UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
NEURO- UND BIOINFORMATIK

From the Institute for Neuro- and Bioinformatics
of the University of Lübeck
Direktor: Prof. Dr. rer. nat. Thomas Martinetz

Towards Understanding Convolutional Neural Networks Through Visualization and Systematic Simplification

Dissertation
for Fulfillment of
Requirements
for the Doctoral Degree
of the University of Lübeck

from the
Department of Computer Sciences and Technical Engineering

Submitted by
Christoph Linse
from Cologne

Lübeck, 2024

IM FOCUS DAS LEBEN

First referee: Prof. Dr. rer. nat. Thomas Martinetz

Second referee: Prof. Dr. rer. nat. Sebastian Otte

Date of the oral examination: 28.05.2025

Approved for printing: Lübeck, 12.06.2025

Acknowledgments

First, I would like to express my sincere thanks to my supervisor Prof. Thomas Martinetz for his invaluable support during my time at the Institute for Neuro- and Bioinformatics in Lübeck. He gave me the freedom to explore my ideas and follow my curiosity while providing me with his feedback that shaped and refined my ideas. I also want to thank Prof. Erhardt Barth for his insightful perspectives on computer vision. I would like to acknowledge Prof. Hammam Alshazly for our countless fruitful discussions and our great collaboration, especially during the difficult time when COVID-19 restrictions applied. He always encouraged me to develop new ideas, continue experimenting, and publish. Mrs. Yvonne Marbach and Mr. Dirk Laggin deserve special thanks for their administrative and technical support. I would like to dedicate a few words to my family and Johannes Bela for their unwavering support and encouragement throughout this journey. Also, I am thankful for the opportunities to work on projects like "Mittelstand 4.0-Kompetenzzentrum Kiel", "Zentrum für Künstliche Intelligenz Lübeck", and "Mittelstand-Digital Zentrum Schleswig-Holstein". These experiences helped me to grow both personally and professionally.

Zusammenfassung

Black-Box-Systeme wie Convolutional Neural Networks (CNNs) haben Computer-Vision grundlegend verändert. Auch wenn Visualisierungsmethoden dabei helfen, CNNs zu erforschen und zu erklären, bleiben ihre inneren Abläufe undurchsichtig. Insbesondere bleibt unklar, wie spezifische Merkmale detektiert werden. Da Deep Learning in verschiedenen Bereichen immer mehr Anwendung findet, wird es zunehmend wichtiger, diese Modelle zu verstehen. So können Fehlinterpretationen und Verzerrungen vermieden werden, die bedeutende gesellschaftliche Folgen haben könnten.

Diese Forschung motiviert umfassende Visualisierungstechniken, die verschiedene Aspekte von CNNs berücksichtigen. Bisherige Ansätze konzentrieren sich oft auf nur wenige Aspekte und beantworten spezifische Fragen. Deren Kombination in einer Software könnte eine umfassendere Sicht auf CNNs und ihre inneren Prozesse ermöglichen. Während der zweidimensionale Raum wegen der begrenzten Bildschirmgröße nicht alle relevanten Informationen darstellen kann, bieten 3D-Visualisierungen neue Ansätze für Darstellung und Interaktion. Daher ermöglichen wir die Visualisierung großer CNNs im dreidimensionalen Raum. Als weiterer Beitrag zum Gebiet der Visualisierung verbessert diese Arbeit die Activation-Maximization-Methode für Merkmalvisualisierung, die zuvor mit lokalen Maxima zu kämpfen hatte.

Neben der Visualisierung verbessert diese Forschung die Transparenz von CNNs durch systematische Vereinfachung. Wir verwenden vordefinierte Faltungsfilter aus der traditionellen Bildverarbeitung in modernen CNN-Architekturen. Anstatt die Filter während des Trainings anzupassen, werden Linearkombinationen der Ausgaben dieser vordefinierten Filter gelernt. Unsere Pre-defined Filter Convolutional Neural Networks (PFCNNs), die nur neun verschiedene Kanten- und Linien-Detektoren nutzen, generalisieren insbesondere auf kleinen Datensätzen besser als Standardnetzwerke. Bei ResNet18 haben wir um $5 - 11$ Prozentpunkte erhöhte Testgenauigkeiten beobachtet auf den Datensätzen Fine-Grained Visual Classification of Aircraft, StanfordCars, Caltech-UCSD Birds-200-2011 und 102 Category Flower, wobei die Anzahl der trainierbaren Parameter gleich geblieben ist. Die Ergebnisse zeigen, dass für viele Bildverarbeitungsprobleme kein Training der Faltungskerne erforderlich ist. In der Praxis können PFCNNs trainierbare sogar Gewichte einsparen.

Abstract

Black-box systems like Convolutional Neural Networks (CNNs) have transformed the field of computer vision. While visualization tools have helped explore and explain CNNs, their inner workings remain opaque, particularly how they detect specific features. As deep learning applications become more widespread across various fields, it becomes crucial to understand these models. This understanding is needed to avoid misinterpretation and bias, which can seriously affect society.

This research motivates holistic visualization approaches, which show various aspects of CNNs. Existing visualizations often focus on a few aspects, answering specific questions. Combining them in comprehensive software could provide a more holistic view of CNNs and their inner processes. While 2D space cannot present all relevant information due to screen size restrictions, 3D environments offer new representation and interaction opportunities. Therefore, we enable the visualization of large CNNs in a virtual 3D space. This work further contributes to the visualization field by improving the activation maximization method for feature visualization, which previously struggled with local maxima.

In addition to visualization, this research increases CNN transparency through systematic simplification. We use pre-defined convolution filters from traditional image processing in modern CNN architectures. Instead of changing the filters during training, the training process finds linear combinations of the pre-defined filter outputs. Our Pre-defined Filter Convolutional Neural Networks (PFCNNs) with nine distinct edge and line detectors generalize better than standard CNNs, especially on smaller datasets. For ResNet18, we observed increased test accuracies ranging from $5-11$ percentage points with the same number of trainable parameters across the Fine-Grained Visual Classification of Aircraft, StanfordCars, Caltech-UCSD Birds-200-2011, and the 102 Category Flower dataset. The results imply that many image recognition problems do not require training the convolution kernels. For practical use, PFCNNs can even save trainable weights.

Contents

Abbreviations

AI	Artificial Intelligence
AM	Activation Maximization
BN	Batch Normalization
BP	Backward Pass
CNN	Convolutional Neural Network
CT	Computed Tomography
FLOPs	FLoating point OPerations
FP	Forward Pass
GPU	Graphics Processing Unit
GUI	Graphical User Interface
t-SNE	t-distributed Stochastic Neighboring Embedding
Grad-CAM	Gradient-weighted Class Activation Mapping
PFCNN	Pre-defined Filter Convolutional Neural Network
PFM	Pre-defined Filter Module
VR	Virtual Reality
XAI	eXplainable Artificial Intelligence

Chapter 1

Introduction

Today, deep learning offers state-of-the-art solutions across various fields, including image recognition. Its popularity increased due to affordable computational resources, the internet's role in data collection and sharing, and the availability of large-scale datasets like ImageNet [80]. Convolutional Neural Networks (CNNs) have demonstrated strong generalization abilities and superior performance in image recognition [54, 63], especially when dealing with unconstrained image data captured in real-world settings [9]. The end-to-end training paradigm eliminates the need for manual feature extraction, as features are learned automatically by minimizing a defined error function on the training data. The features of CNNs can be generic, allowing the transfer to various domains even when training data is limited [33].

1.1 Convolutional Neural Networks are Black-Box Models

However, CNNs are often perceived as black-box models that provide little insight into how they produce their outcomes. In contrast to traditional computer vision methods, CNN outputs cannot be explained by simply analyzing the inference process. Unlike linear systems, non-linear networks cannot be identified by capturing impulse responses. The sheer number of parameters, often in the millions, hinders the manual investigation of learned features or specific network decisions. Recently, there has been an ongoing trend of models to grow bigger, from millions to billions of

parameters, adding to the difficulties of comprehending such systems. The hierarchical structure of stacked layers and non-linearities complicates tracking how distinct features emerge within the network. Other factors affecting the input-output behavior, including regularization, adaptive mechanisms, or loss functions, further complicate the situation. Consequently, CNNs are criticized for making unpredictable decisions and offering limited insight and control to human users [79].

AI systems become increasingly important in our societies. Deep learning is progressively applied in use cases that can have significant implications for human well-being. These applications range from the recognition of individuals [5–7, 9], to medical diagnostics [4, 8]. With a growing number of AI technologies worldwide, they are becoming part of our daily lives, appearing in home assistants, web services, and beyond.

While AI systems can impact human well-being, various risks are associated with opaque models [66]. These systems can be biased towards certain outputs, leading to discriminatory outcomes based on factors like race or gender. Biases in the training data can be transferred to AI systems, perpetuating unfair treatment of certain groups. Another risk is the confusion between correlation and causation, where AI models may rely on spurious correlations that don't reflect real-world causal relationships, resulting in incorrect predictions. Despite these issues, there is growing human overreliance on automated decision-making, known as automation bias. This can lead to errors in critical areas such as medical diagnoses or decisions. Additionally, overreliance can cause deskilling, where strong dependency on AI systems erodes existing human skills or hinders skill development [66].

These risks highlight that without a clear understanding of AI systems, users or decision-makers may not assess risks adequately. This increases the likelihood of undesired outcomes and underscores the need for transparent and interpretable models.

1.2 The Role of Explainable Artificial Intelligence

Explainable Artificial Intelligence (XAI) is a growing research field dedicated to enhancing the interpretability of AI systems [10, 92]. Rather than only prioritizing predictive power, XAI tries to make outcomes more understandable to humans. Although there is no universally accepted definition for XAI [2], it generally addresses concerns about AI transparency and trust. The meaning of "explanation" can vary depending on domain-specific factors such as user context, task requirements, and expectations. XAI systems typically aim to clarify their behavior by explaining their capabilities, ongoing processes, and future steps [27].

Meske et al. [66] summarize various advantages of transparency. First, transparency is essential to verify that machine learning algorithms make decisions based on meaningful and relevant patterns in the input data. Second, explainable models expose their weaknesses, allowing AI researchers and developers to address and improve them.

Third, transparent systems offer opportunities to uncover new insights and previously unknown patterns within the data. Fourth, fields like medicine and law enforcement have to meet regulatory requirements and ethical considerations. To fulfill those regulations, black-box systems like neural networks will need to become more transparent.

1.3 Objectives and Contributions

This dissertation contributes to XAI by studying how features emerge within CNNs. The first contribution presents a use case study that highlights the need for more holistic XAI tools. The next two contributions introduce new visualization techniques that reveal the inner workings of CNNs. Additionally, this work simplifies CNNs by employing pre-defined, interpretable convolution filters, enhancing both their transparency and generalization. Finally, it reduces the number of trainable parameters by utilizing these pre-defined filters. The contributions are detailed below.

First Contribution: Explainable COVID-19 Detection Using Chest CT Scans

The first contribution motivates the development of new, holistic visualization tools for XAI. We critically evaluate the explanations provided by the t-SNE [93] and Grad-CAM [82] techniques in their ability to explain network decisions. t-Distributed Stochastic Neighbor Embedding (t-SNE) is a dimensionality reduction method used to visualize the high-dimensional feature space of trained models. Grad-CAM (Gradient-weighted Class Activation Mapping) generates class-discriminative visualizations by highlighting the image regions that influenced a specific prediction of the model.

Our study investigates the effectiveness of CNNs in diagnosing COVID-19 from chest computed tomography (CT) images. Coronavirus disease 2019 (COVID-19) is an infectious disease caused by the severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2). Chest CTs have been a useful complement to RT-PCR testing, playing a key role in the screening and diagnosis of infections [3, 20, 39]. We fine-tune CNNs on two binary classification datasets: the SARS-CoV-2 CT-scan dataset [85] and the COVID19-CT dataset [32]. t-SNE visualizations show well-separated clusters for COVID-19 and non-COVID-19 cases. Grad-CAM visualizations reveal that the models accurately localize COVID-19-related regions, aligning with the expectations of experienced radiologists.

However, the explanations provided by t-SNE and Grad-CAM address specific aspects of the model. Relevant questions remain. What visual features do models learn during training? What strategies do they use? What are their decisions based on? On a deeper level, it remains unclear how the model's weights influence the features. Addressing these gaps requires new, holistic visualization approaches that capture multiple aspects of the model simultaneously. Queck et al. [76] conducted a study on the

needs of AI users. The most common requests were about tools that improve the debugging and comprehension of AI models. We believe that holistic visualization tools are essential for providing this deeper understanding.

Our findings are detailed in Chapter 3 and have been published in Sensors [8].

Second Contribution: Visualization of Large Neural Networks in 3D Space

We achive holistic visualization by combining existing techniques into a single software tool where they complement each other. Previous work has implemented this in 2D space, where CNN representations were divided into multiple sections due to screen space limitations [55, 97, 102]. In contrast, this work explores visualizations in 3D space as an alternative.

One advantage of 3D over 2D is that more information can be presented clearly. This allows the entire CNN to be visualized as a comprehensive entity without losing details. In a virtual 3D environment, users can move the virtual camera to view different aspects of the model as needed. This way, information density can be increased without adding cognitive load [43]. Users can view the entire network from a distance to understand its computational graph, then zoom in on specific layers for a closer look. Changing perspectives allows users to shift their focus and gain precise insights and interpretations [43]. Moreover, 3D visualizations can be designed to resemble physical objects with haptic input and output for intuitive interactions. As a result, interactive 3D environments provide clearer and more convenient access to CNNs.

Recent prototypes [1, 12, 65, 81, 94] have demonstrated that 3D representations can enhance the understanding of small neural networks. However, these prototypes had limitations. First, real-time rendering of large, popular architectures with thousands of feature maps pose significant computational challenges. Optimization techniques from the video game industry could be used to speed up the rendering process. Second, the interaction designs of these prototypes were not suited for deep CNNs with many layers. Third, previous tools often restricted CNNs to linear structures, excluding more complex architectures with splits or joints. A comprehensive visualization tool should work with arbitrary computational graphs. Finally, developers and researchers require a flexible, user-friendly interface to visualize custom architectures.

This research addresses these limitations by introducing DeepVisionVR, a Unity [28] application designed to visualize large, popular CNNs in an interactive 3D environment. The software enables users to navigate freely within the virtual CNN representation and provides multiple ways to interact with the network. Additionally, a novel PyTorch module dynamically links PyTorch [74] with Unity, offering researchers and developers a convenient interface for visualizing their architectures.

A case study illustrates the effectiveness of DeepVisionVR in analyzing how CNNs memorize images using high-frequency information. Yin et al. [101] found that both low- and high-frequency components in images contribute to classification accuracy.

Later, Wang et al. [96] hypothesized that high-frequency components in natural images, which are not perceived by humans, can be exploited by deep neural networks to boost performance. These high-frequency components do not necessarily lead to overfitting, as they may contain class-relevant information. However, relying on such features can reduce model robustness [37]. According to the texture hypothesis [24], high-frequency, texture-like features dominate models trained on ImageNet. We investigate how CNNs memorize images using high-frequency features and visualize the patterns involved in this process. The results show that high-frequency features can lead to poor generalization, offering insights into the mechanisms of overfitting.

This contribution is detailed in Chapter 4 and has been published in Neural Computing and Applications [50]. Our software is available on Github: `github.com/Criscraft/DeepVisionVR`.

Third Contribution: Leaky ReLUs That Differ in Forward and Backward Pass Facilitate Activation Maximization

In the proposed DeepVisionVR software, features are visualized by activation maximization (AM). This technique generates input stimuli that maximize the output of specific network units. A common application involves optimizing an input image $\mathbf{x} \in [-1, 1]^{C \times H \times W}$ to activate a class neuron i in the context of image classification. The resulting image can then be analyzed and interpreted qualitatively to identify the visual features that represent the class. In this work, the terms AM and feature visualization are used interchangeably. AM can be described as finding

$$\mathbf{x}^* = \underset{\mathbf{x} \in [-1,1]^{C \times H \times W}}{\operatorname{argmax}} f_i(\mathbf{x}) \tag{1.1}$$

with $f_i(\mathbf{x})$, the output of the class neuron for class i. The optimization is achieved iteratively through gradient ascent

$$\mathbf{x} \longleftarrow R(\mathbf{x} + \mu \nabla_x f_i(\mathbf{x})) \tag{1.2}$$

with a learning rate μ and a regularization step R. The regularization serves as an image prior.

A problem with AM is that it is difficult to reliably and consistently find the maximum for different randomly initialized input images. In the early stages of AM, Erhan et al. [19] did not consider this a significant issue. However, modern CNNs have deeper architectures, more parameters, larger input images, and often use rectifiers instead of sigmoid activation functions. Our research questions the efficacy of the AM technique for generating optimal input stimuli. We show that AM struggles to find the optimal stimulus for simple functions that use ReLU or Leaky ReLU [59, 69], due to three optimization issues: a) Sparse gradients can stop the optimization process early, preventing it from reaching the global optimum. b) Different patterns emerge at

different speeds, requiring many iterations to achieve convergence. c) Local maxima often result in suboptimal solutions. Additionally, we observe a strong dependency between the local maximum found by AM and the initialization of the input image. This may explain the success of techniques that use specific initialization strategies for AM. For example, Nguyen et al. [71] initialized the input with the average of an image cluster. Their method successfully generates different aspects of the same class.

Seeing that large negative slopes in Leaky ReLUs can help mitigate the optimization issues, we propose the ProxyGrad algorithm. To achieve higher maxima during AM, we use a secondary network as a proxy for gradient computation. This proxy network incorporates Leaky ReLUs with significantly higher negative slopes, resulting in a smoother loss landscape with fewer local maxima than the original network, thereby enhancing optimization efficiency. The proxy network is an exact copy of the original network, sharing the same weights. In the ProxyGrad algorithm, the original network handles the forward pass, while the proxy network performs the backward pass. Importantly, the proxy network uses the activations from the original network for gradient computation. If the forward pass slope is zero and the backward pass slope is positive, the network produces sparse data representations, similar to ReLU, but dense gradients, similar to Leaky ReLU. This approach differs from simply replacing ReLUs with Leaky ReLUs, as the latter would involve using Leaky ReLUs in both the forward and backward passes.

We demonstrate that class visualizations of ResNet18 [30] trained on ImageNet [80] achieve higher activation values and often offer better visual clarity when using ProxyGrad compared to standard methods. Furthermore, we show that ProxyGrad performs effectively even when training CNN weights for classification, and in some cases, outperforms traditional optimization techniques.

Visualization approaches in the literature also modify the derivative of ReLUs. The Deconvnet approach [104] applies the ReLU to the gradients, allowing only positive "derivatives". Guided backpropagation [87] sets gradients with negative values or inputs to zero, resulting in an even sparser gradient vector. In contrast, ProxyGrad uses distinct negative slopes for Leaky ReLUs in the forward and backward passes, offering a different approach to handling gradients.

Chapter 5 presents the details of the study. The results have been published at the 2024 International Joint Conference on Neural Networks (IJCNN) [52].

Fourth Contribution: Enhancing Generalization in Convolutional Neural Networks through Regularization with Edge and Line Features

Visualizations can deepen our understanding of CNNs but also highlight the enormous complexity of these structures. The third part of the thesis aims at increasing CNN transparency through systematical simplification. We employ understandable, predefined convolution filters to precondition CNNs to extract specific information from

previous layers. A convolution operation can be described as:

$$(f * g)[m, n] = \sum_{c=1}^{C} \sum_{i,j} f_c[i, j] g_c[m - i, n - j]. \tag{1.3}$$

Here, $f \in \mathbb{R}^{C \times k \times k}$ is the filter, and $g \in \mathbb{R}^{C \times W \times H}$ is the input tensor with the number of channels C, the size of the filter k, the width of the filter maps W, and their height H. m and n index the pixels. We express the filters f_c using k^2 pre-defined filters $h \in \mathbb{R}^{k^2 \times k \times k}$ and weights $w \in \mathbb{R}^{k^2 \times C}$:

$$f_c[i, j] = \Big(\sum_{l=1}^{k^2} w_{l,c} \cdot h_l \Big)[i, j]. \tag{1.4}$$

The convolution becomes:

$$(f * g)[m, n] = \sum_{c=1}^{C} \sum_{l=1}^{k^2} w_{l,c} \cdot (h_l * g_c)[m, n]. \tag{1.5}$$

If the set of h_l has a full rank, the network can learn all possible kernels by adjusting the weights $w_{l,c}$. This changes by adding a ReLU [69].

$$\mathrm{PFM}[m, n] = \sum_{c=1}^{C} \sum_{l=1}^{p} w_{l,c} \cdot \mathrm{ReLU}(h_l * g_c)[m, n] \tag{1.6}$$

The additional ReLU nullifies negative values. It removes the information unrelated to the specific pre-defined filter, leading to a well-structured and comprehensible data representation. The intermediate feature maps $\mathrm{ReLU}(h_l * g_c)$ form distinct features, each containing positive filter responses only. Later, we will choose the h_l as edge and line detectors in different orientations. A subsequent linear combination with the trainable weights $w_{l,c}$ combines the distinct features. The entire module is called Pre-defined Filter Module (PFM). A Pre-defined Filter Convolutional Neural Network PFCNN is a CNN with PFM layers instead of traditional convolutional layers. The PFM is implemented as a depthwise 3×3 convolution containing the pre-defined filters. It continues with a batch normalization layer (not shown), a subsequent ReLU, and a pixel-wise 1×1 convolution (trainable linear combination). The number of pre-defined filters p in (1.6) is a free parameter, usually chosen as $p = k^2$. The PFM is similar to depthwise separable convolution [35], which also has depthwise and pointwise convolution parts. However, our method does not adjust the filters in the depthwise part during training. Instead, our approach focuses on learning linear combinations of pre-defined filter outputs.

This research identifies a set of common edge and line detectors as a suitable choice for a broad range of vision problems. In images, edges are boundaries where intensity

values change sharply. Edges are prominent features in traditional computer vision as they can correspond to depth, reflectance, shadow boundaries, and discontinuities in surface orientation [13]. A simple method to extract edge information is convolution with first-order derivative kernels. Similarly, the convolution with second-order derivative kernels can detect thin lines. Common CNNs can develop such kernels during training [22] but it remains unclear how much they rely on these features in practice. The training data might provide incentives to use other features. We demonstrate that the processing of edge and line features in all convolutional layers of PFCNNs can enhance generalization. For PFCNNs, training is bound to finding linear combinations of edge and line filters. This adds transparency to the models and regularizes them, boosting the generalization on small-scale datasets. Our approach leads to increased test accuracies ranging from $5 - 11\%$ across four fine-grained classification datasets.

Additional experiments extract random features using randomly generated pre-defined filters. This setting sometimes increases test accuracy compared to traditional CNNs, although it does not perform as well as using edge and line filters. The results align with findings in the literature that discuss the effectiveness of random filters [23, 78].

Chapter 6 shows the experimental setup and the results of this research in detail. The results are published at the 2024 International Conference on Artificial Neural Networks (ICANN) [53]. The implementation of PFCNNs is available on `github.com/Criscraft/PredefinedFilterNetworks`.

Fifth Contribution: Parameter Reduction Using Pre-Defined Filter Networks

Popular CNN architectures contain millions or even billions of trainable weights, contributing to their opacity. Typically, they operate in the over-parameterized regime, also known as the modern interpolating regime [11]. Pruning experiments show that many of these weights serve no specific function and could be omitted without sacrificing performance [21]. It appears that the superiority of CNNs over traditional image recognition techniques comes at the cost of having a large number of superfluous weights, adding to the opacity of these models. A reduction in parameters would not only simplify the model but also save time, computational resources, and energy.

PFCNNs can significantly reduce the number of trainable parameters, resulting in lightweight models. The PFM, characterized by Equation (1.6), has pCC' parameters with the number of pre-defined filters p, the number of input channels C and the number of output channels C'. $p = 9$ results in the same number of parameters as a 3×3 convolution layer and $p = 1$ reduces the number by $1/9$. The total number of trainable parameters of ResNet18 shrinks from 11 million to only 1.5 million (13%).

However, preliminary experiments with $p = 1$ resulted in inferior performance. Convolving the same pre-defined filter with every input channel in the PFM affected per-

formance negatively. More variation in the filters is required to capture the diverse features in the input data. As a solution, we assign each input channel to one filter from a pool of k unique pre-defined filters. The modified PFM is called reduced PFM

$$\sum_{c=1}^{C} w_c \cdot \mathrm{ReLU}(h_{c \bmod k} * g_c)[m, n]. \tag{1.7}$$

Only the w_c are trainable. CNNs with our reduced PFMs perform similar and sometimes better on the test benchmarks compared to the baseline model Resnet18. The AM technique demonstrates that these simplified models still develop complex, object-specific features during training. The details of this research can be found in Chapter 7. They have been presented at the 2023 International Joint Conference on Neural Networks (IJCNN) [51].

1.4 Related Work

This section provides a literature overview and explains how the prior work is related to this research.

1.4.1 A Taxonomy for Explanation Methods

Ras et al. [79] proposed a useful taxonomy to structure the landscape of explanation methods. It exhibits three main categories: rule-extraction methods, attribution methods, and intrinsic methods. Rule-extraction methods aim to approximate the decision process of models by analyzing input-output pairs. Attribution methods identify relevant components in input samples and explain how they influence the outcome. Attribution methods form a broad group of methods, highlighting different aspects of models [14, 26, 79]. Our third contribution, which improves optimization during AM, as well as the popular Grad-CAM technique [82] fall under attribution methods. Intrinsic methods involve the design of the network architecture or the training procedure to enhance transparency. Our fourth and fifth contributions incorporate pre-defined filters into CNNs, placing them within intrinsic methods.

1.4.2 CNN Visualization in 2D Space

Prior software tools combine and present various visualizations in 2D. The Deep Visualization Toolbox [102] displays activations and per-unit feature visualizations of CNNs in a grid layout. Only a small fraction of the CNN can be displayed on the computer screen simultaneously. As a result, the CNN representation is scattered across several chunks of 2D space. Liu et al. [55] introduced CNNvis, a 2D tool to assess the neural network topology represented as a directed acyclic graph. In this example, the architecture and computational graph are visualized, but information about the activations

and the learned features is lacking. The CNN explainer software [97] shows both architecture and features. However, its 2D approach only works for small networks due to screen space restrictions.

1.4.3 CNN Visualization in 3D Space

Recent prototypes showed that 3D environments are well-suited for displaying small CNNs [12, 76, 81]. The ability to explore CNNs in 3D and immersive Virtual Reality (VR) enabled interactive exploration on different levels of detail. VanHorn et al. [94] developed an immersive environment enabling users to train and test networks with up to 10 layers in Virtual Reality (VR). While this tool was not intended as a generic analysis tool for arbitrary CNN architectures, it enabled users with limited knowledge of computational science to engage in deep learning. Aamir et al. [1] presented a new approach to immersively visualize and interpret deep networks in VR, enabling users to move freely inside an AlexNet [46]. They represented the convolutional layers as a sequence of 2D planes embedded in 3D space, an approach we adopted.

Recently, a novel approach to visualizing loss landscapes of neural networks in 3D space was presented. Li et al. [48] computed the loss landscape across two dimensions in weight space and rendered the loss as a surface in 3D. The characteristics of the shape and the roughness of these landscapes gave insights into the role of architecture design and activation functions in optimization processes.

Like CNNs, large biological networks are also challenging to visualize. Hisham and Mahmood [34] achieved this in 3D space using VR. Their interaction design suited the interaction with large networks. For instance, they proposed how users can reach specific nodes without wandering around in space. Pirch et al. [75] focused on displaying genomic data and molecular interaction networks. They found that 3D network representations allow for quickly and accurately resolving connections that would otherwise be ambiguous. 3D space allowed them to visualize larger networks.

The literature overview shows that there is still a lack of holistic views of CNNs and that 3D space is suited to present complex data. Our second contribution DeepVisionVR enables human-understandable network representations in a virtual 3D space. We augment and optimize previous 3D visualization approaches for displaying large, popular architectures like residual, dense, or Inception networks.

1.4.4 Activation Maximization

In 2009, Erhan et al. used AM to obtain qualitative interpretations of features from neural networks trained on several vision datasets [19]. They aimed to gain insight into what a particular neural network unit represents. The authors initialized an input image with noise and iteratively modified it via back-propagation, as depicted by Equation (1.2), to maximize the activation of specific network components. However, the

generated images often appear unnatural because the CNNs were trained for classification tasks, not image generation. Let us assume a joint probability distribution $p(\mathbf{x}, y)$ with images $\mathbf{x}$ and categorical labels y. The distribution can be decomposed into an image prior and a classification probability distribution: $p(\mathbf{x}, y) = p(\mathbf{x})p(y|\mathbf{x})$. $p(y|\mathbf{x})$ is often modeled via computing the softmax of the network output. Neglecting the image prior $p(\mathbf{x})$ usually leads to uninterpretable images dominated by high-frequency patterns. Prior work proposed methods to regularize the generated images toward certain domains, e.g. smoother images with less high-frequency components. One sophisticated approach is to add regularization terms to the objective function in (1.2) [60, 61]. Natural-looking images tend to have regions of similar intensities, so an L2 loss can prevent spiking intensities. Also, penalizing the variation of neighboring pixels encourages the formation of larger structures in the image. A useful alternative to modifying the objective function is applied in the work of Yosinski et al. [102]. They employed image transformations that map the images to more regularized versions of themselves. Instead of reducing the L2 loss explicitly, they applied L2 normalization to the image between iterations. Analogously, they applied Gaussian blur to the image instead of introducing a loss for neighboring pixel variation. These robustness transformations effectively improve the visual quality of the generated images [68, 70].

Activation maximization plays a prominent role in XAI, focusing on image recognition [51, 72, 83]. Szegedy et al. [90] discovered that AM can produce adversarial examples. For initialization, they chose an image showing a specific class. Subsequently, they slightly modified the image to activate another class. A short transition vector in image space fooled the network to predict the other class, while the visual difference was imperceptibly small to humans. Nguyen et al. [70] generated artificial images that are unrecognizable for humans, but networks trained on ImageNet [80] or MNIST [47] classified them as specific classes with high confidence. These examples highlight the potential vulnerability of deep learning models to small perturbations and the importance of understanding their behavior.

Our third contribution discusses specific optimization issues encountered during activation maximization. The literature discusses optimization issues focusing on training the weights of deep neural networks [18, 57, 98]. To our knowledge, the optimization issues of AM are overlooked in the existing literature. We attribute them to the activation functions ReLU and Leaky ReLU and propose a method to mend these issues, thereby increasing the maxima found by the algorithm.

1.4.5 Pre-defined Filters in CNNs

The concept of pre-defined filters has a rich history in computer vision. In the current era of deep networks, pre-defined filters are occasionally employed as a preprocessing step to enhance recognition performance. For instance, Ma et al. [58] used pre-defined filters to incorporate domain knowledge into their training process by replacing the

first convolutional layer of CNNs with trainable traditional image filters (such as Gabor filters [25]).

Using pre-defined filters also in the intermediate layers of CNNs appears to be a novel concept. Gavrikov and Keuper [23] showed that learning linear combinations of pre-defined, random filters effectively solves image classification problems, especially when the CNNs are wide (many channels per layer). They also showed that these random filters can be shared across layers to reduce the number of weights.

Wimmer et al. [99] expressed 3×3 convolution kernels through various spanning vectors as detailed in equation (1.5). Their approach, termed interspace pruning, aims to reduce the number of trainable parameters by removing a specific number of spanning vectors. The resulting convolution kernels live in a learned subspace optimized using the training data. Our research focuses on regularization rather than pruning. Our approach does not adjust the pre-defined filers to the training data. Instead, we show that the intentional choice of the pre-defined filters introduces biases that can improve generalization and transparency. Nonetheless, our approach can also save trainable parameters as demonstrated in Chapter 7.

1.5 Thesis Organization

Chapter 2 depicts the fundamentals of this study. It overviews the classification problem, datasets, architectures, optimizers, and training procedures. The subsequent Chapter 3 motivates the need for more holistic visualization methods. Chapter 4 depicts the representation of CNNs in immersive 3D environments. Subsequently, Chapter 5 presents optimization challenges encountered during AM and how to overcome them. Then, Chapter 6 introduces pre-defined filters to CNNs to systematically simplify and structure their information processing. Chapter 7 shows how pre-defined filters can reduce the number of parameters of CNNs, leading to lightweight models for image recognition. Chapter 8 provides an in-depth analysis of the findings and discusses their implications. Chapter 9 concludes the dissertation.

Chapter 2

Fundamentals

2.1 The Classification Problem

The machine learning problems discussed in this research live in the domain of image classification. Inspired by a helpful overview of machine learning principles in [64], this section defines the classification problem and derives the cross-entropy loss used in the experiments. Here, a classifier receives inputs x drawn from an input distribution $P(x)$. The classifier assigns a class label y using a classifier function $h_\theta(x)$, typically parameterized by some weights θ. The input x has a true label y with probability $P(y|x)$. For each input x, the classifier produces a loss $\mathcal{L}(y, h_\theta(x)) \in \{0, 1\}$, 0 if the classification is correct and 1 if the classification is incorrect. The error of the classifier on the whole data distribution $P(x, y) = P(y|x)P(x)$ is given by the expected loss

$$E(h_\theta(x)) = \int \mathcal{L}(y, h_\theta(x))P(x, y)dxdy.$$

$E(h_\theta)$ is also called true error, true loss, or true risk.

The classifier h_θ is selected from a function set $\mathcal{H}$, which, for instance, may be determined by the architecture of a neural network. Learning means the process of choosing an h_θ from the set $\mathcal{H}$ to minimize $E(h_\theta)$, realized by choosing a specific θ. This is commonly achieved through Empirical Risk Minimization (ERM). Let $D = \{(\mathbf{x_1}, y_1), ..., (\mathbf{x_k}, y_k)\}$ be a so-called training set comprising k input samples $\mathbf{x_i} \in \mathbb{R}^n$

together with their labels $y_i \in \{1, ..., C\}$, independently drawn from $P(x, y)$. Empirical risk is defined as the average loss on the training set

$$E_D(h_\theta) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(y_i, h_\theta(\mathbf{x_i})). \tag{2.1}$$

$E_D(h_\theta)$ is commonly referred to as the training error or empirical loss. Learning via ERM chooses an $h_\theta \in \mathcal{H}$ that minimizes $E_D(h_\theta)$.

While the loss function in (2.1) describes the classification problem intuitively, it is not useful for optimization via gradient descent. The loss function exhibits plateaus where the gradient is zero. Also, the function reaches its minimum (zero) if all training samples are correctly classified, terminating the training process. A more useful loss function should consider the probability that $\mathbf{x_i}$ belongs to class y:

$$P(y|\mathbf{x_i}, \theta) = (f_\theta(\mathbf{x_i}))_y. \tag{2.2}$$

Therefore, we will train the model f_θ to reflect the distributions of our training data D and classify new datapoints using $h_\theta(x_i) = \arg\max_y (f_\theta(\mathbf{x_i}))_y$. Our model $f_\theta : \mathbb{R}^n \to [0, 1]^C$ should be normalized $\sum_{j=1}^{C} (f_\theta(\mathbf{x_i}))_j = 1$ such that its outputs can be interpreted as probabilities. Let us now view (2.2) as a function of the parameters θ instead of the data $\mathbf{x_i}$. This change of perspective makes (2.2) the likelihood, which is the probability that a specific outcome y is observed when the true parameter is θ. Here, we assume fixed data $\mathbf{x_i}$. Following the principle of maximum likelihood estimation, the likelihood of observing the training data using our model is:

$$L(D, \theta) = \prod_{i=1}^{n} (f_\theta(\mathbf{x_i}))_{y_i}. \tag{2.3}$$

The logarithm of the likelihood is often called the log-likelihood:

$$l(D, \theta) = \sum_{i=1}^{n} log(f_\theta(\mathbf{x_i}))_{y_i}. \tag{2.4}$$

This gives us the negative log-likelihood loss and the corresponding optimization problem of finding:

$$\theta^* = \arg\min_\theta \mathcal{L}(D, \theta) = \arg\min_\theta \; -l(D, \theta). \tag{2.5}$$

The log-likelihood loss is directly related to the cross-entropy, which can be seen when adjusting the formulation of our model:

$$(f_\theta(\mathbf{x_i}))_y = \prod_{j=1}^{C} (f_\theta(\mathbf{x_i}))_j^{(j=y)}. \tag{2.6}$$

Here, the equality $(j = y)$ is evaluated as 1 if the equality holds and 0 if not. This formulation is a handy way to choose an element of the vector of probabilities using exponents. In the negative log-likelihood loss function, the exponent can be moved before the logarithm:

$$\mathcal{L}(D, \theta) = -l(D, \theta) = -\sum_{i=1}^{n}\sum_{j=1}^{C}(j = y_i)\log[(f_\theta(\mathbf{x_i}))_j] = -\sum_{i=1}^{n}\sum_{j=1}^{C}p(j)\log q(j). \quad (2.7)$$

$p(j) = (j = y_i)$ describes the probability density over the true labels. The term $q(j) = (f_\theta(\mathbf{x_i}))_j$ describes the predicted probabilities of the classes. The right-hand side of (2.7) is also the cross-entropy of these distributions. More details about this topic can be found in the article [64].

A common way to confine the model outputs to values between 0 and 1 and to ensure the normalization $\sum_{j=1}^{C}(f_\theta(\mathbf{x_i}))_j = 1$ is given by the softmax function

$$\text{softmax}(z)_i = \frac{\exp z_i}{\sum_{j=1}^{C}\exp z_j} \quad (2.8)$$

with $z \in \mathbb{R}^C$. Notably, the Pytorch module for the cross-entropy loss already applies the softmax function to the model output. Alternatively, the loss can be realized by sequentially applying Pytorch's modules for softmax and the negative log-likelihood loss.

2.2 Datasets

Figure 2.1 presents the image classification datasets used in this research. Table 2.1 provides details about the datasets, including the number of images and classes.

The Caltech101 dataset [49] contains a total of 8677 images collected from the internet. These images are scaled to approximately 300 pixels wide. They cover 101 categories, along with an ignored background class. The categories are highly interpretable for humans and include familiar objects such as consumer products, vehicles, plants, animal species, and various building types. The objects are presented in cluttered environments or against realistic or white backgrounds.

The CIFAR10 dataset [45] contains 50000 training images and 10000 test images of the size 32×32 pixels. The dataset provides 10 classes: plane, car, bird, cat, deer, dog, frog, horse, ship, and truck. For our experiments on the CIFAR10 dataset, we adjust the PFNet18 (see Chapter 7) and ResNet18 [30] (see Section 2.3) architectures to be compatible with small image shapes. In the first layer, this procedure reduces the stride to 1 and the kernel size to 3. In addition, the max pooling layer is removed.

The Caltech-UCSD Birds-200-2011 dataset (CUB) [95] from 2010 consists of 5994 training and 5794 test images with an average width of about 470 pixels. This dataset contains 200 bird species, each represented with around 30 training images. The images

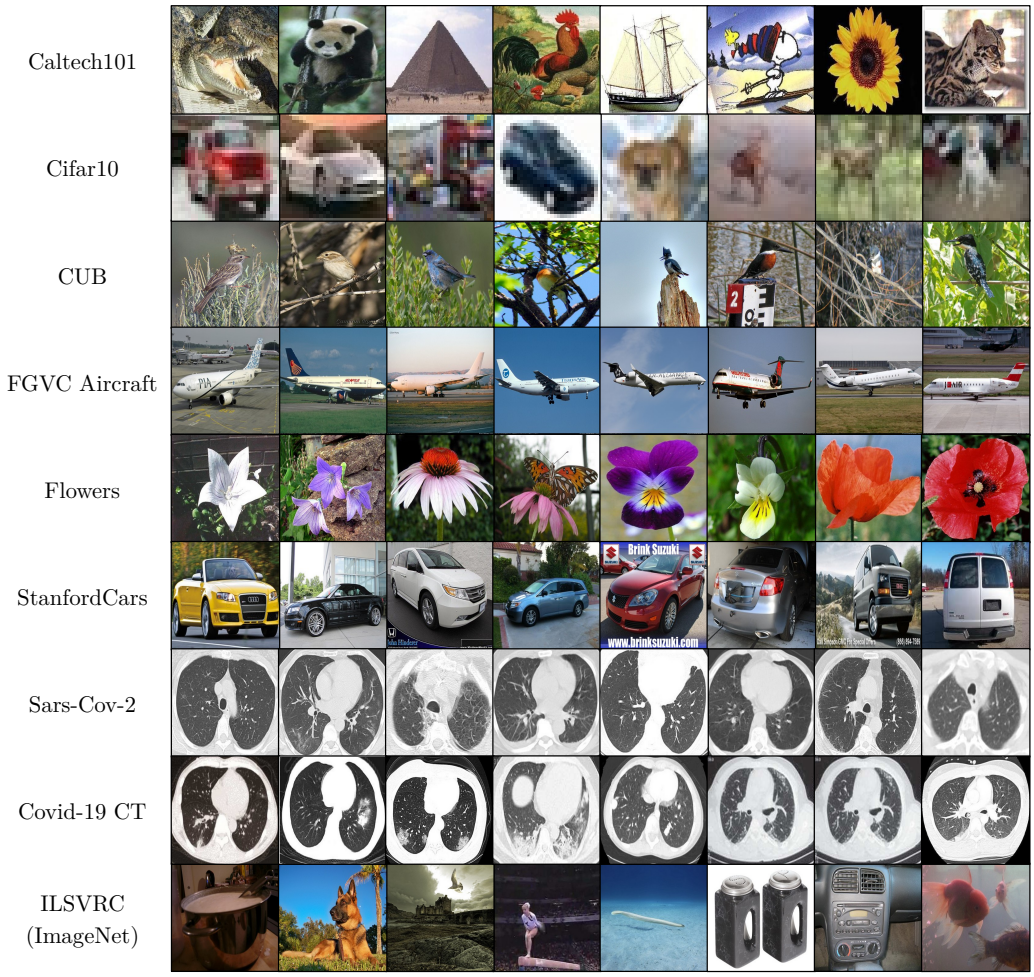


Figure 2.1: Overview of image classification datasets.

Table 2.1: Details of the image classification datasets.

Dataset	Mode	Classes	Number of images in/of... dataset	smallest class	largest class	avg. per class
Caltech101	-	101	8677	31	800	85.91
CIFAR10	train	10	50000	5000	5000	5000
	test		10000	1000	1000	1000
CUB	train	200	5994	29	30	29.97
	test		5794	11	30	28.97
FGVC Aircraft	train	100	6667	66	67	66.67
	test		3333	33	34	33.33
Flowers	train	102	2040	20	20	20.00
	test		6149	20	238	60.28
Stanford Cars	train	196	8144	24	68	41.55
	test		8041	24	68	41.03
SARS-CoV-2	-	2	2482	1230	1252	1241
COVID19-CT	-	2	746	349	397	373
ImageNet (ILSVRC)	train	1000	1281167	732	1300	1281
	test		50000	50	50	50

often show birds in their natural habitats, including clutter, foliage, tree branches, or flowers in the background. Furthermore, the bird categories have much intra-class variation in plumage color, pose, deformation, lighting, and perspective. These factors contribute to the dataset's complexity and make it particularly challenging for object recognition tasks.

The Fine-Grained Visual Classification of Aircraft dataset (FGVC Aircraft) [62] was published in 2013 and consists of 6667 training and 3333 test images showing 100 distinct airplane models. The images have wide aspect ratios and an average width of around 1100 pixels. The dataset is divided into an official training set and a test set with approximately 67 and 33 images per class. Planes are rigid objects that do not deformation. However, significant intra-class variation exists in the visual appearance due to various factors such as advertisement, airlines, and perspective.

The 102 Category Flower dataset (Flowers) [73] contains 102 different blossom types. Each image shows one or several instances of the respective blossom. The dataset was published in 2008, and the images have a mean width of 630 pixels. It contains strong intra-class variations encompassing scaling, perspective, lighting, and color variants. Unlike other datasets, Flowers includes a train, validation, and test set. To maintain consistency with the other datasets, the official training and validation sets are merged into one training set with 2040 images and one test set with 6149 images.

The StanfordCars dataset [44], introduced by Stanford University in 2013, consists

of 8144 training and 8041 test images. It contains 196 different car models with an average width of 700 pixels. Each image typically shows a single car depicted in various environments and perspectives. The dataset shows cars on roads or indoors, captured from the front, side, or rear, introducing rich variability.

The CT scan images of the SARS-CoV-2 dataset [85] were collected from hospitals in Sao Paulo, Brazil. This dataset contains 2482 images from 120 individuals. 1252 images depict CT slices from patients diagnosed with COVID-19, while the other 1230 images show manifestations of other lung diseases. Thus, the image recognition model has to focus on the characteristics of the manifestations of the COVID-19 infection. The CT scans have varying spatial sizes between 104×119 and 416×512 pixels. In Figure 2.1, the first four images show CT slices from patients infected with COVID-19, while the other four images depict other lung diseases.

The COVID19-CT dataset [32] consists of 746 CT images. There are 349 CT images of patients with COVID-19 and 397 CT images showing other pulmonary diseases. The positive CT images were collected from preprints about COVID-19 on medRxiv and bioRxiv, and they feature various manifestations of COVID-19. Since the CT images were taken from different sources, they have varying sizes between 124×153 and 1485×1853 pixels. Similar to the SARS-CoV-2 dataset, Figure 2.1 shows four CT slices from patients infected with COVID-19 and four images of other lung diseases.

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [80] is a well-known benchmark for object classification and detection. The classification dataset contains images depicting 1000 classes. These images were extracted from various platforms, including Flickr, and were manually labeled with one category. Image collection involved queries to image search engines based on WordNet synonyms for each synset. Additionally, the candidate pool of images was diversified by translating these queries into multiple languages. The annotation process relied on human verification via Amazon Mechanical Turk. The ILSVRC evolved as an extension of the bigger ImageNet dataset [16] that continued to expand. The 1000 categories were selected from the broader ImageNet categories, ensuring no overlap between synsets. Over the years, the selection of synsets changed, with 639 synsets remaining consistent across all five ILSVRC challenges. This work uses the ILSVRC from 2012. There are 1281167 images for training and 50000 images for validation. In the training set, the number of images for each synset ranges from 732 to 1300. The images vary in dimensions and resolution.

2.3 Neural Network Architectures

2.3.1 DenseNet

Densely connected convolutional Networks (DenseNets) [36] alleviate the vanishing gradients problem, promote feature reuse, and achieve high recognition performance.

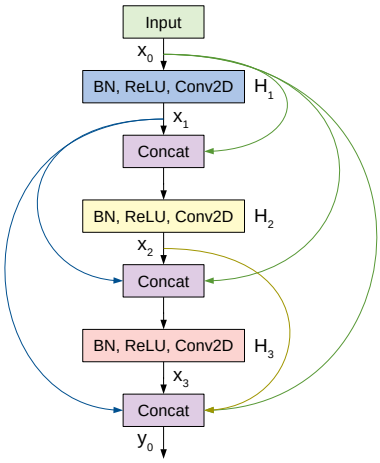


Figure 2.2: Illustration of a dense block.

DenseNets connect each layer to every other layer within a dense block. Figure 2.2 shows a 3-layer dense block where each layer performs a sequence of batch normalization, ReLU activation, and convolution. The input to the next layer will be the outputs of all the preceding layers in a dense block. Each layer generates k new feature maps, where k is a newly introduced hyper-parameter denoted as the growth rate. Let the initial layer of a dense block have c_0 channels. Then, the number of channels at the end of a 3-layer dense block is $c_0 + 3k$. DenseNet introduces a bottleneck layer with 1×1 convolution and $4k$ filters to prevent the number of channels from increasing too rapidly. To tackle the difference in the feature map sizes when transitioning from a large feature map to a smaller one, DenseNet applies a transition layer made of 1×1 convolution and average pooling. A deep DenseNet is constructed by stacking multiple dense blocks with transition layers. Conventional convolution and pooling layers are used at the beginning of the network. DenseNet121, a variant of DenseNet, has about 7 million parameters. It is described in detail in Table 2.2.

We use DenseNets in various experiments. Chapter 6 augments DenseNet121 with pre-defined filters to increase the transparency and the generalization abilities of the architecture. Chapter 3 applies DenseNet to distinguish COVID-19 CT scan images from other lung diseases. Recently, CovidDenseNet [4] was proposed to detect COVID manifestations in CT scan images. It has a total of 1.63 million parameters. Chapter 4 visualizes CovidDenseNet in an immersive 3D environment.

2.3.2 Inception

The Inception network is a deep convolutional architecture introduced as GoogLeNet (Inception V1) in 2014 by Szegedy et al. [89]. The architecture has been refined in

Table 2.2: Details of the DenseNet121 architecture. The output sizes are determined for an input size of 224×224 pixels.

Layers	Output size	Kernel
Convolution	$64 \times 112 \times 112$	7×7, stride 2
Max Pool	$64 \times 56 \times 56$	-
Dense Block 1	$256 \times 56 \times 56$	$\begin{bmatrix} 128 \times 1 \times 1 \\ 32 \times 3 \times 3 \end{bmatrix} \times 6$
Transition 1	$128 \times 56 \times 56$	1×1
	$128 \times 28 \times 28$	Average pool
Dense Block 2	$512 \times 28 \times 28$	$\begin{bmatrix} 128 \times 1 \times 1 \\ 32 \times 3 \times 3 \end{bmatrix} \times 12$
Transition 2	$256 \times 28 \times 28$	1×1
	$256 \times 14 \times 14$	Average pool
Dense Block 3	$1024 \times 14 \times 14$	$\begin{bmatrix} 128 \times 1 \times 1 \\ 32 \times 3 \times 3 \end{bmatrix} \times 24$
Transition 3	$512 \times 14 \times 14$	1×1
	$512 \times 7 \times 7$	Average pool
Dense Block 4	$1024 \times 7 \times 7$	$\begin{bmatrix} 128 \times 1 \times 1 \\ 32 \times 3 \times 3 \end{bmatrix} \times 16$
Classification layer	$1024 \times 1 \times 1$	Adaptive average pool
		fully connected, softmax

various ways, such as adding batch normalization layers to accelerate training (Inception V2 [38]) and factorizing convolutions with larger spatial filters for computational efficiency (Inception V3 [91]). The fundamental building block for all Inception-style networks is the Inception module of which several forms exist. Figure 2.3 shows one variant of the Inception module that is used in the InceptionV3 model. The module branches into four different paths. The input passes through convolutional layers with different kernel sizes (1×1, 3×3, and 5×5) and a pooling operation. Applying different kernel sizes allows the module to capture complex patterns at different scales. The outputs of all branches are concatenated channel-wise.

The overall architecture of the InceptionV3 network is composed of conventional 3×3 convolutional layers at the early stages of the network, where some of these layers are followed by max-pooling operations. Subsequently, a stack of various Inception modules is applied. These modules have different designs concerning the number of applied filters, filter sizes, depth of the module after symmetric or asymmetric factorization of larger convolutions, and when to expand the filter bank outputs. The last Inception module is followed by an average pooling operation and a fully connected layer. The overall architecture has about 22 million parameters.

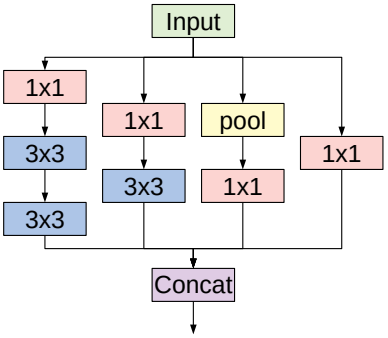


Figure 2.3: A variant of the Inception module used in the InceptionV3 architecture.

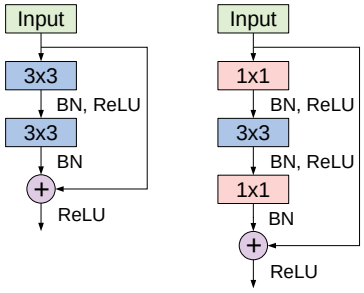


Figure 2.4: Illustration of the basic block (left) and the bottleneck module (right) of ResNet.

This research employs InceptionV3 to distinguish COVID-19 CT scan images from other lung diseases in Chapter 3.

2.3.3 ResNet

Deep Residual Networks (ResNets), proposed by He et al. [30], represent a family of CNN architectures known for their success in various computer vision tasks. ResNets won the ILSVRC-2015 challenges for image recognition, object detection, and localization [80]. The winning network was composed of 152 layers, confirming the beneficial impact of network depth on recognition performance. However, as network depth increases, two challenges emerge: the vanishing gradient problem and performance degradation. They observed that stacking more and more layers can lead to inferior recognition performance. If the additional layers would learn identity mappings, the loss of a deeper model should not be higher than a shallow model. Hence, the performance degradation problem suggested that layers struggle to learn identity mappings during training. Their solution was to learn a desired mapping $H(x)$ by its residual

Table 2.3: Details of the ResNet18 architecture. The output sizes are determined for an input size of 224×224 pixels.

Layers	Output size	Kernel
Convolution	$64 \times 112 \times 112$	7×7, stride 2
MaxPooling	$64 \times 56 \times 56$	-
Basic Block	$64 \times 56 \times 56$	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix} \times 2$
Basic Block	$128 \times 28 \times 28$	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix} \times 2$
Basic Block	$256 \times 14 \times 14$	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix} \times 2$
Basic Block	$512 \times 7 \times 7$	$\begin{bmatrix} 3 \times 3 \\ 3 \times 3 \end{bmatrix} \times 2$
Classification layer	$512 \times 1 \times 1$	Adaptive average pool
		fully connected, softmax

function $H(x) - x$. This formulation preconditions the problem and improves information flow as the network depth increases.

Residual networks are characterized by residual modules, of which two variants are depicted in Figure 2.4. The basic block, utilized in the ResNet18 architecture, consists of two convolutional layers, each employing a 3×3 kernel and a skip connection. The bottleneck block, commonly employed in ResNet50 and deeper variants, contains a sequence of convolutions using both 3×3 and 1×1 kernel sizes. A deep residual network is constructed by stacking multiple residual modules on top of each other along with other conventional convolution and pooling layers. For example, the structure of ResNet18 is outlined in Table 2.3. The architecture has about 11 million trainable parameters.

ResNet is a central architecture in our research. Chapters 5, 6, and 7 employ ResNet18 as a baseline model for comparison against other architectures. Chapters 6 and 7 augment ResNet18 with pre-defined filters to increase the transparency and the generalization abilities of the architecture. Recently, CovidResNet [4] was proposed to detect COVID manifestations in CT scan images. With only 4.98 million trainable parameters, this variant has proven to generalize well on small-scale CT datasets with a few thousand images. Chapter 4 demonstrates how CovidResNet can be represented in an immersive 3D environment.

2.3.4 VGG

In 2014, Simonyan and Zisserman proposed a simple feed-forward CNN architecture VGGNet (Visual Geometry Group Network), or short VGG [84]. It consists of a series of convolutional layers followed by ReLU activation and max-pooling layers.

Block	Layer	Output Size
Block 1	Convolution	$64 \times 224 \times 224$
	Convolution	$64 \times 224 \times 224$
	Max-Pooling	$64 \times 112 \times 112$
Block 2	Convolution	$128 \times 112 \times 112$
	Convolution	$128 \times 112 \times 112$
	Max-Pooling	$128 \times 56 \times 56$
Block 3	Convolution	$256 \times 56 \times 56$
	Convolution	$256 \times 56 \times 56$
	Max-Pooling	$256 \times 28 \times 28$
Block 4	Convolution	$512 \times 28 \times 28$
	Convolution	$512 \times 28 \times 28$
	Max-Pooling	$512 \times 14 \times 14$
Block 5	Convolution	$512 \times 14 \times 14$
	Convolution	$512 \times 14 \times 14$
	Average Pooling	$512 \times 7 \times 7$
Fully Connected	FC, 1024 Neurons	
	FC, 1024 Neurons	
	FC, 100 Neurons	

Table 2.4: Details of a VGG13-based architecture. The output sizes are determined for an input size of 224×224 pixels.

Characterized by its simplicity, VGG is still a popular choice for image classification tasks. Table 2.4 shows the architecture of VGG13, which has 36.3 million parameters.

Our research uses VGG to study Pre-defined Filter Networks in Chapter 6. In our experiments, we augment VGG with batch normalization layers to accelerate training. Moreover, we reduced the size of the fully connected layers to have 1024 neurons to save computational resources and training time.

2.4 LAMB Optimizer

This research employs the LAMB optimizer [103] to adjust the weights of deep neural networks to the training data. LAMB means "Layer-wise Adaptive Moments optimizer for Batch training" and enhances the ADAM optimizer [41]. While LAMB was introduced to have stable convergence on very large batch sizes, it also has advantages for smaller batch sizes. This section introduces the strategy of the optimizer and explains how the algorithm works. The pseudocode of the LAMB optimization algorithm [103] is:

Require: $x_1 \in \mathbb{R}^d$, learning rate $\{\eta_t\}_{t=1}^{T}$, parameters $0 < \beta_1, \beta_2 < 1$, scaling function $\phi, \epsilon > 0$

$m_0 \leftarrow 0$

$v_0 \leftarrow 0$

for $t = 1$ to T **do**
Draw b samples S_t from $\mathbb{P}$.
$g_t \leftarrow \frac{1}{|S_t|} \sum_{s_t \in S_t} \nabla \ell(x_t, s_t)$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
$m_t \leftarrow \frac{m_t}{1-\beta_1^t}$
$v_t \leftarrow \frac{v_t}{1-\beta_2^t}$
$r_t \leftarrow \frac{m_t}{\sqrt{v_t}+\epsilon}$
$x_{t+1}^{(i)} \leftarrow x_t^{(i)} - \eta_t \frac{\phi(||x_t^{(i)}||)}{||r_t^{(i)}+\lambda x_t^{(i)}||}(r_t^{(i)} + \lambda x_t^{(i)})$
end for

In iteration i, the model weights x_i are updated in a certain direction

$$\frac{r_t^{(i)} + \lambda x_t^{(i)}}{|| \, r_t^{(i)} + \lambda x_t^{(i)} \, ||} \tag{2.9}$$

by a certain amount $\eta_t \cdot \phi(|| \, x_t^{(i)} \, ||)$. The latter is determined by the global learning rate η_t and a scaling function $\phi : \mathbb{R}^+ \to \mathbb{R}^+$ that depends on the norm of the weights:

$$\phi(|| \, x_t^{(i)} \, ||) = \min(\max(|| \, x_t^{(i)} \, ||, \lambda_l), \lambda_h)$$

with two constants λ_l and λ_h with $\lambda_l < \lambda_h$. Thus, layers with large weights receive a larger update step than those with small weights. As a result, the LAMB algorithm applies different effective learning rates to each layer.

The direction of the weight update is similar to the ADAM optimizer [41]. $r_t^{(i)}$, the first term in (2.9), contains the moving average of the gradient, providing momentum. The gradient is divided by the square root of its second moment. Intuitively, a parameter with low variance indicates a flat error surface. The update step is increased for faster convergence. A parameter with high variance indicates a steep error surface, requiring a more conservative update. The second term, $\lambda x_t^{(i)}$, implements weight decay. Combining these strategies, the LAMB optimizer converges quickly and smoothly.

2.5 Training Procedures

2.5.1 Default Training Procedure

The network weights are initialized using Kaiming initialization [31]. The LAMB optimizer [103] minimizes the cross-entropy loss with a batch size of 64. The training consists of 300 epochs, during which the initial learning rate of $\mathrm{lr}_{\mathrm{init}} = 0.003$ is stepwise reduced to $\mathrm{lr}_{\mathrm{init}} \cdot 10^{-3}$. Additionally, we set the weight decay parameter to 1. We use the Pytorch framework [74] to implement and train the models. We ensure robustness and reproducibility by repeating the experiments 5 times with different

random seeds. Each seed influences various factors, including weight initialization, mini-batch aggregation, and pseudo-random effects during data augmentation. For image augmentation, we incorporate random cropping and random horizontal flipping. The augmentations enhance the diversity of the training data, contributing to improved model robustness and generalization.

2.5.2 ILSVRC Training Procedure

The networks are trained with a batch size of 48 on five NVIDIA GeForce RTX 4090 GPUs. The remaining training hyperparameters are taken from the training reference of PyTorch [74]. The cross-entropy loss is minimized using stochastic gradient descent for 90 epochs with a momentum of 0.9 and weight-decay 0.0001. The initial learning rate of 0.1 is reduced by a factor of 0.1 every 30 epochs.

2.5.3 Finetuning Training Procedure

The default finetuning procedure utilizes networks that were pre-trained on the ImageNet dataset [80]. All layers of the pre-trained networks are adjusted to the new training data using the LAMB optimizer. As this research applies finetuning only to binary classification problems, we use the binary cross-entropy loss. The initial learning rate is set to 0.0003 and is step-wise reduced to 0.000003. We use a batch size of 32 and a high weight decay of 1. The networks are trained for 100 epochs. Data augmentation methods effectively increase the number of training samples for improved generalization. The augmentation steps include cropping, adding blur with a probability of 25%, adding a random amount of Gaussian noise, changing brightness and contrast, and random horizontal flipping. The images are normalized using the mean and standard deviation of the ImageNet dataset. The finetuning training procedure is applied in Chapter 3 to train deep networks detecting COVID-19 manifestations in chest CT scans.

Chapter 3

Explainable COVID-19 Detection Using Chest CT Scans

This chapter critically discusses the ability of two visualization methods, t-SNE [93] and Grad-CAM [82], to explain CNN decisions in the domain of medical image data, i.e. the detection of COVID-19 using chest CT scans. The chapter starts with depicting the experimental setup in Section 3.1. In Section 3.2 we apply the t-SNE algorithm. Subsequently, Section 3.3 presents the localization maps for highlighting regions relevant for the network decisions using Grad-CAM.

The results of this chapter are published in Sensors [8]. The author contributions include: Conceptualization, H.A. and C.L.; data curation, H.A. and C.L.; formal analysis, H.A.; funding acquisition, T.M.; investigation, H.A. and C.L.; methodology, H.A. and C.L.; project administration, T.M.; validation, H.A. and C.L.; visualization, H.A. and C.L.; writing—original draft, H.A.; writing—review and editing, H.A., C.L., E.B., and T.M. Here, C.L stands for Christoph Linse, H.A. for Hammam Alshazly, E.B. for Erhardt Barth, and T.M. for Thomas Martinetz.

3.1 Experimental Setup

We train the architectures InceptionV3 [91] and DenseNet169 [36] on the SARS-CoV-2 CT [85] and COVID19-CT [32] datasets, respectively. The architectures differ in their design philosophy as depicted in Section 2.3. They are pre-trained on the ImageNet dataset [80] and are subsequently fine-tuned on the SARS-CoV-2 CT or COVID19-CT dataset. The datasets establish binary classification problems with images showing CT slices of COVID-19 infected patients and CT slices of other lung diseases. Section

2.2 exhibits detailed information about the two datasets. All details of the training procedure are described in Section 2.5.3. During training, all layers of the models are adjusted to the new data. As the datasets have no official train and test splits, we apply 5 fold cross-validation. After finetuning, InceptionV3 reaches a test accuracy of 99.1 ± 0.5 percent on the SARS-CoV-2 CT dataset and DenseNet169 a test accuracy of 91.2 ± 1.4 percent on the COVID19-CT dataset.

3.2 t-SNE Visualization Results

The t-SNE algorithm visualizes the distribution of the samples in the high-dimensional feature space. For each image in the SARS-CoV-2 dataset we extract the 2048-dimensional feature vector from the penultimate layer of the InceptionV3 model. Next, t-SNE maps the feature vectors to a 2D space. Figure 3.1 shows two well-separated clusters of COVID-19 and non-COVID-19 samples. The clear and wide margin between the two classes and the similar distributions of train and test samples indicate good generalization capabilities of the model. In the SARS-CoV-2 dataset, some CT images originate from the same individuals. In total, the dataset includes 2482 CT scans acquired from 120 patients. It could lead to bad generalization if the model tries to detect the individuals instead of the disease. t-SNE shows that the feature space of the model is well separated with respect to the classes, not the individuals.

For the COVID19-CT dataset, the feature vectors are extracted from the penultimate layer of the DenseNet169 model (1664 dimensions). Figure 3.2 shows two clusters representing CT images for the COVID-19 and non-COVID-19 classes. Even though the classes are fairly distinguishable, some CT images are misclassified, specifically some non-COVID-19 CT images from the test set.

3.3 Grad-CAM Visualization Results

We provide the Grad-CAM localization maps for various CT images of the COVID-19 class. Figure 3.3 shows CT images from the SARS-CoV-2 CT test set and their localization maps. The InceptionV3 model correctly classifies them as COVID-19 cases and highlights the regions of abnormalities in the CT scans, which are important for the model's decision.

In a similar way, we consider the test CT scans from the COVID-19 dataset using the DenseNet169 model. Figure 3.4 presents CT images and their localization maps. Notably, the model is capable of detecting the COVID-19-related regions as annotated (small square in some images) by expert radiologists.

A wide variety of typical and atypical chest CT abnormalities of COVID-19 patients have been reported in various studies [29, 100]. In order to investigate the ability of our models to identify COVID-19 cases outside the considered datasets and localize their

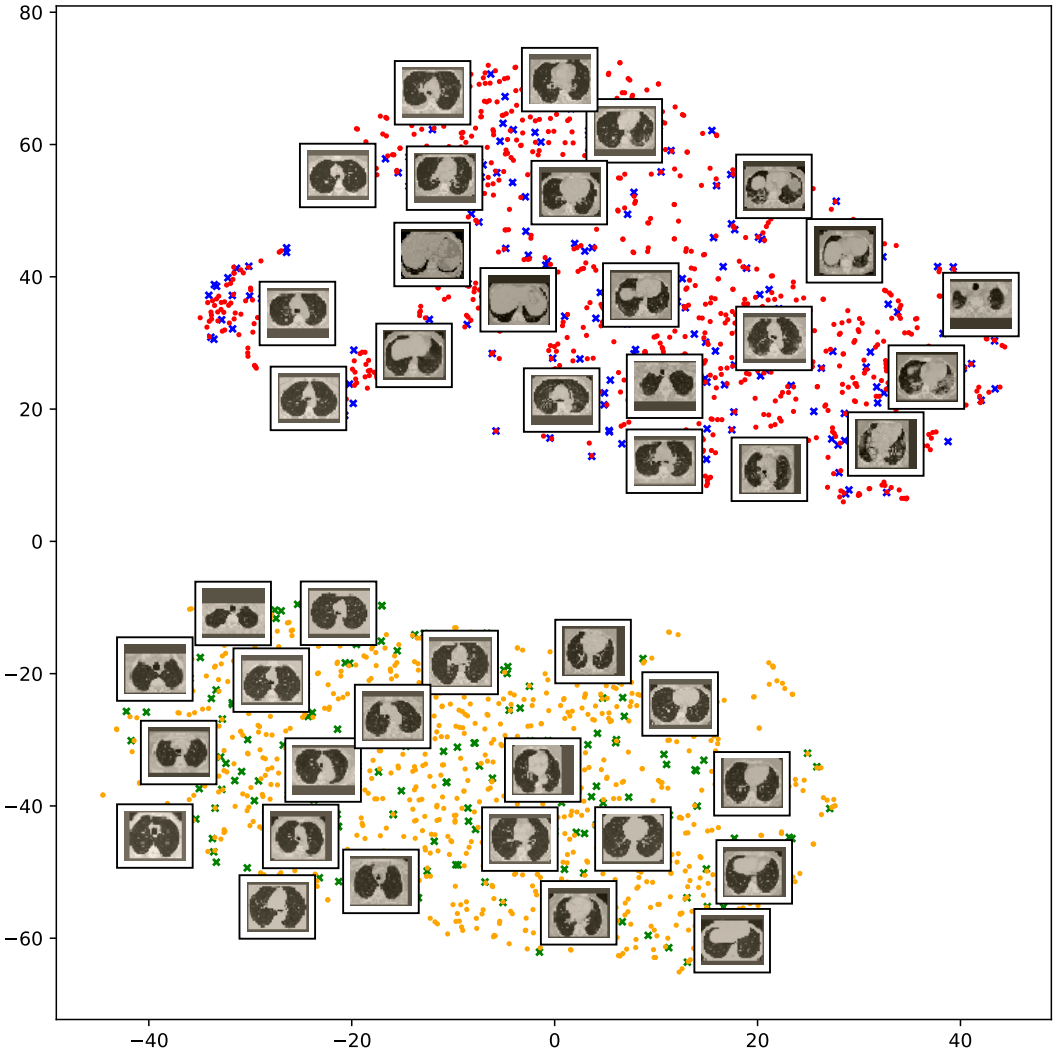


Figure 3.1: Visualization of the t-SNE embeddings for the entire SARS-CoV-2 CT dataset. The clusters represent COVID-19 and non-COVID-19 classes. Red: COVID-19 train samples. Blue: COVID-19 test samples. Yellow: non-COVID-19 train samples. Green: non-COVID-19 test samples.

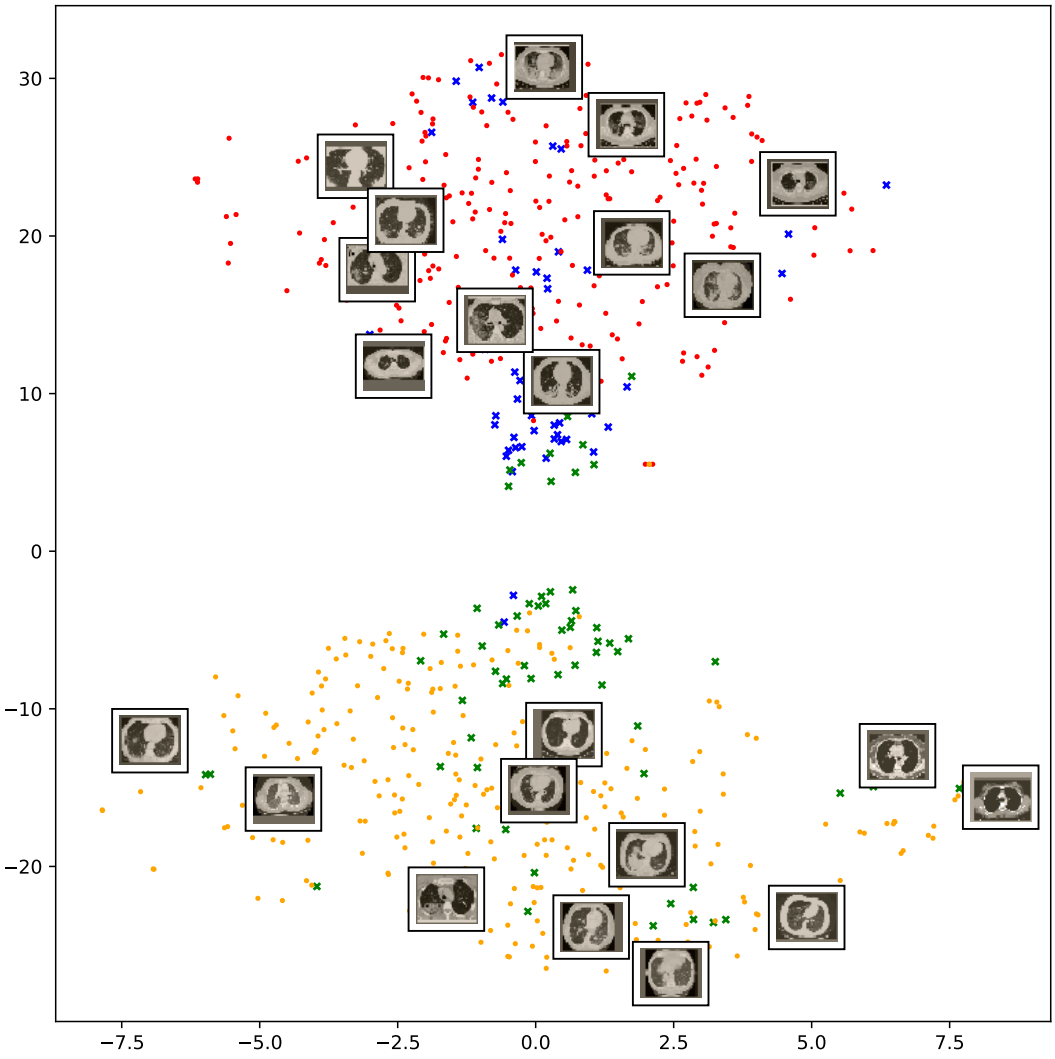


Figure 3.2: Visualization of the t-SNE embeddings for the entire COVID-19 CT dataset. The clusters represent COVID-19 and non-COVID-19 classes. Red: COVID-19 train samples. Blue: COVID-19 test samples. Yellow: non-COVID-19 train samples. Green: non-COVID-19 test samples.

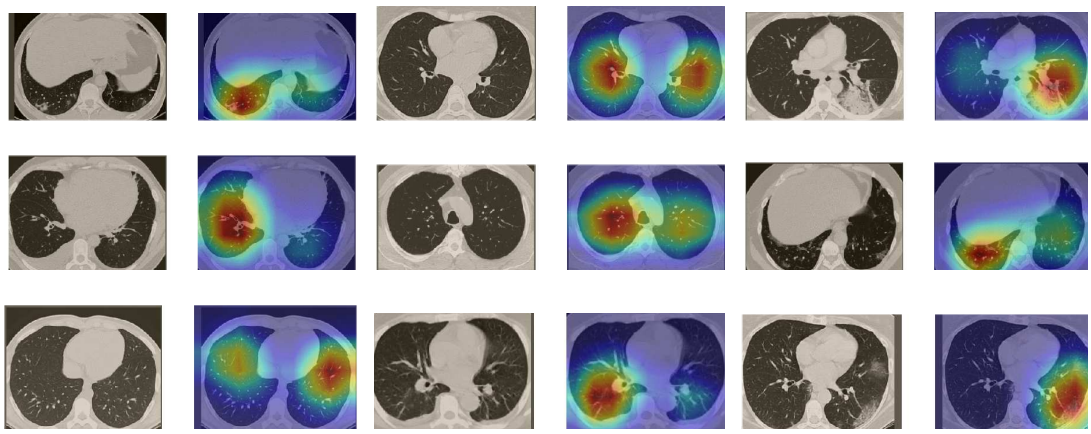


Figure 3.3: Grad-CAM visualizations of test images from the SARS-CoV-2 dataset. The InceptionV3 model correctly classified them as COVID-19 and localized the most relevant regions used for its decision. The first, third, and fifth columns show CT images with COVID-19 findings, whereas the second, fourth, and sixth columns represent their corresponding localization maps.

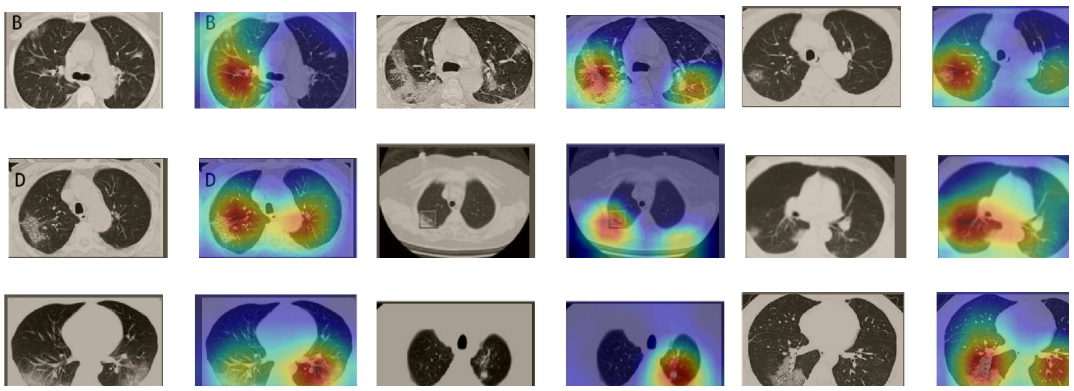


Figure 3.4: Grad-CAM visualizations of test images from the COVID19-CT dataset. DenseNet169 correctly classified them as COVID-19 cases.

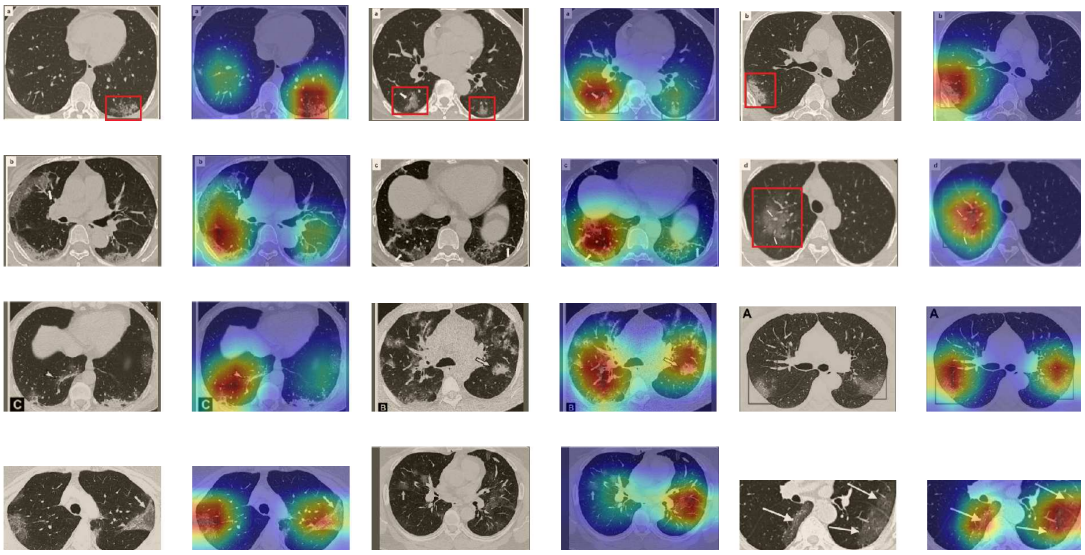


Figure 3.5: Examples of CT images taken from additional publications [29, 100]. The CT images were correctly classified as COVID-19 cases, and the abnormal regions are accurately highlighted in the localization maps.

CT findings, we tested our models on external CT images extracted from these two publications, as they feature typical findings of COVID-19 pneumonia marked by specialists. To make sure that none of the extracted images were unintentionally included in the train sets, specifically the COVID19-CT dataset, we analyzed the InceptionV3 model trained on the SARS-CoV-2 dataset. It was able to correctly classify the given CT images as COVID-19. Another reason against choosing the COVID19-CT dataset is that some of its images already contain visual annotations pointing at or surrounding manifestations of COVID-19 because the images were directly obtained from publications. This could mislead the models to look at these annotations instead of the actual CT image. The Grad-CAM visualizations in Figure 3.5 show that the InceptionV3 model accurately localizes the disease-related regions. The model ignores any specific marks in the images such as letters and only localizes the COVID-19-related regions. These further experiments show the success of the model to learn features related to COVID-19, and to correctly classify CT images outside the domain on which it was trained and tested.

Although we trained our models using CT images where the entire lung is visible in the scans, we tested them on some external CT scans where only one half of the lung is visible. The CT scans were extracted from the paper [100] and show different CT manifestations of COVID-19 marked by red squares or white arrows. Our models were able to classify them correctly as COVID-19 cases. Intriguingly, Figure 3.6 shows that all regions of abnormalities are accurately localized. This also demonstrates the

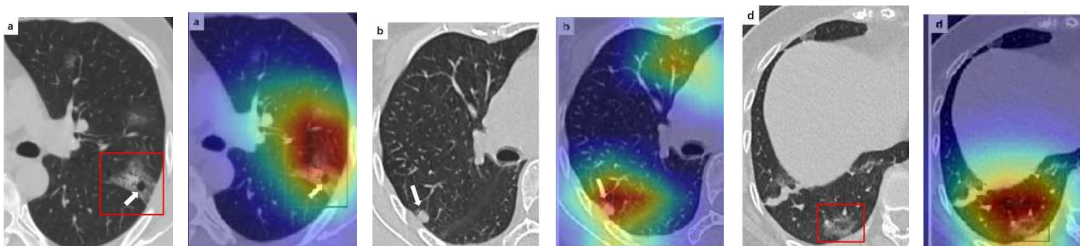


Figure 3.6: Example of annotated CT images with different manifestations of COVID-19 taken from [100]. Only a part of the lung is visible. Still, our models are able to identify them as COVID-19 cases and accurately localize the COVID-19-associated regions.

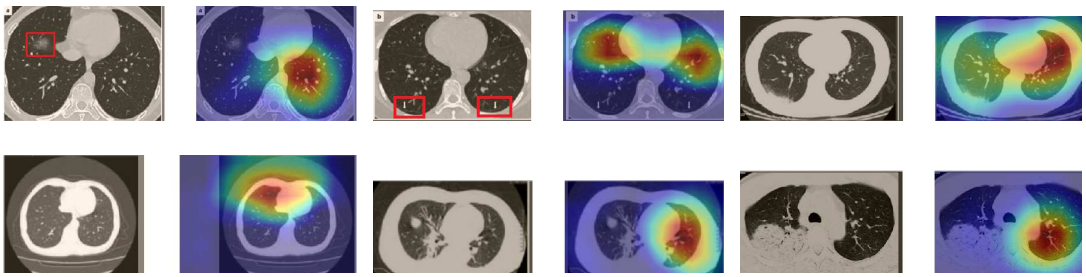


Figure 3.7: CT images and their Grad-CAM localization maps showing cases in which the model failed to localize the most relevant COVID-19 regions.

potential of the models to detect COVID-19 abnormalities in CT images outside the training set.

For a comprehensive analysis, we include some cases where the models failed to localize the exact COVID-19-associated regions. Figure 3.7 illustrates examples of CT scans that were correctly identified by our models as COVID-19 cases, but where our models failed to localize the most relevant regions associated with COVID-19 as marked in some CT images. In some cases the models only localized the findings in one lung, and failed to highlight the disease-related regions in the other lung.

Chapter 4

A Walk in the Black-Box: 3D Visualization of Large Neural Networks in Virtual Reality

This chapter develops a novel visualization approach for a more holistic understanding of deep neural networks. We combine various aspects of CNNs in one software and visualize large CNNs in a virtual 3D space. However, rendering large architectures like ResNet50 in 3D space presents several challenges. First, optimizing the software to render numerous network layers and feature maps poses significant challenges. Also, large networks require a suitable user interaction design. Then, there is the problem of dynamically representing linear structures, splits, or junctions in the computational graph. Additionally, the tool must provide a flexible and intuitive interface for developers and researchers to visualize their custom architectures. This chapter addresses these challenges and presents our software *DeepVisionVR*.

The chapter is structured as follows. Section 4.1 explains our approach of visualizing CNNs and gives details on the design principles. Section 4.2 uses our software to visualize models with different levels of generalization ability. Section 4.3 presents the results of this study. The findings in this chapter are published in Neural Computing and Applications [50]. The software is available on GitHub `github.com/Criscraft/DeepVisionVR`.

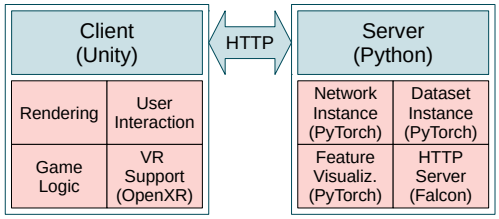


Figure 4.1: DeepVisionVR architecture.

4.1 Visualization Software

Rendering all feature maps within a deep network in real-time is a computationally challenging task. To address this challenge, we use the Unity game engine for rendering CNNs and processing interactions. Unity offers a flexible framework and supports deployment on various platforms including VR through the OpenXR framework.

While Unity creates immersive 2D and 3D experiences, its current functionalities lack managing neural networks in a flexible manner. To overcome this limitation and to facilitate convenient workflows for developers and researchers, we keep the neural network handling in Python. Consequently, we provide a Python module to connect the PyTorch framework [74] and the Unity game engine. This approach grants full access to the network's internals. Furthermore, we provide a programming interface for convenient software reconfiguration across different scenarios, including architecture and dataset modifications.

4.1.1 Software Architecture

The architecture of our software is shown in Figure 4.1. The software implements a client-server pattern where the Unity client handles the rendering and the user interactions, while a Python web server manages the CNN models using the PyTorch framework. This setup has several advantages. First, users can effortlessly modify or replace the network architecture through a server-side Python script. The Python server determines the visualization based on the PyTorch implementation of the network. The client only receives and interprets the network data. Second, distributing the computational load and memory usage between two machines can enhance performance. Third, the server can be made available on the web if desired. It establishes a Python-driven Flask web server to communicate with the client via HTTP and a REST API. Fourth, Python users can remain within their domain without needing Unity skills to configure the image data or the CNN models.

The rendering of large CNN architectures such as ResNets needs to be optimized. Our strategy combines common techniques utilized in game development including

Unity prefabs, leveraging multithreading, and employing mipmaps to reduce the detail of distant layers. To render the feature maps efficiently, we use a fast unlit shader that applies a colormap without resource-intensive routines such as illumination or shadows. The shader implements single-pass instanced rendering, resulting in efficient stereo rendering and reduced power consumption for the integrated VR mode. Since each feature map is rendered using a quad with only two triangles, the number of vertices is not a concern. Thanks to these optimizations, we successfully visualize ResNet18 in VR using an NVIDIA GTX 1080, achieving a stable frame rate of 90 frames per second on an Oculus Rift. This performance was achieved using input images of regular size with dimensions of 224×224 pixels.

4.1.2 Developer Interface

To visualize a custom CNN, users modify the Python script that implements the PyTorch model. There is no need to export the model to different formats or tables. The key component of our interface is a PyTorch module called TrackerModule, which is a subclass of PyTorch's Identity module. This module is added after layers or tensor operations that should be tracked in the visualization. The tracker modules form a graph by tracking the predecessor trackers. In addition, a custom marker module adds landmarks to divide the CNN into sections. It also subclasses the Identity class. Importantly, both the tracker module and the marker module do not alter the network's behavior and can therefore be included in the training and inference processes. The module-based approach grants developers and researchers fine-grained control over precisely what should be visualized.

Easy reconfiguration of the software is crucial. Therefore, we discuss three distinct use cases: (a) reconfiguring the network weights, (b) adjusting the network architecture, and (c) modifying the dataset. For point (a), users modify the Python script that defines and instances the Pytorch model. The new script should load custom network weights from disk, a standard procedure for PyTorch users. For point (b), users replace this script with their own architecture. For point (c), the server instantiates a PyTorch dataset class and transmits the contained images to the Unity client. To change the dataset, users replace the PyTorch script implementing the dataset. Notably, the images must adhere to the network's requirements, such as input size or the number of input channels.

4.1.3 Visualization of the Network Architecture

Figure 4.2 shows the CNN as a directed graph, similar to a conveyor system on a workshop floor. The input layer is represented by a floating canvas where users can insert images (here a sunflower). The classification results appear above the input layer. Behind the input layer, the RGB channels are shown. The subsequent layer

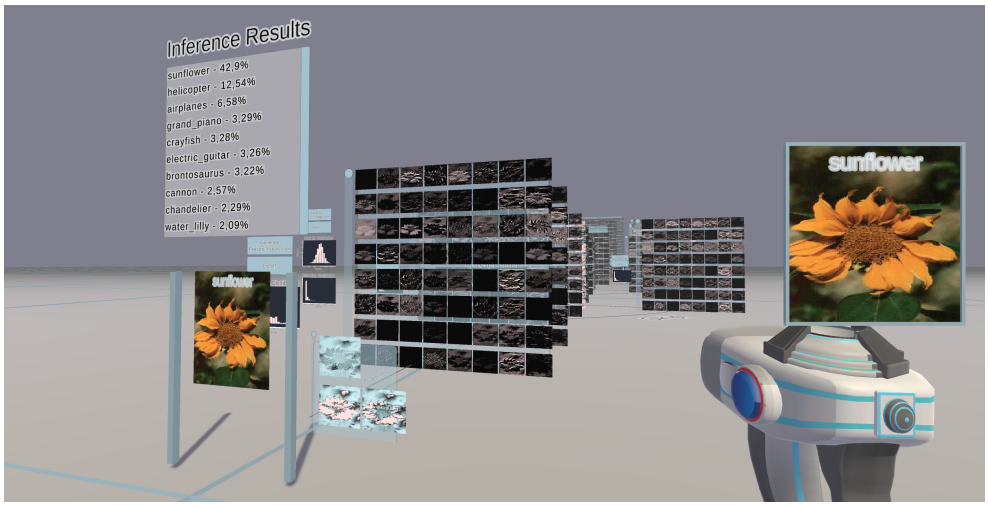


Figure 4.2: Representation of CovidResNet in 3D space. Each 2D panel shows the feature maps (channels) of a specific layer. Negative activations are colored blue, zero activations are black, and positive activations are white.

exhibits the feature maps after the first convolutional layer. Multiple networks can be visualized side by side within the same world space for comparison. The network layers are positioned automatically in 3D space to reflect the model architecture. We address the issue of branches in the computational graph by arranging them next to each other and connecting layers with labeled edges.

Figure 4.3 shows how our approach visualizes various CNN architectures. The top left image shows a basic block from the ResNet architecture. The input layer is shown on the left-hand side. The computational graph splits into an upper branch with two consecutive convolution operations and a lower branch, the skip connection. The branches are summed at the end of the block. The labels on the ground explain the computations. The basic block is also illustrated in Figure 2.4 as a 2D diagram. The top right image of Figure 4.3 shows a dense block, which is also shown in Figure 2.2 in 2D. Here, the input layer is located at the bottom left. The outputs of the first convolution operation are concatenated with the input feature maps. This process is repeated two more times, with the final layer comprising all previous feature maps. The bottom image depicts an Inception module, where the computational graph divides into four different branches with filters of varying kernel sizes. These branches are then concatenated to form the output of the Inception module.

4.1.4 Interaction Design

The interaction design focuses on handling large architectures with thousands of feature maps within a 3D environment. Users have plenty of options to immersively interact

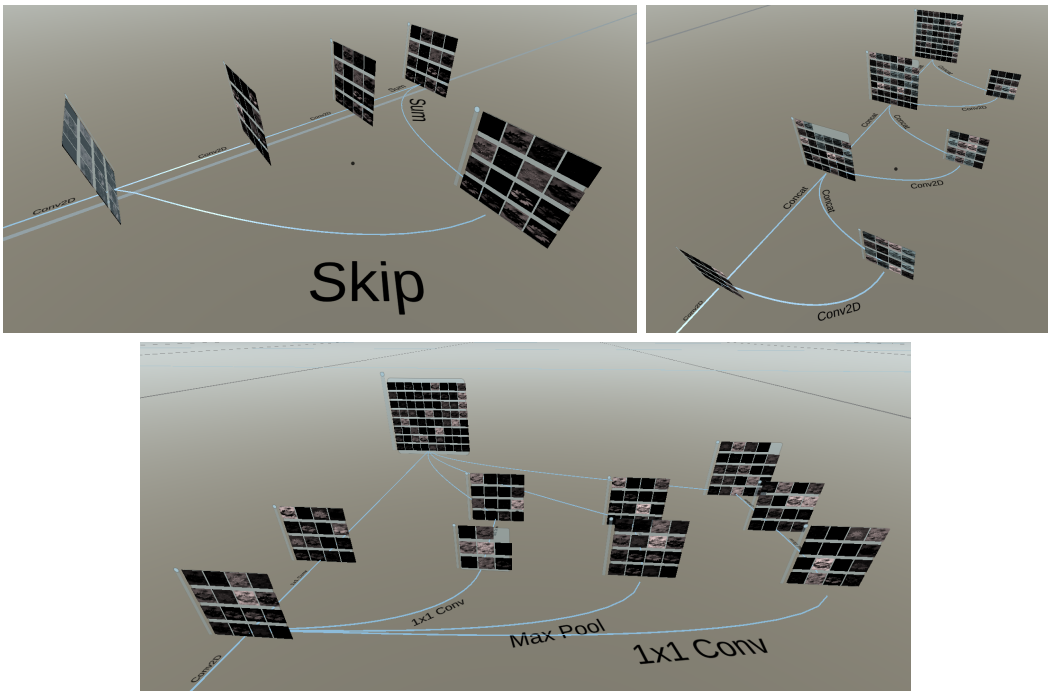


Figure 4.3: Representation of different architectures. Left: residual basic block. Right: dense block. Bottom: block of the Inception network.

with CNNs, including walking through the network and manipulating images.

Handheld Tool The primary interface for interaction is a handheld tool. This tool facilitates the selection of images from the dataset panel, as illustrated on the left-hand side of Figure 4.4. The selected image appears as a floating holograph above the tool. When inserting the image into the network, the software automatically initiates the forward pass and updates the visual representation of the CNN. Additionally, images generated via feature visualization can be picked up and utilized as regular input to the network. Additionally, the handheld tool interacts with floating GUI planes in the world.

Environment Customization Users can organize the 3D environment according to their preferences by manipulating and scaling layers, improving handling, enhancing readability, and hiding undesired elements.

Visual Inspection Addressing limitations of current VR headsets, which restrict per-eye resolution, we have implemented a floating holograph above the handheld tool, functioning as a magnifying glass. This feature enables inspecting images more closely.

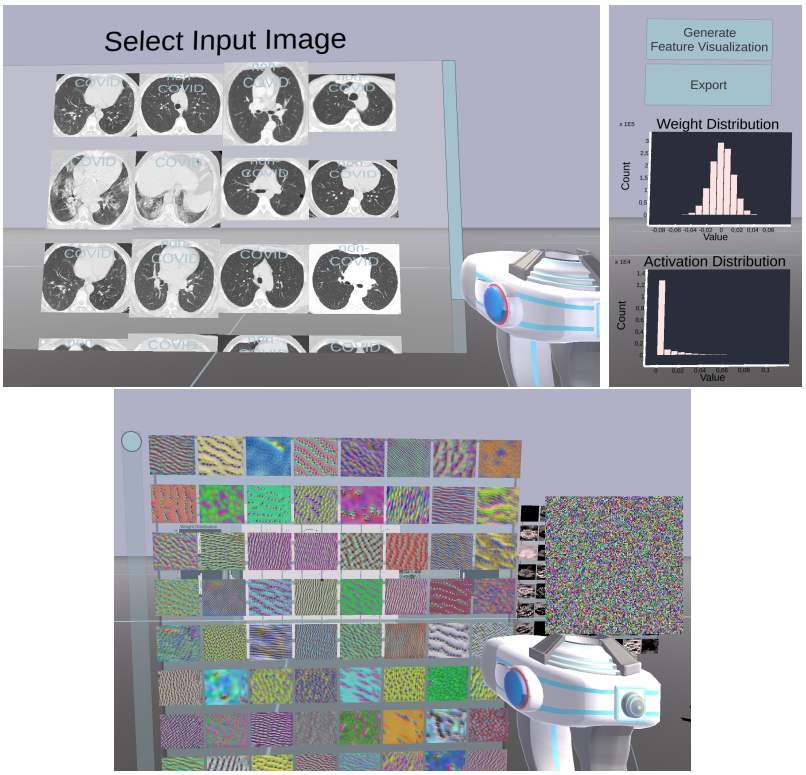


Figure 4.4: Top left: Dataset panel for selecting input images. Top right: User interface and statistics for a selected network layer. Bottom: Feature visualization. Each tile corresponds to one generated image maximizing the respective channel.

Organizing Additional Information Users can request additional information about the layers through floating user interface panels. This involves activation distributions or weight distributions, as shown in the center of Figure 4.4.

Locomotion Free locomotion and teleportation are available through VR controllers. Also, traditional mouse and keyboard inputs can be used, common to 3D games.

4.1.5 Feature Visualization

Section 1.3 gives a detailed introduction to activation maximization, a popular technique for feature visualization. DeepVisionVR offers feature visualization through a two-step process. First, users insert an input image (from a dataset or noise) into the network. Second, users press the feature visualization button of a layer they are interested in as shown in Figure 4.4. Automatically, the activation maximization algorithm optimizes the input image to maximize the average activation of each channel within

this layer, respectively. When applied to class neurons in the classification layer, feature visualization produces images that the network associates with specific classes. These generated images can be manipulated using the hand tool and inserted into the network to verify if they activate the channel.

To generate more naturally looking images, our software applies a series of transformations to the generated image between each update step:

1. Padding: Add 10 pixels of padding on each side.
2. Rotation: Randomly rotate the image within a range of 1 to 10 degrees with a probability of 30%.
3. Scaling: Randomly scale the image by a factor ranging from 1.0 to 1.05 with a probability of 5%.
4. Blurring: Apply Gaussian blur with a (5, 5) kernel and a sigma between 0 and 0.5.
5. Cropping: Center crop the image to remove the padding.
6. Pixel rolling: Roll the pixels along the x and y axes with a random number between 1 and 5 pixels and a probability of 30%.
7. Pixel distribution adjustment: Shift and scale the pixel distribution to match the mean and standard deviation of the original dataset. This normalized image is then blended with the original version using a 5% factor. This step ensures a gradual and smooth normalization process, counteracting exploding pixel values and promoting a natural color distribution.

4.2 Experimental Setup

As explained in the introduction in Chapter 1, we explore how CNNs learn spurious features to memorize images. By visualizing these features we provide insights into the mechanisms of overfitting and highlight the importance of choosing appropriate data augmentation strategies to improve model performance. In the following, we present three distinct training strategies, each yielding models with varying levels of generalization. The three strategies are depicted in Figure 4.5.

First Training Strategy The first training strategy follows a common approach to training robust models. As depicted on the left side of Figure 4.5, training starts with a randomly initialized CNN. We augment the input images to effectively increase the training data and foster the development of generic features. The augmentation includes cropping, random location shifts, adding Gaussian noise, brightness and contrast jittering, and random horizontal flipping.

Second Training Strategy The second training strategy employs shuffled training labels, destroying any underlying patterns in the data. Consequently, this approach is

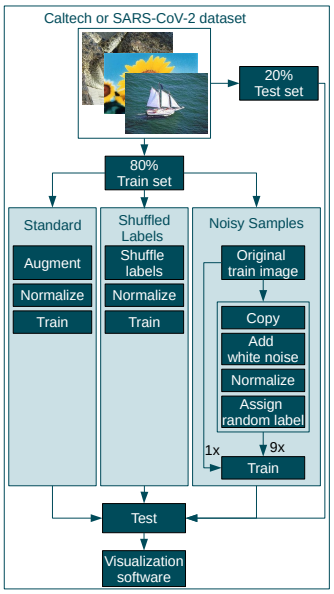


Figure 4.5: Three training strategies for obtaining models with different levels of generalization abilities.

expected to provide a model without generalization capabilities. During training, the labels remain consistent. No image augmentation is applied.

Third Training Strategy Bad global minima, where the training error is zero while the test error remains high, exist [56]. However, finding such minima in practice is challenging [54]. The third training strategy aims to find such bad minima. It creates a new training set from an original labeled dataset by copying each image n times, resulting in a total of $n + 1$ versions for each image. One version retains the original image with its correct label, while the other n versions receive random, consistent labels with random, consistent, white noise. Consistent means that the modifications do not change over time during training. The new train set should move the network's attention away from learning universal features towards memorizing the noise patterns.

Training Details We use Pytorch [74] to train our models using the Lamb optimizer [103]. The learning rate is reduced from the initial value $\mathrm{lr}_{\mathrm{init}}$ to $0.001 \cdot \mathrm{lr}_{\mathrm{init}}$. The network weights are initialized using Kaiming initialization [30]. The training hyperparameters are summarized in Table 4.1.

Datasets We perform the experiments on the SARS-CoV-2 dataset [85] and the Caltech101 dataset [49], which cover medical and non-medical domains, respectively.

Table 4.1: Training hyperparameters.

Experiment	Mode	Epochs	Initial lr	Weight decay	Augmentation	Image shape
Caltech101 CovidResNet	standard	200	0.002	on	on	175×150
	labels shuffled	250	0.001	off	off	
	noisy samples	1000	0.001	off	off	
SARS CovidResNet	standard	150	0.002	on	on	250×180
	labels shuffled	250	0.001	off	off	
	noisy samples	1000	0.001	off	off	
SARS CovidDenseNet	standard	150	0.002	on	on	250×180
	labels shuffled	250	0.001	off	off	
	noisy samples	1000	0.001	off	off	

Table 4.2: Performance metrics of the three different training strategies.

Architecture	Dataset	Type	Test accuracy
CovidResNet	Caltech101	Standard	0.78
CovidResNet	Caltech101	Shuffled labels	0.01
CovidResNet	Caltech101	Noisy samples	0.42
CovidResNet	SARS-CoV-2	Standard	0.96
CovidResNet	SARS-CoV-2	Shuffled labels	0.49
CovidResNet	SARS-CoV-2	Noisy samples	0.97
CovidDenseNet	SARS-CoV-2	Standard	0.97
CovidDenseNet	SARS-CoV-2	Shuffled labels	0.50
CovidDenseNet	SARS-CoV-2	Noisy samples	0.99

Section 2.2 describes the datasets in detail. As there are no official test splits available, we create our own using 80% for training and 20% for testing. We also rescale the images to an input size of (175×150) pixels. We counteract the class imbalance of the Caltech101 dataset by performing undersampling.

Architectures We apply two recent architectures in our analysis, CovidResNet and CovidDenseNet [4]. They were recently proposed to detect COVID manifestations in CT scan images and have proven to generalize well on small-scale CT datasets with a few thousand images. CovidResNet and CovidDenseNet have a total of 4.98 million parameters and 1.63 million parameters, respectively. Section 2.3 gives a detailed overview of both architectures.

4.3 Results

4.3.1 Model Performance

After training, the models reach zero error on the train set. Table 4.2 summarizes the performance metrics of the trained models. The results show that the three train-

ing strategies lead to models with different levels of generalization abilities, which is in line with our expectations. The first training strategy applied to Caltech101 using CovidResNet reaches a test accuracy of 78%. CovidResNet and CovidDenseNet achieve impressive test accuracies of 96% and 97% respectively on the SARS-CoV-2 dataset. The second learning strategy with shuffled training labels performs similarly to a random classifier with 1% test accuracy on Caltech101 and about 50% on SARS-CoV-2. Interestingly, the third strategy exhibits a test accuracy of only 42% on the Caltech101 dataset. Its ability to correctly classify test data in nearly half of the cases across 101 classes is still remarkable. The test accuracies on the SARS-CoV-2 dataset are even similar to the first training strategy. Notably, this model managed to memorize images by capturing their specific noise patterns and developed generic features, simultaneously. Finding a network with zero training error but bad generalization seems to be challenging, proving the good bias of CovidResNet and CovidDenseNet for image recognition problems. Subsequently, we visualize these networks to learn how memorization and generalization can coexist within CNNs in the subsequent section.

4.3.2 Memorization Correlates with the Processing of Local Information

Table 4.3 visualizes the activation patterns within the networks. The input image shows a sunflower (Caltech101) or a CT image of a COVID-19-infected patient (SARS-CoV-2). High activation is represented by white, negative values by blue, and zero activation by black.

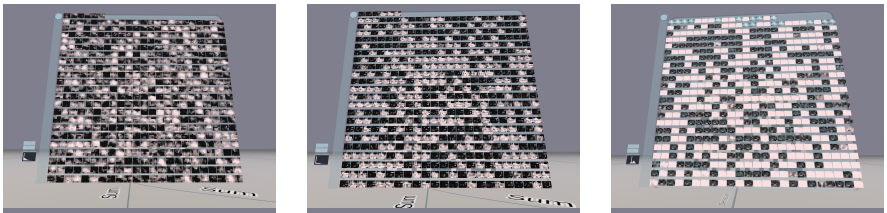
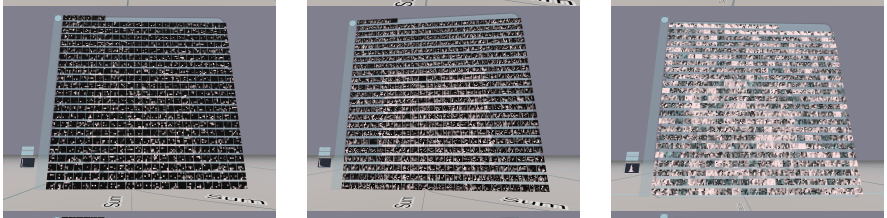
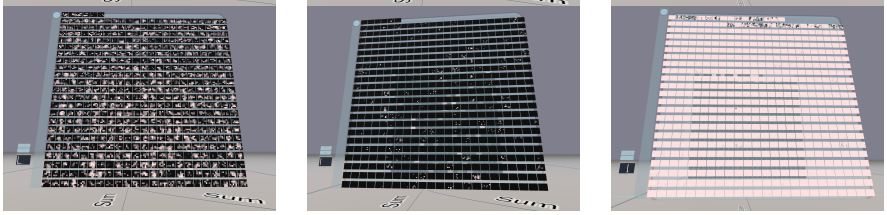
First Training Strategy

The top row of Table 4.3 displays the activations after the last convolutional layer of CovidResNet and CovidDenseNet, trained using the first training strategy. The channels tend to be either activated or not. In activated channels, the neighboring pixels usually have similar activation levels. This behavior is consistent across different test images.

Second Training Strategy

The second row of Table 4.3 displays the activation patterns in the networks trained with shuffled train labels. The feature maps contain a lot of spatial information. The 512 filters respond to various local regions within the image. There is no indication that the channels mark semantic image regions like a sunflower or a part of the loung. We hypothesize that these channels may process specific textures or local pixel arrangements.

Table 4.3: Activations, showing the 512 feature maps in the last convolutional layer. Best viewed with zoom.

Training Strategy	Caltech101 CovidResNet	SARS-CoV-2 CovidResNet	SARS-CoV-2 CovidDenseNet
First			
Second			
Third			

Third Training Strategy

The third row of Table 4.3 displays the third training strategy. The feature maps contain plenty of spatial information. The activations seem to be triggered by local patterns in the input as they do not follow shapes or semantic regions in the input images. This characteristic is found in all three networks trained with the third training strategy. CovidResNet trained on the SARS-CoV-2 dataset has sparser activations compared to the Caltech101 model. CovidDenseNet is similar to CovidResNet with a flipped sign (black and white switched).

4.3.3 Feature Visualization on the Caltech101 Dataset

We employ feature visualization to generate input images that maximize the activation of specific feature maps within the CNNs. Figure 4.6 presents the resulting images. We strongly recommend viewing the visualizations digitally with significant zoom.

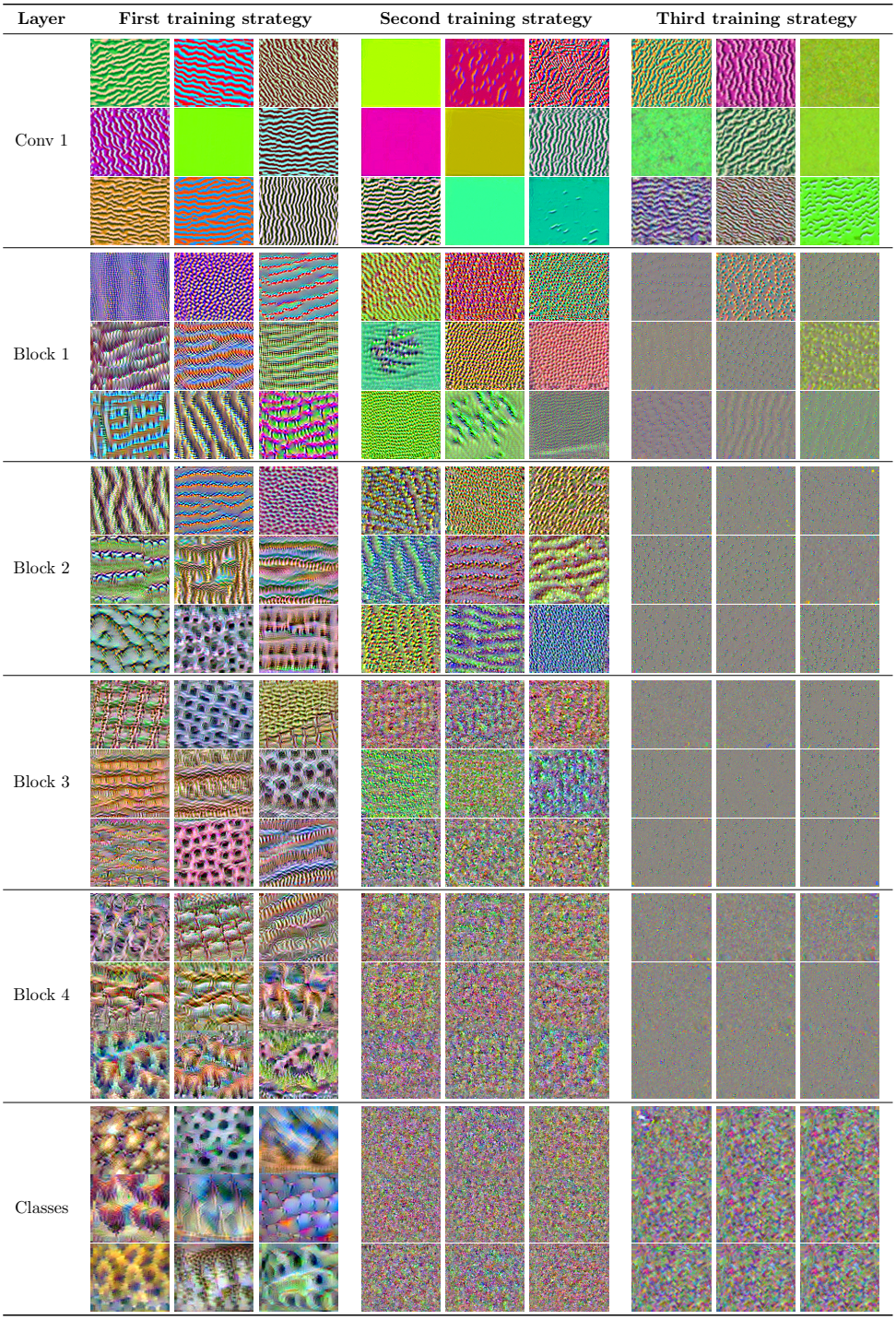


Figure 4.6: Feature visualization of CovidResNet trained on the Caltech101 dataset with different training strategies. Best viewed in color with zoom.

First Training Strategy

Figure 4.6 shows images maximizing the activation of CovidResNet trained on the Caltech101 dataset. The first column focuses on the first training strategy. The initial convolutional layer contains various edge filters. Some channels focus more on color information than on structure.

In the first block, simplistic textures emerge, evolving into more complex versions in subsequent blocks. In blocks 3 and 4, the visualizations start to resemble objects or animals. This hints at the processing of semantics within these layers. Some of the patterns seem to originate from previous ones, combined with horizontal stripe or grid patterns. The well-defined objects in block 4 show the high-level features emerging in this layer.

The feature visualization of the classification neurons unveils the visual concepts the CNN has learned from the classes. See Figure 2.1 for image references. Although somewhat subjective, we allow ourselves to provide some interpretations of the class visualizations in Figure 4.6. The image of a crocodile head (top left channel in the three-by-three grid) exhibits large scales and eyes. The spatial arrangement seems to be irrelevant to the network. Similarly, the panda (top center) shows prominent black spots, typical of panda faces. Again, the arrangement seems to be irrelevant. The image of a pyramid (top right) shows triangular shapes against sandy ground and a blue sky. Further images display roosters, a schooner, and the cartoon character Snoopy, identified by specific features such as the snout and nose. Lastly, images of sunflowers, a wild cat, and a yin-yang are shown. For the cat, the network seems to focus on dotted fur patterns. No body parts or heads are visible, hinting that they are not important for network decisions.

Second Training Strategy

In the first layer, the feature visualizations show edge and high-frequency patterns. The latter are only visible using significant zoom. In the subsequent blocks, the high-frequency patterns become more pronounced. Channels increasingly prioritize combinations of colorful pixels over structure or shape. The deeper layers are dominated by specific high-frequency patterns and local pixel arrangements. There is no indication that the CNN processes large-scale structures or semantic information.

Third Training Strategy

The feature visualization of CovidResNet trained with the third training strategy provides similar results to the second training strategy. Nevertheless, in the first block, patterns significantly diverge from the first two models, where distinct clusters with high-frequency patterns appear. The images of subsequent layers exhibit significant high-frequency patterns while lacking global structure. Possibly these patterns corre-

spond to the noise patterns memorized from the training images, aiding the network to identify them.

4.3.4 Feature Visualization on the SARS-CoV-2 Dataset

The feature visualization for CovidResNet and CovidDenseNet trained on the SARS-CoV-2 dataset is shown in Figures 4.7 and 4.8, respectively. As the results are similar for both architectures, they are discussed jointly.

First Training Strategy

The feature visualizations of the first layer show edge and grid-like patterns. Deeper in the network, the stripes and grid patterns evolve into characteristic shapes with different levels of roughness and smoothness. In contrast to Caltech101, the SARS-CoV-2 models lose feature diversity in the deeper layers. We found that the channels in the fourth layer can be roughly clustered into two groups, possibly due to the binary classification problem.

Second Training Strategy

Similar to the models trained on the Caltech101 dataset, the generated images contain high-frequency patterns, which can only be seen with zoom enabled. There is no hint for the processing of semantic information.

Third Training Strategy

The feature visualizations contain faint larger structures, hinting at the processing of low-frequency information. This suggests that the network might still process semantic relationships alongside high-frequency patterns. However, in the visualization of the class neurons, high-frequency patterns dominate, and COVID-19 manifestations are not visible.

4.3.5 The Coexistence of Generic and Spurious Features

The third training strategy produced models with mediocre and good generalization abilities. In contrast to the first training strategy, the visualizations showed strong high-frequency features and, to a lesser extent, low-frequency features. The results suggest that the models learned to memorize the train images through high-frequency information contained in the noise that we artificially introduced to the train data. Similarly, the second training strategy used high-frequency information to memorize the shuffled training data. Notably, in models trained with the third training strategy generic and spurious features coexisted. Both modalities appeared to be entangled, as no distinct channels were identified that exclusively processed one or the other.

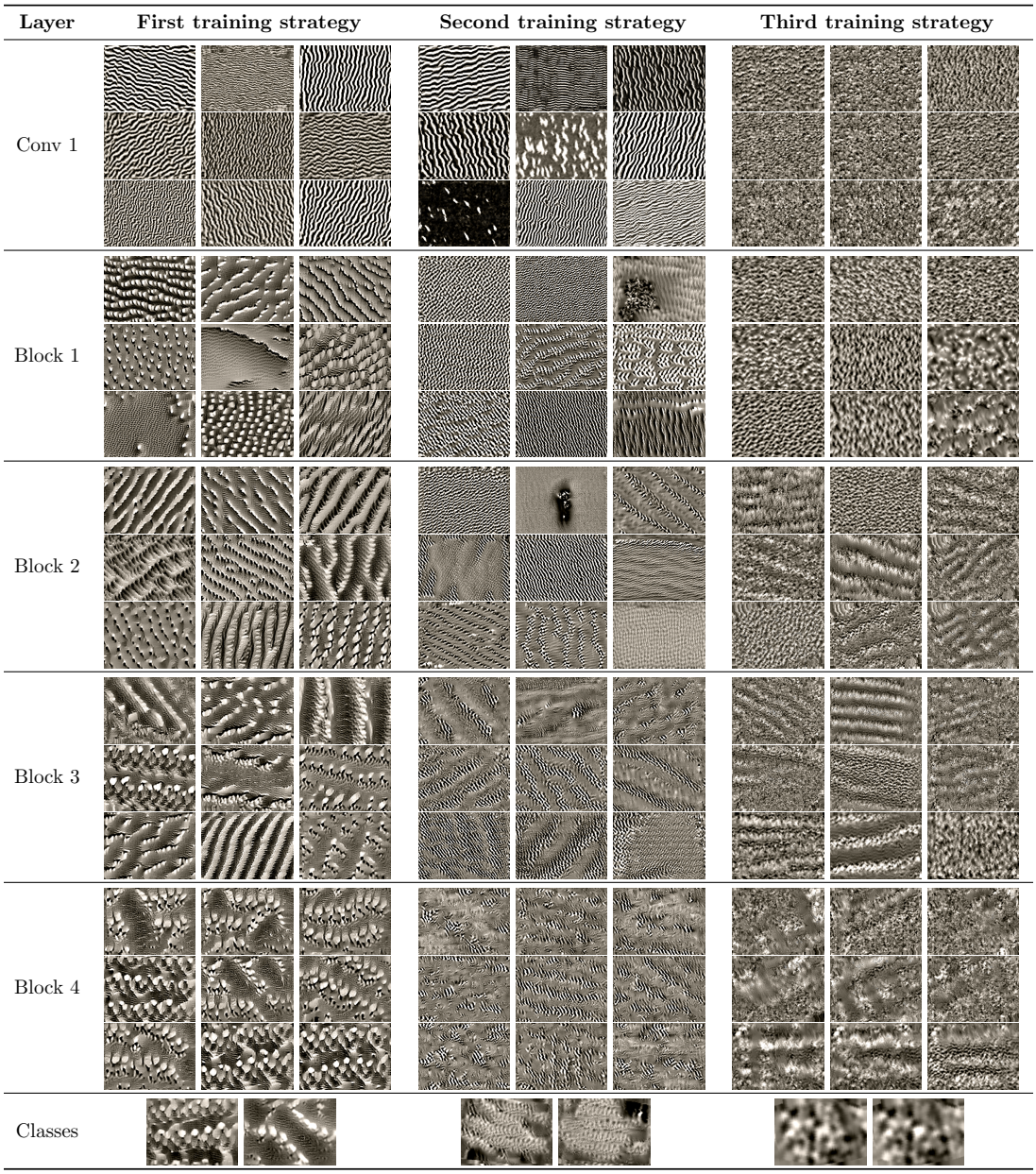


Figure 4.7: Feature visualization of CovidResNet trained on the SARS-CoV-2 CT-scan dataset with different training strategies. Best viewed with zoom.

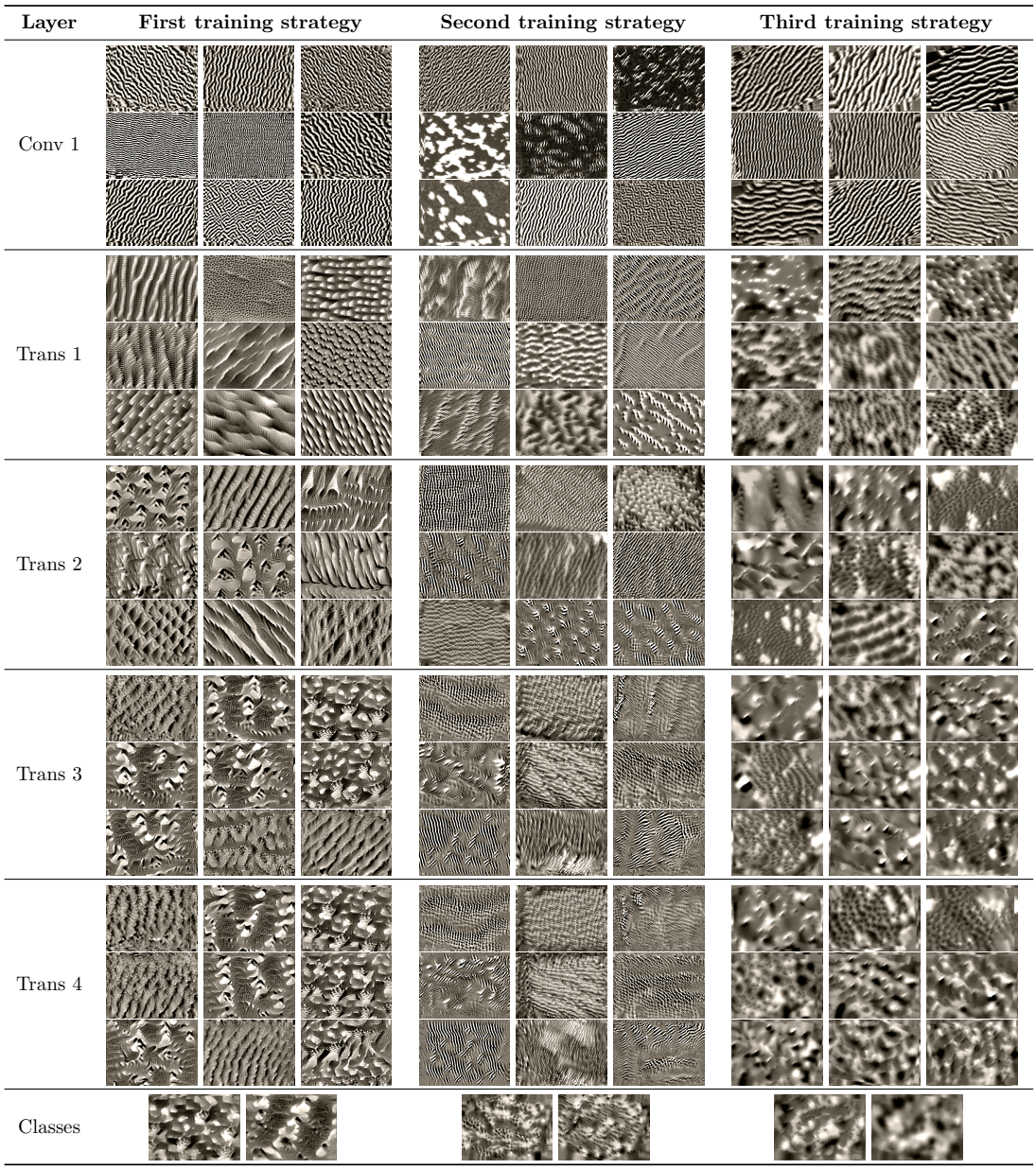


Figure 4.8: Feature visualization of CovidDenseNet trained on the SARS-CoV-2 CT-scan dataset with different training strategies. Best viewed with zoom.

Chapter 5

Leaky ReLUs That Differ in Forward and Backward Pass Facilitate Activation Maximization in Deep Neural Networks

Activation Maximization (AM) is a popular method in XAI. In the previous chapter, AM visualized features of CNNs. The technique solves a specific optimization problem introduced in Section 1.3. However, optimizing functions that contain ReLUs [69] or Leaky ReLUs [59] remains a central challenge. While Leaky ReLU mitigates gradient sparsity, challenges persist in AM.

The ReLU [69] produces sparse gradient vectors:

$$\begin{aligned} \mathrm{ReLU}(x) &= \max(x, 0) \\ \frac{\partial \mathrm{ReLU}(x)}{\partial x} &= (\mathrm{ReLU}(x) > 0). \end{aligned}$$

Here, the inequality expression is evaluated as 1 if true, else as 0. To improve the gradient flow, Leaky ReLU does not set the derivative to zero [59]:

$$\begin{aligned} \mathrm{LReLU}_s(x) &= \max(x, sx) \\ \frac{\partial \mathrm{LReLU}_s(x)}{\partial x} &= \max((\mathrm{LReLU}(x) > 0), s). \end{aligned}$$

s is called the negative slope. It holds $0 < s < 1$.

We hypothesize that AM of modern CNNs with ReLUs or Leaky ReLUs severely suffers from optimization issues. We show that AM does not find the optimal stimulus

(white image) for simple functions containing ReLU or Leaky ReLU due to three optimization issues a) sparse gradients, b) the race of patterns, and c) local maxima. The issues a) - c) are explained in detail in Section 5.1.

The chapter is structured as follows. Section 5.1 presents the optimization issues of ReLU and LeakyReLU concerning AM. Subsequently, Section 5.2 introduces a partial solution, the ProxyGrad algorithm. Section 5.3 demonstrates the performance of ProxyGrad when visualizing classes from ImageNet. In Section 5.4, we apply ProxyGrad to train the weights of CNNs for image classification. Section 5.5 explains why ProxyGrad provides larger gradients with higher slopes within ResNet. The results are published at the 2024 International Joint Conference on Neural Networks (IJCNN) [52].

5.1 Optimization Issues During Activation Maximization

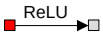
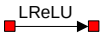
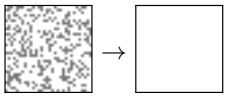
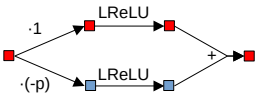
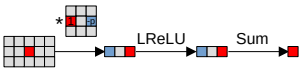
This section presents three optimization issues and demonstrates their relevance for AM using "white image problems" as toy examples. A white image problem is an AM problem where the optimal input pattern is $\mathbf{x}^* = \mathbf{1}$ (a white image). In the following, $\mathbf{x} \in [-1, 1]^{H \times W}$ is a gray scale image. The white image problem consists of a function $f(\mathbf{x}) : [-1, 1]^{H \times W} \to \mathbb{R}$ and a loss function $E(\mathbf{x}) = -f(\mathbf{x})$. Examples are shown in Table 5.1.

a) 'sparse gradients': The function $f_1(\mathbf{x}) = \sum \mathrm{ReLU}(\mathbf{x})$ represents a white image problem where the sum is over all pixels. However, AM will not generate a white image for negative input values because of sparse gradients. This phenomenon is depicted in the second row of Table 5.1. Leaky ReLUs with a positive slope s do not suffer from sparse gradients and generate the white image successfully.

b) 'race of patterns': The derivative of the Leaky ReLU implies that input elements are changed during gradient ascent at different speeds. The third row of Table 5.1 shows the results of AM using the function $f_2(\mathbf{x}) = \sum \mathrm{LReLU}_s(\mathbf{x})$. After few iterations, the positive values in the input increase their value, seen as white pixels. The negative values also increase, but at a much lower rate. Additional iterations are needed to produce the optimal input stimulus, the white image. In a deep CNN with many layers of Leaky ReLUs, the effect may significantly increase the iterations needed to reach local maximum.

c) 'local maxima': Some simple functions containing Leaky ReLUs have local maxima. The function $f_3(\mathbf{x}) = \sum \mathrm{LReLU}_s(\mathbf{x}) + \mathrm{LReLU}_s(-p\mathbf{x})$ with $1 > p > s > 0$ in Table 5.1 represents a white image problem. As before, the sum is over the pixels. It consists of the Leaky ReLU plus another one scaled down by a factor p. The derivative after some specific pixel x_i is $\frac{\partial f_3(\mathbf{x})}{\partial x_i}(1) = 1 - ps > 0$ and $\frac{\partial f_3(\mathbf{x})}{\partial x_i}(-1) = s - p < 0$, implying that gradient ascent will get stuck in a local maximum if the input contains negative values. Figure 5.1 plots the function. One can see that the optimization will either

Table 5.1: AM problems that are not solved via gradient ascent. The optimal solution is a white image. A red (blue) box indicates positive (negative) values. The right column shows the result of AM. The initial image is shown at the very top. It is $1 > p > s > 0$ with the negative slope s. The sums are over the pixels.

Function	Illustration	
		
$f_1(\mathbf{x}) = \sum \mathrm{ReLU}(\mathbf{x})$		
$f_2(\mathbf{x}) = \sum \mathrm{LReLU}(\mathbf{x})$		
$f_3(\mathbf{x}) = \sum \mathrm{LReLU}(\mathbf{x}) + \sum \mathrm{LReLU}(-p\mathbf{x})$		
Convolution		

end up in a black or a white pixel, depending on the initialization. Table 5.1 shows an implementation of f_3 that applies two different weights on a single channel. No white image is generated. The optimization gets stuck in a local maximum. Another example for local maxima uses a convolution operation with the kernel $\left(\begin{smallmatrix} 0 & 0 & 0 \\ 1 & 0 & -p \\ 0 & 0 & 0 \end{smallmatrix}\right)$ as shown in Table 5.1. In the example, we chose the kernel specifically such that the network output is maximal when the input image is white (ignoring border effects). The large visual gap between the generated image and the optimal stimulus questions the practical usefulness of AM. Also, very different patterns are generated depending on the initialization of the input.

Nevertheless, as the negative slope s increases toward 1, the race of patterns mends and the number of local maxima decreases. A slope of 1 leads to a linear network, making the AM problem convex. However, modifying the negative slope of a trained CNN also changes the predictions, adding to the difficulty of visually interpreting the visualization. Therefore, we propose to increase the slope in the backward pass only.

5.2 The ProxyGrad Algorithm

To cope with the optimization issues, our ProxyGrad algorithm employs a secondary network as a proxy for gradient computation. The proxy network should possess a

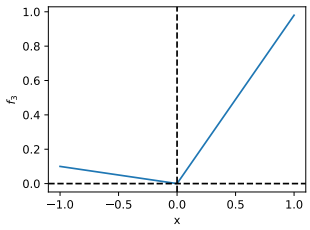


Figure 5.1: Illustration of function $f_3(\mathbf{x}) = \sum \text{LReLU}_s(\mathbf{x}) + \sum \text{LReLU}_s(-p\mathbf{x})$ with $p = 0.2$ and $s = 0.1$ for an input image with a single pixel.

Table 5.2: The forward pass (FP) and the backward pass (BP) for ReLU, Leaky ReLU, and ReLU with ProxyGrad. The gradient R_i^l of the network output $f^{\text{out}}(\mathbf{x})$ after the ith feature map of layer l is $R_i^l = \partial f^{\text{out}} / \partial f_i^l$.

FP ReLU with ProxyGrad:	$f_i^{l+1} = \max(f_i^l, 0)$	3 2 -1 -1 1 3 -2 -3 4	3 2 0 0 1 3 0 0 4
FP Leaky ReLU:	$f_i^{l+1} = \max(f_i^l, s \cdot f_i^l)$	3 2 -1 -1 1 3 -2 -3 4	3 2 -s -s 1 3 -2s -3s 4
BP ReLU:	$R_i^l = (f_i^l > 0) \cdot R_i^{l+1}$	1 -1 0 0 2 3 0 0 -3	1 -1 -2 1 2 3 2 -1 -3
BP ReLU with ProxyGrad:	$R_i^l = \max((f_i^l > 0), s) \cdot R_i^{l+1}$	1 -1 -2s -s 2 3 -2s -s -3	1 -1 -2 1 2 3 2 -1 -3

simplified loss landscape with fewer local maxima than the original network, thereby enhancing optimization efficiency. Our proxy shares the architecture and the weights with the original network but incorporates a higher negative slope. During optimization, the loss is computed based on the original network. Subsequently, the gradient of the loss is computed based on the proxy but with the activations of the original network. Table 5.2 visualizes how ProxyGrad handles the backward pass with ReLUs in the original network and Leaky ReLUs in the proxy network. As seen in the example, the output remains sparse while the gradient becomes dense. Please find the implementation of the ProxyGrad algorithm in Appendix A.1.

5.3 Activation Maximization Using ProxyGrad

To demonstrate the visualization capabilities of ProxyGrad, we perform AM on the class neurons of a ResNet18 trained on ImageNet [80]. The original network contains ReLUs. The proxy network uses Leaky ReLUs with a negative slope $s > 0$ in the

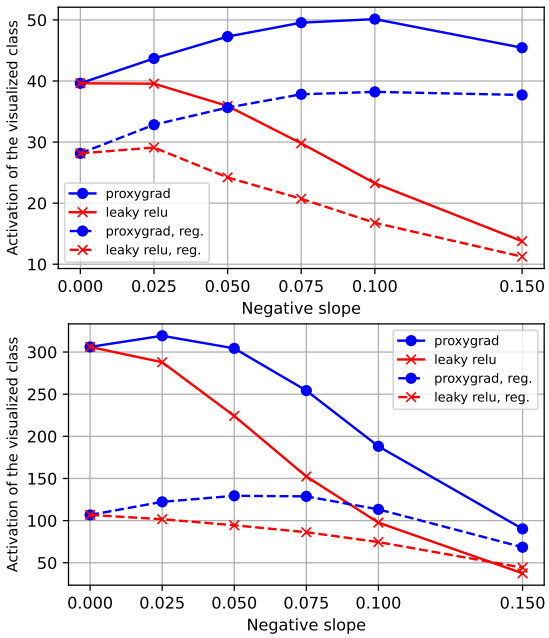


Figure 5.2: Mean activation of the generated images as measured by the original ReLU network. In the legend, "proxygrad" refers to using the ReLU network with ProxyGrad. "reg." means that regularization was applied to the images during AM. Top: after 10 iterations. Bottom: after 500 iterations.

backward pass. We call this experiment ReLU with ProxyGrad. The experiments compare three methods: AM using the original ReLU network, AM using ReLU with ProxyGrad, and AM using Leaky ReLU networks. In the latter experiment, we replace all ReLUs in the original network with Leaky ReLUs. All experiments use the same network weights. The generated images are quantitatively evaluated using the class outputs of the original ReLU network. The initial image shows Gaussian noise on a gray background with $3 \times 224 \times 224$ pixels. The optimization lasts for 10 or 500 iterations and uses a learning rate of $\mu = 25$. We either apply no regularization or weak image blur and small rotations after each iteration, a common strategy to find robust maxima [83, 102]. For comparability, the optimization uses normalized gradient vectors, as motivated by Section 5.5, where the magnitude of gradients is discussed for different slopes. The update step for gradient ascent becomes

$$x \longleftarrow R\left(x + \mu \frac{\nabla_x f(\mathbf{x})}{||\nabla_x f(\mathbf{x})||_2}\right). \tag{5.1}$$

Figure 5.2 presents the activation scores using the original ReLU network. For each experiment, AM generates $C = 1000$ images $\{x_1^*, ..., x_C^*\}$, one for each class. Then, the

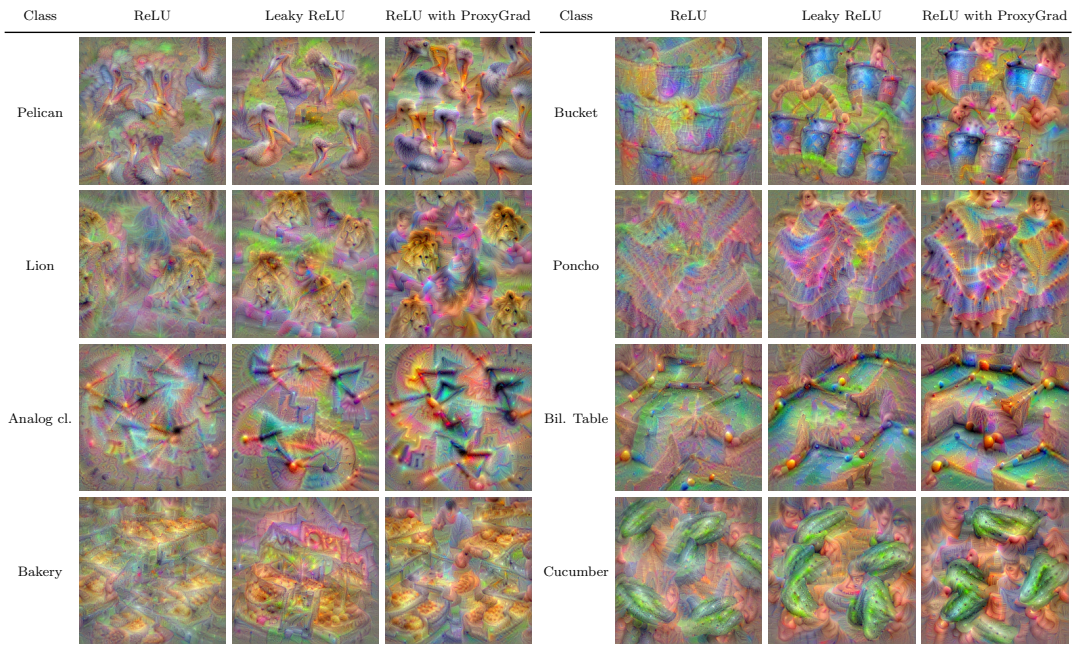


Figure 5.3: Activation maximization for class neurons of ResNet18 trained on ImageNet. The negative slope is 0.075. Best viewed digitally with zoom.

average activation score

$$\text{score} = \sum_{i=1}^{C} N_i(x_i^*)/C \tag{5.2}$$

is computed over the classes i. Here, N_i represents the ith class neuron of the original ReLU network. The score is always computed using the ReLU network, no matter what strategy generated the images x_i^*. The experiment on the left-hand side of the two diagrams ($s = 0$) considers the images generated using the ReLU network (baseline). Images generated by ReLU with ProxyGrad are illustrated in blue and images generated by Leaky ReLU networks are in red. ReLU with ProxyGrad reaches higher activation scores with an increasing slope, showing its ability to facilitate optimization. Also, the ReLU network classifies the generated image as the target class in almost all cases. After 10 iterations, images generated using ReLU with ProxyGrad have a mean activation of about 50 (40), while the images generated without ProxyGrad only reach 40 (30) without (with) regularization. After 500 iterations, ProxyGrad images have an activation of about 330 (130), while the images generated without ProxyGrad only have 300 (100) without (with) regularization. Because of gradient normalization, all optimization steps have equal size. This implies that the direction of the gradient returned by ProxyGrad is more stable during AM.

However, for very high negative slopes, the activation scores of the original ReLU

Figure 5.4: AM with a high negative slope of 0.3, far beyond the optimal range. In this negative example, the class 'poncho' is visualized. Best viewed digitally with zoom. Left: Leaky ReLU network. Right: ReLU with ProxyGrad

network decrease. An optimal choice for the slope exists. For the experiments with Leaky ReLU networks, no such optimum seems to exist, and the generated images tend to produce lower activations than those generated by the original ReLU network. This underscores the ability of ProxyGrad to facilitate optimization.

For a qualitative comparison, Figure 5.3 presents class visualizations for ReLU, Leaky ReLU, and ReLU with ProxyGrad after 2500 iterations.

For ReLU with ProxyGrad, the visual quality of the generated images tends to improve they develop stronger key features of the respective class and more distinct shapes. Also, the textures look cleaner and less noisy.

Interestingly, for high negative slopes far beyond the optimal range, ReLU with ProxyGrad generates objects resembling animal faces. This phenomenon is demonstrated in Figure 5.4 where the class poncho is visualized. This occurs also for other classes unrelated to animals or faces. We speculate that a dominant subnetwork that mainly processes facial features exists within the trained ResNet18. When performing AM using the Leaky ReLU network, very high negative slopes make the synthesized images collapse to simple, repetitive textures. Objects do not appear, as seen in Figure 5.4.

5.4 Recognition Performance

As a generic optimization strategy, ProxyGrad can also be applied to adjust the weights of deep neural networks. This section addresses the performance of ProxyGrad for training the weights of CNNs on the Caltech101 dataset [49], the Caltech-UCSD Birds-200-2011 (CUB) dataset [95], and the 102 Category Flower dataset (Flowers) [73]. The datasets are presented in detail in Section 2.2. In a comprehensive comparison, we train ReLU networks, ReLU networks with ProxyGrad, and Leaky ReLU networks.

Table 5.3: Average test performance of ResNet18 and PFNet18 on three benchmark datasets. s denotes the negative slope.

Caltech101 dataset

Architecture	Activation	s	Accuracy	Std
ResNet18	ReLU with ProxyGrad	0.01	59.3	1.7
	ReLU with ProxyGrad	0.1	59.5	1.5
	ReLU with ProxyGrad	0.2	**62.9**	**0.8**
	Leaky ReLU	0.01	58.3	1.4
	Leaky ReLU	0.1	59.3	1.6
	Leaky ReLU	0.2	62.2	0.8
	ReLU	0	56.0	0.6
PFNet18	ReLU with ProxyGrad	0.01	63.5	1.0
	ReLU with ProxyGrad	0.1	63.7	0.7
	ReLU with ProxyGrad	0.2	**64.6**	**1.0**
	Leaky ReLU	0.01	63.7	1.4
	Leaky ReLU	0.1	63.4	0.9
	Leaky ReLU	0.2	63.5	0.7
	ReLU	0	63.9	0.7

CUB dataset

Architecture	Activation	s	Accuracy	Std
ResNet18	ReLU with ProxyGrad	0.01	58.9	0.4
	ReLU with ProxyGrad	0.1	58.8	0.3
	ReLU with ProxyGrad	0.2	56.6	0.3
	Leaky ReLU	0.01	59.6	0.3
	Leaky ReLU	0.1	**61.2**	**0.4**
	Leaky ReLU	0.2	**61.2**	**0.6**
	ReLU	0	58.8	0.5
PFNet18	ReLU with ProxyGrad	0.01	53.3	0.4
	ReLU with ProxyGrad	0.1	53.3	0.2
	ReLU with ProxyGrad	0.2	52.7	0.4
	Leaky ReLU	0.01	53.3	0.3
	Leaky ReLU	0.1	53.2	0.5
	Leaky ReLU	0.2	**53.4**	**0.3**
	ReLU	0	53.2	0.3

Flowers dataset

Architecture	Activation	s	Accuracy	Std
ResNet18	ReLU with ProxyGrad	0.01	74.1	0.2
	ReLU with ProxyGrad	0.1	74.7	0.5
	ReLU with ProxyGrad	0.2	75.3	0.4
	Leaky ReLU	0.01	74.3	0.6
	Leaky ReLU	0.1	75.7	0.4
	Leaky ReLU	0.2	**76.1**	**0.3**
	ReLU	0	74.0	0.3
PFNet18	ReLU with ProxyGrad	0.01	80.9	0.4
	ReLU with ProxyGrad	0.1	80.8	0.5
	ReLU with ProxyGrad	0.2	80.7	0.5
	Leaky ReLU	0.01	**81.0**	**0.4**
	Leaky ReLU	0.1	80.9	0.4
	Leaky ReLU	0.2	**81.0**	**0.4**
	ReLU	0	80.8	0.2

The experiments are conducted for the ResNet18 architecture [30] and the Pre-defined Filter Network 18 (PFNet18) architecture [51]. For an introduction to the ResNet architecture, please refer to Section 2.3. PFNet18 is a ResNet18 variant with depthwise convolution, where the spatial convolutional part does not change during training. The architecture is presented in Chapter 7 in detail. We chose PFNet18 because it follows a different learning philosophy and potentially exhibits different training dynamics. The networks are trained until convergence with the hyper-parameters shown in Section 2.5.1.

Table 5.3 presents the mean test accuracy and the standard deviation of five experiments with different seeds. ReLU with ProxyGrad is well-suited for adjusting the weights of deep neural networks. ProxyGrad achieves the highest test accuracies for the Caltech101 dataset with both architectures. On the other two datasets, the Leaky ReLU is the best performer. However, the performance differences are minor.

High negative slopes between 0.1 and 0.2 tend to have the best test performance in our experiments. A negative slope of 0.01, the default setting used in many applications, belongs to the top performers only once (PFNet18 trained on the Flowers dataset). In the remaining five cases, a negative slope of 0.1 or higher reaches the highest test accuracies. However, higher slopes do not always perform better. In preliminary experiments, we found that negative slopes higher than 0.2 lead to inferior performance.

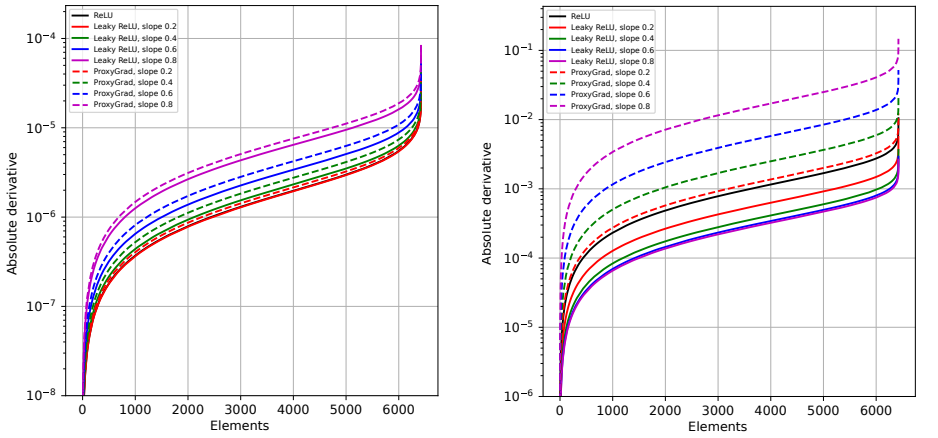
5.5 ProxyGrad Provides Large Gradients

We also study how the negative slope affects the magnitude of the gradients within ResNet18 in different layers. We found that ReLU with ProxyGrad provides larger gradients with increasing slope. In Leaky ReLU networks, the gradients decrease with increasing slope. This finding has implications for the choice of hyper-parameters and supports our proposal to normalize the gradient during AM in Section 5.3.

To show this phenomenon, we compute the derivatives of the cross-entropy loss in a randomly initialized ResNet18. For computation, we use randomly selected samples from the Caltech101 dataset. The experiment is repeated 50 times with different network initializations and input batches to obtain statistically robust results. Table 5.4 shows the absolute derivatives after the output of the first convolutional layer in the first block. We observed similar results at different positions in the network. Without batch normalization, the derivatives increase with a higher negative slope. With batch normalization layers, the derivatives decrease for Leaky ReLU networks. We attribute this phenomenon to the normalization step. Increasing the slope increases the variance of the output of a Leaky ReLU (see Appendix A.2). The subsequent batch normalization layer divides the upstream gradient by the high standard deviation during the backward pass. Thus, the downstream gradient magnitude decreases. This

Table 5.4: Derivatives of the cross-entropy loss after the output of the first convolutional layer in the first block of ResNet18. The x axis shows the sorted elements of the gradient vector.

Without batch normalization	With batch normalization



effect accumulates with every batch normalization layer, leading to small gradients in the early layers within the network. During training, the effect can be reduced by choosing a higher learning rate. ProxyGrad, however, does not suffer from gradient degradation and produces larger gradients.

Chapter 6

Enhancing Generalization in Convolutional Neural Networks through Regularization with Edge and Line Features

The previous chapters employed visualization to advance our comprehension of CNNs. Another approach to increase CNN transparency involves systematic simplification. The idea is to introduce understandable, pre-defined filters to deep CNNs. Instead of traditional convolutional layers, we incorporate Pre-defined Filter Modules (PFMs), which convolve the input data using a fixed set of 3×3 pre-defined filters. A subsequent ReLU erases negative filter responses. Then, a 1×1 convolutional layer generates linear combinations of the rectified filter outputs. The channel-wise and point-wise convolutions implement our approach described by Equation (1.6). Our method increases the test accuracies by $5 - 11$ percentage points across four fine-grained classification datasets.

The chapter is structured as follows. Section 6.1 demonstrates the effectiveness of our regularization method using a new toy dataset for binary image classification. Subsequently, implementation details are presented in Section 6.2. Section 6.3 shows that the generalization abilities of ResNet [30] and DenseNet [36] improve on the Fine-Grained Visual Classification of Aircraft dataset (FGVC Aircraft) [62], StanfordCars

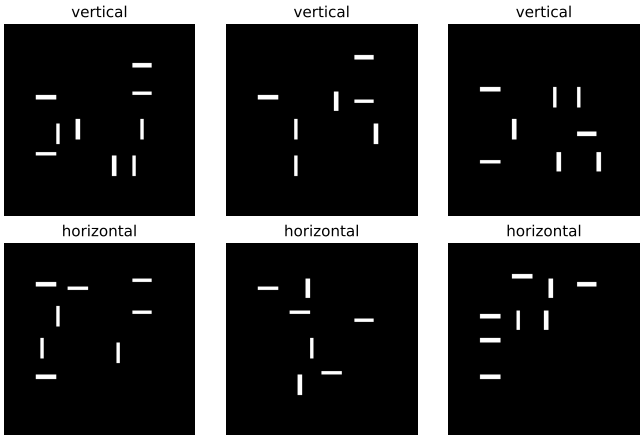


Figure 6.1: Samples from the toy dataset with the classes *vertical* and *horizontal*.

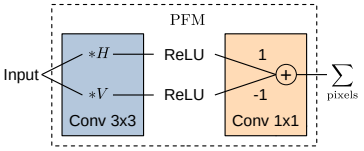
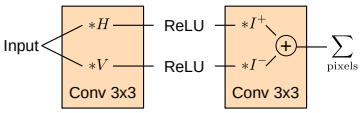
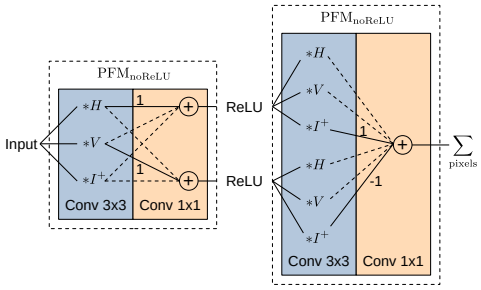
[44], Caltech-UCSD Birds-200-2011 (CUB) [95], and the 102 Category Flower dataset (Flowers) [73] for edge and line detectors as pre-defined filters. In Section 6.4, we argue why adding linearly dependent edge kernels further increases the performance on the benchmark datasets. Furthermore, we show that the number of dimensions spanned by the set of filters has a minor effect on performance. Section 6.5 studies how many pre-defined filters are needed for optimal recognition performance. Section 6.6 discusses the relevance of skip connections in PFCNNs. Specifically, we show that aliasing effects in the skip connections or missing skip connections can negatively impact the performance of PFCNNs.

The research is published at the 2024 International Conference on Artificial Neural Networks (ICANN) [53]. The implementation of PFCNNs is available on `github.com/Criscraft/PredefinedFilterNetworks`. The dataset is available on `github.com/Criscraft/Oriented_Dashes_Classification_Dataset`.

6.1 Toy Dataset

Image recognition on a toy dataset further motivates the usage of pre-defined filters in CNNs. We study a simplified dataset for binary image classification that requires the processing of gradient information. The task is determining whether the image has more horizontal than vertical dashes, requiring the network to identify their orientations. Thus, effective utilization of gradient information is essential for solving this problem. We show that one PFM with pre-defined edge filters can solve the problem with only two trainable parameters, while a fully convolutional network with 3×3 kernels and ReLUs would require at least two layers and 36 parameters. Therefore, PFMs seem to suit problems where the orientations of edges are relevant.

Table 6.1: Different implementations of (6.1). p denotes the number of trainable parameters. Orange boxes contain trainable parameters. Blue boxes contain pre-defined filters. A dashed line corresponds to the weight value zero. I^+ denotes the identity filter and $I^- = -I^+$

Model	p	Illustration
PFM	2	
CNN	36	
$\text{PFM}_{\text{noReLU}}$	12	

The dataset contains 1024 images featuring various horizontal and vertical dashes. Figure 6.1 shows some example images. The grayscale images have a shape of 48×48 pixels. The dashes are one pixel thick and five pixels long. Scenarios with an equal number of horizontal and vertical dashes do not occur. The function $f : \mathbb{R}^{H \times W} \to \mathbb{R}$

$$f(\mathbf{x}) = \sum_{i,j} \left(\text{ReLU}(H * \mathbf{x}) - \text{ReLU}(V * \mathbf{x})\right)[i, j] \tag{6.1}$$

solves the problem with the pre-defined kernels

$$H = \begin{pmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{pmatrix}, \quad V = \begin{pmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{pmatrix}. \tag{6.2}$$

A non-negative output refers to the *horizontal* class, while a negative one refers to the *vertical* class. Indeed, Function (6.1) correctly classifies all images in the dataset.

Table 6.1 presents three variants to implement (6.1) as a CNN. For the sake of simplicity, we ignore padding in our examples. The first variant uses our approach. It employs only one PFM consisting of two pre-defined edge kernels, a ReLU, and a

trainable 1×1 convolution with two parameters. The second variant uses two common convolutional layers connected by a ReLU. The first convolutional layer has one input and two output channels. The second layer has two input channels and one output channel. The architecture needs a total of 36 trainable parameters. The third variant employs two $\text{PFM}_{\text{noReLU}}$ implementing (1.5) without the intermediate ReLU function. It needs a third pre-defined kernel (the identity filter) to implement (6.1). Here, all layers share the same set of pre-defined filters. The third variant has 12 trainable parameters.

The first variant using one PFM has the fewest trainable parameters. It appears well-suited for image recognition problems where image gradients are relevant. In the subsequent section, we apply PFMs in deep CNN architectures and demonstrate that edge and line filters provide a beneficial bias for challenging real-world image recognition problems.

6.2 Architecture and Sets of Filters

Starting from ResNet18 described in Section 2.3, all convolutional layers are substituted with PFMs. The resulting network is denoted PFNet18. ResNet18 contains three skip connections with a 1×1 convolution and a stride of two. As motivated by Section 6.6, these three skip connections are enhanced by smoothing their inputs using a Gaussian filter. This step addresses aliasing issues and increases the performance of PFNets.

To further study the applicability of our regularization method, we introduce PFMs to DenseNet121 [36] as an additional backbone architecture. Similar to ResNet, DenseNet is a widely used architecture in image recognition. It consists of 121 layers and contains 12.7 million trainable parameters. For a detailed explanation of DenseNet, see Section 2.3.

All PFMs within a single network share the same pre-defined kernels, which remain unchanged throughout training, if not mentioned otherwise. The experiments involve edge and line detectors in various orientations, as shown in Figure 6.2. The filters are mean-free, and the sum of their absolute elements is one.

We also employ random filters where the filter elements are drawn from a uniform distribution [-1, 1] without normalization. All PFMs of the same network share the same random filters. Nevertheless, distinct random seeds lead to different filters. We also utilize translating filters. Their corresponding kernels have one element being one and the other elements being zero.

In additional experiments, we vary the number of pre-defined filters p in Equation (1.6). The number of trainable parameters of a PFM is $p \cdot c_{\text{in}} \cdot c_{\text{out}}$, the number of input and output channels. By default, $p = k^2$ where $k \times k$ is the filter size. Table 6.2 shows the sizes of ResNet18-based models for different p. PFNet with nine pre-defined

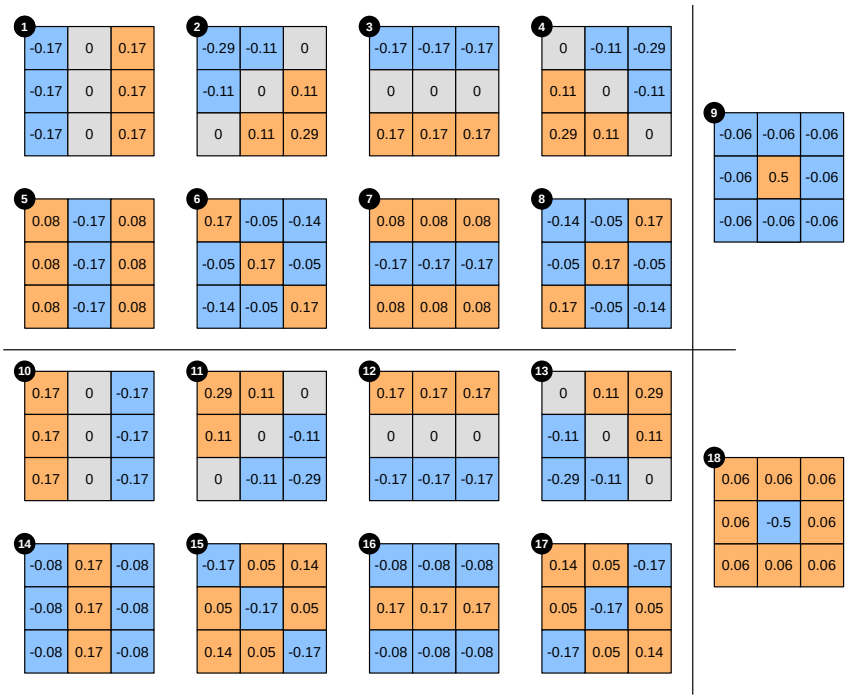


Figure 6.2: Set of pre-defined filter kernels used in the experiments.

filters exhibits the same number of parameters as the baseline, ResNet18. Table 6.2 also demonstrates the time needed for the forward and backward pass. The times were measured for input tensors of shape $48 \times 3 \times 224 \times 224$ on an NVIDIA GeForce RTX 4090 GPU. Our regularization method tends to be slower compared to the baseline. One reason is that each convolutional layer in the original network is replaced by a PFM with two convolution steps, leading to more nodes in the computational graph.

6.3 Performance on Benchmark Datasets

The models are trained and tested on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [80], Fine-Grained Visual Classification of Aircraft dataset (FGVC Aircraft) [62], StanfordCars [44], Caltech-UCSD Birds-200-2011 (CUB) [95], and the 102 Category Flower dataset (Flowers) [73] described in Section 2.2.

The trainable network weights are initialized using Kaiming initialization [31]. The training hyperparameters for the ILSVRC are summarized in Section 2.5.2. For the remaining datasets, we use the default training hyperparameters from Section 2.5.1 with the batch size set to 32 and the initial learning rate set to $\mathrm{lr}_{\mathrm{init}} = 0.0015$. Table 6.3 presents the average test performance of five models trained with different seeds. The filter type 'edge, line' uses the edge and line detectors from Figure 6.2 starting

Table 6.2: Number of trainable parameters of PFNet18 in millions. Below, the time for the forward pass (FP) and backward pass (BP) for input tensors of shape $48 \times 3 \times 224 \times 224$ in milliseconds on an NVIDIA GeForce RTX 4090 GPU.

# Filters:	2	4	8	9	13	18	ResNet18
# Parameters:	2.7	5.2	10.1	11.3	16.2	22.4	11.3
Time FP:	7	11	21	24	33	46	6
Time BP:	14	21	36	40	54	73	14

Table 6.3: Average test accuracy. The pre-defined filters are not adjusted to the training data.

Backbone	Filter type	# Filters	Flowers	CUB	FGVC Aircraft	StanfordCars
ResNet18	Translating	9	73.01±0.61	55.59±0.62	73.72±0.39	77.47±0.44
ResNet18	Random	9	78.82±1.99	60.29±0.62	74.59±3.74	79.80±2.64
ResNet18	Random	18	81.16±1.80	62.66±1.01	77.60±1.59	81.96±1.45
ResNet18	Edge, line	9	84.28±0.23	61.28±0.28	79.66±0.20	82.64±0.17
ResNet18	Edge, line	18	**85.16±0.15**	**63.01±0.50**	**81.66±0.23**	**83.66±0.20**
ResNet18	–	–	73.4±0.34	58.51±0.53	73.32±1.06	77.9±0.37
DenseNet121	Edge, line	9	81.46±0.38	60.43±0.18	78.94±0.29	80.62±0.32
DenseNet121	Edge, line	18	**81.74±0.35**	**62.10±0.37**	**80.44±0.17**	**81.04±0.34**
DenseNet121	–	–	75.19±0.66	58.26±0.60	74.03±0.38	77.43±0.26

from index one. The sets with 9 and 18 detectors both span 9 dimensions.

PFNet18 with edge and line detectors exhibits performance improvements, achieving margins of up to 11 percentage points compared to the baseline. The enhancement is consistent across all four datasets and is attributed to processing edge and line features in the convolutional layers, contributing to the regularization of the models. The experiments with DenseNet121 as a backbone show similar results. Interestingly, having more than 9 filters enhances the test performance further, even though the additional kernels in the set of pre-defined filters are linearly dependent. Section 6.4 studies this phenomenon in detail.

Experiments with pre-defined filters, randomly drawn from a uniform distribution around zero, are also shown in Table 6.3. Occasionally, PFNet18 with random filters surpasses the baseline model. However, the networks exhibit a high standard deviation in test accuracy, reaching up to 3.74 percentage points for the FGVC Aircraft dataset. Some random filter sets are more or less suited to the specific recognition tasks.

In alternative experiments, we employ translating filters instead of edge, line, or random filters. Referring to (1.6), the translating filters mimic the learning of the

Table 6.4: Accuracy on the validation set of the ILSVRC (ImageNet). The pre-defined filters are not adjusted to the training data.

Filter type	# Filters	Top 1	Top 5
ResNet18 [30]	–	72.12	–
ResNet18 (pre-trained Pytorch model [74])	–	69.76	89.08
ResNet18 (baseline)	–	69.60	89.13
Edge, line	9	67.59	88.13
Edge, line	18	70.16	89.49

Table 6.5: Average test accuracy. The first column describes the initialization of the pre-defined filters. The filters are adjusted to the training data.

Filter type	# Filters	Flowers	CUB	FGVC Aircraft	StanfordCars
Translating	9	74.22±0.92	59.35±0.80	74.56±0.38	79.29±0.70
Random	9	79.61±1.96	60.92±1.58	75.19±3.25	80.08±2.31
Random	18	80.65±1.35	62.94±0.89	78.01±1.89	81.81±1.61
Edge, line	9	84.29±0.33	61.93±0.49	78.09±0.36	81.54±0.45
Edge, line	18	**84.62±0.41**	**63.17±0.48**	**79.54±0.55**	**82.74±0.27**
ResNet18	–	73.4±0.34	58.51±0.53	73.32±1.06	77.9±0.37

convolutional filters in the canonical basis. Compared to ResNet18, the performance drops up to 3 percentage points. We attribute the decline to the ReLU in the first layer, which sets approximately half of the pixels of the original input image to zero. The impact is different on each dataset.

Table 6.4 presents the test accuracies on the ILSVRC (ImageNet). Our regularized model with nine filters exhibits a performance that is 2 percentage points below the baseline. We attribute this to the high similarity between training and test accuracy. Our regularization might not be necessary in this scenario. Instead, the model's capacity should ideally increase. Indeed, when we augment the network with more filters, thereby expanding its capacity, the model shows a slight improvement over the baseline.

Furthermore, we adjust the pre-defined filters to the train data. We allow each PFM to learn its own set with nine or 18 filters. However, these experiments do not improve the performance metrics, as shown in Table 6.5. The training process struggled to identify filters that outperformed our pre-defined edge and line detectors, underscoring their good generalization abilities.

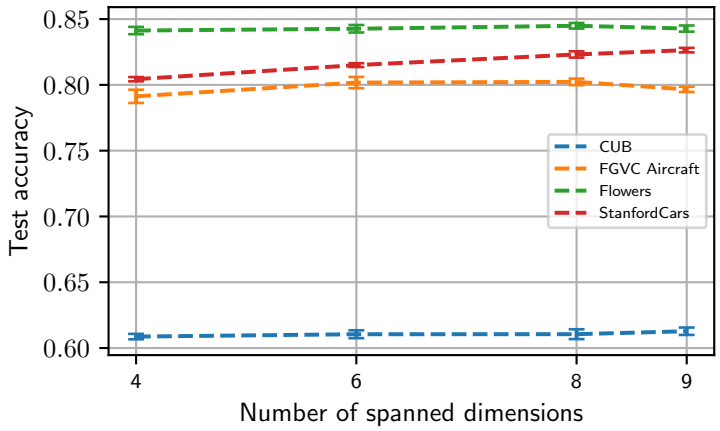


Figure 6.3: Average test accuracy when using nine edge and line detectors that span a variable number of dimensions.

6.4 Number of Dimensions Spanned by the Set of Pre-defined Filters

This section investigates how the number of dimensions spanned by the set of pre-defined filters affects the recognition performance. The number of dimensions means how many of the nine possible dimensions are spanned by the set of pre-defined filters. It is obtained by counting the number of linearly independent filters. In the following experiments, PFNet18 is trained using nine pre-defined kernels that span a varying number of dimensions. To get four dimensions, we choose the filters 1, 3, 5, 7, 10, 12, 14, and 16, and the sum of 14 and 16 from Figure 6.2. Figure 6.3 presents the average test accuracies from five runs with different seeds. The results suggest that the dimensionality spanned by the pre-defined filters has minimal influence on performance, with four dimensions already yielding satisfactory results.

Interestingly, Table 6.3 unveils a notable performance gain between using nine and 18 pre-defined kernels by margins of $1-3$ percentage points. The nine 3×3 pre-defined filters already span all nine dimensions. We suggest that the additional ReLU in the PFM can make the features linearly independent, even if the filter kernels are linearly dependent. A PFM with one input channel and two pre-defined filters $\tilde{w}_1, \tilde{w}_2 \in \mathbb{R}^{k\times k}$ contains the two functions

$$\begin{aligned} f^{(\tilde{w}_1,m,n)}(\mathbf{x}) &= \mathrm{ReLU}(\tilde{w}_1 * \mathbf{x})[m,n] \\ f^{(\tilde{w}_2,m,n)}(\mathbf{x}) &= \mathrm{ReLU}(\tilde{w}_2 * \mathbf{x})[m,n] \\ f^{(\tilde{w}_1,m,n)}, f^{(\tilde{w}_2,m,n)} &: \mathbb{R}^{M\times N} \to \mathbb{R} \end{aligned} \tag{6.3}$$

with pixel coordinates $m, n \in N$. As shown in the appendix A.3, if the filters are

Table 6.6: Average test performance on the benchmark datasets. All experiments but the baseline use $\text{PFM}_{\text{noReLU}}$ modules without the additional ReLU. The pre-defined filters are not adjusted to the training data.

Filter type	# Filters	Flowers	CUB	FGVC Aircraft	Stanford Cars
Translating	9	74.93±0.72	59.62±0.52	74.51±0.58	79.84±0.40
Random	9	78.51±1.64	59.28±1.55	73.62±4.21	79.11±2.53
Random	18	78.54±2.27	**60.55±1.02**	**75.05±2.13**	**80.04±1.47**
Edge, line	9	78.29±0.38	50.01±0.69	71.25±0.40	72.56±0.28
Edge, line	18	**79.38±0.35**	50.41±0.46	72.43±0.59	73.68±0.49
ResNet18	–	73.4±0.34	58.51±0.53	73.32±1.06	77.9±0.37

linearly dependent with $a\tilde{w}_1 = \tilde{w}_2$ and $a < 0$, then the functions in (6.3) are linearly independent. In our experiments, the PFM benefits from having a negative copy of the nine linearly independent pre-defined filters because the resulting 18 rectified convolution outputs become linearly independent. The PFM (ignoring normalization layers) can be written as

$$\begin{aligned} \text{PFM}[m,n] &= \sum_{c=1}^{c_{\text{in}}} \sum_{l=1}^{2} q_{cl} \text{ReLU}(\tilde{w}_l * \mathbf{x})[m,n] \\ &= q_{11}\text{ReLU}(\tilde{w}_1 * \mathbf{x})[m,n] \\ &\quad + q_{12}\text{ReLU}(a\tilde{w}_1 * \mathbf{x})[m,n] \\ \text{Case 1: } &(\tilde{w}_1 * \mathbf{x})[m,n] \geq 0 : q_{11}(\tilde{w}_1 * \mathbf{x})[m,n] \\ \text{Case 2: } &(\tilde{w}_1 * \mathbf{x})[m,n] < 0 : aq_{12}(\tilde{w}_1 * \mathbf{x})[m,n]. \end{aligned} \tag{6.4}$$

The convolution output is either multiplied with q_{11} or aq_{12}. Here, the ReLU acts like a switch, deciding which weight to apply. The ReLU also ensures that only one of the two channels is active. We conclude that the set of pre-defined filters should incorporate pairs of filter kernels with inverted signs. For instance, the filters $\tilde{w}_1$ and $\tilde{w}_2 = -\tilde{w}_1$ could represent two edge detectors of opposing directions (for reference, see filters one and ten in Figure 6.2).

To study the effect of linearly dependent filters on performance, we repeat the experiments conducted in Section 6.3 in an ablation study. This time, we employ $\text{PFM}_{\text{noReLU}}$ modules without the additional ReLU function as described in (1.5). Since the pre-defined filters span all nine dimensions, the network can still effectively learn all convolution kernels. Regularization does not occur. As expected, the results presented in Table 6.6 exhibit a notable performance drop for edge and line filters and a weaker drop for random kernels. The baseline model ResNet18 outperforms the edge and line filters on three of the four datasets. These results underline the relevance of the additional ReLU in the PFM.

Table 6.7: Filter subsets of different sizes and types.

Filter type	# Filters	Filter selection (see Fig. 6.2)
even	2	5, 7
even	4	5, 7, 14, 16
even	8	5 - 8, 14 - 17
uneven	2	1, 3
uneven	4	1, 3, 10, 12
uneven	8	1 - 4, 10 - 13
even-uneven	9	1 - 9
even-uneven	13	1 - 9, 11, 13, 15, 17
even-uneven	18	1 - 18

Furthermore, Table 6.6 shows that translating filters achieve test accuracies comparable to the baseline. This result complements the prior experiments with translation filters where an additional ReLU function decreased the recognition rates by 3 percentage points. The additional ReLU lowered the performance by setting dark input pixels to zero, erasing half of the original image's information.

6.5 Nine or More Filters Provide Optimal Results

This section studies the effect of the number of pre-defined filters on the recognition performance. We train PFNet18 on the CUB and the Flowers dataset using the filter subsets in Table 6.7. As illustrated in Figure 6.4, the best results are obtained utilizing all 18 filters. The test accuracies drop when choosing four or fewer filters. It is worth noting that a reduction in the number of pre-defined filters limits the information transferred to the subsequent layer and also leads to a decrease in trainable parameters, thereby diminishing the model's capacity, as shown in Table 6.2. The edge filters (green color) often outperform the line filters (yellow color) or the random filters (red color). Given the abundance of edges in images, we hypothesize that edge filters convey more information than lines for the vision problems discussed.

6.6 The Relevance of Skip Connections

This section highlights the relevance of residual connections in small PFCNNs. He et al. [30] introduced residual connections to address the vanishing gradient problem and performance degradation in deep networks. Section 2.3 describes residual networks in detail. We train PFNet18 with nine edge and line filters using the default training hyperparameters, with and without skip connections. For comparison, we also augment the VGG13 architecture with pre-defined filters (PFVGG13). This architecture lacks skip connections by default, as described in Section 2.3.

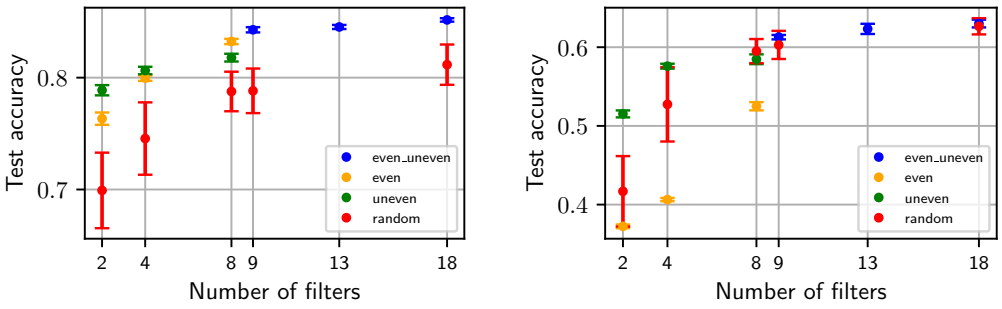


Figure 6.4: Average test accuracy when using a variable number of filters. Left: Flowers dataset. Right: CUB dataset.

Table 6.8: Ablation study on the relevance of skip connections. *PFNet default* denotes the PFNet model with 9 line and edge filters. *PFNet w/o skip con.* denotes the same model but without skip connections.

	Flowers	CUB	FGVC Aircraft	Stanford Cars
PFNet default	84.28±0.23	61.28±0.28	79.66±0.20	82.64±0.17
PFNet with aliasing	77.54±0.32	52.64±0.67	77.99±0.38	80.60±0.41
PFNet w/o skip con.	42.41±0.64	30.55±1.12	76.12±0.64	77.37±0.48
ResNet18	73.40±0.34	58.51±0.53	73.32±1.06	77.90±0.37
PFVGG13	40.16±1.60	30.26±1.40	67.09±0.36	63.72±0.52
VGG13	74.79±0.57	56.11±0.78	73.53±0.68	74.69±0.29

Table 6.8 shows low performance on all datasets when skip connections are lacking. The accuracy drops from 84.28% to 42.41% for PFNet18 on the Flowers dataset. The CUB dataset shows a similar drop from 61.28% to 30.55%. Similar drops occur when comparing the PFVGG13 model to VGG13 on the Flowers and CUB datasets. In contrast, the FGVC Aircraft and Stanford Cars datasets are less affected, possibly due to the lack of organic shapes and natural textures in the aircraft and car models.

We hypothesize that the performance drop in PFNet18 without skip connections is due to the inability of the PFMs to learn the identity mapping. As our set of pre-defined filters does not include the identity filter $\left(\begin{smallmatrix}0&0&0\\0&1&0\\0&0&0\end{smallmatrix}\right)$, learning the identity mapping does not occur, inhibiting the reuse of features in deeper layers. Learning an approximation of the identity might be challenging depending on the choice of pre-defined filters. Future experiments could test this hypothesis by including an additional identity filter in the set of pre-defined filters.

Aliasing in the skip connections can also degrade the performance of PFCNNs. In PFNet18 and ResNet18, three skip connections exist where the resolution is reduced by spatial sub-sampling. Here, a 1×1 convolution with a stride of two discards

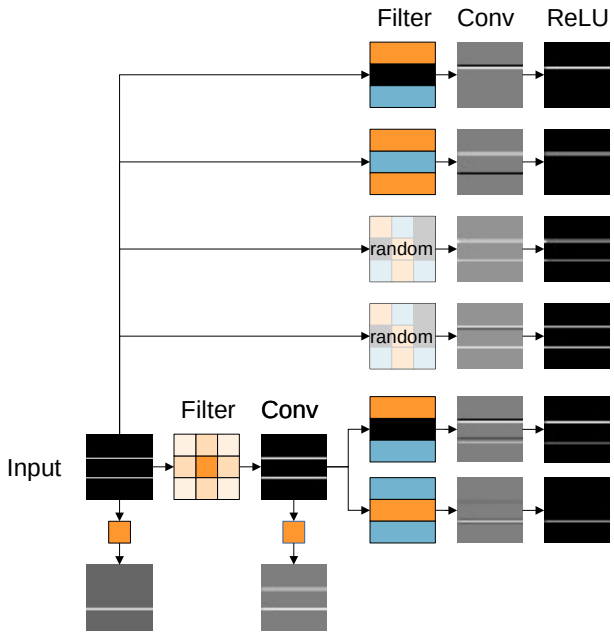


Figure 6.5: Aliasing effects when applying stride 2 convolutions on an input image showing two horizontal lines. Orange color denotes positive kernel weights, black denotes zero, and blue negative weights. The convolution operations do not always capture both lines.

information. To mend aliasing in our default PFNet18, we blur the input data of these three convolution operations with a 3×3 Gaussian filter. Without this step, we observe performance drops as demonstrated by Table 6.8, where the performance drops by almost 10 percentage points on the CUB dataset. Again, the FGVC Aircraft and Stanford Cars datasets are less affected. For the original ResNet18, we found no significant change in the test accuracies when blurring the three skip connections, suggesting that ResNet18 adapts to handle aliasing at the cost of additional resources.

Figure 6.5 illustrates the aliasing issues in more detail. It shows different convolution operations with stride two applied to a single input image. The image renders two white lines on a black background. Random 3×3 filters often capture both lines. The 1×1 convolution (bottom) only captures one line due to spatial subsampling. Convolution operations with a stride of two and 3×3 pre-defined filters also do not always capture both lines. Information can be irrevocably lost. Thus, PFCNNs rely on the skip connections to compensate for aliasing effects, and aliasing in the skip connections must be avoided.

Chapter 7

Parameter Reduction using Pre-Defined Filter Networks

The previous chapter introduced pre-defined filters to deep CNNs to improve their transparency and increase their generalization abilities. This chapter aims to reduce the number of trainable weights in CNNs by utilizing pre-defined filters. Equation (1.6) suggests that having only one pre-defined filter per input channel ($p = 1$) reduces the number of parameters. Following this idea, we build lightweight models for image recognition.

The chapter is structured as follows: Section 7.1 provides all details about our reduced PFNet18 architecture. Section 7.2 explains the experimental setup. Section 7.3 shows that the reduced PFNet18 with edge and line filters outperforms ResNet18 on some fine-grained image datasets. Additionally, we show that the reduced PFNet18 can learn complex, discriminative features. Also, we find that edge and line filters outperform random filters. The work in this chapter was presented at the 2023 International Joint Conference on Neural Networks (IJCNN) [51].

7.1 Reduced Pre-defined Filter Module

The reduced Pre-defined Filter Module, defined by Equation (1.7), is illustrated in Figure 7.1. We utilize $k = 16$ unique pre-defined edge and line filters in our reduced PFMs. Specifically, we employ the filters 1-8 and 10-17 from Figure 6.2. We distribute the filters across the input channels in a fixed order, with the input channel indexed by c convolved with the kernel indexed by $c \bmod k$ as can be seen in (1.7).

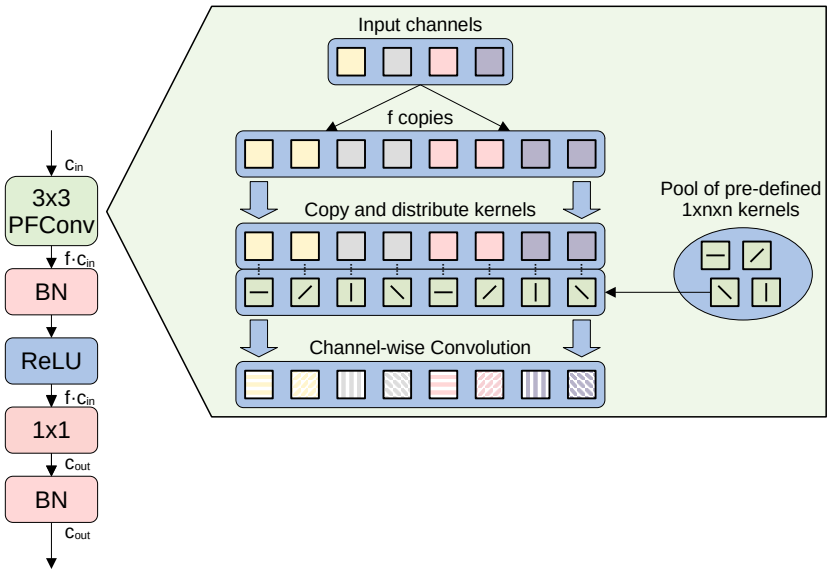


Figure 7.1: Illustration of the reduced Pre-defined Filter Module as described in (1.7). The order in which the pre-defined filters are distributed over the input channels is fixed. The reduced PFM uses $f = 1$.

We construct the reduced PFNet18 architecture similar to the PFNet18 in the previous Chapter 6 but with reduced PFMs in all layers except in the first layer, where we use a default PFM. The reduced PFNet18 requires only 13% of the training parameters of a ResNet18.

A general PFM framework

The default PFM and the reduced PFM can combined by a general PFM framework. It is defined by

$$\sum_{c=1}^{f \cdot c_{\text{in}}} w_c \cdot \text{ReLU}(h_{c \bmod k} * g_{\lfloor c/f \rfloor})[m, n]. \tag{7.1}$$

with the growth factor f. Conceptually, f determines the number of copies of the c_{in} input channels before convolution, as illustrated in Figure 7.1. The vanilla PFM is obtained by $f = k$ with k, the number of pre-defined filters in the filter pool. This setting includes each combination of input channel and pre-defined filter as shown in Equation (1.6). $f = k = 9$ leads to the same number of trainable parameters as the baseline model, ResNet18. $f = 1$ provides the reduced PFM where each input channel only receives one pre-defined filter. After applying the pre-defined filters, the number of intermediate channels is $c_{\text{int}} = c_{\text{in}} \cdot f$. Our implementation requires $c_{\text{int}} \mod k = 0$ with the number of pre-defined kernels k. Another requirement is $c_{\text{int}} \geq c_{\text{in}}$.

The order of the pre-defined filters in the filter pool $(h_1, ..., h_a, ..., h_b, ..., h_k)$ should not impact the set of functions F that can be learned by the network. Similarly, permuting the channels in the input image should not be reflected in the network's architecture. This requirement has implications for the choice of f, which we will discuss in the following. First, we study the first PFM in the network. For $f = k$, each kernel is applied to each input channel. In this case, the order of the input channels or the pre-defined filters does not affect the set of learnable functions F. If we permute the input channels or the pre-defined filters, we can adjust the weights in the subsequent 1×1 layer to get the same outcome. In contrast, the setting $f = 1$ produces different functions for permuted input channels. Thus, we use $f = k$ in the first PFM of our reduced networks.

Second, we discuss the second and subsequent PFMs in the network. For $f = 1$, the arrangement of pre-defined filters doesn't influence the network's learnable functions F. If two filters at positions a and b swap, this operation can be reversed in two steps. First, the preceding 1×1 layer swaps its output channels a and b by permuting its weights. If skip connections exist, all preceding PFMs adjust their 1×1 layer. Subsequently, the inputs of the following 1×1 layer need swapping, which can be accomplished by permuting the corresponding weights. For $f = k$, the set of learnable functions F is also not affected because of the commutativity of the sum in the 1×1 convolution.

The order of kernels can matter for $f \in [2, k-1]$, which we avoid. Please remember, that the input channel $\lfloor c/f \rfloor$ is convolved with the kernel $c \bmod k$ where c goes from 1 to fC. A specific input channel can only have one of f different filter combinations. However, if the order of how the pre-defined filters would be random, we had $\binom{k}{f}$ possible filter combinations for a specific input channel. This exceeds k for $f \in [2, k-1]$. Consequently, reordering filters can lead to new filter combinations, potentially altering the set of functions F.

7.2 Experimental Setup

We train and test the reduced PFNet18 architecture on several image classification datasets and compare the performance against ResNet18. We use the training hyper-parameters depicted in Section 2.5.1. Training occurs on the Caltech101, CIFAR10, CUB, FGVC Aircraft, Flowers, and Stanford Cars datasets described in Section 2.2. As no official split is available for the Caltech101 dataset, we randomly pick 20 training and 10 test images per class.

Table 7.1: Details of the reduced PFNet18 architecture with 16 pre-defined filters.

Layers	Output size	Kernel
Pre-defined Filter Module	$\begin{bmatrix} 3 \times 224 \times 224 \\ 48 \times 112 \times 112 \\ 64 \times 112 \times 112 \end{bmatrix}$	$\begin{bmatrix} \text{input}, \\ 3 \times 3, \text{depthwise} \\ 1 \times 1, \text{pixelwise} \end{bmatrix}$
MaxPooling	$64 \times 56 \times 56$	3×3, 64, stride 2
Pre-defined Filter Module	$64 \times 56 \times 56$	$\begin{bmatrix} 3 \times 3, \text{depthwise} \\ 1 \times 1, \text{pixelwise} \end{bmatrix} \times 2$
Pre-defined Filter Module	$128 \times 28 \times 28$	$\begin{bmatrix} 3 \times 3, \text{depthwise} \\ 1 \times 1, \text{pixelwise} \end{bmatrix} \times 2$
Pre-defined Filter Module	$256 \times 14 \times 14$	$\begin{bmatrix} 3 \times 3, \text{depthwise} \\ 1 \times 1, \text{pixelwise} \end{bmatrix} \times 2$
Pre-defined Filter Module	$512 \times 7 \times 7$	$\begin{bmatrix} 3 \times 3, \text{depthwise} \\ 1 \times 1, \text{pixelwise} \end{bmatrix} \times 2$
Classification layer	$512 \times 1 \times 1$	Adaptive average pool
		fully connected, softmax

7.3 Results

7.3.1 Benchmarks

The performance metrics of the PFNet18 and ResNet18 models are presented in Table 7.2. PFNet18 outperforms ResNet18 on the Caltech101, FGVC Aircraft, and Flowers datasets. On Caltech101, PFNet18 has a test accuracy that is almost 10 percentage points higher than ResNet18. On the Flowers dataset, the improvement is 7 percentage points. However, on the Stanford Cars dataset, both architectures perform similarly. ResNet18 shows higher accuracies on CIFAR10 and CUB. The experiments show that the edge and line filters provide a good bias for some image recognition problems. Notably, the reduced PFNet18 achieves these results with only 13% of the parameters of ResNet18.

7.3.2 Feature Visualization

This section shows that the reduced PFNet18 learned complex visual features. Figure 7.2 shows feature visualizations of the reduced PFNet18 and the original ResNet18 trained on the Caltech101 and Flowers datasets. The visualizations were created using activation maximization as implemented in Section 4.1.5. The Figure is best viewed

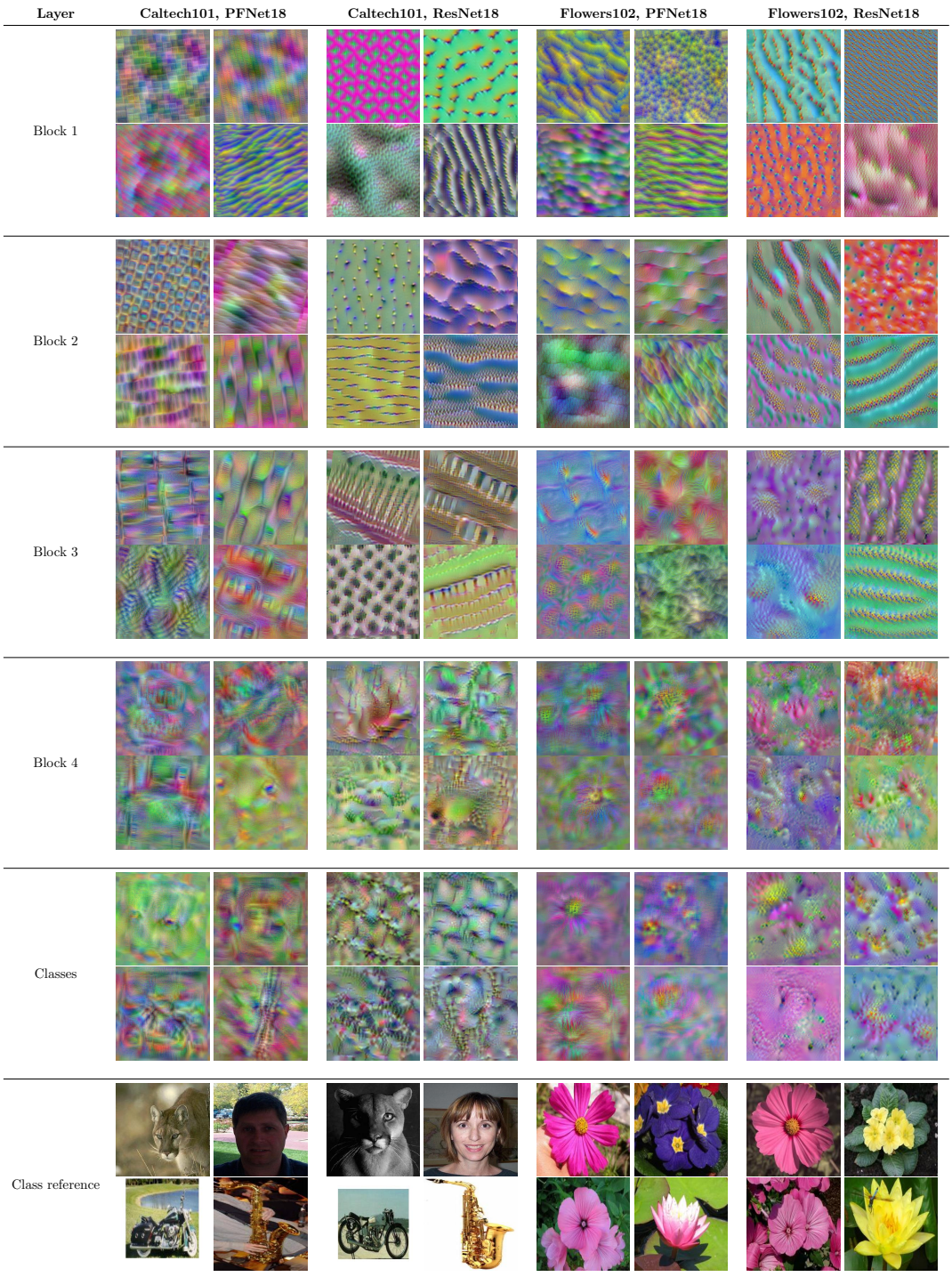


Figure 7.2: Feature visualization for models trained on the Caltech101 and Flowers datasets. Best viewed digitally with significant zoom.

Table 7.2: Test accuracy for training from scratch and 5 different seeds.

Dataset	PFNet18	ResNet18
Caltech101	**65.60±0.66**	57.19±0.78
CIFAR10	92.15±0.18	**94.33±0.11**
CUB	53.00±0.55	**58.51±0.53**
FGVC Aircraft	**75.49±0.29**	73.32±1.06
Flowers	**80.66±0.35**	73.40±0.34
Stanford Cars	77.06±0.42	**77.90±0.37**

Table 7.3: Computational efficiency of the reduced PFNet18 considering model size and speed. FP denotes forward pass and BP denotes backward pass. The input tensors have the shape $64 \times 3 \times 224 \times 224$.

Model	Parameters (10^6)	Size (MB)	FP (ms)	BP (ms)	GPU Memory FP (GB)	Mult-Adds (10^9)
PFNet18	1.46	6	37	70	3.9	0.26
ResNet18	11.23	45	27	65	3.0	1.81

digitally with significant zoom. The features processed at the end of the first block consist of rudimentary textures. At the end of the second block, more detailed textures are visible. The models trained on the Flowers dataset show leaf and flower prototypes. The models trained on Caltech101 include rectangular shapes and more variety in general. Similar observations apply for the third and fourth blocks, where the first object instances appear. The feature visualization of the classification layer reveals what the networks expect to look like cougar face, face, motorbike, and saxophone (Caltech101) and Mexican aster, primula, tree mallow, and water lily (Flowers). On the Caltech101 and Flowers datasets, the reduced PFNets18's feature visualizations look convincing: On Caltech101, one instance of the target class with a plausible shape is visible. The cougar's face consists of a head with two eyes and ears. The human face has a nose, eyes, hair, and a flat chin. The motorbike has two wheels, a saddle, and a handle. The saxophone is a long stick with buttons and a shimmering surface. In contrast, ResNet18 does not generate such shapes and seems to focus on textures. The results indicate that reduced PFCNNs can learn complex, object-specific features from only 20 training images per class.

7.3.3 Computational Efficiency

Computational efficiency and model size are relevant for training and deploying deep networks, especially on limited hardware with harsh energy and memory requirements.

Table 7.3 summarizes relevant computational aspects. The reduced PFNet18 has only 1.46 million parameters and requires only 6 MB of space on the disk, whereas ResNet18 consumes 45 MB with 11 million parameters. The reduction of parameters does not lead to a significant change in training time, as the duration of a backward pass is around 65-70 ms for a batch size of $64 \times 3 \times 224 \times 224$ on an NVidia GTX 1080 Ti. PFNet18 needs an additional time of 10 ms per batch. The long processing time is interesting because the number of mult-adds of PFNet18 is only $0.26 \cdot 10^9$ while ResNet18 has $1.81 \cdot 10^9$ mult-adds. Counting mult-adds means counting the FLOPs of a model and dividing the result by 2. Although PFNet18 requires much fewer computations, its implementation requires more nodes in the computational graph and more distinct GPU calls. Our implementation of reduced PFMs seems to lack efficiency on current GPU hardware, which is fast for few but large tensor operations. Therefore, we assume there is still much room for optimizations.

7.3.4 Ablation Study

We also study the relevance of single elements in the reduced PFM. Table 7.4 presents the results of an ablation study. The experiments are conducted on the Flowers dataset with the same hyper-parameters as in the previous experiments. The results show average values for five different seeds.

First, we show that edge and line filters achieve higher test accuracies than random filters. We pick 16 random pre-defined kernels from a uniform distribution. These kernels are constant during training. Only the 1×1 convolutions are adjusted to the Flowers dataset. In this setting, the performance drops to $73.98 \pm 1.24\%$. Interestingly, it is still as good as ResNet18. These results are consistent with the findings in the previous Chapter 6.

In another experiment, the random pre-defined kernels are allowed to adjust to the training data. We give each reduced PFM its own 16 kernels to optimize. This model has a test accuracy of $74.54 \pm 0.88\%$, performing better than the baseline model ResNet18 but worse than our edge and line filters. When the pre-defined filters are initialized with the edge filters and allowed to adapt during training, the test accuracy is $80.11 \pm 0.62\%$, less than the default experiment where we keep the pre-defined filters constant. Our findings indicate that the edge and line filters provide a beneficial bias for the Flowers dataset and that adjusting the kernels during training is unnecessary.

We also study the importance of the additional ReLU in the reduced PFMs by removing it. The convolution with the pre-defined kernels and the subsequent 1×1 convolution form one linear operation. The performance drops to $75.21 \pm 0.23\%$. The importance of the ReLU is consistent with our findings in Section 6.4.

As in the previous chapter, we analyze the role of skip connections in the reduced PFNet18. In the architecture, three skip connections with a 1×1 convolution and a stride of two drop information due to spatial subsampling. We train the models with

Table 7.4: Test accuracy of variants of the reduced PFNet18 and the original ResNet18 on the Flowers dataset.

Model	Description	Accuracy
PFNet18	No aliasing, default	**80.66±0.35**
PFNet18	Aliasing	72.40±0.59
PFNet18	First ReLU removed, no aliasing	75.21±0.23
PFNet18	16 Trainable filters, edge init., no aliasing	80.11±0.62
PFNet18	16 Trainable filters, random init., no aliasing	74.54±0.88
PFNet18	16 frozen filters, random init., no aliasing	73.98±1.24
ResNet18	Default	73.40±0.34
ResNet18	No aliasing	**74.36**±0.57

and without blurring the input data of these three convolution operations with a 3×3 Gaussian filter. For ResNet18, there is no significant change in the test accuracy on the Flowers dataset. The reduced PFNet18 improves its test accuracy from $72.40 \pm 0.59\%$ to $80.66 \pm 0.35\%$. This performance gap indicates that the aliasing effects in the skip connections are more harmful for PFNet18 than for ResNet18, which is consistent with the findings in Section 6.6.

Chapter 8

Discussion

We Need Holistic Visualizations to Comprehend CNNs

Chapter 3 evaluated the t-SNE and Grad-CAM visualization techniques in their ability to explain CNNs trained for COVID-19 detection using CT images. First, we validated that the trained models could distinguish COVID-19 from other lung diseases. Each visualization method explained a specific aspect of the models: In Section 3.2, t-SNE unveiled that the InceptionV3 model trained on the SARS-CoV-2 dataset had a wide margin between the two classes. Also, the DenseNet169 model trained on the COVID-19 CT dataset had a distinguishable feature space. The Grad-CAM technique showed that both models focus on the relevant image regions marked by specialists when detecting COVID-19 (see Section 3.3). Moreover, we tested our models on external CT images from different publications, and the models accurately localized the COVID-19-associated regions as marked by expert radiologists.

Nonetheless, the visualization methods lacked a holistic view. Some relevant model aspects were not covered: a) Information about the learned features and the strategies to identify COVID-19 manifestations. b) The computational graph, which is crucial for understanding the network architecture and data flow. c) Information about the activations. d) The distribution of the weights. e) The training process, including the loss landscape, the gradient flow, and the training data. To understand how specific features emerge within CNNs, we need holistic visualization methods that cover several aspects. Together, such methods should complement each other and provide users with an overview of the network internals.

Holistic Visualization of CNNs in 3D Space

We proposed DeepVisionVR, a holistic approach to visualize CNNs in a virtual 3D space. The software combines the visualization of various model aspects as detailed in Chapter 4. We designed our visualization to arrange comprehensive data in 3D space clearly and provided suitable user interactions (see Section 4.1.3). For instance, users can move to specific layers to focus on the information they want. From a distance, users gain a high-level overview of the computational graph. Switching between these

perspectives allows for presenting much information without risking cognitive overload. Consistent with prior work [1, 12, 43, 65, 81, 94] we think that interactive 3D environments have the potential to augment our understanding of CNNs. Given our brain's comprehension of 3D environments, visualizing network architectures in this manner may help manage their complexity.

There is much room for future research: a) The design principles of the visualization are tailored to image data. Other data types like text, sound, video, or time series are not covered yet. b) Currently, the software displays feed-forward networks. Recurrent or graph networks require changes in the visualization design. c) As the deep learning community moves toward larger architectures, the computational costs for rendering CNNs will increase, and the user interaction will be more challenging. Consequently, adjustments to the visualization design will be required. d) A user study would help assess and improve the visualization design. e) The visualization could be integrated into general-purpose immersive VR platforms where graphical rendering, geometry, software maintenance, and interactivity issues are taken care of by the provider. This step could simplify future development [17]. f) The software could be extended to support the visualization of the training process, including the loss landscape, the gradient flow, and the training data.

Optimization Issues During AM

We found that solving the optimization problem of AM as in Equation (1.1) is a challenging task. Section 5.1 showed that gradient ascent does not find the optimal stimuli reliably for simple functions containing ReLUs or Leaky ReLUs. We attributed these optimization issues to sparse gradients, a phenomenon called the race of patterns, and local maxima. Toy examples illustrated how these issues lead to striking differences in the generated images and the optimal solutions in Section 5.1. These findings question the practical application of AM and the visual interpretation of the generated images: a generated image might be a local maximum that looks very dissimilar from the optimal input stimulus.

Section 5.3 showed that a high negative slope of the Leaky ReLU can mitigate these issues. Our proposed ProxyGrad algorithm reached higher maxima by using different slopes in the forward and backward passes. This strategy increased the maxima found by AM using a ResNet18 pre-trained on ImageNet. The visual quality of the class visualizations also improved: The generated images showed more clarity and structure compared to not using ProdyGrad.

Section 5.4 implies that ProxyGrad can also be used to train the weights of neural networks. Two ReLU networks with ProxyGrad performed similarly to Leaky ReLU networks on three challenging image classification tasks. Notably, ProxyGrad uses a modified gradient of the loss function. Nonetheless, ProxyGrad can be an alternative

to Leaky ReLU networks. ProxyGrad does not always offer superior classification performance but provides some conceptual advantages. It can produce sparse outputs while providing dense gradients for back-propagation. Also, a high negative slope in the backward pass of ProxyGrad does not lead to a linearization of the network, as Leaky ReLUs do.

According to the results in Section 5.4, training the weights of CNNs without using ProxyGrad did not appear to suffer from the optimization issues mentioned above. We speculate that mini-batch optimization introduces noise into the gradient estimation during model training. A common view is that this gradient noise pushes the learning process out of sharp local minima [40]. As a result, the optimization process is more likely to find flat minima, where the gradient noise does not cause an escape. In contrast, AM optimization relies on only a single sample, making the optimization issues potentially more pronounced.

This research comes with some limitations and raises further questions. a) We observed an optimal range for the negative slope in the backward pass of ProxyGrad. An extremely high negative slope can lead to low maxima and unplausible images. Therefore, ProxyGrad users should determine the optimal range by trying out different negative slopes. b) While this work considers CNNs with ReLU and Leaky ReLU activation functions, the discussion does not include other activation functions such as Exponential Linear units (ELUs) [15], SELU [42], Swish [77], or Mish [67]. Also, this work does not study different optimizers for AM. c) In preliminary experiments, we observed that different pooling methods strongly affect the generated patterns during AM. Analyzing these effects could lead to further insights and improvements in AM and pooling techniques. d) Moving all activation functions within ResNet to the convolutional branches could improve gradient flow. With this modification, the skip connections form a linear path from the input to the output layer of the network, potentially simplifying optimization. e) It is known that CNNs require more parameters to consistently converge than they theoretically require for optimally solving a problem. One explanation comes from the lottery ticket hypothesis [21] that quick convergency to a solution relies on lucky random initial weights of subnetworks. Springer and Kenyon [88] further studied the phenomenon by letting CNNs learn the rules of the Game of Life. They found that the number of parameters required to obtain these rules is often higher than the minimal network. Future work could study if ProxyGrad can decrease the number of parameters required for CNNs to converge to a solution.

Interpreting Images Generated by AM Remains a Challenge

This discussion would not be complete without noting the challenges of interpreting the images generated by AM. Much research exists to find image priors for AM (see Section

1.3). Unfortunately, priors can create visual patterns that must not be confused with the network's features. Therefore, the generated images might not always represent the features learned by the network.

Another problem arises when CNNs process features that are unintuitive to us. The work of Yin et al. [101] found that CNNs can exploit high-frequency components that improve classification performance but are not interpretable by humans. Our experiments in Section 4.2 about memorizing train images through spurious patterns support their findings. In our experiments, networks learned high-frequency features that looked unintuitive to us but helped decrease the loss function. Consequently, the feature visualizations showed high-frequency information and looked unintuitive to us.

We hypothesize a potential third problem of interpreting images generated by AM. A prominent group of understandable features is shape. When CNNs process shape, they may require convolution operations on different feature map scales (high and low-resolution feature maps). The complexity of CNNs that recognize simple shapes might already surpass our capacity when inspecting the network weights. The example of shape indicates a gap between simple features that are mathematically accessible and visually understandable. Closing this gap will be crucial for verifying our interpretations of AM-generated images.

Visualizing how CNNs Memorize Training Data

In Section 4.2, we proposed a new training strategy to obtain models with poor generalization abilities. This method involved multiple copies of data with random labels and additive Gaussian noise. Then, AM visualized the high-frequency information used to memorize the train images. We observed that individual CNNs acquired both generic and spurious features. This finding poses a challenge for image recognition, as sub-networks susceptible to spurious input patterns could be triggered unintentionally without the presence of the actual object. Based on the hypothesis that deep neural networks can exploit high-frequency components not perceived by humans [101], models predominantly processing such high-frequency information will not be understandable. The reliance on spurious high-frequency inputs can also lead to fragility, where practically invisible variations in the input can drastically change network decisions. New data augmentation methods, architectures, and training techniques are needed to improve model robustness.

Future work could encourage CNNs to process low-frequency information like shape. While focusing on low-frequency information does not guarantee optimal model performance or robustness [101], we believe it could improve generalization and transparency in some applications. From the perspective of explainable AI, it might make sense to encourage models to use information that we humans find intuitive. Common approaches to bias models toward low-frequency information include Gaussian data

augmentation and adversarial training. Another option is to augment the training set with Fourier-basis perturbations of high frequencies similar to [86]. However, these approaches do not rely on naturally occurring high-frequency distributions. A more suitable approach might be augmenting training data with high-frequency bands from real images. The idea involves combining two images from the train set, x_a and y_b, into a new image, z_a. x_a and z_a have label a and y_b has label b. z_a could be chosen as x_a with an additional high-frequency band from x_b.

$$z = \mathcal{F}^{-1}\left(X(u,v) + H(u,v) \odot Y(u,v)\right) \tag{8.1}$$

Here, $X(u,v)$ and $Y(u,v)$ are the Fourier transforms of the images x_a and y_b, where u and v represent the frequency components. The operation $\odot$ denotes element-wise multiplication. $H(u,v)$ is a binary mask for keeping frequencies above some choosable threshold c.

$$H(u,v) = \begin{cases} 1 & \text{if } \sqrt{u^2+v^2} > c, \\ 0 & \text{if } \sqrt{u^2+v^2} \leq c. \end{cases} \tag{8.2}$$

Another proposal to obtain models that predominantly process low-frequency information is to regularize the architecture. Processing high-frequency patterns in image data requires convolutional kernels with high-frequency components. Therefore, smoothing the convolutional kernels could bias the CNN toward processing low-frequency patterns. Indeed, prior work smoothed the kernels in CNNs to increase robustness [96]. Chapter 6 showed how pre-defined filters can boost generalization in CNNs. Using pre-defined filters with few high-frequency components could bias the model to focus on low-frequency information.

Pre-defined Filters in CNNs Enhance Generalization

We introduced pre-defined filters to CNNs and proposed the Pre-defined Filter CNNs (PFCNNs) in Section 6.2. Processing edge and line features within ResNets and DenseNets improved generalization (see Section 6.3). In the case of ResNet18, we observed a noteworthy increase in test accuracy ranging from $5-11$ percentage points across four fine-grained classification datasets with the same number of trainable parameters. This improvement came from making the CNNs process edge and line features. The learning process found linear combinations of the pre-defined filter outputs without changing the pre-defined filters. The superior recognition performance of our models shows that adjusting the filters in CNNs to the training data is unnecessary for many image recognition problems. These findings can guide the design and optimization of CNN architectures for various computer vision tasks.

Removing the residual connections from ResNet-based PFCNNs can worsen their performance considerably (see Sections 6.6 and 7.3.4). We observed test performance drops of up to 42 percentage points on the Flowers dataset for PFCNNs without

residual connections. We attributed the drop to the inability of PFMs to learn the identity under our choice of pre-defined filters. The identity is crucial for reusing features and skipping additional layers. The performance drop is small in traditional CNNs, suggesting that they adapt to missing skip connections or aliasing at the cost of additional resources. A question for future research is which pre-defined filters allow or do not allow learning the identity. We applied pre-defined filters in ResNets and DenseNets, where the identity is a central part of the architecture. We also found that aliasing in the skip connections can also harm performance. When working with PFCNNs, we strongly recommend applying Gaussian smoothing to the particular skip connections where the feature map resolution is reduced through subsampling.

The Choice of Pre-defined Filters Matters

We assessed the performance of different types of pre-defined filters in Section 6.3. Edge and line filters achieved top performance across all datasets. Allowing the pre-defined filters to adapt to minimize the training loss did not lead to further performance improvements. This observation suggests that our edge and line filters capture generic features suiting many image recognition tasks.

Random pre-defined filters performed similar or better than the baseline model ResNet18, underscoring the usefulness of random filters (see Section 6.3). Nonetheless, the test accuracies were below those of edge and line filters. The results were also confirmed for the reduced PFCNNs in Section 7.3.4. These findings align with the literature discussing the potential of random filters [23, 78].

We also studied how the size of the set of pre-defined filters affects recognition performance (see Section 6.5). The number of pre-defined filters in the set mattered with optimal results obtained with nine or more pre-defined filters. Doubling the number of filters resulted in further performance improvements of $1-2$ percentage points across all datasets. However, doubling the filters also doubled the computation time, limiting the attractiveness for platforms with sensitive energy and speed requirements.

The number of dimensions spanned by the set of pre-defined filters appeared to have a low impact on recognition performance. Four dimensions achieved comparable recognition values to those obtained with nine dimensions. This observation was attributed to the nature of the ReLU activation function as explained in Section 6.4. When applied to the outputs of convolution operations with linearly dependent filter kernels, ReLU can produce linearly independent results. Therefore, we recommend using pairs of pre-defined filters with inverted signs in PFCNNs.

Parameter Reduction Using Pre-Defined Filter Networks

We simplified CNNs further by reducing their parameters in Chapter 7. We introduced the reduced PFM with only 13% of the weights of ResNet18, outperforming ResNet18 on the Caltech101, FGVC-Aircraft, and Flowers datasets by margins of up to 8 percentage points (see Section 7.3). On the CIFAR10, CUB, and Stanford Cars datasets, PFNet18 performed similar or slightly worse than ResNet18 by up to 5 percentage points on CUB.

Activation maximization unveiled that the reduced network yielded complex, object-specific features from only 20 training images per class. On the Caltech101 dataset, it processed the shape of recognized objects, while ResNet18 seemed to prioritize textures (see Section 7.3).

Ideas for Future Research on PFCNNs

PFCNNs come with open questions and new ideas for future research. a) Determining the optimal set of pre-defined filters for specific image recognition tasks remains challenging. Adjusting the filters to the training data did not yield better test performance than using simple edge and line filters. Thus, it is an open question of how to find generic filters that provide better performance. b) Applying pre-defined filters to diverse tasks beyond image recognition, such as sound or video analysis, is left for future research. Specialized features may offer significant benefits in these domains. c) Investigating the compatibility of our method with architectures beyond ResNet and DenseNet requires more experiments. d) Future research should also explore transfer learning with PFCNNs. e) We showed that PFCNNs can save trainable parameters. From a compression perspective, it would be interesting to compare our method against existing compression or pruning techniques in terms of compression rate and accuracy. f) We found that increasing the number of pre-defined filters improved recognition performance at the cost of computation time. Future research could investigate if further increasing the number of filters to 32 or 64 would lead to better performance.

Chapter 9

Conclusion

We demonstrated that current visualization techniques typically focus on specific model aspects and are insufficient for a comprehensive understanding of CNNs. We motivated a holistic approach that captures multiple facets of network behavior, from learned features to the data flow through the network. Our software, DeepVisionVR, visualizes large, popular CNNs in 3D space, providing a first step toward this goal. The tool combines visualization tools and interaction possibilities, offering a new way to explore trained CNN models. We hope our approach will boost the general understanding of CNNs for newcomers and experienced members of the deep learning community.

We showed that feature visualization through activation maximization suffers from optimization issues for neural networks with rectifiers. Leaky ReLUs with different negative slopes in the forward and backward pass lead to higher maxima and clear visualizations. We named our method ProxyGrad because it utilizes a proxy function for gradient computation. ProxyGrad also offers a promising alternative for training models: It can substitute traditional Leaky ReLU networks in image recognition, providing similar performance while featuring conceptual advantages in gradient computation.

We visualized how CNNs simultaneously learned generic and spurious features. We systematically trained models sensitive to memorized high-frequency patterns, leading to fragile decisions. This observation motivates new training techniques and architectures that prioritize robustness. We gave ideas to encourage CNNs to focus on understandable low-frequency information, such as shape, which might enhance model generalization and transparency.

This research systematically simplified CNNs through the introduction of understandable, pre-defined filters. With nine simple edge and line detectors, our Pre-defined Filter CNN (PFCNN) boosted the test accuracy of ResNet18 by $5 - 11$ percentage points across four classification benchmark datasets with the same number of trainable parameters. The generalization improvements came from restricting the learning process to finding linear combinations of pre-defined filter outputs. The edge and line detectors added a beneficial bias to the models, demonstrating that training the filter kernels of CNNs is unnecessary for many image recognition problems.

A slight reconfiguration of PFCNNs significantly reduced the number of trainable parameters without sacrificing performance on small datasets. Our Models outper-

formed ResNet18 with only 13% of its weights on the Caltech101, FGVC-Aircraft, and Flowers102 datasets by margins of up to 8 percentage points. On the CIFAR10, CUB, and Stanford Cars datasets, our models performed similar or slightly worse than ResNet18 by up to 5 percentage points on CUB. Activation maximization indicated the emergence of complex, discriminative features despite the small number of parameters. Once again, it becomes clear that many weights in conventional CNNs are redundant.

This research gave new ideas for visualizing and designing CNNs to increase transparency. Collectively, these contributions advance our understanding of CNNs by unveiling their internal mechanisms.

Bibliography

[1] A. Aamir, M. Tamosiunaite, and F. Wörgötter. "Caffe2Unity: Immersive Visualization and Interpretation of Deep Neural Networks". In: *Electronics* 11.1 (Dec. 2021), p. 83.

[2] A. Adadi and M. Berrada. "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)". In: *IEEE Access* 6 (2018), pp. 52138–52160.

[3] T. Ai, Z. Yang, H. Hou, C. Zhan, C. Chen, W. Lv, Q. Tao, Z. Sun, and L. Xia. "Correlation of Chest CT and RT-PCR Testing for Coronavirus Disease 2019 (COVID-19) in China: A Report of 1014 Cases". In: *Radiology* 296.2 (Aug. 2020), E32–E40.

[4] H. Alshazly, C. Linse, M. Abdalla, E. Barth, and T. Martinetz. "COVID-Nets: deep CNN architectures for detecting COVID-19 using chest CT scans". In: *PeerJ Computer Science* 7 (July 2021), e655.

[5] H. Alshazly, C. Linse, E. Barth, S. A. Idris, and T. Martinetz. "Towards Explainable Ear Recognition Systems Using Deep Residual Networks". In: *IEEE Access* 9 (2021), pp. 122254–122273.

[6] H. Alshazly, C. Linse, E. Barth, and T. Martinetz. "Deep Convolutional Neural Networks for Unconstrained Ear Recognition". In: *IEEE Access* 8 (2020), pp. 170295–170310.

[7] H. Alshazly, C. Linse, E. Barth, and T. Martinetz. "Ensembles of Deep Learning Models and Transfer Learning for Ear Recognition". In: *Sensors* 19.19 (Jan. 2019), p. 4139.

[8] H. Alshazly, C. Linse, E. Barth, and T. Martinetz. "Explainable COVID-19 Detection Using Chest CT Scans and Deep Learning". In: *Sensors* 21.2 (Jan. 2021), p. 455.

[9] H. Alshazly, C. Linse, E. Barth, and T. Martinetz. "Handcrafted versus CNN Features for Ear Recognition". In: *Symmetry* 11.12 (Dec. 2019), p. 1493.

[10] A. B. Arrieta, N. Diaz-Rodriguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information Fusion* 58 (2020), pp. 82–115.

[11] M. Belkin, D. Hsu, S. Ma, and S. Mandal. "Reconciling modern machine-learning practice and the classical bias–variance trade-off". In: *Proceedings of the National Academy of Sciences* 116.32 (2019), pp. 15849–15854.

[12] M. Bock and A. Schreiber. "Visualization of neural networks in virtual reality using Unreal Engine". In: *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*. Tokyo Japan: ACM, Nov. 2018, pp. 1–2.

[13] M. Brady. "Computational Approaches to Image Understanding". In: *ACM Computing Surveys* 14.1 (Mar. 1982), pp. 3–71.

[14] J. Choo and S. Liu. "Visual analytics for explainable deep learning". In: *IEEE Computer Graphics and Applications* 38.4 (2018), pp. 84–92.

[15] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: *arXiv:1511.07289 [cs]* (Feb. 2016).

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. Miami, FL: IEEE, June 2009, pp. 248–255.

[17] C. Donalek, S. G. Djorgovski, A. Cioc, A. Wang, J. Zhang, E. Lawler, S. Yeh, A. Mahabal, M. Graham, A. Drake, S. Davidoff, J. S. Norris, and G. Longo. "Immersive and collaborative data visualization using virtual reality platforms". In: *2014 IEEE International Conference on Big Data (Big Data)*. Washington, DC, USA: IEEE, Oct. 2014, pp. 609–614.

[18] S. C. Douglas and J. Yu. "Why RELU Units Sometimes Die: Analysis of Single-Unit Error Backpropagation in Neural Networks". In: *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. Pacific Grove, CA, USA: IEEE, Oct. 2018, pp. 864–868.

[19] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. "Visualizing higher-layer features of a deep network". In: *University of Montreal* 1341.3 (2009), pp. 1–13.

[20] Y. Fang, H. Zhang, J. Xie, M. Lin, L. Ying, P. Pang, and W. Ji. "Sensitivity of Chest CT for COVID-19: Comparison to RT-PCR". In: *Radiology* 296.2 (Aug. 2020), E115–E117.

[21] J. Frankle and M. Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *arXiv:1803.03635 [cs]* (Mar. 4, 2019).

[22] P. Gavrikov and J. Keuper. "CNN Filter DB: An Empirical Investigation of Trained Convolutional Filters". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. New Orleans, LA, USA: IEEE, June 2022, pp. 19044–19054.

[23] P. Gavrikov and J. Keuper. “Rethinking 1x1 Convolutions: Can we train CNNs with Frozen Random Filters?” In: *arXiv:2301.11360 [cs]* (Jan. 2023).

[24] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel. “ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness”. In: *arXiv:1811.12231 [cs]* (2018).

[25] G. H. Granlund. “In search of a general picture processing operator”. In: *Computer Graphics and Image Processing* 8.2 (Oct. 1978), pp. 155–173.

[26] F. Grün, C. Rupprecht, N. Navab, and F. Tombari. “A taxonomy and library for visualizing learned features in convolutional neural networks”. In: *arXiv:1606.07757* (2016).

[27] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G.-Z. Yang. “XAI—Explainable artificial intelligence”. In: *Science Robotics* 4.37 (Dec. 2019), eaay7120.

[28] J. K. Haas. “A history of the unity game engine”. In: Worcester Polytechnic Institute, 2014.

[29] C. Hani, N. Trieu, I. Saab, S. Dangeard, S. Bennani, G. Chassagnon, and M.-P. Revel. “COVID-19 pneumonia: A review of typical CT findings and differential diagnosis”. In: *Diagnostic and Interventional Imaging* 101.5 (May 2020), pp. 263–268.

[30] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778.

[31] K. He, X. Zhang, S. Ren, and J. Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, Chile: IEEE, Dec. 2015, pp. 1026–1034.

[32] X. He, X. Yang, S. Zhang, J. Zhao, Y. Zhang, E. Xing, and P. Xie. “Sample-Efficient Deep Learning for COVID-19 Diagnosis Based on CT Scans”. In: *medRxiv* (Apr. 17, 2020).

[33] L. Hertel, E. Barth, T. Käster, and T. Martinetz. “Deep Convolutional Neural Networks as Generic Feature Extractors”. In: *arXiv:1710.02286 [cs]* (Oct. 2017).

[34] A. Hisham and B. Mahmood. “VR-Based Visualization for Biological Networks”. In: *International Research Journal of Innovations in Engineering and Technology* 07.05 (2023), pp. 48–68.

[35] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv:1704.04861* (2017).

[36] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. "Densely connected convolutional networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4700–4708.

[37] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry. "Adversarial Examples Are Not Bugs, They Are Features". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019.

[38] Ioffe, Sergey and Szegedy, Christian. "Batch normalization: accelerating deep network training by reducing internal covariate shift". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, 2015, pp. 448–456.

[39] J. P. Kanne. "Chest CT Findings in 2019 Novel Coronavirus (2019-nCoV) Infections from Wuhan, China: Key Points for the Radiologist". In: *Radiology* 295.1 (Apr. 2020), pp. 16–17.

[40] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: *International Conference on Learning Representations*. 2017.

[41] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs]* (Jan. 2017).

[42] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. "Self-normalizing neural networks". In: *Advances in neural information processing systems* 30 (2017).

[43] E. H. Korkut and E. Surer. "Visualization in virtual reality: a systematic review". In: *Virtual Reality* 27.2 (June 2023), pp. 1447–1480.

[44] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. "3D Object Representations for Fine-Grained Categorization". In: *2013 IEEE International Conference on Computer Vision Workshops*. Sydney, Australia: IEEE, Dec. 2013, pp. 554–561.

[45] A. Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: *University of Toronto* (May 2012).

[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012.

[47] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[48] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. “Visualizing the Loss Landscape of Neural Nets”. In: *Proceedings of the 32nd Conference on Neural Information Processing System*. 2018, pp. 1–11.

[49] Li Fei-Fei, R. Fergus, and P. Perona. “Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories”. In: *2004 Conference on Computer Vision and Pattern Recognition Workshop*. Washington, DC, USA: IEEE, 2004, pp. 178–178.

[50] C. Linse, H. Alshazly, and T. Martinetz. “A walk in the black-box: 3D visualization of large neural networks in virtual reality”. In: *Neural Computing and Applications* (Aug. 2022).

[51] C. Linse, E. Barth, and T. Martinetz. “Convolutional Neural Networks Do Work with Pre-Defined Filters”. In: *2023 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2023, pp. 1–8.

[52] C. Linse, E. Barth, and T. Martinetz. “Leaky ReLUs That Differ in Forward and Backward Pass Facilitate Activation Maximization in Deep Neural Networks”. In: *2024 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2024.

[53] C. Linse, B. Brückner, and T. Martinetz. “Enhancing Generalization in Convolutional Neural Networks through Regularization with Edge and Line Features”. In: *ICANN 2024, Lecture Notes in Computer Science*. Springer, 2024.

[54] C. Linse and T. Martinetz. “Large Neural Networks Learning from Scratch with Very Few Data and without Explicit Regularization”. In: *Proceedings of the 2023 15th International Conference on Machine Learning and Computing*. Zhuhai China: ACM, Feb. 2023, pp. 279–283.

[55] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. “Towards better analysis of deep convolutional neural networks”. In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2016), pp. 91–100.

[56] S. Liu, D. Papailiopoulos, and D. Achlioptas. “Bad global minima exist and SGD can reach them”. In: *Proceedings of the 34th Conference on Neural Information Processing System*. 2020.

[57] L. Lu. “Dying ReLU and Initialization: Theory and Numerical Examples”. In: *Communications in Computational Physics* 28.5 (June 2020), pp. 1671–1706.

[58] Y. Ma, Y. Luo, and Z. Yang. “PCFNet: Deep neural network with predefined convolutional filters”. In: *Neurocomputing* 382 (Mar. 2020), pp. 32–39.

[59] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. “Rectifier nonlinearities improve neural network acoustic models”. In: *Proc. icml*. Vol. 30. 1. Atlanta, GA. 2013, p. 3.

[60] A. Mahendran and A. Vedaldi. "Understanding deep image representations by inverting them". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, June 2015, pp. 5188–5196.

[61] A. Mahendran and A. Vedaldi. "Visualizing deep convolutional neural networks using natural pre-images". In: *International Journal of Computer Vision* 120.3 (2016), pp. 233–255.

[62] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi. "Fine-Grained Visual Classification of Aircraft". In: *arXiv:1306.5151 [cs]* (June 2013).

[63] J. Martinetz, C. Linse, and T. Martinetz. "Rethinking generalization of classifiers in separable classes scenarios and over-parameterized regimes". In: *2024 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2024.

[64] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher, and D. J. Schwab. "A high-bias, low-variance introduction to Machine Learning for physicists". In: *Physics Reports* 810 (May 2019), pp. 1–124.

[65] N. Meissler, A. Wohlan, N. Hochgeschwender, and A. Schreiber. "Using Visualization of Convolutional Neural Networks in Virtual Reality for Machine Learning Newcomers". In: *2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*. San Diego, CA, USA: IEEE, Dec. 2019, pp. 152–1526.

[66] C. Meske, E. Bunde, J. Schneider, and M. Gersch. "Explainable Artificial Intelligence: Objectives, Stakeholders, and Future Research Opportunities". In: *Information Systems Management* 39.1 (Jan. 2022), pp. 53–63.

[67] D. Misra. "Mish: A Self Regularized Non-Monotonic Activation Function". In: *arXiv:1908.08681 [cs, stat]* (Aug. 2020).

[68] A. Mordvintsev, C. Olah, and M. Tyka. "Inceptionism: Going deeper into neural networks". In: *Google research blog* 20.14 (2015), p. 5.

[69] V. Nair and G. E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Madison, WI, USA: Omnipress, 2010, pp. 807–814.

[70] A. Nguyen, J. Yosinski, and J. Clune. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, June 2015, pp. 427–436.

[71] A. Nguyen, J. Yosinski, and J. Clune. "Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks". In: *arXiv:1602.03616 [cs]* (2016).

[72] A. Nguyen, J. Yosinski, and J. Clune. “Understanding Neural Networks via Feature Visualization: A Survey”. In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Ed. by W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, and K.-R. Müller. Vol. 11700. Cham: Springer International Publishing, 2019, pp. 55–76.

[73] M.-E. Nilsback and A. Zisserman. “Automated Flower Classification over a Large Number of Classes”. In: *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*. Bhubaneswar, India: IEEE, Dec. 2008, pp. 722–729.

[74] A. Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035.

[75] S. Pirch, F. Müller, E. Iofinova, J. Pazmandi, C. V. R. Hütter, M. Chiettini, C. Sin, K. Boztug, I. Podkosova, H. Kaufmann, and J. Menche. “The VRNetzer platform enables interactive network analysis in Virtual Reality”. In: *Nature Communications* 12.1 (Apr. 2021), p. 2432.

[76] D. Queck, A. Wohlan, and A. Schreiber. “Neural Network Visualization in Virtual Reality: A Use Case Analysis and Implementation”. In: *Human Interface and the Management of Information: Visual and Information Design*. Ed. by S. Yamamoto and H. Mori. Vol. 13305. Cham: Springer International Publishing, 2022, pp. 384–397.

[77] P. Ramachandran, B. Zoph, and Q. V. Le. “Searching for Activation Functions”. In: *arXiv:1710.05941 [cs]* (Oct. 2017).

[78] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari. “What’s hidden in a randomly weighted neural network?” In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11893–11902.

[79] G. Ras, M. van Gerven, and P. Haselager. “Explanation methods in deep learning: Users, values, concerns and challenges”. In: *Explainable and interpretable models in computer vision and machine learning* (2018), pp. 19–36.

[80] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115.3 (Dec. 2015), pp. 211–252.

[81] A. Schreiber and M. Bock. “Visualization and Exploration of Deep Learning Networks in 3D and Virtual Reality”. In: *HCI International 2019 - Posters*. Ed. by C. Stephanidis. Vol. 1033. Cham: Springer International Publishing, 2019, pp. 206–211.

[82] Selvaraju, Ramprasaath R. and Cogswell, Michael and Das, Abhishek and Vedantam, Ramakrishna and Parikh, Devi and Batra, Dhruv. “Grad-CAM: Visual Explanations From Deep Networks via Gradient-Based Localization”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[83] A. Shahroudnejad. “A Survey on Understanding, Visualizations, and Explanation of Deep Neural Networks”. In: *arXiv:2102.01792 [cs]* (2021).

[84] K. Simonyan and A. Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *Proceedings of the International Conference on Learning Representations*. 2015, pp. 1–14.

[85] E. Soares, P. Angelov, S. Biaso, M. H. Froes, and D. K. Abe. “SARS-CoV-2 CT-scan dataset: A large dataset of real patients CT scans for SARS-CoV-2 identification”. In: *medRxiv* (2020).

[86] R. Soklaski, M. Yee, and T. Tsiligkaridis. “Fourier-Based Augmentations for Improved Robustness and Uncertainty Calibration”. In: *NeurIPS 2021 Workshop on Distribution Shifts: Connecting Methods and Applications*. 2021.

[87] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. “Striving for Simplicity: The All Convolutional Net”. In: *ICLR (workshop track)*. 2015.

[88] J. M. Springer and G. T. Kenyon. “It’s Hard for Neural Networks to Learn the Game of Life”. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. 2021 International Joint Conference on Neural Networks (IJCNN). Shenzhen, China: IEEE, July 18, 2021, pp. 1–8.

[89] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. “Going deeper with convolutions”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, June 2015, pp. 1–9.

[90] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. “Intriguing properties of neural networks”. In: *arXiv:1312.6199 [cs]* (Feb. 19, 2014).

[91] Szegedy, Christian and Vanhoucke, Vincent and Ioffe, Sergey and Shlens, Jon and Wojna, Zbigniew. “Rethinking the Inception Architecture for Computer Vision”. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016, pp. 2818–2826.

[92] E. Tjoa and C. Guan. “A survey on explainable artificial intelligence (xai): Toward medical xai”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.11 (2020), pp. 4793–4813.

[93] Van der Maaten, Laurens and Hinton, Geoffrey. “Visualizing data using t-SNE.” In: *Journal of machine learning research* 9.11 (2008).

[94] K. C. VanHorn, M. Zinn, and M. C. Cobanoglu. “Deep Learning Development Environment in Virtual Reality”. In: *arXiv:1906.05925 [cs, stat]* (June 2019).

[95] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. *The Caltech-UCSD Birds-200-2011 Dataset*. Tech. rep. CNS-TR-2011-001. California Institute of Technology, 2011.

[96] H. Wang, X. Wu, Z. Huang, and E. P. Xing. “High-Frequency Component Helps Explain the Generalization of Convolutional Neural Networks”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle, WA, USA: IEEE, June 2020, pp. 8681–8691.

[97] Z. J. Wang, R. Turko, O. Shaikh, H. Park, N. Das, F. Hohman, M. Kahng, and D. H. Polo Chau. “CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (Feb. 2021), pp. 1396–1406.

[98] T. Whitaker and D. Whitley. “Synaptic Stripping: How Pruning Can Bring Dead Neurons Back to Life”. In: *2023 International Joint Conference on Neural Networks (IJCNN)*. Gold Coast, Australia: IEEE, June 2023, pp. 1–8.

[99] P. Wimmer, J. Mehnert, and A. Condurache. “Interspace Pruning: Using Adaptive Filter Representations to Improve Training of Sparse CNNs”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 12527–12537.

[100] Z. Ye, Y. Zhang, Y. Wang, Z. Huang, and B. Song. “Chest CT manifestations of new coronavirus disease 2019 (COVID-19): a pictorial review”. In: *European Radiology* 30.8 (Aug. 2020), pp. 4381–4389.

[101] D. Yin, R. Gontijo Lopes, J. Shlens, E. D. Cubuk, and J. Gilmer. “A Fourier Perspective on Model Robustness in Computer Vision”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019.

[102] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. “Understanding Neural Networks Through Deep Visualization”. In: *arXiv:1506.06579 [cs]* (June 22, 2015).

[103] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, and C.-J. Hsieh. "Large batch optimization for deep learning: Training BERT in 76 minutes". In: *International Conference on Learning Representations*. 2020.

[104] M. D. Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Networks". In: *Computer Vision – ECCV 2014*. Ed. by D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars. Vol. 8689. Cham: Springer International Publishing, 2014, pp. 818–833.

Appendix A

Appendix

A.1 Implementation of the ProxyGrad Algorithm

The following code implements the forward and backward pass of the ProxyGrad algorithm for using ReLUs in the forward pass and Leaky ReLUs in the backward pass.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class ProxyGradRELUFN(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, negative_slope):
        ctx.x = x
        ctx.negative_slope = negative_slope
        return F.relu(x)

    @staticmethod
    def backward(ctx, grad_output):
        mask = ctx.x < 0.
        grad_output[mask] *= ctx.negative_slope
        return grad_output, None

class ProxyGradRELU(nn.Module):
    def __init__(self, negative_slope):
        super().__init__()
        self.negative_slope = negative_slope

    def forward(self, x):
        return ProxyGradRELUFN.apply(
            x, self.negative_slope)
```

A.2 Leaky ReLUs Applied on a Gaussian Distribution

This section computes the mean and the variance of the output distribution of a Leaky ReLU applied on the mean-free Gaussian variable X with variance 1.

$$
\begin{aligned}
E(X^{\mathrm{LReLU}}) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{0} sxe^{-\frac{x^2}{2}} dx + \frac{1}{\sqrt{2\pi}} \int_{0}^{\infty} xe^{-\frac{x^2}{2}} dx \\
&= -\frac{s}{\sqrt{2\pi}} + \frac{1}{\sqrt{2\pi}} = \frac{1-s}{\sqrt{2\pi}} \\
V(X^{\mathrm{LReLU}}) &= E(X^{\mathrm{LReLU}^2}) - E(X^{\mathrm{LReLU}})^2 \\
&= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{0} s^2x^2e^{-\frac{x^2}{2}} dx + \frac{1}{\sqrt{2\pi}} \int_{0}^{\infty} x^2e^{-\frac{x^2}{2}} dx \\
&\quad - E(X^{\mathrm{LReLU}})^2 \\
&= \frac{s^2}{2} + 1 - \frac{1}{2} - \frac{(1-s)^2}{2\pi} = \frac{s^2+1}{2} - \frac{(1-s)^2}{2\pi}
\end{aligned}
$$

The variance increases with increasing negative slope s. We used the following integrals where $\Phi(x)$ denotes the CDF of the Gaussian distribution:

$$
\int_a^b xe^{-\frac{x^2}{2}} dx = e^{-\frac{a^2}{2}} - e^{-\frac{b^2}{2}}
$$

$$
\int_a^b x^2e^{-\frac{x^2}{2}} dx = \sqrt{2\pi}(\Phi(b) - \Phi(a)) + (ae^{-\frac{a^2}{2}} - be^{-\frac{a^2}{2}})
$$

Furthermore, in ResNet18, we measure the standard deviation of the input to different batch normalization layers. We randomly initialize the network and select a random batch from the Caltech101 dataset as input. The experiment is repeated with different seeds to get statistically robust results. As shown in Figure A.1 the standard deviation increases for higher negative slopes, which supports our hypothesis.

A.3 Linear Independency of ReLU-based Functions

Two functions $f_1, f_2 : X \to Y$ are linearly independent if

$$
(\forall \mathbf{x} \in X : c_1 f_1(\mathbf{x}) + c_2 f_2(\mathbf{x}) = 0) \Leftrightarrow c_1 = c_2 = 0. \tag{A.1}
$$

Consider the functions from (6.3) that occur in the PFM module. The functions are linearly dependent if the pre-defined filters $\tilde{w}_1$ and $\tilde{w}_2$ are linearly dependent and $a\tilde{w}_1 = \tilde{w}_2, a \geq 0$.

Proof. Choose some arbitrary $\mathbf{x} \in \mathbb{R}^{M \times N}$. Choose $c_1 \in \mathbb{R}\backslash\{0\}$ and $c_2 = -c_1/a$. Then,

$$
\begin{aligned}
&c_1 f^{(\tilde{w}_1,m,n)}(\mathbf{x}) + c_2 f^{(\tilde{w}_2,m,n)}(\mathbf{x}) \\
&= c_1 \mathrm{ReLU}(\tilde{w}_1 * \mathbf{x})[m,n] - c_1 \frac{a}{a} \mathrm{ReLU}(\tilde{w}_1 * \mathbf{x})[m,n] = 0.
\end{aligned} \tag{A.2}
$$

□

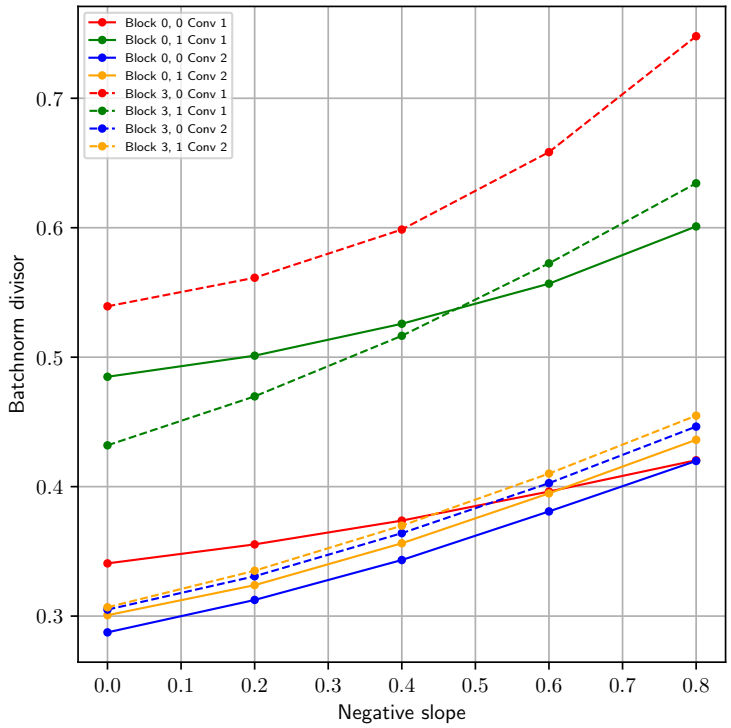


Figure A.1: Standard deviation of the input to batch normalization layers taken from different locations in ResNet with Leaky ReLUs.

The functions in (6.3) are linearly independent if $\tilde{w}_1$ and $\tilde{w}_2$ are linearly dependent and $a\tilde{w}_1 = \tilde{w}_2, a < 0$.

Proof. The $\leftarrow$ direction is clear. To show the $\rightarrow$ direction, let $\mathbf{x} \in \mathbb{R}^{M \times N}$.

$$\begin{aligned}
&c_1 f^{(\tilde{w}_1,m,n)}(\mathbf{x}) + c_2 f^{(\tilde{w}_2,m,n)}(\mathbf{x}) = 0 \\
&\Leftrightarrow c_1 \mathrm{ReLU}(\tilde{w}_1 * \mathbf{x})[m,n] + c_2 \mathrm{ReLU}(a\tilde{w}_1 * \mathbf{x})[m,n] = 0 \\
&\Leftrightarrow c_1 \mathrm{ReLU}(\tilde{w}_1 * \mathbf{x})[m,n] - ac_2 \mathrm{ReLU}(-\tilde{w}_1 * \mathbf{x})[m,n] = 0 \\
&\text{Case1} : (\tilde{w}_1 * \mathbf{x})[m,n] \geq 0 \implies c_1 = 0 \\
&\text{Case2} : (\tilde{w}_1 * \mathbf{x})[m,n] < 0 \implies c_2 = 0
\end{aligned} \tag{A.3}$$

The sum has to be zero for all $\mathbf{x} \in \mathbb{R}^{M \times N}$. This means that both coefficients c_1 and c_2 have to be zero. □

References

[1] C. Linse, E. Barth, and T. Martinetz. "Leaky ReLUs That Differ in Forward and Backward Pass Facilitate Activation Maximization in Deep Neural Networks". In: *2024 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2024.

[2] C. Linse, B. Brückner, and T. Martinetz. "Enhancing Generalization in Convolutional Neural Networks through Regularization with Edge and Line Features". In: *ICANN 2024, Lecture Notes in Computer Science*. Springer, 2024.

[3] J. Martinetz, C. Linse, and T. Martinetz. "Rethinking generalization of classifiers in separable classes scenarios and over-parameterized regimes". In: *2024 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2024.

[4] C. Linse, E. Barth, and T. Martinetz. "Convolutional Neural Networks Do Work with Pre-Defined Filters". In: *2023 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2023, pp. 1–8.

[5] C. Linse and T. Martinetz. "Large Neural Networks Learning from Scratch with Very Few Data and without Explicit Regularization". en. In: *Proceedings of the 2023 15th International Conference on Machine Learning and Computing*. Zhuhai China: ACM, Feb. 2023, pp. 279–283.

[6] C. Linse, H. Alshazly, and T. Martinetz. "A walk in the black-box: 3D visualization of large neural networks in virtual reality". en. In: *Neural Computing and Applications* (Aug. 2022).

[7] H. Alshazly, C. Linse, M. Abdalla, E. Barth, and T. Martinetz. "COVID-Nets: deep CNN architectures for detecting COVID-19 using chest CT scans". en. In: *PeerJ Computer Science* 7 (July 2021), e655.

[8] H. Alshazly, C. Linse, E. Barth, S. A. Idris, and T. Martinetz. "Towards Explainable Ear Recognition Systems Using Deep Residual Networks". In: *IEEE Access* 9 (2021), pp. 122254–122273.

[9] H. Alshazly, C. Linse, E. Barth, and T. Martinetz. "Explainable COVID-19 Detection Using Chest CT Scans and Deep Learning". en. In: *Sensors* 21.2 (Jan. 2021), p. 455.

[10] H. Alshazly, C. Linse, E. Barth, and T. Martinetz. "Deep Convolutional Neural Networks for Unconstrained Ear Recognition". In: *IEEE Access* 8 (2020), pp. 170295–170310.

[11] H. Alshazly, C. Linse, E. Barth, and T. Martinetz. "Ensembles of Deep Learning Models and Transfer Learning for Ear Recognition". en. In: *Sensors* 19.19 (Jan. 2019), p. 4139.

[12] H. Alshazly, C. Linse, E. Barth, and T. Martinetz. "Handcrafted versus CNN Features for Ear Recognition". en. In: *Symmetry* 11.12 (Dec. 2019), p. 1493.