Guestrin Carlos.* "Why Should I Trust You?": Explaining the Predictions of Any Classifier // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York, NY, USA: Association for Computing Machinery, 2016. 1135–1144. (KDD '16).
- Richter Mats L., Byttner Wolf, Krumnack Ulf, Wiedenroth Anna, Schallner Ludwig, Shenk Justin.* (Input) Size Matters for CNN Classifiers // Artificial Neural Networks and Machine Learning – ICANN 2021. Cham: Springer International Publishing, 2021a. 133–144.
- Richter Mats L., Malihi Leila, Windler Anne-Kathrin Patricia, Krumnack Ulf.* Exploring the Properties and Evolution of Neural Network Eigenspaces during Training // "The 12th Iranian and the second International Conference on Machine Vision and Image Processing MVIP (in print)". 2022.
- Richter Mats L., Schöning Julius, Krumnack Ulf.* Should You Go Deeper? Optimizing Convolutional Neural Network Architectures without Training by Receptive Field Analysis (in print) // 20th IEEE International Conference on Machine Learning and Applications, ICMLA 2021, Miami, FL, USA, December 13-16, 2021. 2021b.
- Richter Mats L., Shenk Justin, Byttner Wolf, Arpteg Anders, Huss Mikael.* Feature Space Saturation during Training (in print) // "32st British Machine Vision Conference 2021, BMVC 2021, Virtual Event, UK, November 22-25, 2021". 2021c.
- Ruder Sebastian.* An overview of gradient descent optimization algorithms // CoRR. 2016. abs/1609.04747.
- Rumelhart David E., Hinton Geoffrey E., Williams Ronald J.* Learning Representations by Back-propagating Errors // Nature. 1986. 323, 6088. 533–536.
- Sandler Mark, Howard Andrew, Zhu Menglong, Zhmoginov Andrey, Chen Liang-Chieh.* MobileNetV2: Inverted Residuals and Linear Bottlenecks // 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2018. 4510–4520.

- Scheidegger Florian, Istrate Roxana, Mariani Giovanni, Benini Luca, Bekas Costas, Malossi A. Cristiano I.* Efficient image dataset classification difficulty estimation for predicting deep-learning accuracy // *Vis. Comput.* 2021. 37, 6. 1593–1610.
- Selvaraju Ramprasaath R., Cogswell Michael, Das Abhishek, Vedantam Ramakrishna, Parikh Devi, Batra Dhruv.* Grad-CAM: Visual Explanations From Deep Networks via Gradient-Based Localization // *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Oct 2017.
- Shen Yiqiu, Wu Nan, Phang Jason, Park Jungkyu, Liu Kangning, Tyagi Sudarshini, Heacock Laura, Kim S. Gene, Moy Linda, Cho Kyunghyun, Geras Krzysztof J.* An interpretable classifier for high-resolution breast cancer screening images utilizing weakly supervised localization // *Medical Image Anal.* 2021. 68. 101908.
- Shenk Justin.* Spectral Decomposition for Live Guidance of Neural Network Architecture Design. Osnabrück, Germany, 2018.
- Shenk Justin, Richter Mats L., Arpteg Anders, Huss Mikael.* Spectral Analysis of Latent Representations // *CoRR*. 2019. abs/1907.08589.
- Shenk Justin, Richter Mats L., Byttner Wolf.* Delve: Neural Network Feature Variance Analysis (under review) // *Journal of Open Source Software*. 2021.
- Simonyan Karen, Zisserman Andrew.* Very Deep Convolutional Networks for Large-Scale Image Recognition. 2015.
- Smith Leslie N., Topin Nicholay.* Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates // *CoRR*. 2017a. abs/1708.07120.
- Smith Leslie N., Topin Nicholay.* Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates // *CoRR*. 2017b. abs/1708.07120.
- Szegedy Christian, Ioffe Sergey, Vanhoucke Vincent, Alemi Alexander A.* Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning // *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. 2017. 4278–4284. (AAAI'17).

- Szegedy Christian, Liu Wei, Jia Yangqing, Sermanet Pierre, Reed Scott, Anguelov Dragomir, Erhan Dumitru, Vanhoucke Vincent, Rabinovich Andrew.* Going deeper with convolutions // 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2015. 1–9.
- Szegedy Christian, Vanhoucke Vincent, Ioffe Sergey, Shlens Jon, Wojna Zbigniew.* Rethinking the Inception Architecture for Computer Vision // 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. 2818–2826.
- Tan Mingxing, Le Quoc V.* EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks // Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA. 97. 2019. 6105–6114. (Proceedings of Machine Learning Research).
- Tan Mingxing, Le Quoc V.* EfficientNetV2: Smaller Models and Faster Training // Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event. 139. 2021. 10096–10106. (Proceedings of Machine Learning Research).
- Tan Mingxing, Pang Ruoming, Le Quoc V.* EfficientDet: Scalable and Efficient Object Detection // 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020. 2020. 10778–10787.
- Touvron Hugo, Cord Matthieu, Douze Matthijs, Massa Francisco, Sablayrolles Alexandre, Jégou Hervé.* Training data-efficient image transformers & distillation through attention // Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event. 139. 2021a. 10347–10357. (Proceedings of Machine Learning Research).
- Touvron Hugo, Cord Matthieu, Sablayrolles Alexandre, Synnaeve Gabriel, Jégou Hervé.* Going deeper with Image Transformers // CoRR. 2021b. abs/2103.17239.
- Vaswani Ashish, Shazeer Noam, Parmar Niki, Uszkoreit Jakob, Jones Llion, Gomez Aidan N, Kaiser Łukasz, Polosukhin Illia.* Attention is All you Need // Advances in Neural Information Processing Systems. 30. 2017.

- Wang Huan, Keskar Nitish Shirish, Xiong Caiming, Socher Richard.* Identifying Generalization Properties in Neural Networks // CoRR. 2018. abs/1809.07402.
- Woo Sanghyun, Park Jongchan, Lee Joon-Young, Kweon In So.* CBAM: Convolutional Block Attention Module // Proceedings of the European Conference on Computer Vision (ECCV). September 2018.
- Wu Nan, Phang Jason, Park Jungkyu, Shen Yiqiu, Huang Zhe, Zorin Masha, Jastrzebski Stanislaw, Févry Thibault, Katsnelson Joe, Kim Eric, Wolfson Stacey, Parikh Ujas, Gaddam Sushma, Lin Leng Leng Young, Ho Kara, Weinstein Joshua D., Reig Beatriu, Gao Yiming, Toth Hildegard, Pysarenko Kristine, Lewin Alana, Lee Jiyon, Airola Krystal, Mema Eralda, Chung Stephanie, Hwang Esther, Samreen Naziya, Kim S. Gene, Heacock Laura, Moy Linda, Cho Kyunghyun, Geras Krzysztof J.* Deep Neural Networks Improve Radiologists' Performance in Breast Cancer Screening // IEEE Trans. Medical Imaging. 2020. 39, 4. 1184–1194.
- Xie Saining, Girshick Ross, Dollár Piotr, Tu Zhuowen, He Kaiming.* Aggregated Residual Transformations for Deep Neural Networks // 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017. 5987–5995.
- Yosinski Jason, Clune Jeff, Bengio Yoshua, Lipson Hod.* How Transferable Are Features in Deep Neural Networks? // Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2. Cambridge, MA, USA: MIT Press, 2014. 3320–3328. (NIPS'14).
- Zagoruyko Sergey, Komodakis Nikos.* Wide Residual Networks // Proceedings of the British Machine Vision Conference (BMVC). September 2016. 87.1–87.12.
- Zeiler Matthew D.* ADADELTA: An Adaptive Learning Rate Method // CoRR. 2012. abs/1212.5701.
- Zeiler Matthew D., Fergus Rob.* Visualizing and Understanding Convolutional Networks // Computer Vision – ECCV 2014. Cham: Springer International Publishing, 2014a. 818–833.
- Zeiler Matthew D., Fergus Rob.* Visualizing and Understanding Convolutional Networks // Computer Vision – ECCV 2014. Cham: Springer International Publishing, 2014b. 818–833.

Zhang Chiyuan, Bengio Samy, Hardt Moritz, Recht Benjamin, Vinyals Oriol. Understanding deep learning requires rethinking generalization // 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. 2017.

Zhou Bolei, Khosla Aditya, Lapedriza Agata, Oliva Aude, Torralba Antonio. Learning Deep Features for Discriminative Localization // 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. 2921–2929.

Learning Deep Features for Discriminative Localization. // . 12 2015.

Zhou Daquan, Kang Bingyi, Jin Xiaojie, Yang Linjie, Lian Xiaochen, Hou Qibin, Feng Jiashi. DeepViT: Towards Deeper Vision Transformer // CoRR. 2021. abs/2103.11886.

Zilly Julian Georg, Srivastava Rupesh Kumar, Koutník Jan, Schmidhuber Jürgen. Recurrent Highway Networks // Proceedings of the 34th International Conference on Machine Learning. 70. 06–11 Aug 2017. 4189–4198. (Proceedings of Machine Learning Research).

Learning Transferable Architectures for Scalable Image Recognition. // . 06 2018. 8697–8710.

A Full t -Test Tables of VGG11, VGG13, VGG16 and ResNet18

These are the full experimental results of the experiments in section 7.4 The result of this section were published previously as part of the supplementary material of Richter et al. (2021c).

Table 14: Sum of Projections in VGG11 ($n=15$). $\mu \neq 0$ ($p=.99$) in **bold**.

Explained σ	Mean difference	sample σ	t-stat	p-value
0.9999	-0.0001	0.0007	-0.714	0.487
0.9998	-0.0005	0.0008	-2.44	0.029
0.9997	-0.0007	0.0012	-2.07	0.058
0.9996	-0.0005	0.0018	-1.1	0.292
0.9995	-0.0010	0.0018	-2.16	0.049
0.9994	-0.0008	0.0016	-1.8	0.094
0.9993	-0.0010	0.0015	-2.72	0.017
0.9992	-0.0013	0.0014	-3.65	0.003
0.9991	-0.0011	0.0018	-2.51	0.025
0.999	-0.0015	0.0021	-2.74	0.016
0.998	-0.0019	0.0029	-2.49	0.026
0.997	-0.0013	0.0034	-1.46	0.166
0.996	-0.0009	0.0039	-0.888	0.390
0.995	0.0005	0.0037	0.537	0.600
0.994	0.0022	0.0038	2.2	0.045
0.993	0.0056	0.0071	3.03	0.009
0.992	0.0114	0.0110	4.03	0.001
0.99	0.0260	0.0179	5.61	0.000
0.98	0.1073	0.0253	16.4	0.000
0.97	0.2824	0.0954	11.5	0.000
0.96	0.4356	0.0767	22	0.000
0.95	0.5118	0.0616	32.2	0.000
0.94	0.5658	0.0568	38.6	0.000
0.93	0.6385	0.0476	52	0.000
0.92	0.7070	0.0510	53.7	0.000
0.91	0.7574	0.0240	122	0.000
0.9	0.7727	0.0090	333	0.000

Table 15: Sum of projections in VGG13 (n=26). $\mu \neq 0$ ($\alpha = 0.01$) in **bold**.

Explained σ	Mean difference	sample σ	t-stat	p-value
0.9999	-0.0004	0.0008	-2.42	0.023
0.9998	-0.0005	0.0009	-2.81	0.010
0.9997	-0.0010	0.0010	-5.26	0.000
0.9996	-0.0009	0.0010	-4.92	0.000
0.9995	-0.0011	0.0010	-5.46	0.000
0.9994	-0.0012	0.0012	-4.91	0.000
0.9993	-0.0012	0.0012	-4.83	0.000
0.9992	-0.0013	0.0013	-5.17	0.000
0.9991	-0.0016	0.0015	-5.48	0.000
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	-0.910	0.371
0.994	0.0037	0.0043	4.45	0.000
0.992	0.0096	0.0062	7.91	0.000
0.99	0.0178	0.0136	6.68	0.000
0.98	0.1123	0.0377	15.2	0.000
0.97	0.2254	0.0578	19.9	0.000
0.96	0.4803	0.1022	24.0	0.000
0.95	0.7026	0.0368	97.3	0.000
0.94	0.7536	0.0227	169	0.000
0.93	0.7654	0.0202	193	0.000
0.92	0.7785	0.0164	242	0.000
0.91	0.7867	0.0143	280	0.000
0.9	0.7929	0.0117	345	0.000

Table 16: Sum of projections in VGG19 (n=40). $\mu \neq 0$ (p=.99) in **bold**.

Explained σ	μ_{diff}	σ_{sample}	t-stat	p-value	μ_{Sat}	σ_{Sat}	$\mu(\sum dim E_l^k)$
0.9999	-0.0003	0.0008	-2.65	0.011	60.0	0.6	2613 \pm 102
0.9998	-0.0006	0.0011	-3.31	0.002	54.5	0.6	2268 \pm 97
0.9997	-0.0006	0.0014	-2.82	0.008	51.2	0.7	2071 \pm 93
0.9996	-0.0003	0.0016	-1.28	0.208	48.8	0.6	1938 \pm 88
0.9995	-0.0001	0.0017	-0.352	0.727	47.1	0.7	1841 \pm 86
0.9994	0.0007	0.0019	2.18	0.035	45.6	0.7	1766 \pm 84
0.9993	0.0009	0.0022	2.62	0.012	44.5	0.7	1705 \pm 83
0.9992	0.0012	0.0031	2.42	0.020	43.4	0.7	1653 \pm 80
0.9991	0.0016	0.0032	3.14	0.003	42.5	0.7	1608 \pm 79
0.998	0.0107	0.0148	4.57	0.000	36.0	0.7	1318 \pm 73
0.996	0.0771	0.0585	8.33	0.000	30.0	0.7	1074 \pm 67
0.994	0.1873	0.0812	14.6	0.000	26.3	0.7	934 \pm 62
0.992	0.2754	0.0822	21.2	0.000	23.7	0.6	837 \pm 58
0.99	0.3643	0.0900	25.6	0.000	21.8	0.6	765 \pm 54
0.98	0.6176	0.0413	94.6	0.000	16.1	0.5	556 \pm 41
0.97	0.6559	0.0386	107	0.000	13.1	0.4	451 \pm 32
0.96	0.7008	0.0384	115	0.000	11.2	0.3	385 \pm 27
0.95	0.7351	0.0337	138	0.000	9.8	0.3	339 \pm 24
0.94	0.7550	0.0265	180	0.000	8.8	0.2	303 \pm 21
0.93	0.7639	0.0231	209	0.000	7.9	0.2	275 \pm 19
0.92	0.7727	0.0167	293	0.000	7.2	0.2	252 \pm 17
0.91	0.7775	0.0143	344	0.000	6.6	0.2	233 \pm 16
0.9	0.7796	0.0127	387	0.000	6.1	0.2	215 \pm 15

B On the Evolution of Saturation Patterns in Fully Connected and Convolutional Neural Networks During Training

This section can be seen as supplementary to the results in chapter 8 and was initially omitted from this chapter, since its insights do not directly affect the primary results of this work. However, since the experiments of this section demonstrate interesting behavior in overfitting scenarios, we find that these results are worth to be included.

The tail pattern we discussed earlier in this work allows for the identification of inefficiencies caused by mismatches between the neural architecture and the input resolution. The analysis of saturation patterns is generally done after training has concluded. However, since saturation can be computed live during training with little overhead (see section 6.1.3) it might be interesting to see how these patterns emerge during the training process. Furthermore, we are interested in how saturation reflects overfitting, when a dense neural network is strongly overtrained on a given dataset. The result of this section were published previously as part of the supplementary material of Shenk et al. (2019).

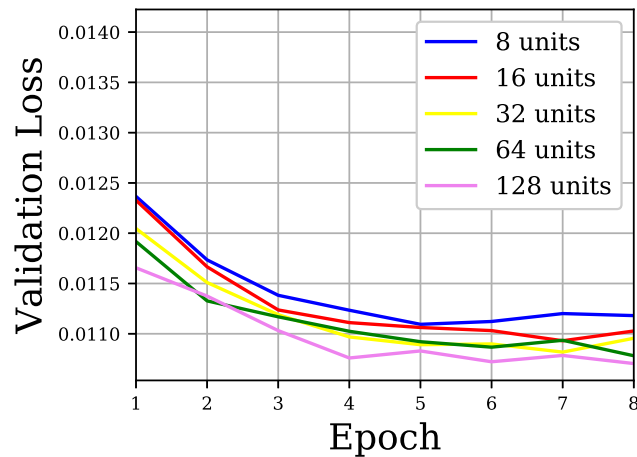
B.1 Methodology

We first investigate how the saturation level evolves in a single layer for different feature space dimensionalities. We train a 3 layer fully connected neural network, the first layer has 256 units, the second layer has 8, 16, 32, 64 and 128 units (a different number for each run). We train these networks using the ADAM optimizer and a batch size of 128. The training is conducted twice. Once using 8 epochs, which is enough for all models to converge, the second experiment is run for 20 epochs, which results in the loss increasing again due to overfitting. We compute the saturation of the hidden-layer of the 3-layer architecture after each epoch to observe the evolution of the model. We further compute the loss of the architecture to observe a potential relationship between loss and saturation convergence.

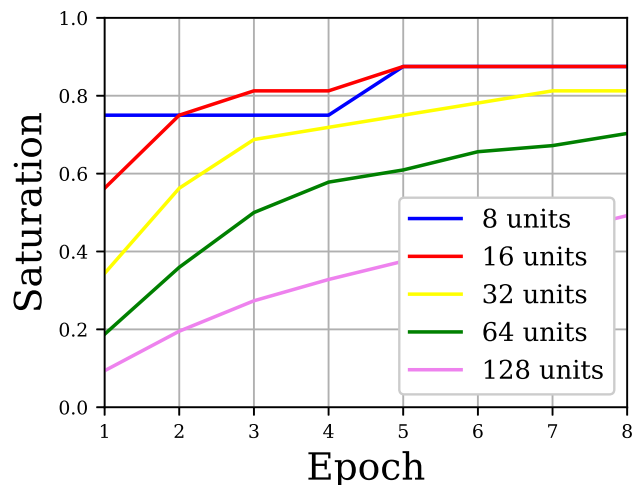
Based on these observations, we repeat the experiment on VGG11 and 19 as well as sparse (low capacity) versions of these models with $\frac{1}{8}$ of the original number of filters. We do this to observe whether the evolution of saturation patterns depends on the architecture, depth and capacity of the network.

B.2 Results

We observe that an increase in the number of units in the fully connected layer will result in an increased saturation during every epoch (see figure 66). Furthermore, the saturation converges at a similar pace than the loss, starting from a low-saturation level and increasing epoch by epoch.



(a) Validation loss during training.

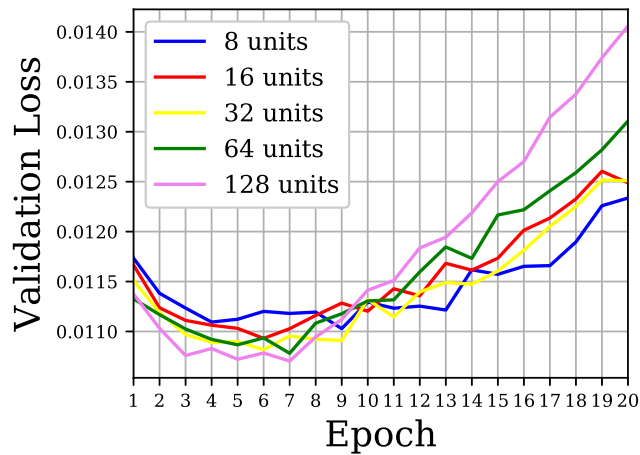


(b) Saturation of layer 2 during training.

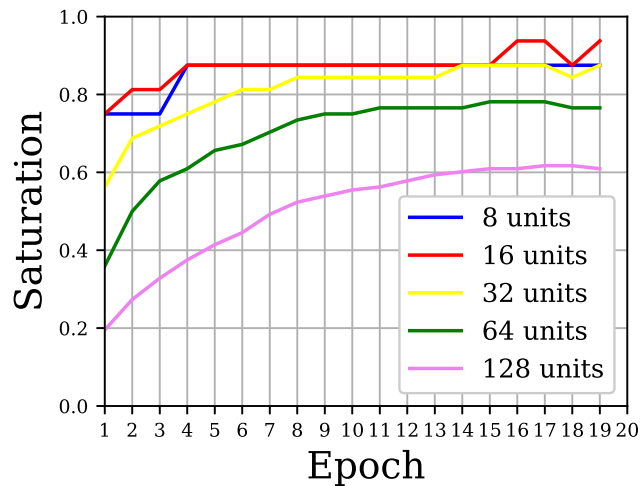
Figure 66: Saturation and loss converge at a similar pace. These results are also published in Shenk et al. (2019). © 2022 IEEE

This joint convergence is interrupted, when the model starts to overfit. In figure 67 we can see that the increase in validation loss is not followed by a drastic change in saturation. Instead, the curve of saturation values flattens for all sizes of the second layer. The fact that overfitting is not reflected in saturation values indicates that the changes to the way the data is processed when the model starts to overfit are subtle but have a drastic effect on gener-

alization.



(a) Validation loss during training.



(b) Saturation of layer 2 during training.

Figure 67: When overfitting the saturation pattern keeps converging, indicating that overfitting is not drastically altering the lossless eigenspace. © 2022 IEEE

The converging behavior that is observed on the fully connected network can be observed in a fully convolutional network as well (see figure 68). This converging behavior is independent of the position of the layer in the network, the number of layers and the capacity of the network. However, it is harder to observe in low capacity convolutional neural networks due to their low feature space dimensionality, which limits the numbers of possible saturation values.

Another interesting observation is that tail pattern seem to be observable rather early during training, which indicates that an on-line analysis during training allows the data scientist to detect inefficiencies early before training has concluded.

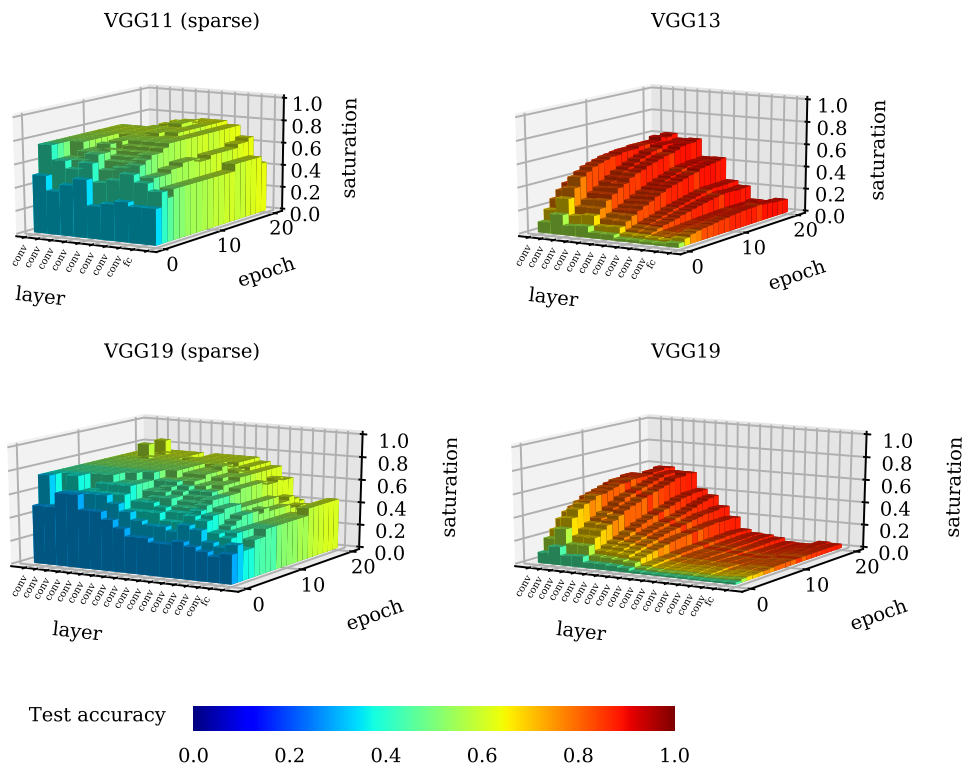


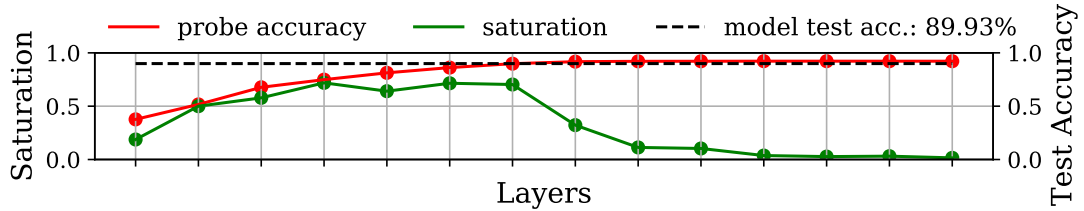
Figure 68: Saturations of convolutional neural networks show a converging behavior regarding saturation similar to previous observations in figure 66. This figure was also published in Shenk et al. (2019). © 2022 IEEE

C Different Types of Tail Patterns - A Brief Explanation

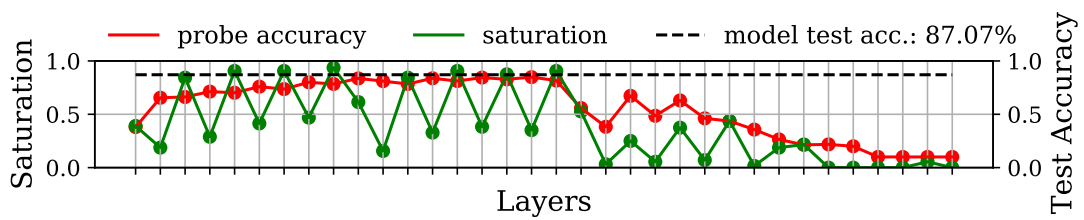
This section is complementary to section 8 and demonstrates different types of tail patterns and how they are reflected in logistic regression probe accuracy and saturation. We find that saturation is subject to noise induced by certain features of the neural architecture like the increase or decrease in filters from layer to layer, the use of 1×1 convolutions and downsampling layers are common culprits for zigzag-like behavior or sudden dips and spikes in saturation, an example for the latter is DenseNet18 in figure 69 (b). It has to be stressed that these factors are neither random nor non-reproducible. Instead, they usually result in anomalous patterns that are very stable over multiple runs (which is exemplified in section 8.2), indicating that more insights may be gained from analyzing these in later works.

Logistic regression probes are considerably more robust against the aforementioned properties. However, they are influenced by the path the information takes during the forward pass, revealing different *types* of tail patterns that can be differentiated based on the processing in the tail-layers. The three examples found commonly are depicted in figure 69. These examples also provide insights into the ways neural network process information differently, which is the main reason why we dedicate an additional section to these findings. All the networks are trained on Cifar10 using a 32×32 pixel input resolution. In figure 69 (a) we find a pass-through tail, where the layers process the information but do not advance the quality of the intermediate solution. We find this type of tail pattern is typical for sequential neural networks (which you can see from other results in the supplementary material). The second type of tail, depicted in figure 69 (b), is caused by the multiple pathways inside the DenseBlock of DenseNet. Information can pass from any previous layer to the current layer within the DenseBlock, effectively allowing the information to skip layers. When layers are skipped, the intermediate solution quality degrades and instantaneously recovers after the skipped section is over. The latter is apparent in the depicted example by the high model performance relative to the probe performance of the last DenseBlock layers. This phenomenon was initially observed on a simple MLP-example by Alain, Bengio (2017) and is reproduced by this work in figure 34 of chapter 4. If necessary, the signal may jump more than a single building block in the architecture. An example of which can be seen in figure 69 (c) on a ResNet34 architecture. This jumping is indicated by the zigzag-pattern in the

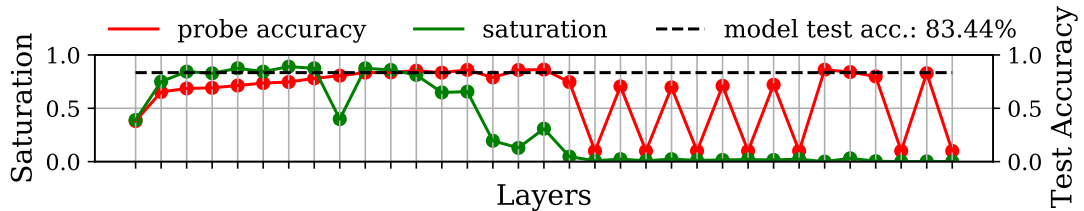
probe performance, where the higher performing layer resembles the first and lower performing layer the second layer of a residual block. The result of this section were published previously as part of the supplementary material of Richter et al. (2021c).



(a) VGG16 tail layers maintain the quality of the intermediate solution



(b) The tail of DenseNet18 shows a decay in probe performance, indicating that the last DenseBlock is skipped entirely (Alain, Bengio (2017)).



(c) ResNet34 skips most residual blocks in the tail, which is apparent by the zick-zack pattern in probe performances caused by the starts and end of skip-connections (Alain, Bengio (2017)).

Figure 69: Depending on the neural architecture, tail patterns may deviate in their appearance in probe performance. In sequential architectures (a) the layers maintain the quality of the intermediate solution. If shortcut connections exist in the architecture, layers may be *skipped*. Skipped layers are apparent by their decaying probe performance (Alain, Bengio (2017)). This is apparent on DenseNet18 (b) and ResNet34 (b) where a single DenseBlock and multiple ResidualBlocks are skipped respectively. All models are trained on Cifar10 at native resolution. © 2022 IEEE

This shows that architecture decisions, influencing the potential pathway’s information can take from input to output, can have a significant influence on the way the model processes (or chooses not to process) information. In any case, the semantic of the tail-pattern remains unchanged, since a skipped layer

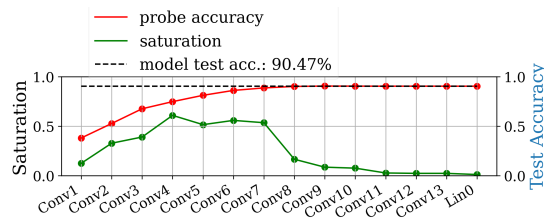
and an unproductive layer can both be considered a parameter and computational inefficiency.

D Resolution Shifts the Distribution of the Inference Process on Different Models

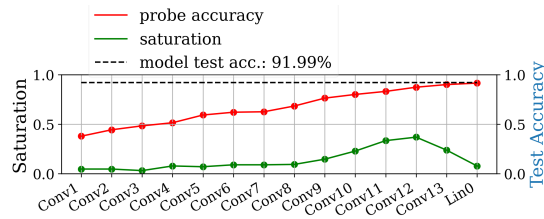
In chapter 9 we show on a ResNet18 architecture how resolution affects the distribution of the inference process by training ResNet18 on three different resolutions on Cifar10. By analyzing probe performances and saturation, we show that the inference is distributed differently on different resolutions, even if not information is added by upscaling.

To show that this observation does generalize, we repeat the experiment. We use TinyImageNet as a more complex but still small resolution problem. We further show that the hypothesis holds also true for VGG16 and ResNet50 models. The result of this section were published previously as part of the supplementary material of Richter et al. (2021c).

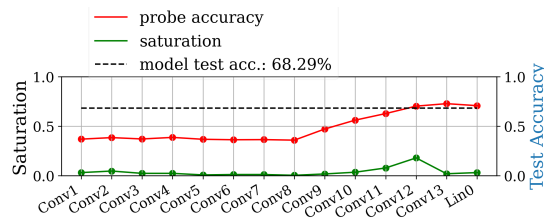
D.1 VGG16 - Cifar10



(a) 32×32 (Cifar10 native resolution)



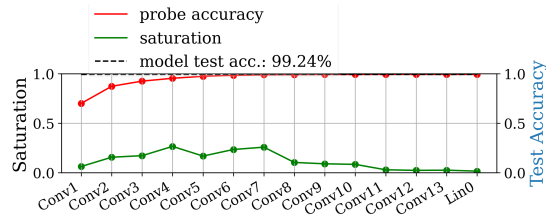
(b) 160×160



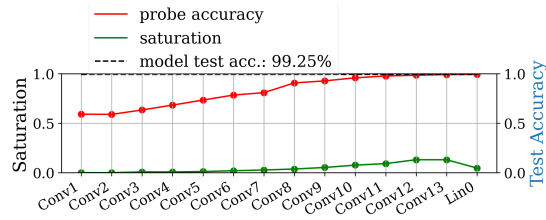
(c) 1024×1024

Figure 70: Reproduction of the experiment depicted in figure 50 using the VGG16 architecture. Because of the larger memory footprint of the model, the batch size of this experiment is reduced to 20. The observed pattern stays the same. Low-resolution results in a tail close to the output, while high resolution exhibits a tail at the input. The (medium-sized) resolution of 160×160 pixels performs best. © 2022 IEEE

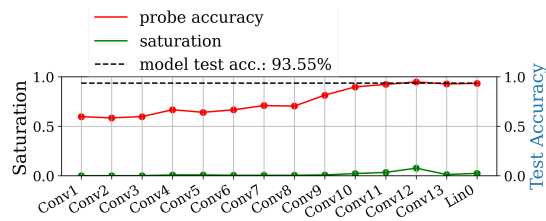
D.2 VGG16 - MNIST



(a) 32×32 (Cifar10 native resolution)



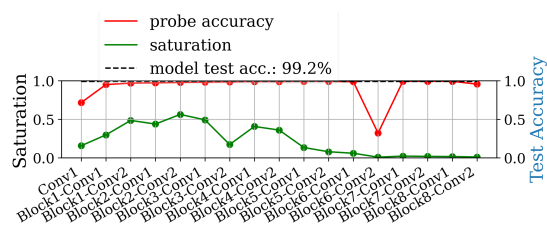
(b) 160×160



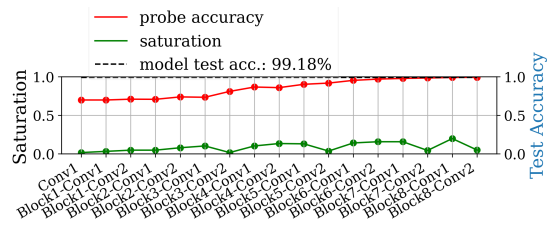
(c) 1024×1024

Figure 71: Reproduction of the experiment depicted in figure 50 using the VGG16 architecture. Because of the larger memory footprint of the model, the batch size of this experiment is reduced to 20. The observed pattern stays the same. Low-resolution results in a tail close to the output, while high resolution exhibits a tail at the input. The (medium-sized) resolution of 160×160 pixels performs best.

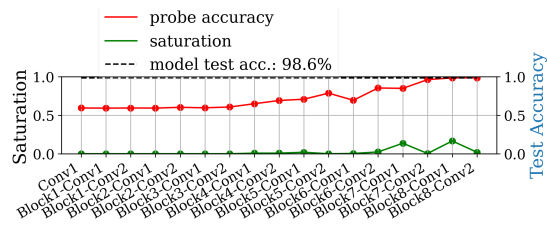
D.3 ResNet18 - MNIST



(a) 32×32



(b) 224×224

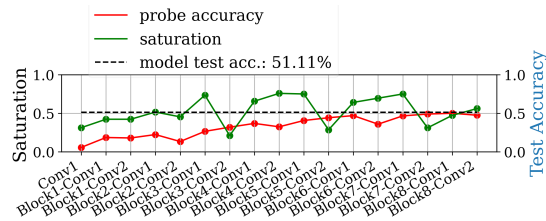


(c) 1024×1024

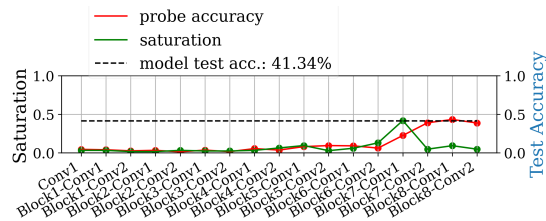
Figure 72: Resolution Experiment reproduced on ResNet18 using the MNIST dataset

D.4 ResNet18 - TinyImageNet

In this experiment, the setup varies slightly from the original setup of chapter 9 based on properties of the TinyImageNet dataset. TinyImageNet consists of 200 classes of 64×64 pixel images. Therefore, the first experiment is conducted on 64×64 pixel input resolution instead of 32×32 pixel resolution, which is the native resolution of Cifar10.



(a) 64×64 (TinyImageNet native resolution)

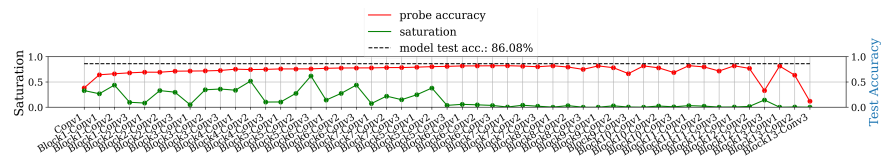


(b) 1024×1024

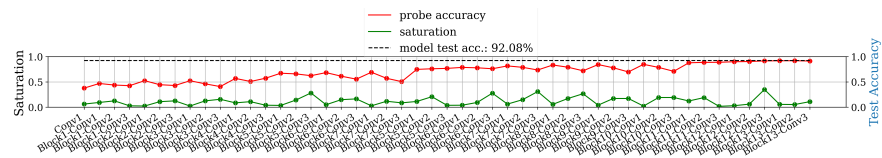
Figure 73: The higher native resolution combined with the residual connection removes the tail pattern even at 64×64 pixels for ResNet18. We see that a tail pattern can be produced in the input part of the network when the resolution is increased drastically to 1024×1024 , confirming observations from Cifar10.

D.5 ResNet50 - Cifar10

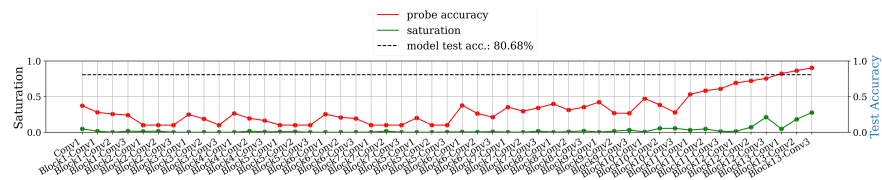
Due to an error in the setup, the mid-sized image experiment was conducted with a 160×160 pixel resolution instead of 224×224 . However, even though the resolution was altered it still shows the expected behavior observed with other models. The performance on 32×32 pixel resolution is still suboptimal and later layers are low saturated and do not contribute to the inference process. In contrast, the low saturated tail is gone at a resolution of 160×160 , which is also the resolution with the best overall predictive performance. At high resolution the overall performance decreases again, the tail pattern is present close to the input and the probes show a stagnating (and noisy) behavior, similar to other models.



(a) 32×32 (Cifar10 native resolution)



(b) 160×160 (ResNet standard)



(c) 1024×1024

Figure 74: Producing a tail pattern in the layers near the input and the output is also possible for very deep models like ResNet50 with the expected effects on the performance. The Bottleneck-Modules with 1×1 convolutions and varying filter sizes induce additional noise into saturation patterns.

E Object Size Experiments on MNIST and ResNet models

In section 9.2 we make the claim that the observed relation of input resolution and the performance of the model is an artifact caused by the underlying interaction of object size and neural architecture. We repeat the experiments in section 9.2 that demonstrate this claim on ResNet-style models as well as on the MNIST dataset. As we can see in figure 75, figure 76 and figure 77, the results are consistent with the observations shown in section 9.2 in all three scenarios. This means that the saturation patterns and probe performances of models trained 32×32 pixel images placed on 160×160 black canvases more closely resemble the saturation patterns and probe performances of the models trained on 32×32 pixel images rather than models trained on upscaled 160×160 pixel images from the same dataset.

E.1 ResNet18 - Cifar10

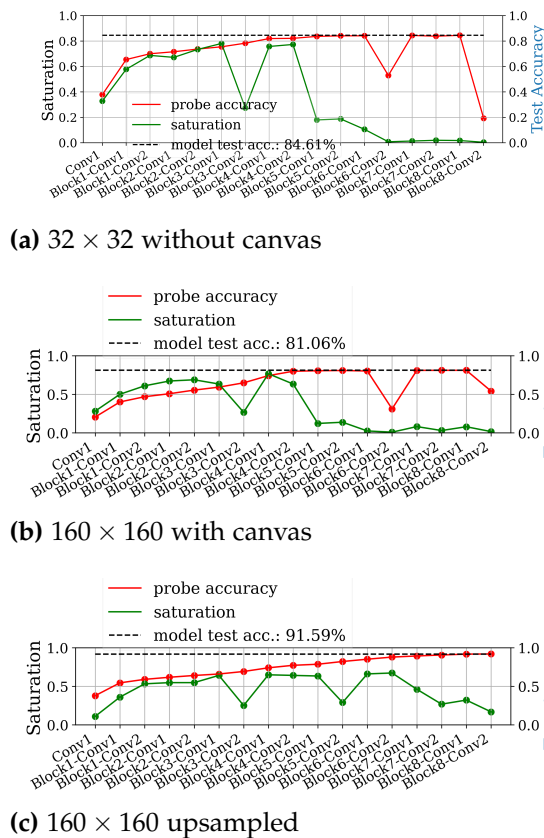
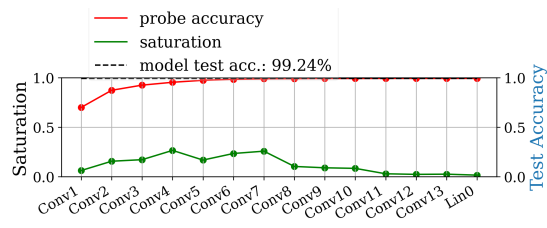
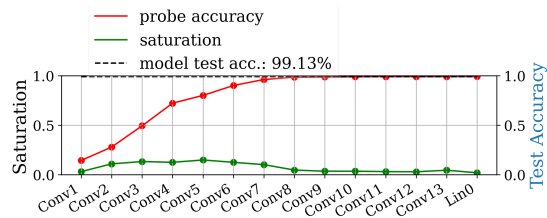


Figure 75: Random Positioning Experiments conducted with ResNet18 on Cifar10 data.

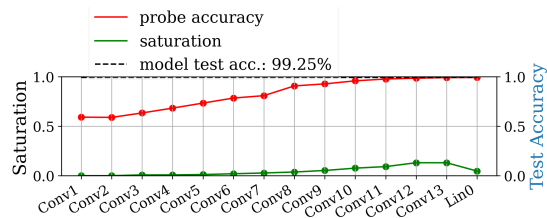
E.2 VGG16 - MNIST



(a) 32×32 without canvas



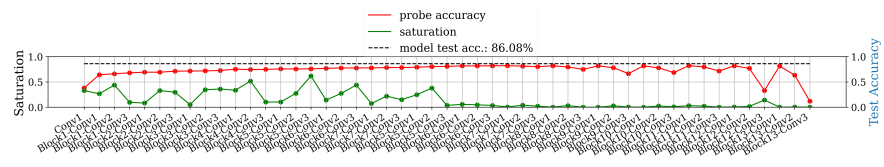
(b) 160×160 with canvas



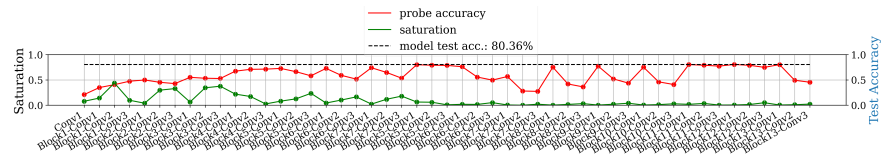
(c) 160×160 upsampled

Figure 76: Random Positioning Experiments conducted with VGG16 on MNIST data.

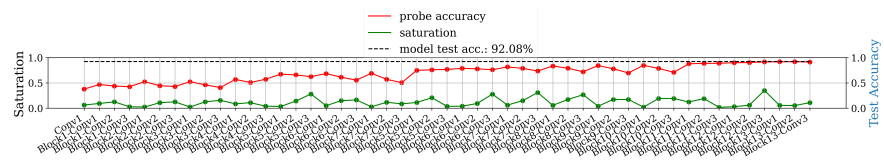
E.3 ResNet50 - Cifar10



(a) 32×32 (Cifar10 native resolution)



(b) 160×160 with canvas

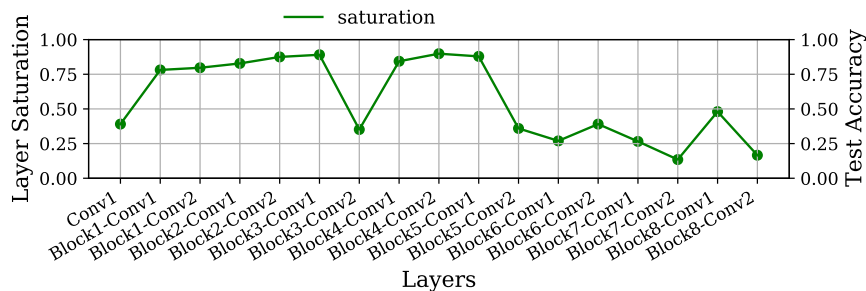


(c) 160×160 upsampled

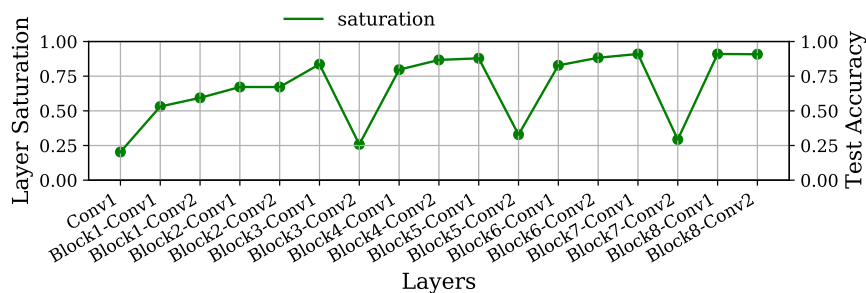
Figure 77: Random Positioning Experiments repeated on ResNet50.

E.4 Reproducing Tail Patterns on ImageNet and iNaturalist

This set of experiments is a recreation of the tail pattern phenomenon on ImageNet and iNaturalist to demonstrate that this pattern is no quirk of low-resolution datasets like Cifar10 and TinyImageNet. For these experiments computing probe performances was not feasible due to resource limitations. For this reason, only saturation is provided. Each model is trained 2 times. Once on the design resolution of 224×224 pixels of the respective models (for reference purposes, we do not expect to see a tail pattern at this resolution) and once on 32×32 pixels, which reliably results in tail patterns for these models. The training setup from section 9.3.1 is reused for these experiments. The result of this section were published previously as part of the supplementary material of Richter et al. (2021c).

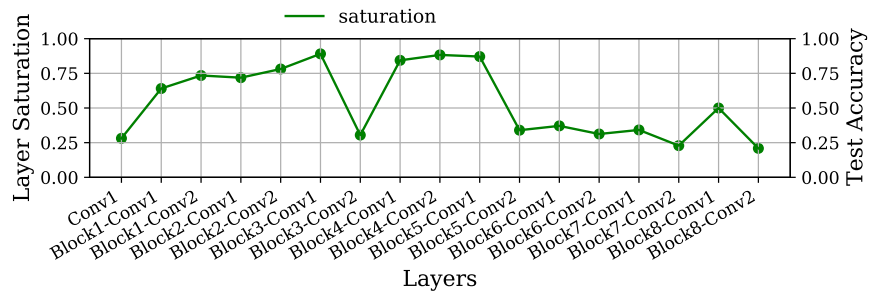


(a) ResNet18 - ImageNet - 32×32

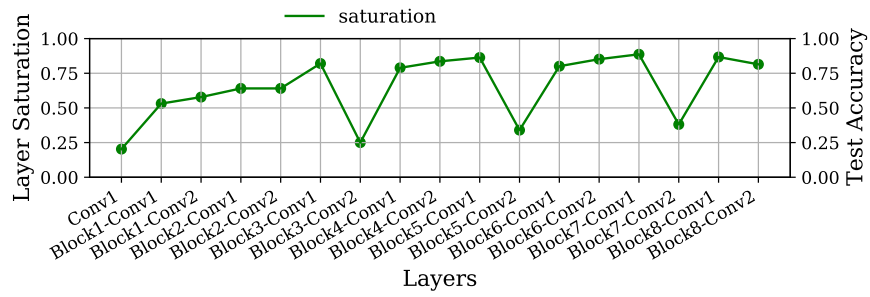


(b) ResNet18 - ImageNet - 224×224

Figure 78: ResNet18 trained on ImageNet.

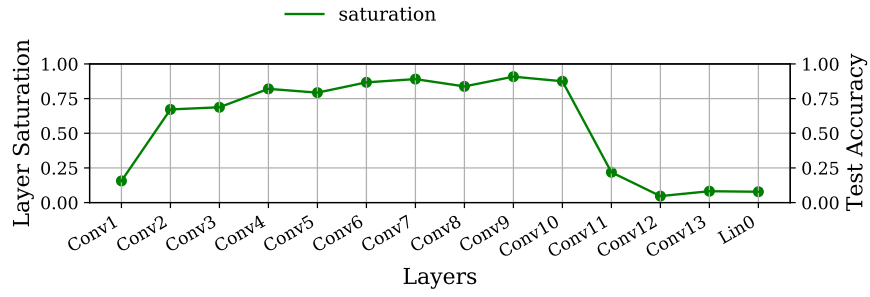


(a) ResNet18 - iNaturalist - 32×32

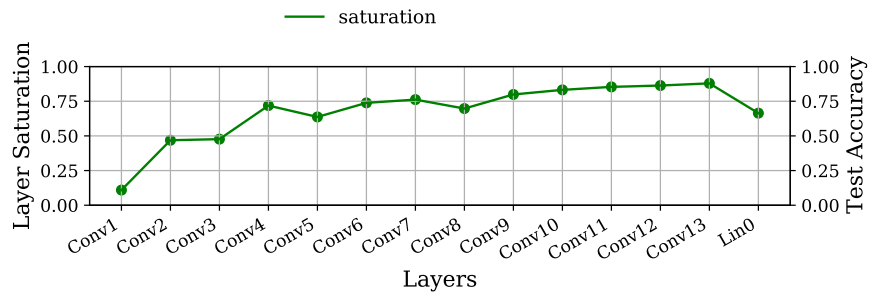


(b) ResNet18 - iNaturalist - 224×224

Figure 79: ResNet18 trained on iNaturalist.

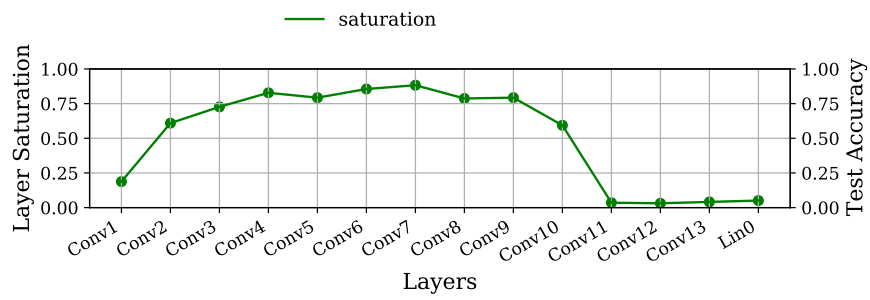


(a) VGG16 - ImageNet - 32×32

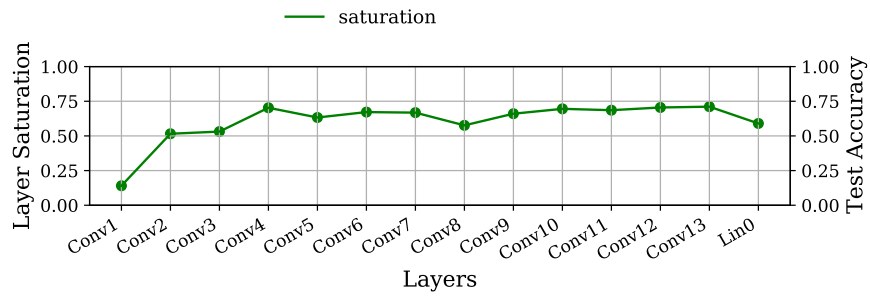


(b) VGG16 - ImageNet - 224×224

Figure 80: VGG16 trained on ImageNet.



(a) VGG16 - iNaturalist - 32×32



(b) VGG16 - iNaturalist - 224×224

Figure 81: VGG16 trained on iNaturalist.

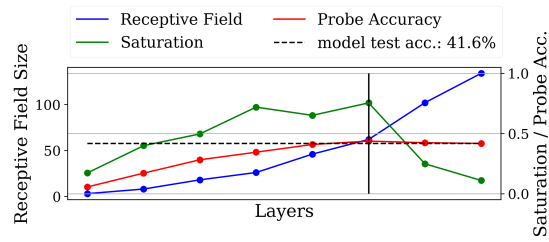
F Predicting the Border Layers in Different Scenarios Using VGG

These are reproductions of the experiments in section 9.3 using the VGG-family of networks. The first set of experiments uses VGG-style networks with $\frac{1}{8}$ of their original filter sizes in each layer. This effectively reduces the capacity of the network. The question we try to answer with this experiment is whether it is possible to distribute the inference process more by reducing the capacity (even if it is very uneconomical regarding the loss in predictive performance). The second experiment uses the standard-VGG architectures again, but uses the TinyImageNet-Dataset instead to demonstrate that the observed behavior is not a quirk of the Cifar10 dataset. The result of this section were published previously as part of Richter et al. (2021c).

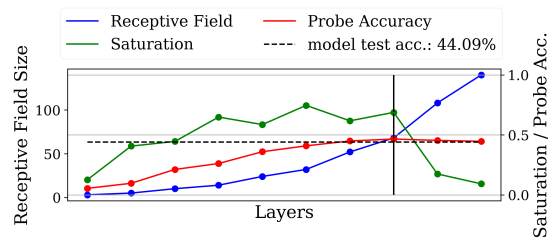
F.1 Results

We find that reducing the capacity of the network is not affecting the border layer and the distribution of the inference process (see figure 82 and 83). This indicates that the influence of the receptive field is still the dominating factor, and that optimizing the depth and the width of the network can be optimized independently. We expand upon this topic in chapter 11.

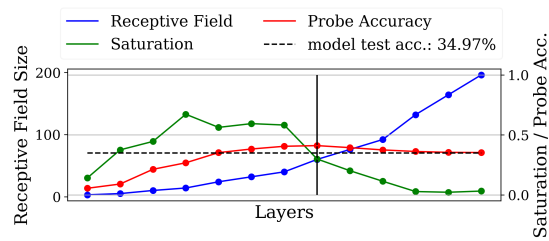
E.3 VGG11 / VGG16 - TinyImageNet



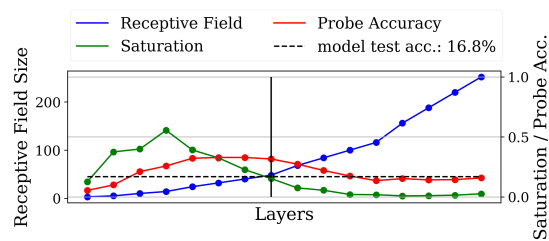
(a) VGG11



(b) VGG11



(c) VGG16



(d) VGG11

Figure 83: Repeating the experiment depicted in figure 52 on TinyImageNet yields consistent results regarding the border layer.

F.4 DenseNet18, 65 - Cifar10

This part of the appendix contains old results regarding tail patterns on DenseNet-architectures. Note that the start of the degradation is predicted by b_{min} as the border layer. The border layers depicted in the plots are b_{max} , the receptive field size depicted in these experiments is also $r_{l,max}$, thus the printed border layer is not marking the end of the productive part of the network. The main reason for this is that these experiments are based on experiments in chapter 9. At the point where this section was written and the experiments conducted, the differentiation between $r_{l,min}$ and $r_{l,max}$ was unknown, and the then-conclusion that tails in DenseNet-architectures cannot be predicted like ResNets tail patterns was resolved by using b_{min} instead of b_{max} (see section 10.2). However, we think that it is still important to include these results to show that tail pattern can be reproduced on other architectures such as DenseNet on multiple datasets. A compilation of different types of tail patterns can also be found in appendix C.

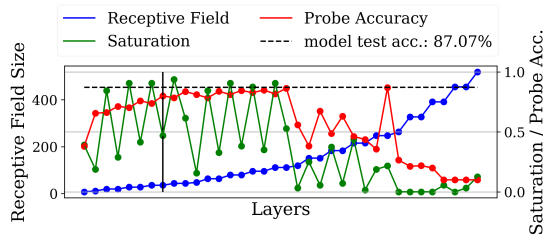


Figure 84: DenseNet18 - Cifar10 - 32×32 input resolution.

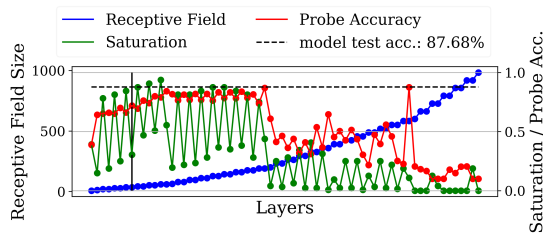
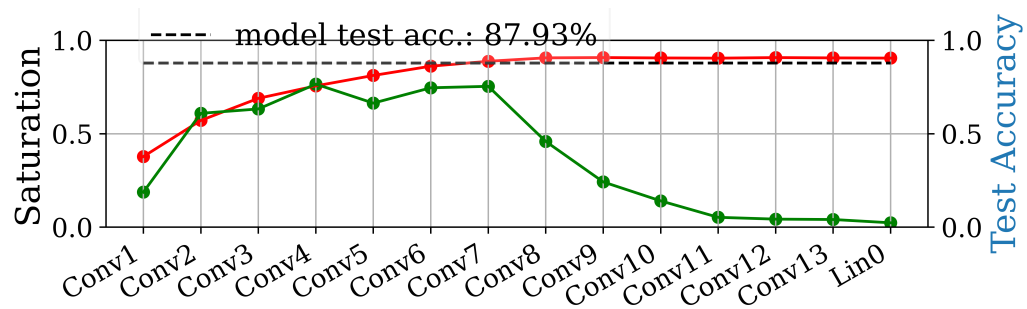


Figure 85: DenseNet65 - Cifar10 - 32×32 input resolution.

G Receptive Field Analysis With Partial Solution Heatmaps (Compilations from Chapter 9)



(a) VGG16 trained on Cifar10 using 32×32 pixel input size

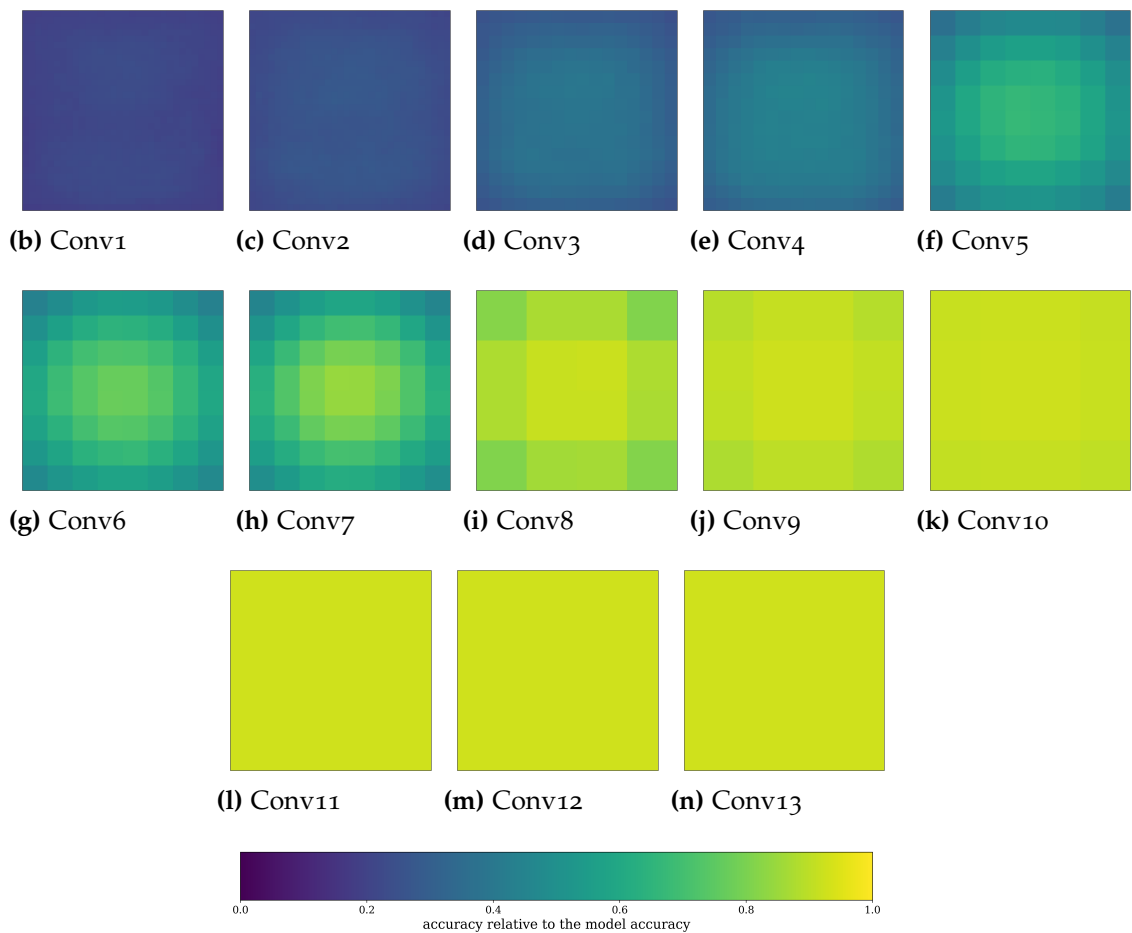
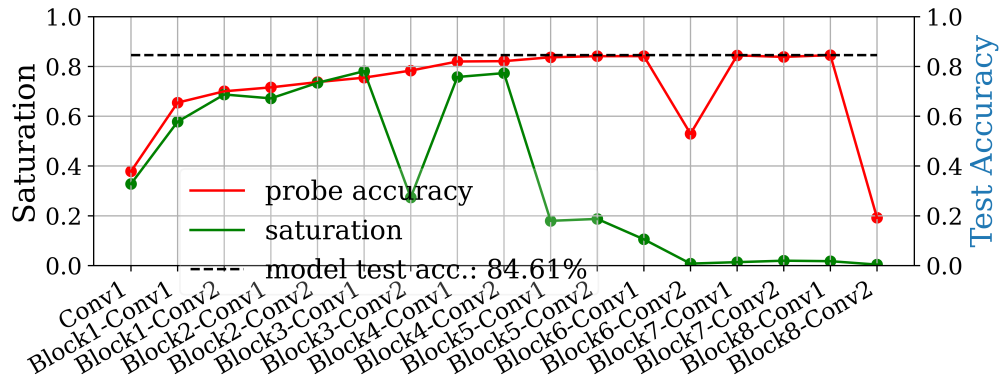


Figure 86: VGG16 trained on Cifar10 alongside the heatmaps generated from the relative accuracy of the partial solutions in each layer. Note that the border layer (Conv8) is the first layer to have partial solutions of equal quality to the model's solution (measured in accuracy).



(a) ResNet18 trained on Cifar10 using 32×32 pixel input size

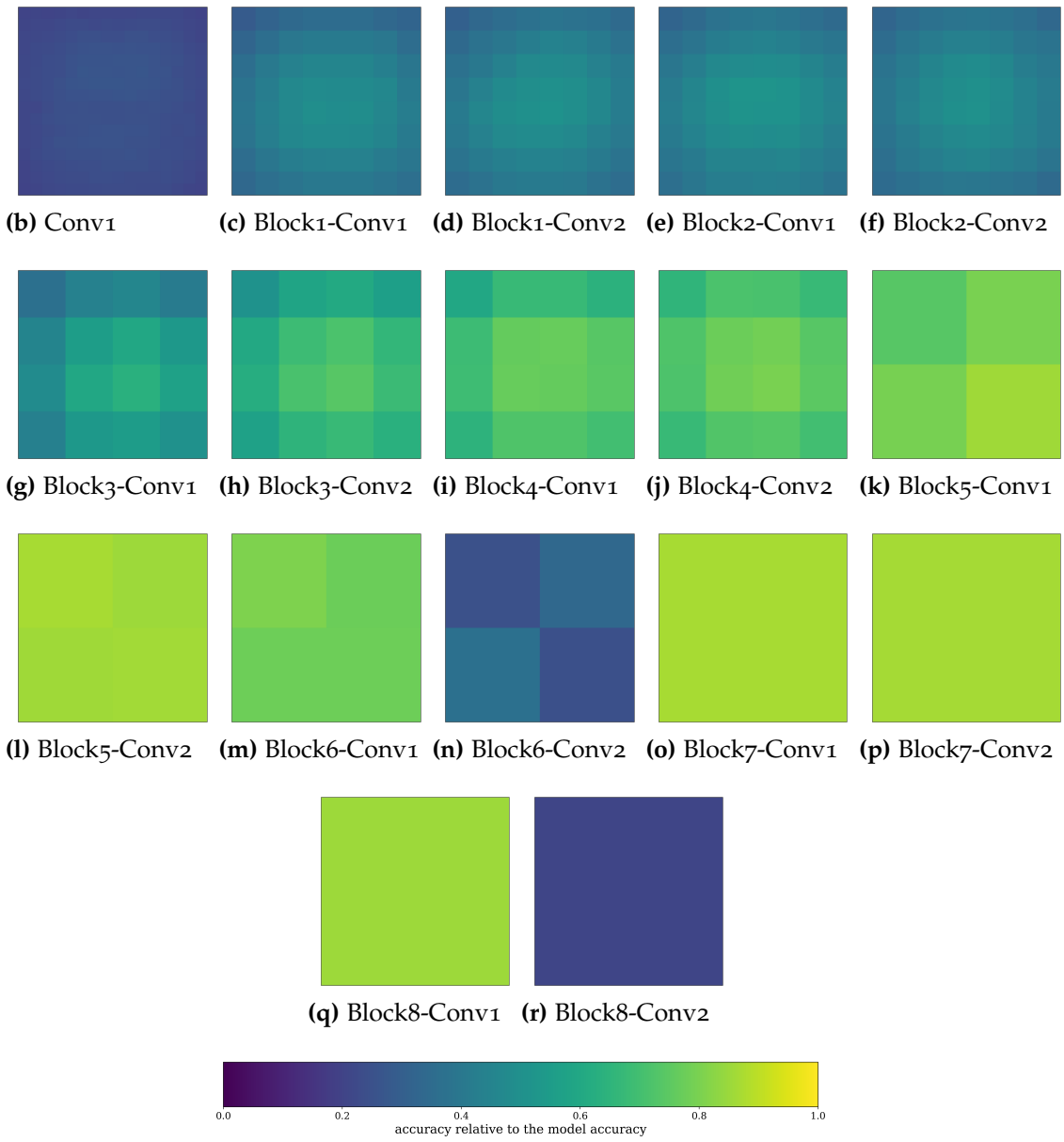
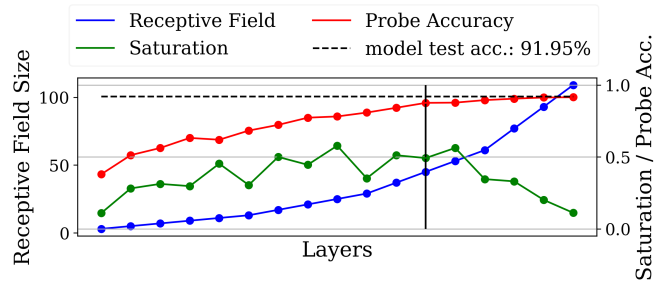


Figure 87: ResNet18 trained on Cifar10 alongside the heatmaps generated from the relative accuracy of the partial solutions in each layer. Note that "skipped" layers (Block6-Conv2 and Block8-Con2) have a worse performance and worst partial solutions.



(a) ResNet18 (Cifar10 Optimized) trained on Cifar10 using 32×32 pixel input size

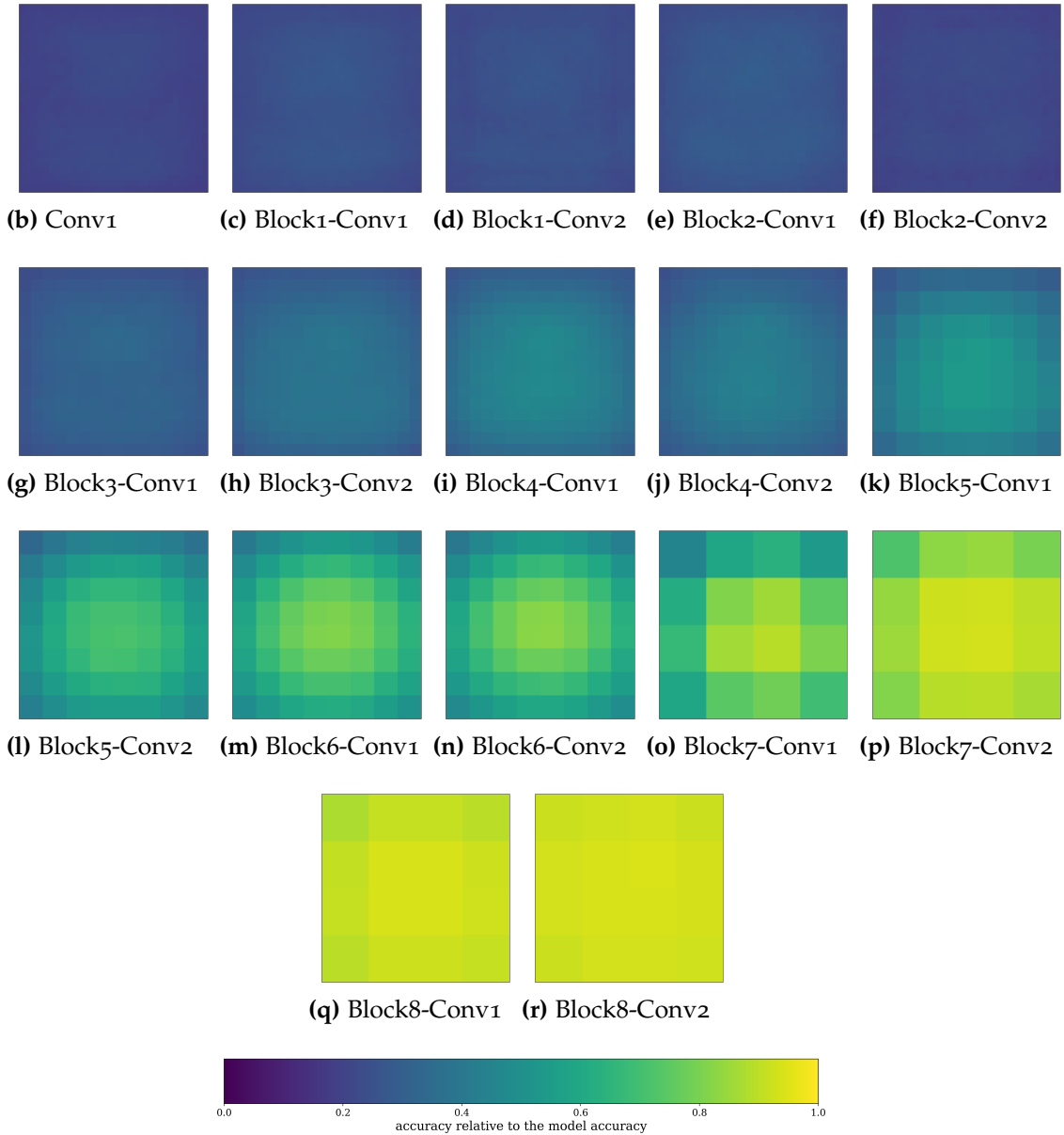


Figure 88: Cifar10 optimized ResNet18 trained on Cifar10 alongside the heatmaps generated from the relative accuracy of the partial solutions in each layer. Note that no layers are skipped and the solution quality of the partial solution develops up until the final residual block.

Ceterum censeo Carthaginem esse delendam.