TECHNISCHE UNIVERSITÄT
ILMENAU

Fakultät für Informatik und Automatisierung
Fachgebiet Neuroinformatik und Kognitive Robotik

Transfer Learning für die Erkennung von Straßenschäden

Masterarbeit zur Erlangung des akademischen Grades Master of Science

Christoph Balada

Betreuer:	Dipl.-Inf. Markus Eisenbach
Verantwortlicher Hochschullehrer:	Univ.-Prof. Dr. Horst-Michael Groß
	FG Neuroinformatik und Kognitive Robotik

Die Masterarbeit wurde am 29. Juni 2018 bei der Fakultät für Informatik und Automatisierung der Technischen Universität Ilmenau eingereicht.

Abstract

Eine der größten Herausforderungen bei dem Training tiefer Neuronaler Netze ist die Bereitstellung einer ausreichenden Datenmenge. Zu wenige oder qualitativ zu schlechte Daten führen automatisch zu ungenügenden Ergebnissen. Um diesem Problem zu begegnen, kann Transfer Learning eingesetzt werden. Diese Arbeit fokussiert sich darauf, die Eignung von Transfer Learning für die Detektion von Straßenschäden zu untersuchen. Die Ergebnisse zeigen, dass der Nutzen von Transfer Learning abhängig von der verwendeten Architektur ist, jedoch bei keiner ein negativer Einfluss festgestellt werden konnte. Im Vergleich zu einem Training mit zufällig initialisierten Gewichten konnte die Leistung auf Test-Daten bis zu 5% im F1-Score gesteigert werden. Weitere Untersuchungen der Architekturen haben gezeigt, dass der Effekt einer Feature-Selektion, welcher in anderen Arbeiten beobachtet wurde, nicht reproduziert werden konnte. Zu diesem Zweck wurden mehrere Experimente vor und nach dem Finetuning auf Basis der Gewichte und der Aktivierungen der Netze durchgeführt.

Danksagungen

An erster Stelle möchte ich meinem Betreuer Markus Eisenbach für die hervorragende Unterstützung und viel hilfreiches Feedback danken. Des Weiteren geht mein Dank an Prof. Dr.-Ing. Horst-Michael Groß für die Ermöglichung und Begutachtung meiner Arbeit.

Darüber hinaus möchte ich mich bei allen bedanken, die mich bei der Erstellung meiner Masterarbeit auf verschiedenem Wege unterstützt haben.

Abschließend geht mein Dank insbesondere an Celina von Eiff und Linda Frömmel, die mich durch ihre Hilfe beim Korrekturlesen großartig unterstützt haben.

Selbstständigkeitserklärung: ”Hiermit versichere ich, dass ich diese Masterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe. Alle von mir aus anderen Veröffentlichungen übernommenen Passagen sind als solche gekennzeichnet.“

Ilmenau, 29. Juni 2018

Christoph Balada

Inhaltsverzeichnis

Einleitung 1

Deep Learning Techniken konnten in den letzten Jahren bemerkenswerte Leistungssteigerungen in den verschiedensten Bereichen erzielen. Für den großen Erfolg ist maßgeblich die Verfügbarkeit großer Datenmengen, wie zum Beispiel durch das ImageNet [Deng et al., 2009] Projekt, verantwortlich. Neben der stetigen Entwicklung neuer Netzarchitekturen und deren Optimierung bilden die verwendeten Daten nach wie vor die Basis für den Erfolg jedes Deep Learning Projekts. Insbesondere die Menge der verfügbaren Daten ist dabei entscheidend, denn erst durch die Verwendung enormer Datenmengen ist es möglich, die State-of-the-Art-Ergebnisse zu erreichen.

1.1 Motivation

Plattformen wie zum Beispiel Google oder Facebook generieren jeden Tag neue Daten, welche für ein Training von tiefen Neuronalen Netzen (NN) verwendet werden können. Die so gesammelten Daten werden genutzt, um die Leistung der Netze stetig zu verbessern. Abseits solcher großen Unternehmen bzw. Organisationen ist die Datengewinnung jedoch nach wie vor ein kritischer Punkt bei der Umsetzung intelligenter Deep Learning Systeme. Um diesem Problem zu begegnen, wurden verschiedene Initiativen, wie z.B. die ImageNet [Deng et al., 2009] Datenbank, gegründet. Ziel des ImageNets war es, Bildmaterial zu den verschiedensten allgemeinen Bereichen zu sammeln und über ein Crowd Worker Konzept labeln zu lassen.
Die ImageNet Datenbank umfasst momentan circa 14 Millionen Bildbeispiele, welche in 21841 sogenannte Synsets unterteilt sind. Darüber hinaus konnte sich ImageNet

[Deng et al., 2009] als ein Benchmark in der Objekterkennung etablieren und auf ImageNet-Daten vortrainierte Netze dienen als Ausgangspunkt für den Start vieler Deep Learning (DL)-Projekte. Der Vorteil vieler Trainingsbeispiele liegt dabei in einer höheren Genauigkeit und besseren Generalisierungsfähigkeit im Vergleich zu einer kleinen Datenbasis. Insbesondere die Eigenschaft der Generalisierungsfähigkeit ist ein Kernelement der Neuronalen Netze, durch welches sie eine so große Bedeutung erlangen konnten. Allgemein lässt sich somit sagen, dass je mehr qualitativ hochwertige Daten für ein Training existieren, umso besser funktionieren Neuronale Netze.

1.2 Transfer Learning

Trotz Datenbanken, wie z.B. dem ImageNet, ist es nur selten der Fall, dass ausreichend viele Trainingsbeispiele vorhanden sind. Dies gilt insbesondere bei speziellen Anwendungen, wie z.B. für eine bestimmte Materialprüfung, für welche solche allgemeinen Daten nicht genutzt werden können.
Ein anschauliches Beispiel für breite Verfügbarkeit von Bildbeispielen sind Katzenbilder innerhalb des Internets. Es existiert eine unüberschaubare Anzahl und sie sind leicht zugänglich. Betrachtet man jedoch die Anzahl und Verfügbarkeit von speziellerem Bildmaterial, wie z.B. Bakterien unter einem Mikroskop [Längkvist et al., 2014], so ist die Lage eine andere. Ebenso verhält es sich mit Bildbeispielen von schadhaften Stellen auf Asphalt [Eisenbach et al., 2017] oder Fehlern auf Werkstoffen [Kim et al., 2017].
Diese Problematik ist das Kernproblem, dessen Lösung Transfer Learning (TL) zum Ziel hat. Dabei wird ein größerer Quelldatensatz verwendet, um ein Neuronales Netz einem sog. Pretraining zu unterziehen. Anschließend wird das Netz mit einem meist kleineren Zieldatensatz hinsichtlich der eigentlichen Anwendung trainiert und optimiert. Die Übernahme der im Pretraining gelernten Gewichte und Features für eine andere Aufgabe im Zieldatensatz wird Transfer Learning genannt. Abbildung 1.1 illustriert das allgemeine Vorgehen, wie es beispielsweise in [Kim et al., 2017] durchgeführt wurde. Der Quelldatensatz kann dabei auch gänzlich andere Inhalte und Objektklassen als der Zieldatensatz enthalten. Durch das Pretraining soll das Neuronale Netz

intelligent initialisiert werden, weshalb ein kleiner Zieldatensatz ausreichend ist, um das Netz auf die neue Zielaufgabe anzupassen.

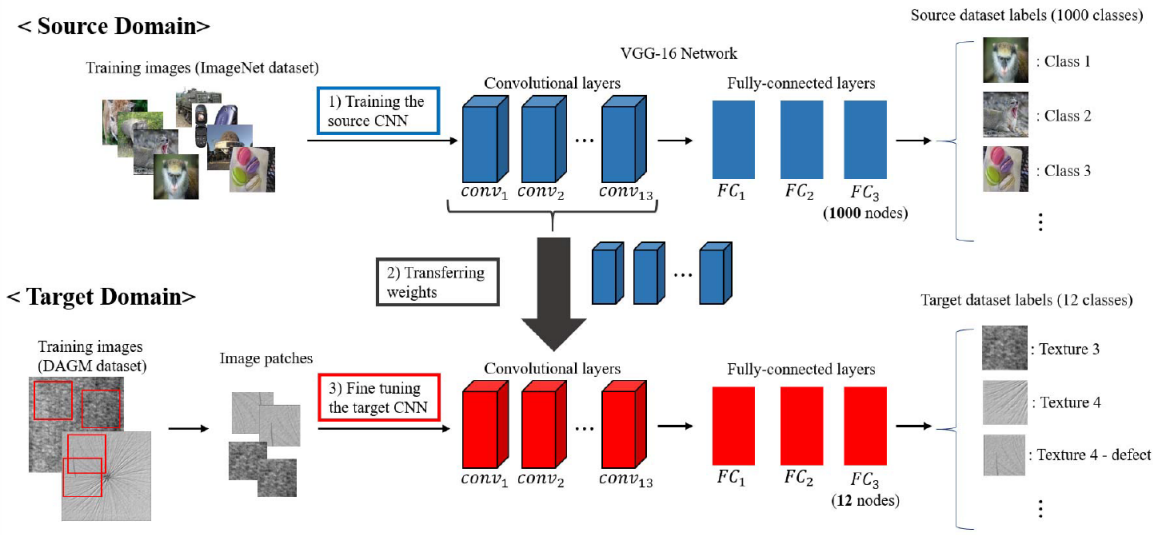


Abbildung 1.1: Allgemeines Vorgehen des Transfer Learnings
Zu sehen ist der prinzipielle Ablauf des Transfer Learnings, welcher unabhängig vom Anwendungsfall ist. Die obere Hälfte illustriert das Pretraining eines tiefen Faltungsnetzes auf einem Quell- oder Ursprungsdatensatz, wohingegen die untere Hälfte das Finetuning auf einem Zieldatensatz repräsentiert. Vom Netz, welches während des Pretrainings trainiert wird, werden die Gewichte der Faltungsschichten übernommen. In diesem Beispiel ist der Quelldatensatz ImageNet und der Zieldatensatz DAGM, welcher Werkstoffe mit und ohne fehlerhafte Stellen zeigt. Bildquelle: [Kim et al., 2017]

1.3 Aufbau der Arbeit

Die Arbeit ist in sechs Kapitel unterteilt. Das erste Kapitel dient der Einleitung und grundlegenden Einordnung des Themas. Im Folgenden werden die weiteren Kapitel kurz erläutert.

Kapitel 2. - Grundlagen
Das Kapitel "Grundlagen" geht ausführlich auf die Grundlagen der verwendeten Verfahren ein. Außerdem werden die verwendeten Architekturen der Neuronalen Netze präsentiert sowie deren Besonderheiten erläutert.

Kapitel 3. - State-of-the-Art
Das Kapitel "State-of-the-Art" gibt einen Überblick über weitere Arbeiten zu dem Thema Transfer Learning. Es werden frühere Untersuchungen zu dem Nutzen von Transfer Learning präsentiert und Effekte beschrieben, welche in anderen Arbeiten beobachtet wurden. Darüber hinaus werden verschiedene andere Einsatzmöglichkeiten von Transfer Learning beschrieben.

Kapitel 4. - Transfer Learning für die Erkennung von Straßenschäden
Das Kapitel "Transfer Learning für die Erkennung von Straßenschäden" beschreibt die Durchführung des Transfer Learnings in dieser Arbeit und gibt Aufschluss über die Experimente sowie erwarteten Effekte.

Kapitel 5. - Experimente
Das Kapitel "Experimente" präsentiert die Ergebnisse des durchgeführten Transfer Learnings sowie der aufbauenden Experimente und diskutiert die gewonnenen Erkenntnisse.

Kapitel 6. - Fazit
Das Kapitel "Fazit" resümiert die wichtigsten Erkenntnisse der vorliegenden Arbeit. Darüber hinaus werden, auf diesen Erkenntnissen aufbauend, weitere mögliche Untersuchungen diskutiert.

Grundlagen 2

Wie bereits einleitend in Kapitel 1 kurz erläutert, beschreibt der Begriff Transfer Learning lediglich die Weiterverwendung von bereits existierenden Gewichten eines Neuronalen Netzes. Diese Gewichte können das Resultat eines unabhängigen anderen Trainings oder auch aus einem gezielten Pretraining hervorgegangen sein. Um einen Einblick in den Nutzen von Transfer Learning für die Klassifikation von Straßenschäden zu erhalten, werden sowohl verschiedene Netzarchitekturen als auch verschiedene Finetuning-Varianten untersucht. Zum besseren Verständnis soll dazu in diesem Kapitel ein Überblick gegeben werden, welche Möglichkeiten es gibt, auf das Vorgehen des Transfer Learnings Einfluss zu nehmen. Abschnitt 2.2 geht dazu auf die verschiedenen Varianten von Transfer Learning ein. In Abschnitt 2.1 und 2.3 wird in aller Kürze allgemein auf Faltungsnetze eingegangen. Darauf aufbauend werden in diesen beiden Abschnitten außerdem die in dieser Arbeit verwendeten Architekturen sowie ihre Besonderheiten behandelt. Darüber hinaus werden in Abschnitt 2.4 einige Verfahren erläutert, welche für die Auswertung bzw. die Analyse der trainierten Netze notwendig sind.

2.1 Convolutional Neural Nets

Convolutional Neural Nets bzw. Faltungsnetze, wie sie für die Computer Vision erstmals von Yann LeCun 1989 in [LeCun et al., 1989] beschrieben wurden, haben in den vergangenen Jahren einen großen Durchbruch erzielt. Zwar sind über die Zeit verschiedene Formen bzw. Architekturen, wie die Inception- (2.3.3) oder Residual-Architektur

(2.3.4), entstanden, der Grundgedanke ist jedoch gleich geblieben.
Ein Faltungsnetz beschreibt eine Aneinanderreihung mehrerer Faltungsschichten, welche einen Input erhalten und aus diesem eine Klasseninformation berechnen. Die Faltungsschichten reichen untereinander sog. Featuremaps weiter, welche den Input nach der Faltung repräsentieren. Die Parameter des Faltungskernel, mit welchen die Faltung durchgeführt wird, werden im Folgenden auch Gewichte genannt und sind darüber hinaus die zu optimierenden Trainingsparameter.
Während des Trainings sucht das Neuronale Netz Faltungskernel, welche möglichst optimal dafür geeignet sind, aus dem sehr hochdimensionalen Input eine Repräsentation zu generieren, welche eine möglichst allgemeingültige Klassifikation ermöglicht. Diese schichtweise Reduktion der unnötigen Information wird beispielsweise in der Arbeit von [Shwartz-Ziv and Tishby, 2017] als Informationsflaschenhals formuliert. In dieser Betrachtung ist es die Aufgabe jeder einzelnen Schicht, eine gewisse Menge an irrelevanter oder redundanter Information zu entfernen, welche in [Shwartz-Ziv and Tishby, 2017] auch als Rauschen betrachtet wird.
Neben den reinen Faltungsschichten verfügt ein Faltungssnetz meist noch über weitere Schichttypen. Sehr häufig werden auch Poolingschichten verwendet, die ein Subsampling im Sinne einer Auflösungspyramide realisieren. Außer Acht gelassen werden an dieser Stelle verschiedene Unterformen, wie die Generativen Neuronalen Netze [Goodfellow et al., 2014] oder die (Variational) Autoencoder [Kingma and Welling, 2013], welche innerhalb dieser Arbeit keine Anwendung finden. Abschnitt 2.3 geht im Detail auf die Architekturen ein, welche innerhalb dieser Arbeit verwendet werden.

2.2 Allgemeiner Ansatz und Stellschrauben bei TL

Während des Trainings eines Netzes werden die Gewichte der Schichten eines Neuronalen Netzes so angepasst, dass der Input möglichst korrekt auf den Sollwert der Ausgabe abgebildet wird. Dies geschieht über den Backpropagation Algorithmus [Rumelhart et al., 1986]. Dieser führt einen Gradientabstieg durch und ermittelt zu jedem Gewicht des Netzes den Beitrag zu der resultierenden Abweichung zwischen Ist- und Sollwert der Netzausgabe. Auf Basis dieses Fehlerbeitrages jedes einzelnen Gewichtes

wird das entsprechende Gewicht entlang des Gradienten so angepasst, dass der Fehler reduziert wird.
Der Grundgedanke des Transfer Learnings ist, wie auch in [Yosinski et al., 2014] beschrieben, relativ simpel. Die gelernten Gewichte eines Netzes werden als Initialisierung für ein neues Netz verwendet und bilden dementsprechend einen besseren Startpunkt als eine zufällige Initialisierung der Gewichte. Wie einleitend bereits erwähnt, bestehen dabei verschiedene Möglichkeiten, auf das Transfer Learning Einfluss zu nehmen. Variiert werden kann zum einen wie viele Schichten des vortrainierten Netzes übernommen werden (z.B. in [Yosinski et al., 2014]) und zum anderen welche Schichten im anschließenden Finetuning trainiert bzw. fixiert werden ([Kim et al., 2017]). Darüber hinaus ist die Learning Rate (LR) relevant, welche während des Finetunings verwendet wird. Da sich die transferierten Gewichte bereits in einem guten lokalen Optimum befinden, besteht die Gefahr, dass bei einer zu großen LR das Optimum verlassen wird und sich das Netz in einen schlechteren Zustand begibt. Der Trainingserfolg im Pretraining wäre in diesem Fall zerstört.
Bei der Überlegung, welche Schichten des vortrainierten Netzes übernommen werden, stimmen sowohl [Kim et al., 2017] als auch [Yosinski et al., 2014] darüber überein, dass eine Übernahme aller Schichten sinnvoll, jedoch nicht zwingend notwendig, ist. In [Yosinski et al., 2014] wird jedoch explizit ermittelt, dass der Leistungszugewinn durch TL am größten ist, wenn alle vortrainierten Gewichte übernommen werden.

2.3 Verwendete Architekturen

Um eventuelle Abhängigkeiten zwischen Architekturmustern, wie z.B. den Inception-Modulen (s. 2.3.3) und der Effektivität des Transfer Learnings, zu berücksichtigen, werden die Experimente auf sieben verschiedenen und sehr verbreiteten Architekturen parallel durchgeführt. Bei allen verwendeten Architekturen handelt es sich um tiefe Feed-Forward Faltungsnetze, in denen Neuronen einer Schicht lediglich Verknüpfungen zu Neuronen einer nachfolgenden Schicht aufweisen. Darüber hinaus besitzen die jeweiligen Architekturen jedoch verschiedene Besonderheiten.
Abgeschlossen werden fast alle Netze mit zumindest einer vollverschalteten Schicht,

welche die Klassifikation über eine Softmaxausgabe realisiert. Zu beachten ist, dass die Architekturen, die in dieser Arbeit verwendet werden, eine leicht geringere Anzahl Gewichte aufweisen als die ursprünglichen Versionen in den Veröffentlichungen. Diese Abweichung geht darauf zurück, dass die Architekturen ursprünglich für eine Netzeingabe mit drei Kanälen für die Farbinformationen erstellt wurden und dementsprechend in der ersten Faltungsschicht auch Gewichte für drei Kanäle besitzen. Der innerhalb dieser Arbeit verwendete Zieldatensatz besteht allerdings aus Schwarzweißbildern, wodurch in der ersten Faltungsschicht Gewichte eingespart werden. Weitere Details zu dieser Abweichung und wie die Netze von einem Dreikanal-Input zu einem Einkanal-Input transformiert wurden, können Abschnitt 4.4.1 entnommen werden.
Im Nachfolgenden soll ein Überblick über die Architekturen gegeben werden. Detailliertere Informationen über die entsprechenden Architekturen können den angegebenen originalen Arbeiten entnommen werden, etwaige Abweichungen von diesen werden in Abschnitt 4.6 genauer erläutert.

2.3.1 AlexNet

Das AlexNet [Krizhevsky et al., 2012] ist die älteste verwendete Architektur und stellt damit auch den klassischsten Vertreter dar. Es handelt sich dabei um ein reines Feed-Forward-Netz, welches aus fünf Faltungsschichten und drei Maximumpoolingschichten besteht. Anders als in der ursprünglichen Arbeit von [Krizhevsky et al., 2012] wird jedoch keine Lokal-Response-Normalization verwendet. Abbildung 2.1 zeigt dazu eine schematische Darstellung des Netzes.
Um eine gewisse Skaleninvarianz zu erreichen, wird ein Subsampling in der ersten Faltungsschicht sowie den Poolingschichten angewendet. Dazu wird ein sog. Stride eingesetzt, welcher die Schrittweite des Filters auf 2 bzw. 4 Pixel setzt, sodass dieser Filter gewisse Pixel überspringt. Abgeschlossen wird das Netz durch drei vollverschaltete Schichten, welche außerdem den Großteil der zu lernenden Gewichte stellen. Des Weiteren kommt vor der ersten und zweiten vollverschalteten Schicht ein Dropout (s. 2.3.1) zur Anwendung, welches Overfitting vermeiden und die Generalisierungsfähigkeit verbessern soll. Außer für die Softmaxausgabe der Klassifikation wird in jeder

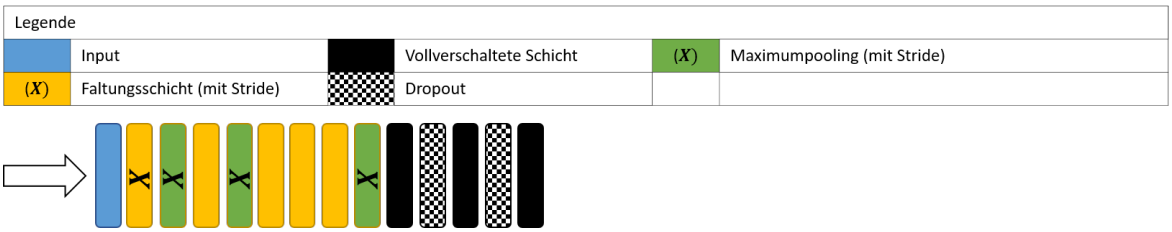


Abbildung 2.1: AlexNet Architektur

Zu sehen ist eine schematische Darstellung der AlexNet Architektur mit Legende. Das AlexNet ist eine der ältesten Architekturen und beschreibt ein sehr simples Faltungsnetz ohne spezielle Besonderheiten. Um Overfitting zu reduzieren, wird Dropout eingesetzt. Stride wird innerhalb der ersten Faltungsschicht und den Maximumpoolingschichten verwendet, um ein Subsampling zu realisieren.

Schicht eine ReLU-Ausgabe (Rectifying Linear Unit) zur Berechnung der Aktivierung verwendet.

Dropout

Dropout [Hinton et al., 2012] ist ein Regularisierungsverfahren, welches als eigenständige Schicht auf vollverschaltete Schichten angewendet wird. Es reduziert mit einer festgelegten Wahrscheinlichkeit (in den meisten Fällen 50%) die Ausgabe eines vorhergehenden Neurons zu Null. Ziel des Dropouts ist es, ein Overfitting, also das Auswendiglernen von bestimmten Mustern, zu reduzieren, was zu einer Verbesserung der Generalisierung führt. Darüber hinaus kann das Lernen mit Dropout als eine Art Ensemble-Lernen betrachtet werden. Während des Trainings sieht kein Neuron alle Beispiele und das Resultat ähnelt daher einem gewichteten Mehrheitsentscheid. Im Laufe jedes Lernschrittes entsteht somit ein anders verschaltetes Neuronales Netz.

2.3.2 VGG19

Das VGG19 [Simonyan and Zisserman, 2014] der Visual Geometry Group der Universität Oxford ist ebenfalls eines der älteren Netze und folgt dem Grundgedanken des AlexNet. Es besteht aus 16 Faltungs-, fünf Pooling- und drei vollverschalteten Schichten. Im Vergleich zum AlexNet zeichnet sich das VGG19 durch eine höhere Tie-

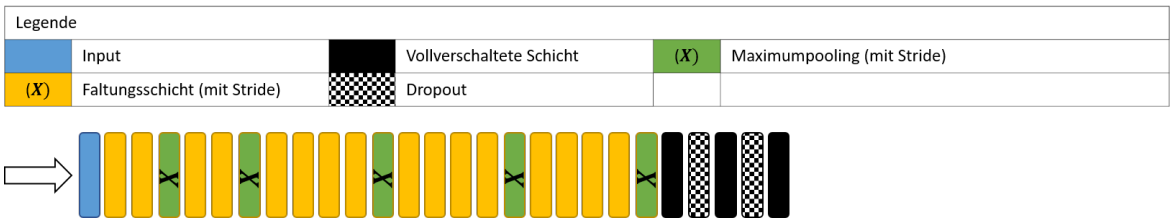


Abbildung 2.2: VGG19 Architektur

Zu sehen ist eine schematische Darstellung der VGG19 Architektur mit Legende. Analog zum AlexNet (2.1) handelt es sich bei der VGG19 Architektur um eine der älteren, sehr einfachen Architekturen. Das VGG19 besitzt keine speziellen Besonderheiten, ist im Vergleich zum AlexNet jedoch um einiges tiefer. Um Overfitting zu reduzieren, wird Dropout eingesetzt. Stride wird innerhalb der Maximumpoolingschichten verwendet, um ein Subsampling zu realisieren.

fe aus. Ansonsten ist der Aufbau weitgehend analog. Insbesondere durch die Tiefe und relativ großen Featuremaps im Übergang zu den vollverschalteten Schichten besitzt das VGG19 mit über 143 Mio. die meisten Gewichte unter den verwendeten Architekturen. Ebenfalls analog ist die Verwendung von Dropout und Maximumpooling, der ReLU-Ausgabefunktion sowie die Softmaxausgabe zur Klassifikation. Abbildung 2.2 zeigt eine schematische Darstellung des Netzes.

2.3.3 InceptionNetV3

Die ursprüngliche Inception-Architektur geht zurück auf die Arbeit von [Szegedy et al., 2015] und wird in dieser Arbeit in der Version "V3" [Szegedy et al., 2016] verwendet. Auch in der dritten Version des Netzes bilden die sog. Inception-Module (s. 2.3.3) die Netzbasis. Zehn dieser Inception-Module werden miteinander verkettet und realisieren somit das Netz, welches eine erheblich höhere Tiefe aufweist als das bisherige AlexNet (s. 2.3.1) oder das VGG19 (s. 2.3.2). Der Größenunterschied zu den vorher beschriebenen Netzen wird insbesondere durch Abbildung 2.3 deutlich.
Wichtig zu beachten ist dabei, dass trotz der deutlich höheren Tiefe das InceptionNetV3 mit über 23 Mio. deutlich weniger Gewichte aufweist als das VGG19 (s. 2.3.2). Neben den Inception-Modulen ist die Verwendung von Batch Normalization-Schichten eine weitreichende Neuerung, welche in Abschnitt 2.3.3 weiterführend be-

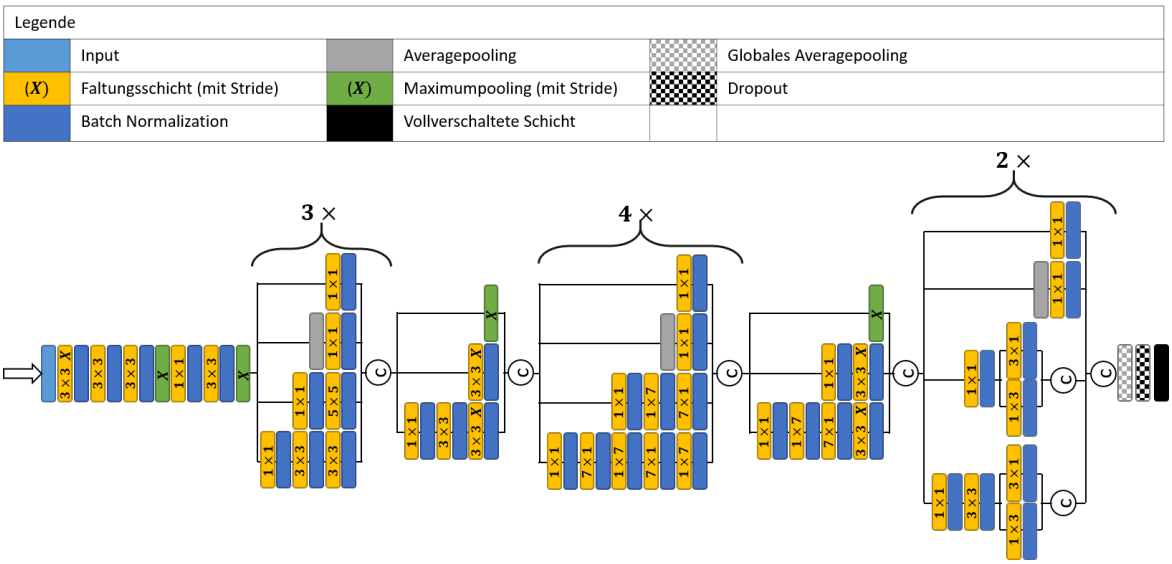


Abbildung 2.3: InceptionNetV3 Architektur

Zu sehen ist eine schematische Darstellung der InceptionNetV3 Architektur mit Legende. Auch in der dritten Version des InceptionNets ist der Aufbau der ersten Version grundlegend übernommen worden. Das Netz setzt sich aus einem Anfangs- bzw. Endteil und insgesamt fünf verschiedenen Inception-Modulen zusammen, welche in Abschnitt 2.3.3 genauer erläutert werden. Jedes Inception-Modul teilt sich in mehrere Zweige auf, welche anschließend konkateniert werden. Um Overfitting zu reduzieren, wird Dropout und Batch Normalization (s. Abschnitt 2.3.3) eingesetzt. Stride wird innerhalb der Maximumpoolingschichten verwendet, um ein Subsampling zu realisieren.

schrieben wird. Unabhängig von der Tiefe des InceptionNets wird ebenfalls die ReLU-Ausgabefunktion sowie eine Softmaxausgabe zur Klassifikation genutzt. Um ein Subsampling zu realisieren, wird sowohl Average- als auch Maximumpooling verwendet.

Inception-Module

Jedes Inception-Modul bildet eine abgeschlossene Einheit und kann beliebig mit anderen verkettet werden. Ein Inception-Modul besteht aus mehreren sog. Branches, welche sich sowohl in ihrem Aufbau als auch in den verwendeten Filtergrößen der Faltungsschichten unterscheiden. Das InceptionNetV3 besteht aus fünf verschiedenen Typen von Inception-Modulen, deren Aufbau in Abbildung 2.4 schematisch darge-

stellt ist. Am Ende jedes Inception-Moduls werden alle resultierenden Featuremaps als Kanäle konkateniert, sodass der darauffolgenden Schicht alle Featuremaps als Input präsentiert werden. Ziel des besonderen Aufbaus der Inception-Module ist einerseits die Reduktion der Features, um Rechenkapazität einzusparen, und andererseits eine Regularisierung, um Overfitting zu vermeiden. Um diese Ziele zu erreichen, werden zu Beginn jedes Branches 1×1 Faltungen verwendet. Die 1×1 Faltungen bewerten, wie auch in [Chollet, 2016] beschrieben, Korrelationen zwischen den Kanälen und helfen somit, die Anzahl der Featuremaps, zu Beginn jedes Branches, zu reduzieren. Durch diese Reduktion wird innerhalb des Branches eine erhebliche Menge Gewichte eingespart, wodurch ebenfalls die nötige Rechenzeit verringert wird. Darüber hinaus werden in den Branches verschiedene Filtergrößen verwendet, um eine Empfindlichkeit auf verschieden große Bildstrukturen zu ermöglichen. Während der Anwendung ist meist unbekannt, welche Strukturgrößen das zu erkennende Objekt auf dem Bildmaterial besitzt. Der Einsatz mehrerer Filtergrößen wirkt diesem Problem entgegen. Nachdem alle Branches eines Moduls konkateniert wurden, folgt das nächste und kann aus der Menge der Features mit unterschiedlichen Filtergrößen die beste Kombination auswählen. Die verwendeten Größen sind: (1×1), (1×3), (3×1), (1×7), (7×1), (3×3), (5×5).

Batch Normalization

Das Training eines Netzes erfolgt aus Effizienzgründen batchweise, was bedeutet, dass während des Trainings immer mehrere Trainingsbeispiele parallel von dem Netz berechnet werden. Batch Normaliziation (BN, [Ioffe and Szegedy, 2015]) ist dabei ein Normalisierungsschritt, welcher jeweils auf ein einzelnes Batch angewendet wird, und eine eigene Schicht innerhalb des Netzes bildet. Die BN folgt meist auf jede Hidden-Schicht und besitzt Gewichte, welche während des Trainings gelernt werden. Diese Gewichte sind die Normalisierungsparameter, also Mittelwert, Varianz und Skalierung sowie Shift eines jeden Kanals. Die Besonderheit des Mittelwertes und der Varianz ist, dass sie während des Trainings nicht angewendet, sondern nur als gleitender Mittelwert bzw. Varianz gelernt werden. Stattdessen werden während der Trainingsphase

der Mittelwert und die Varianz, die für die Normierung benötigt werden, direkt aus dem aktuellen Batch ermittelt. Während der Anwendungsphase werden hingegen die im Training gelernten Werte für Mittelwert und Varianz verwendet, um den Input zu normieren.

Ziel der Normalisierung ist es, die numerische Abhängigkeit zwischen den Schichten zu reduzieren und den Fluss des Gradienten zu verbessern. Durch die Normierung jeder

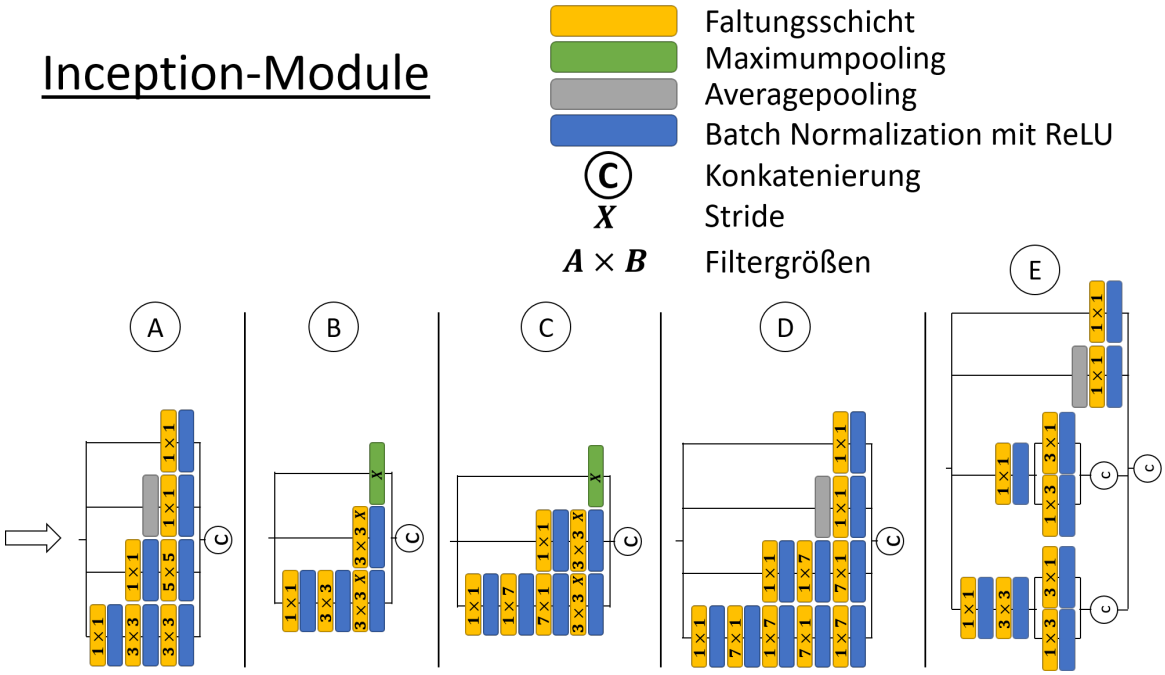


Abbildung 2.4: Überblick der Inception-Module

Zu sehen ist eine schematische Darstellung der fünf Inception-Module, welche im InceptionNetV3 verwendet werden. Alle Module sind in ihrem Aufbau relativ ähnlich, unterscheiden sich jedoch in der Anzahl der Branches, der Anzahl der Schichten pro Branch und der verwendeten Filtergrößen pro Branch. Durch die Aufteilung in verschiedene Branches innerhalb eines Moduls ist es möglich, verschiedene Filtergrößen zu verwenden. Dies hat den Vorteil, dass in jedem Modul Features trainiert werden können, welche auf Bildstrukturen unterschiedlicher Größe empfindlich sind. Die Module B und C realisieren darüber hinaus durch die Verwendung von Stride ein Subsampling. Abgeschlossen werden alle Module durch die Konkatenierung aller Featuremaps entlang der Kanäle.

Ausgabe sind nachfolgende Schichten nicht darauf angewiesen, die Ausgabekodierung der jeweils vorhergehenden Schicht zu lernen. Würde eine starke Abhängigkeit von der Höhe der Aktivierung vorliegen, hätte das einen großen Einfluss auf die nachfolgenden Schichten. Der Effekt einer Änderung zu Beginn des Netzes würde sich daher über das Netz hinweg potenzieren. Die Normierung zielt darauf ab, dies zu minimieren. Eine Änderung der Gewichte in einer vorderen Schicht, z.B. durch das Training, hat dadurch geringeren Einfluss auf das Verhalten nachfolgender Schichten. Des Weiteren wird durch die stetige Normalisierung ebenfalls der Fluss des Gradienten durch das Netz verbessert. Das ermöglicht zum einen, tiefere Netze effektiv zu trainieren, und zum anderen eine bessere Trainingskonvergenz. Ein weiterer positiver Effekt der Batch Normalization ist die Regularisierung der Gewichte. Durch die stetige Normierung der Aktivierung wird ebenfalls die Gefahr reduziert, dass die gelernten Gewichte betragsmäßig zu groß werden.

2.3.4 ResNet50

Das ResidualNet50 (ResNet50) [He et al., 2016] beschreibt grundlegend ein Netz, welches, dem VGG19 (s. Abschnitt 2.3.2) ähnelnd, aus hintereinander geschalteten Faltungsschichten besteht. Im Gegensatz zu dem VGG19 ist das ResNet jedoch erheblich tiefer, verwendet Batch Normalization (s. Abschnitt 2.3.3) und besitzt sog. Residual- oder Skip-Connections. Diese Skip-Connections fassen mehrere Faltungsschichten jeweils zu einem Residual-Block bzw -Modul, wie in Abschnitt 2.3.4 beschrieben, zusammen. Abbildung 2.5 zeigt eine schematische Darstellung des Netzes. Ziel der Skip-Connections ist es, das Problem der verschwindenden Gradienten bei tiefen Neuronalen Netzen zu reduzieren.

In [He et al., 2016] werden mehrere Versionen des ResNets beschrieben, die sich jedoch nur in der Anzahl der Residual-Blöcke unterscheiden. Die Architektur, welche in dieser Arbeit verwendet wird, ist das ResNet50. Es besitzt, seinem Namen entsprechend, insgesamt 50 Faltungsschichten. Damit verfügt es über dreimal so viele Faltungsschichten wie das VGG19. Ein vollständiges ResNet besteht somit aus einer Eingabefaltungsschicht, einer Abfolge der Residual-Blöcke und abschließend aus einem

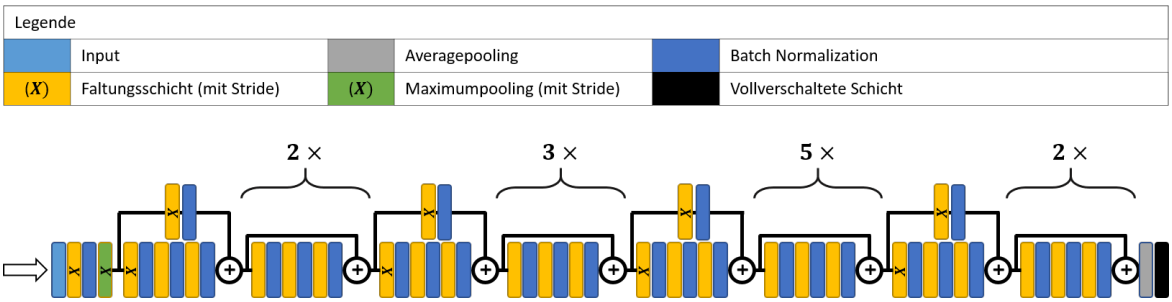


Abbildung 2.5: ResNet50 Architektur

Zu sehen ist eine schematische Darstellung der ResNet50 Architektur mit Legende. Im Vergleich zu einem klassichen Faltungsnetz, wie dem VGG19 (s. 2.3.2), ist die ResNet50 Architektur um ein vielfaches tiefer. Mehrere Faltungsschichten werden zu einem Residual-Block (s. 2.3.4) zusammengefasst und von einer Skip-Connection übersprungen. Durch diese Überbrückungen konnte das Problem der verschwindenden Gradienten ausreichend minimiert werden, um selbst Netze mit solch einer hohen Tiefe effektiv trainieren zu können. Um ein Subsampling zu realisieren, wird, zusätzlich zu der ersten Faltungs- und Poolingschicht, auch in einigen Residual-Modulen ein Stride angewendet.

Averagepooling und einer vollverschalteten Schicht mit Softmaxausgabe. Subsampling wird verteilt, über verschiedene Residual-Blöcke innerhalb des Netzes, realisiert. Über die Tiefe des Netzes hinweg wird die Anzahl der Filter pro Schicht mehrfach variiert, sodass die Anzahl der Featuremaps im Verlauf von 64 pro Schicht auf bis zu 2048 pro Schicht ansteigt.

Residual-Block/-Modul

Trotz der scheinbaren Ähnlichkeit mit der VGG19-Architektur sind die Skip-Connections ein ausschlaggebender Unterschied. Durch diese Verbindungen werden einige Faltungsschichten übersprungen, wodurch, wie in Abbildung 2.6 zu sehen, ein Bypass für das Netz gebildet wird. Ziel des Bypass ist die Realisierung eines besseren Flusses des Gradienten. Erreicht wird dies durch Überbrückung der Faltungsschichten, was einem linearen Zusammenhang entspricht bzw. abgeleitet in einer Einheitsfunktion resultiert. Somit kann der Gradient, selbst durch sehr tiefe Netze, deutlich besser

fließen und ermöglicht ein gutes Konvergieren des Trainings.
Die ResNets können, wie auch in Abbildung 2.6 zu erkennen ist, in zwei verschiedene Module unterteilt werden. Die beiden verschiedenen Residual-Module unterscheiden sich lediglich hinsichtlich des Subsamplings. Während Modul B kein Subsampling besitzt, verfügt Modul A über eine Faltungsschicht innerhalb der Skip-Connection, welche durch einen Stride das Subsampling durchführt. Um die Featuremapgröße gleichmäßig zu halten, wird der Stride innerhalb des Residual-Moduls ebenfalls verwendet.
Der entscheidende Unterschied zur Inception-Architektur, an welche diese Parallelität erinnern könnte, ist, dass die Featuremaps am Ende eines Residual-Blocks nicht konkateniert, sondern addiert werden. Das führt zu einer gänzlich anderen Funktion und sollte nicht mit der Inception-Architektur verwechselt werden.

2.3.5 XceptionNet

Die Xception-Architektur [Chollet, 2016] hat seinen geistigen Ursprung in der Inception-Architektur. Der Name Xception leitet sich daher auch von Xtreme Inception ab. Im Gegensatz zu dem InceptionNet beruht die Xception-Architektur allerdings hauptsächlich auf Separable Convolutions (s. 2.3.5), welche innerhalb der Arbeit von [Chollet, 2016] als eigenes Inception-Modul in sich selbst verstanden werden.
Nichtsdestotrotz erinnert das XceptionNet in seinem Aufbau eher an ein ResNet. Neben den Separable Convolutions besitzt das XceptionNet, analog zur ResNet-Architektur, Skip-Connections, welche die periodischen Blöcke parallel überbrücken. Insgesamt ergeben sich somit 14 Module mit 36 Faltungsschichten, welche bis auf das erste und letzte Modul, jeweils durch eine Skip-Connection überbrückt werden. Darüber hinaus wird in der Xception-Architektur ebenfalls Batch Normalization eingesetzt. Mit 22,8 Millionen zu lernenden Gewichten ist das XceptionNet hinsichtlich seiner Größe mit dem InceptionNetV3 insgesamt vergleichbar.
Um ein Subsampling zu realisieren, wird insbesondere in den vorderen Modulen Stride innerhalb der Faltungs- und Maximumpooling-Schichten verwendet. Abgeschlossen

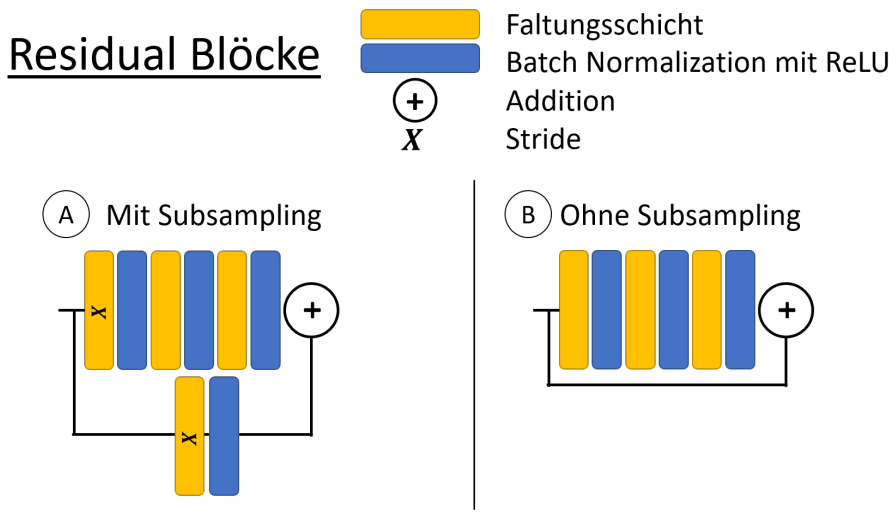


Abbildung 2.6: Überblick der Residual-Blöcke

Zu sehen ist eine schematische Darstellung der beiden Residual-Blöcke, welche innerhalb der ResNet Architektur verwendet werden. Beide sind in ihrem Aufbau einander sehr ähnlich und bestehen aus einer Reihe Faltungsschichten mit Batch Normalization (s. 2.3.3). Block A realisiert darüber hinaus durch einen Stride auch ein Subsampling. Im Anschluss an beide Module werden die resultierenden Featuremaps beider Pfade addiert. Durch den einfachen additiven Zusammenhang ergibt sich für den Gradienten der Skip-Connections eine Einheitsfunktion, wodurch der Gradient insgesamt sehr gut durch das Netz fließen kann.

wird das Netz durch ein globales Averagepooling sowie eine vollverschaltete Schicht mit Softmaxausgabe. Eine schematische Darstellung der Architektur kann Abbildung 2.7 entnommen werden.

Separable Convolutions

Eine Separable Convolution, wie auch in [Chollet, 2016] beschrieben, beinhaltet, dem Namen entsprechend, die Zerlegung einer normalen Faltung in zwei einzelne Faltungen. Eine normale Faltung wird aufgeteilt in jeweils eine zweidimensionale Faltung mit räumlicher Ausdehnung über einen Kanal hinweg und eine eindimensionale Faltung entlang der Kanäle.

Beispielsweise erfordert die Faltung eines Dreikanalfarbbildes mit einem Filter der

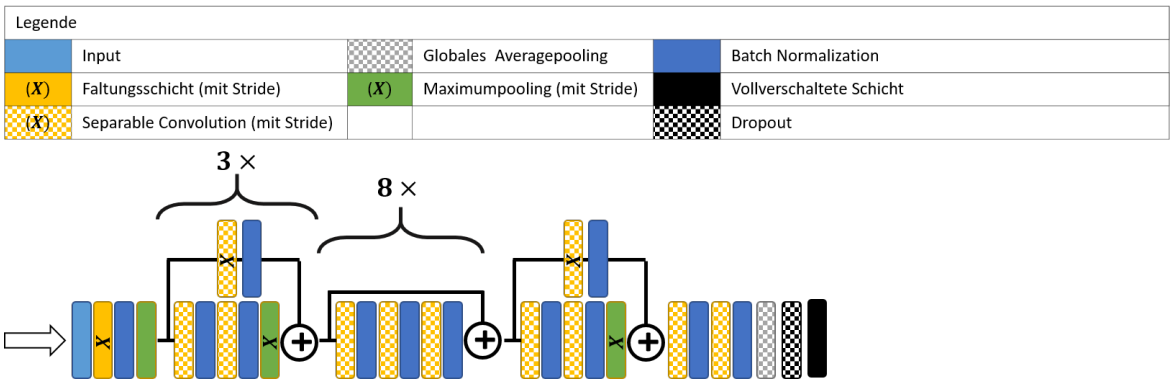


Abbildung 2.7: Xception Architektur

Zu sehen ist eine schematische Darstellung der Xception Architektur mit Legende. Trotz der großen Ähnlichkeit mit der ResNet Architektur (s. 2.3.4) ist das Vorbild für das XceptionNet das InceptionNet.

Wie auch in Abschnitt 2.3.5 beschrieben, werden die Separable Convolutions in ihrer Funktion als eine Art Inception-Modul aufgefasst. Darüber hinaus ähnelt das XceptionNet in seinem Aufbau jedoch sehr der ResNet Architektur. Es werden ebenfalls zwei verschiedene Typen von Residual-Blöcken verwendet, wobei ein Typ ein Subsampling durch eine Faltung mit Stride innerhalb der Skip-Connection realisiert. Der große Unterschied zum ResNet besteht lediglich in der konsequenten Verwendung von Separable Convolutions.

Größe 3×3 einen Kernel mit $3 \times 3 \times 3$ Gewichten. Es ist jedoch möglich, diese Faltung durch eine $3 \times 3 \times 1$ sowie eine $1 \times 1 \times 3$ Faltung zu ersetzen, ohne gleichzeitig die Ausgabegröße zu verändern. Diese mathematische Umformulierung spart in diesem Beispiel mit 12 anstelle von 27 Gewichten erheblich an Parametern.

Die Aufteilung ist dabei so interpretierbar, dass zunächst relevante Strukturen, z.B. Ecken, Kanten bzw. Intensitäten, auf dem Input ermittelt werden. Anschließend fließen Korrelationen über die Kanäle hinweg, in das gelernte Feature mit ein. Es ist jedoch zu beachten, dass die Separable Convolution mathematisch nicht exakt der gleichen Operation entspricht wie eine normale Faltung. Ziel der Separable Convolution ist es, durch die eingesparten Parameter eine deutlich höhere Anzahl Neuronen zu ermöglichen, ohne dass die Masse der Gewichte eines Netzes zu stark zunimmt. Durch eine höhere Anzahl Neuronen wird es dem Netz ermöglicht, mehr verschiedene Features

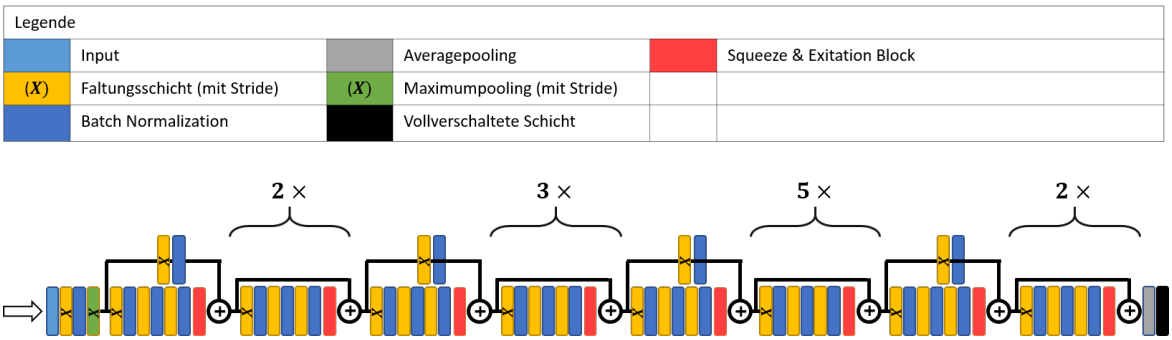


Abbildung 2.8: SE-ResNet Architektur

Zu sehen ist eine schematische Darstellung der SE-ResNet Architektur mit Legende. Im grundlegenden Aufbau folgt die SE-ResNet Architektur exakt dem normalen ResNet und ergänzt dieses nur um die Squeeze & Excitation Blöcke (s. 2.3.6).

auszuprägen, welche für eine Klassifikation verwendet werden können.

2.3.6 Squeeze-and-Excitation-ResidualNet50

Das Squeeze-and-Excitation-ResidualNet50 (SE-ResNet50) [Hu et al., 2017] folgt in seinem Aufbau grundlegend exakt dem originalen ResNet50 und bildet lediglich eine Erweiterung dessen. Dazu wird nach jedem Residual-Block ein sog. Squeeze-and-Excitation-Block (SE-Block) (s. 2.3.6) eingefügt. Aufgabe des SE-Blocks ist die Gewichtung der Featuremaps, welche durch den Residual-Block ermittelt wurden. Alle anderen Eigenschaften des ResNets bleiben analog zu dem originalen ResNet50. Abbildung 2.8 zeigt eine schematische Darstellung des SE-ResNet50.

Squeeze-and-Excitation Blocks

Ein Squeeze-and-Excitation Block bzw. SE-Block nach [Hu et al., 2017] besteht aus einem globalen Averagepooling und zwei vollverschalteten Schichten. Die erste der beiden vollverschalteten Schichten besitzt dabei um den Faktor 16 weniger Neuronen als die zweite.

Durch das Pooling wird zunächst jede Featuremap auf lediglich einen Pixelwert in Höhe der durchschnittlichen Aktivierung reduziert. Der resultierende Vektor bildet die Eingabe für die beiden vollverschalteten Schichten, wobei die erste dieser beiden durch

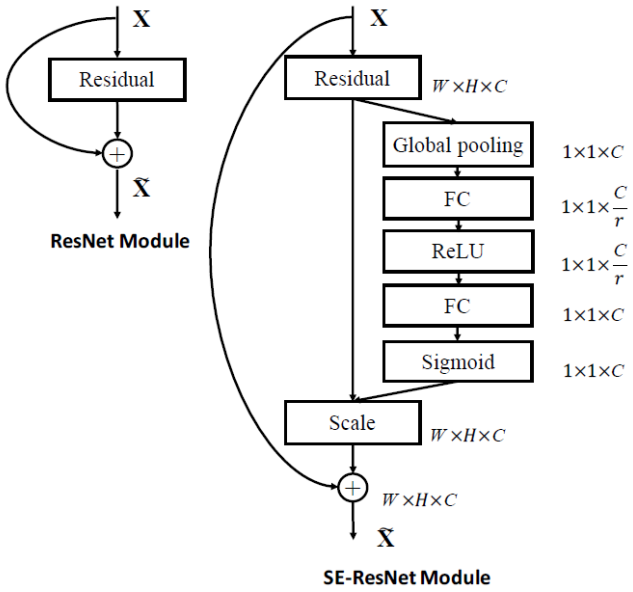


Abbildung 2.9: Squeeze-and-Excitation Block

Zu sehen ist die schematische Darstellung eines Squeeze-and-Excitation Blocks eines SE-ResNets. Der SE-Block folgt auf einen normalen Residual-Block, wie in einem ResNet, und berechnet jeweils eine Gewichtung für jede Featuremap. Genauere Details zur Funktion können Abschnitt 2.3.6 entnommen werden.

die verringerte Anzahl Neuronen einen künstlichen Flaschenhals bildet. Die Sigmoidausgabe der hinteren vollverschalteten Schicht, welche Ausgaben zwischen Null und Eins erzeugt, entspricht den Gewichtungsfaktoren, mit denen die Featuremaps multipliziert bzw. gewichtet werden. Abbildung 2.9 zeigt den schematischen Ablauf eines SE-Blocks.

Während des Trainings versucht das Netz trotz des Flaschenhalses alle für die Klassifikation relevanten Informationen zu erhalten und wird gezwungen, eine Gewichtung der Featuremaps zu lernen. Dadurch wird eine Korrelation zwischen mittlerer Aktivierung einer Featuremap und dem Beitrag zur korrekten Klassifikation ermittelt.

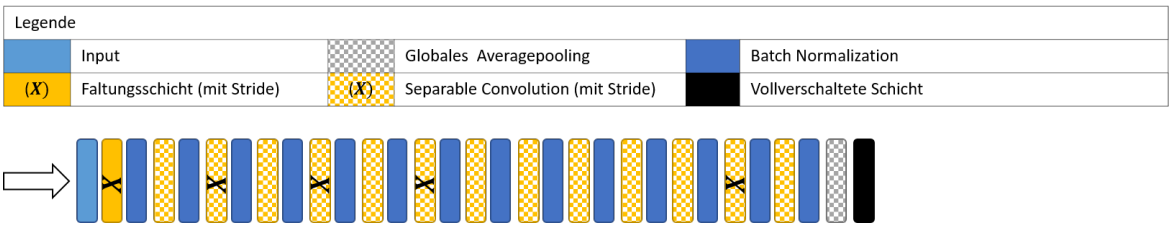


Abbildung 2.10: MobileNet Architektur

Zu sehen ist eine schematische Darstellung der MobileNet Architektur mit Legende. Der Aufbau des MobileNets erinnert stark an das VGG19, wurde jedoch gezielt vor dem Hintergrund entworfen, dass es auf mobilen Plattformen zum Einsatz kommt. Um eine hohe Effizienz zu erreichen, wird im Vergleich zum VGG19 konsequent auf den Einsatz von Separable Convolutions gesetzt und lediglich eine vollverschaltete Schicht verwendet. Insgesamt konnte die Menge der nötigen Berechnungen somit deutlich reduziert werden, ohne dass die Leistung des Netzes zu stark einbricht.

2.3.7 MobileNet

Die MobileNet-Architektur [Howard et al., 2017] wurde speziell für den Einsatz auf mobilen Endgeräten entwickelt. Der einfachen VGG19-Architektur ähnelnd, besitzt das MobileNet keine Skip-Connections oder Inception-Module und besteht aus einer einfachen Verkettung mehrerer Faltungsschichten. Im Gegensatz zum VGG19 werden jedoch Separable Convolutions eingesetzt. Die MobileNet-Architektur ist außerdem mit 13 Faltungsschichten weniger tief als das VGG19. Des Weiteren wird im Anschluss an die Faltungsschichten des MobileNet Batch Normalization (s. 2.3.3) eingesetzt. Abgeschlossen wird das Netz durch ein Averagepooling sowie eine vollverschaltete Schicht mit Softmaxausgabe. Abbildung 2.10 zeigt eine schematische Darstellung des MobileNets, wie es in dieser Arbeit verwendet wird. Die verwendete Version weicht lediglich geringfügig von der originalen Version ab. Kapitel 4.6 geht auf die Unterschiede der Architekturen genauer ein.

Mit circa 4 Mio. Gewichten ist das MobileNet die Architektur mit der geringsten Anzahl an Parametern. Neben der geringen Größe des Netzes existiert darüber hinaus ein Parameter α, welcher verwendet wird, um die Anzahl der Neuronen des Netzes anzupassen. Dieser ist dabei lediglich ein Faktor, welcher bei der Erstellung des Netzes

die Filteranzahl für die jeweiligen Schichten beeinflusst. Ferner besitzt das MobileNet keine spezifischen Besonderheiten.

2.3.8 Vergleich der Anzahl der Gewichte

Anzahl der zu trainierenden Parameter der jeweiligen Netze:

Architektur	Anzahl der Gewichte
MobileNet	4,253,864
InceptionNetV3	23,851,784
ResNet50	25,636,712
SE-ResNet50	28,141,144
XceptionNet	22,910,480
AlexNet	60,965,224
VGG19	143,667,240

Tabelle 2.1: Vergleich der Anzahl der Parameter aller verwendeten Architekturen

2.4 Analyse Vorgehen

Neben dem Training der verschiedenen Architekturen sollen innerhalb dieser Arbeit Effekte des Transfer Learnings untersucht werden. Die PCA (s. Abschnitt 2.4.1) und t-SNE (s. Abschnitt 2.4.2) werden dazu verwendet, Featuremaps eines Faltungsnetzes zweidimensional darzustellen. Der Fisher Score (s. Abschnitt 2.4.3) wird benötigt, um die lineare Trennbarkeit der Ausgabe eines Klassifikators zu bewerten, und die Feature-Attention-Analyse (s. Abschnitt 2.5) soll zeigen, welche Regionen eines Inputs für die gelernten Features eine hohe Aktivierung hervorrufen. Diese Verfahren werden im Folgenden erläutert.

2.4.1 Principal Component Analysis

Die Principal Component Analysis (PCA) [Pearson, 1901] formuliert ein Verfahren, um zu einer Datenmenge ein neues orthogonales Koordinatensystem zu ermitteln, welches mittelwertfrei ist und den Achsen entlang die höchst mögliche Varianz aufweist. Abbildung 2.11 zeigt dazu ein Beispiel für eine zweidimensionale Datenverteilung.
Ein durch die PCA ermitteltes neues Ursprungssystem kann verwendet werden, um die Dimensionalität eines Problems zu reduzieren, indem Achsen, auf denen kaum bis keine Varianz vorhanden ist, entfernt werden. Des Weiteren kann somit eine Visualisierung von hochdimensionalen Daten in einer zweidimensionalen Grafik erstellt werden. Abbildung 2.12 zeigt dazu ein Beispiel. In dieser Arbeit wird die PCA verwendet, um Featuremaps, die durch ein Faltungsnetz berechnet wurden, zweidimensional darzustellen. Ziel ist es, innerhalb dieser Features Cluster zu finden.

2.4.2 t-Distributed Stochastic Neighbor Embedding

t-Distributed Stochastic Neighbor Embedding (t-SNE) [Maaten and Hinton, 2008] ist eine Erweiterung der Arbeit von [Hinton and Roweis, 2003] und bildet ein Verfahren zur Visualisierung hochdimensionaler Datenpunkte. Ziel ist es, die Daten auf zwei bzw. drei Dimensionen zu reduzieren und dabei lokale bzw. globale Nachbarschaften zu erhalten. In dieser Arbeit wird t-SNE analog zur PCA verwendet und soll Featuremaps,

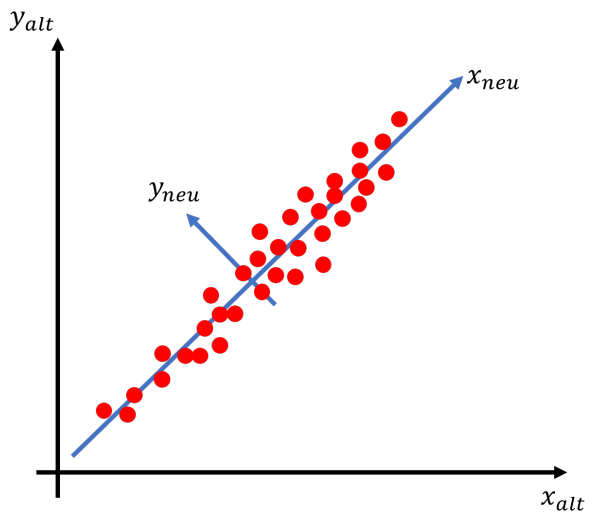


Abbildung 2.11: PCA im zweidimensionalen Raum

Zu sehen ist eine beispielhafte Visualisierung einer PCA für Daten in einem zweidimensionalen Raum. Anhand der gegebenen Daten ist zu erkennen, dass der Großteil der Varianz entlang der Achse x_{neu} verläuft. Ausgehend von den Daten im Koordinatensystem mit den Achsen x_{alt} und y_{alt}, berechnet die PCA eine neue Repräsentation auf Basis des neuen Koordinatensystems x_{neu} und y_{neu}. Die Länge der Basisvektoren, welche durch die Eigenvektoren beschrieben werden, entspricht den Eigenwerten, welche durch die PCA ermittelt wurden. Die Eigenwerte stehen dabei im direkten Verhältnis zur Varianz, welche die Daten entlang dem zugehörigen Eigenvektor aufweisen.

die durch ein Faltungsnetz berechnet wurden, zweidimensional darstellen. Ziel ist es, innerhalb dieser Features Cluster zu finden.

In der Arbeit von [Maaten and Hinton, 2008] wird die Beziehung zwischen den Datenpunkten als Wahrscheinlichkeit abgebildet. Mit anderen Worten, betrachtet man einen Punkt A, existiert eine gewisse Wahrscheinlichkeit, dass sich ein Punkt B in einem bestimmten Abstand zu diesem aufhält. Diese Wahrscheinlichkeit wird zunächst für die Kombination aller Datenpunkte zueinander im hochdimensionalen Raum ermittelt. Anschließend werden in einem z.B. zweidimensionalen Raum die gleiche Anzahl zufälliger Punkte initialisiert und diese solange angepasst, bis die Aufenthaltswahrscheinlichkeiten dieser mit den Daten im hochdimensionalen Raum vergleichbar sind.

Als Maß für die Ähnlichkeit der Wahrscheinlichkeitsverteilungen wird die Kullback-

Leibler-Divergenz (KLD) verwendet, welche im Verlauf des Verfahrens minimiert werden soll. Um die KLD zu minimieren, werden die anfangs zufällig initialisierten Punkte im niedrigdimensionalen Raum schrittweise variiert. Um zu bestimmen, wie diese Punkte variiert werden müssen, um die KLD zu verringern, wird ein Gradientenabstieg durchgeführt. Die Funktion, in [Maaten and Hinton, 2008] auch Kostenfunktion genannt, an der der Gradientenabstieg durchgeführt wird, ist die Summe über alle KL-Divergenzen.

Erreicht man einen Zustand, in dem für die Kostenfunktion ein gutes Optimum ge-

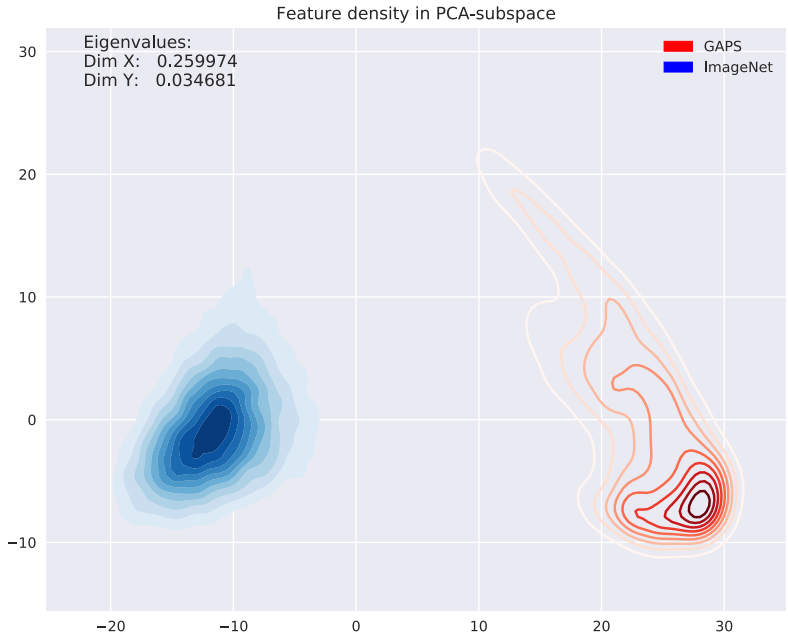


Abbildung 2.12: PCA Visualisierung von Featuremaps

Zu sehen ist die Visualisierung der Dichte von Datenpunkten innerhalb eines zweidimensionalen Unterraums. Für die Erstellung der Grafik wurden die Featuremaps zweier Datensätze durch ein Faltungsnetz berechnet und durch eine PCA auf zwei Dimensionen heruntergebrochen. Darüber hinaus werden die normierten Eigenwerte der beiden Achsen angegeben. Wie zu erkennen ist, lassen sich beide Datensätze gut voneinander trennen, was für einen gewissen Unterschied der Bilddaten spricht.

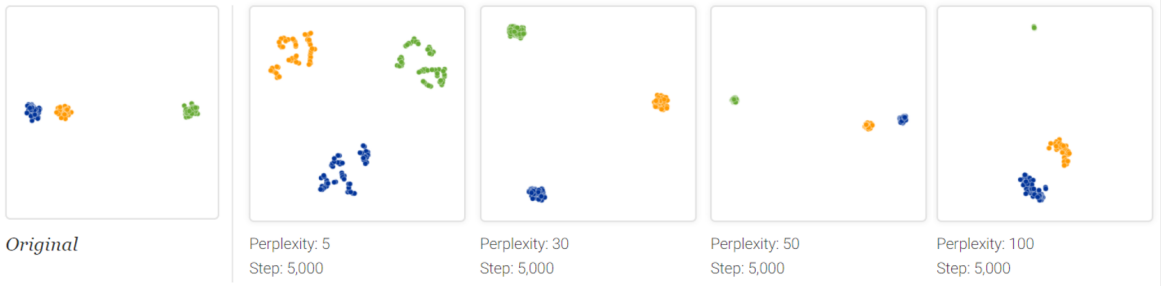


Abbildung 2.13: t-SNE Clustergrößen und Abstände
Zu sehen ist eine t-SNE Visualisierung der gegebenen Daten (links) nach jeweils 5000 Iterationen, jedoch mit verschiedenen Perplexities. Wie zu erkennen ist, ist es kaum bis gar nicht möglich, auf Basis des Ergebnisses Rückschlüsse auf die Clustergrößen oder Abstände in der Originalverteilung zu ziehen.

funden wurde, ist das Verfahren abgeschlossen. Trotz eines möglicherweise guten Optimums gibt es für die Anwendung des t-SNE Verfahrens jedoch einige Dinge, die zu beachten sind.
Grundlegend handelt es sich bei t-SNE um ein Verfahren, aus dessen Lösung zunächst lediglich abgelesen werden kann, ob gewisse Cluster im hochdimensionalen Raum trennbar sind. Die Repräsentation im niedrigdimensionalen Raum gibt, wie in [Wattenberg et al., 2016] beschrieben, nicht unbedingt die korrekten Relationen des Abstandes zwischen Clustern oder die Clustergrößen wieder. Abbildung 2.13 verdeutlicht die Problematik.

2.4.3 Fisher Score

Der Fisher Score ist ein Maß zur Bewertung der Inter- und Intraklassenvarianz. In dieser Arbeit wird der Fisher Score verwendet, um die Ausgabe eines Klassifikators für zwei verschiedene Klassen zu bewerten. Anhand dieser Bewertung soll ein Ausgabeneuron des vortrainierten Klassifikators ermittelt werden, welches die Daten des Zieldatensatzes linear trennbar kodiert.
Um zwei Datenmengen voneinander linear trennen zu können, ist es zunächst sinnvoll, zu betrachten, ob diese überhaupt linear trennbar sind. Abbildung 2.14 zeigt dazu ein Beispiel, was Inter- und was Intraklassenvarianz im Hinblick auf eine gute Trennbarkeit

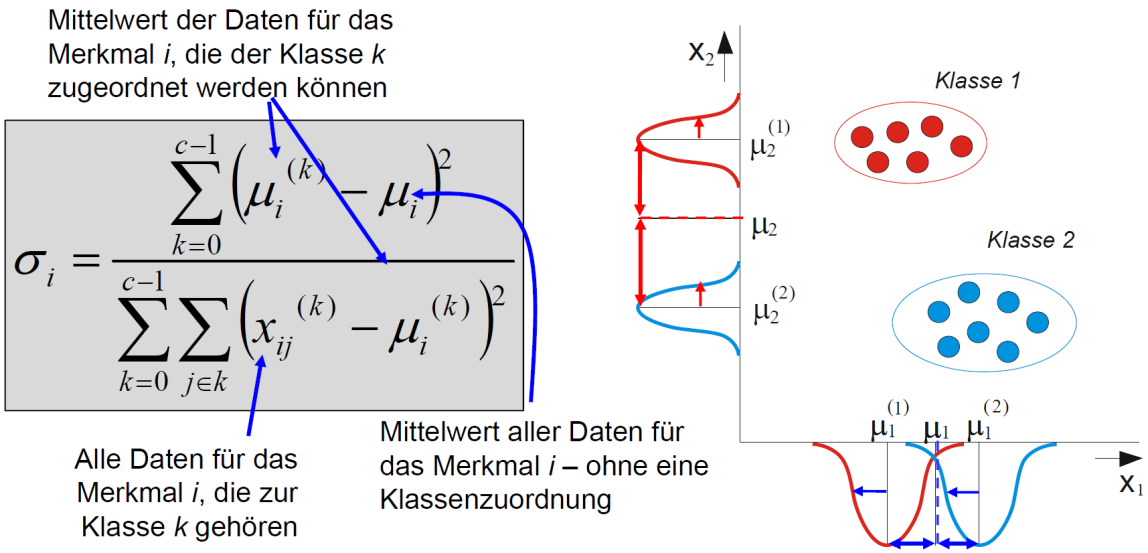


Abbildung 2.14: Berechnung des Fisher-Scores für zwei Klassen

Zu sehen ist eine zweidimensionale Datenverteilung von zwei Klassen und die Berechnungsvorschrift für den Fisher-Score bzw. die Fisher-Diskriminante.

Quelle: Vorlesungsskript Angewandte Neuroinformatik

Stand: Sommersemester 2015

https://www.tu-ilmenau.de/neurob/teaching/ss/ani/ani-ss-18/

bedeuten. Liegt eine gute Trennbarkeit vor, also ist die Interklassenvarianz ausreichend hoch und die Intraklassenvarianz ausreichend gering, lassen sich die gegebenen Daten durch eine lineare Klassifikation gut unterscheiden.

Ziel ist es, einen möglichst hohen Fisher Score und damit gute Trennbarkeit zu erreichen. Der Fisher Score berechnet sich dabei als Quotient aus der Interklassenvarianz und der Summe der Intraklassenvarianz, wie in Abbildung 2.14 beschrieben.

2.5 Feature-Attention bzw. Verdeckungsmethode

Um einen tieferen Einblick in die vom Neuronalen Netz gelernten Features zu erhalten, wird eine Feature-Attention-Analyse, wie in [Zeiler and Fergus, 2014], verwendet. Diese Analyse liefert einen qualitativen Eindruck davon, auf welche Strukturen des Inputs die Features empfindlich reagieren.

Der Grundgedanke für diesen Ansatz ist ein sehr simpler und entspricht einer Art

On-Off-Vergleich. Zunächst wird der Input unverändert von dem Neuronalen Netz verarbeitet und das Ergebnis, also die Featuremaps, einer gewünschten Schicht als Referenz abgegriffen. Anschließend wird der gleiche Input dem Neuronalen Netz erneut präsentiert, jedoch ein Teil des Inputs mit dem Mittelwert überschrieben, also von einer z.B. grauen Fläche abgedeckt.

Die Featuremaps werden erneut an der gleichen Schicht entnommen. Im Sinne eines On-Off-Vergleichs ergibt sich dadurch eine On-Featuremap, welche ohne Verdeckung erstellt wurde, und eine Off-Featuremap mit Abdeckung. Die jeweiligen On- und Off-Featuremaps werden nun subtrahiert und es resultiert als Differenz eine Aus-

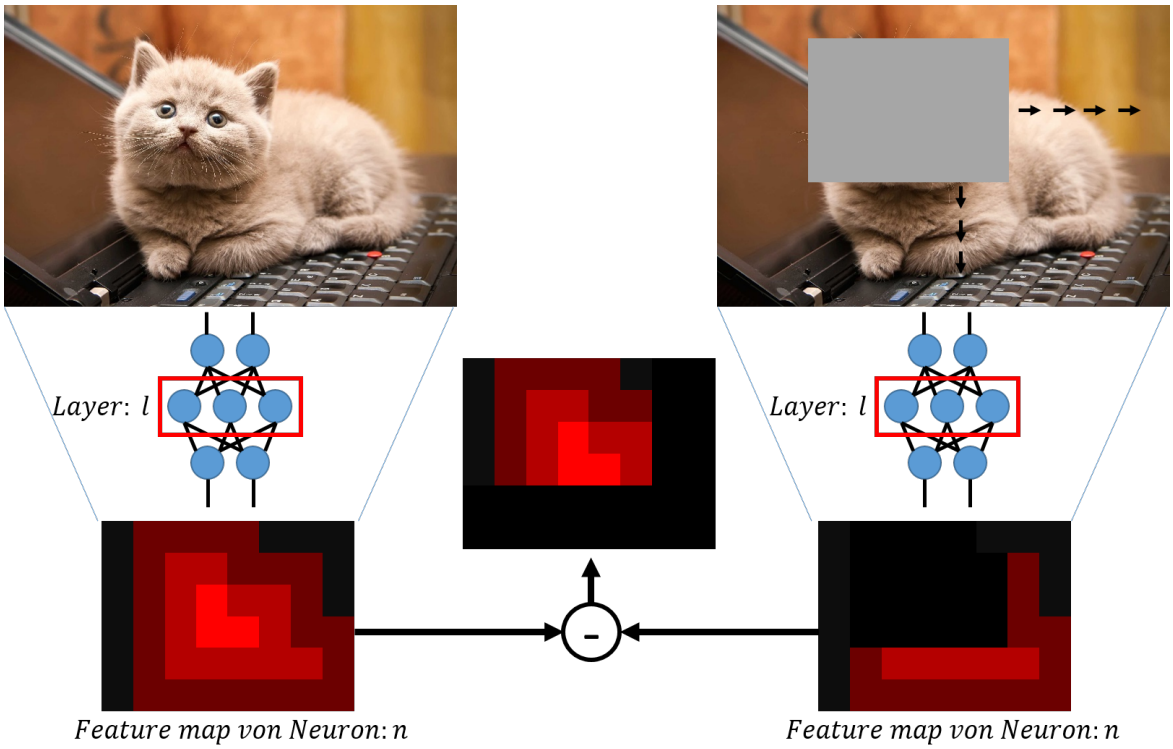


Abbildung 2.15: Feature-Attention-Methode

Zu sehen ist jeweils im oberen Teil der Input für ein gegebenes Faltungsnetz und im unteren Teil die potentielle Ausgabe, also eine Featuremap, eines spezifischen Neurons einer Faltungsschicht. Subtrahiert man beide Resultate voneinander, entspricht die resultierende Featuremap sozusagen den Kosten, welche dem Neuron dadurch entstehen, dass ein gewisser Bildausschnitt verdeckt ist. Umso höher die jeweiligen Kosten, umso stärker reagiert das Neuron auf den Inhalt, welcher verdeckt wurde.

sage darüber, wie stark die Abdeckung die Aktivierung auf der Featuremap reduziert hat. Daraus lässt sich folgern, dass falls die Differenz besonders groß ist, sich das jeweilige Neuron, auf dessen Featuremap sich die hohe Differenz ergibt, sehr für den abgedeckten Bereich "interessiert". Abbildung 2.15 zeigt dazu ein Beispiel, wie solch ein On-Off-Vergleich einer Featuremap aussehen könnte.
Wie bereits erwähnt, sollte die Differenz auf einer Featuremap besonders hoch sein und ist beispielsweise, wie in Abbildung 2.15, zur Zeit eine Katze abgedeckt, so ist davon auszugehen, dass das entsprechende Neuron insbesondere auf katzenförmige Strukturen reagiert. Da die ausgeprägten Features jedoch auch auf sehr feine Strukturen reagieren können, ist dies nicht automatisch als Beweis zu werten, dass das entsprechende Neuron ausschließlich für Katzen zuständig ist. Ein solcher Zusammenhang lässt sich lediglich durch eine Untersuchung entsprechend vieler verschiedener Inputbilder unterstützen oder ggf. auch ablehnen. An dieser Stelle sei betont, dass es sich bei diesem Verfahren lediglich um eine qualitative Betrachtung handelt. Abbildung 2.16 zeigt ein weiteres Beispiel, das dem Vorgehen in [Zeiler and Fergus, 2014] entspricht.

2.6 Fazit

Um einen tieferen Einblick in den Nutzen von Transfer Learning in Bezug auf eine bestimmte Anwendung zu erhalten, können verschiedene Varianten des Transfer Learnings verglichen werden. Dies umfasst eine Variation der Anzahl der Schichten, die innerhalb des Finetunings gelernt werden sowie eine Variation der Lernrate. Außerdem kann untersucht werden, ob die Architektur einen Einfluss auf die Effektivität des Transfer Learnings besitzt.
Des Weiteren stellt sich die Frage, welche Auswirkungen das Finetuning auf das bereits Gelernte hat. In [Kim et al., 2017] werden verschiedene Effekte beschrieben, welche auch innerhalb dieser Arbeit untersucht werden sollen. Dazu werden Verfahren wie t-SNE (s. 2.4.2) oder auch eine Feature-Attention-Analyse (s. 2.5) verwendet.

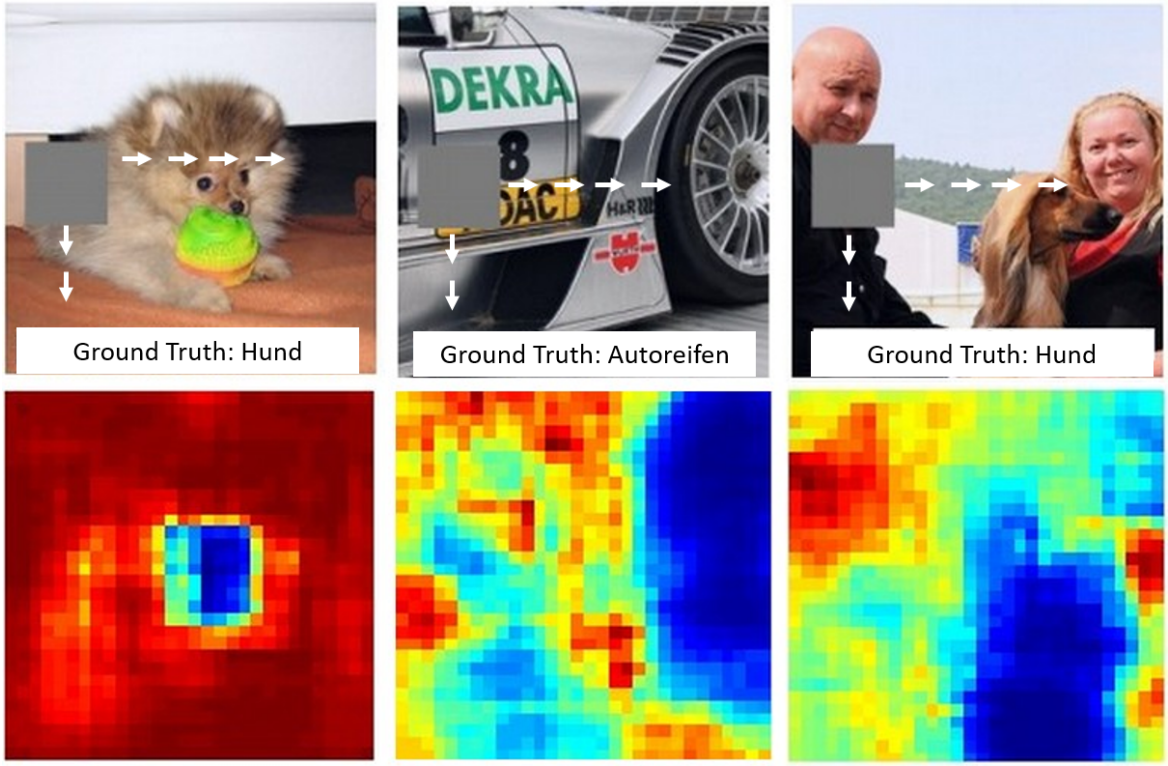


Abbildung 2.16: Beispiele der Feature-Attention-Methode

Zu sehen sind im oberen Teil verschiedene Inputbilder, über welche pixelweise eine Maske geschoben wird. Im unteren Teil befinden sich Visualisierungen jeweils eines spezifischen Neurons einer Schicht. Ein Pixel einer solchen Visualisierung entspricht dabei den Kosten, welche dem Neuron für die Verdeckung des Inputs an einer bestimmten Position entstehen. Der untere Teil zeigt dementsprechend keine Featuremaps, sondern eine Kostenkarte, die zeigt, welche Region des Inputs für das Neuron besonders "interessant" ist. Blau symbolisiert dabei hohe Kosten, also eine hohe Aktivierung für das Neuron, falls der Bereich nicht verdeckt ist, und rot geringe Kosten.

State-of-the-Art 3

Schon zu Beginn des Erfolges tiefer Faltungsnetze (Deep Convolutional Neural Nets - DCNNs) in der Bildverarbeitung wurde gezeigt, dass gelernte Features eines Netzes auch auf andere Aufgaben übertragbar sind. Um die Eigenschaften einer solchen Übertragung näher kennenzulernen, wird in Abschnitt 3.1 zunächst eine Arbeit zum Grundgedanken der Übertragung von Gewichten ohne ein Finetuning präsentiert. In Abschnitt 3.2 wird darauf aufbauend eine Arbeit betrachtet, in der untersucht wurde, wodurch die Übertragbarkeit von Gewichten bzw. Features begünstigt wird. Abschnitt 3.5 betrachtet anschließend verschiedene Arbeiten, die Transfer Learning auf die Detektion von Straßenschäden angewendet haben.
Des Weiteren wird in Abschnitt 3.3 eine Arbeit präsentiert, in der die gelernten Features vor und nach dem Finetuning analysiert werden. Abschließend werden in Abschnitt 3.6 weitere mögliche Anwendungsfälle abseits der Bildklassifikation in aller Kürze betrachtet.

3.1 Generelles Transfer Learning

Vor der Zeit der tiefen Neuronalen Netze wurden Features, wie z.B. das HOG-Feature in [Dalal and Triggs, 2005], meist händisch erstellt. Dieser Prozess zur Ermittlung eines guten Features für ein Klassifikationsproblem stellte nicht selten eine große Hürde dar. Insbesondere auch, da die Qualität des Features für die Leistung und Generalisierungsfähigkeit des Klassifikators entscheidend ist. Mit dem Aufkommen von tiefen Neuronalen Netzen und dem selbstständigen Lernen der Features konnte in [Krizhevs-

ky et al., 2012] ein Leistungssprung bei der Bildklassifikation beobachtet werden. Aufbauend auf den Erfolgen des Deep Learnings stellte sich wenig später die Frage, wie allgemeingültig bzw. allgemein anwendbar die selbstständig gelernten Features sind. Dazu wurde in [Razavian et al., 2014] ein Transfer Learning ohne nachfolgendes Finetuning durchgeführt. Es konnte gezeigt werden, dass ein Neuronales Netz, welches auf dem ImageNet-Datensatz trainiert wurde, auch in der Lage ist, andere Klassifikationsaufgaben mit bemerkenswerter Leistung zu bewältigen. Selbst ohne jedes Finetuning des Netzes konnten andere Verfahren, welche händisch erstellte Features besitzen, direkt übertroffen werden. Ähnliches wurde ebenfalls in [Donahue et al., 2014], [Zhang et al., 2018] und [Gopalakrishnan et al., 2017] beobachtet. Nichtsdestotrotz wurde in diesen Arbeiten auch festgestellt, dass ein Transfer Learning mit selbst nur wenigen Epochen Finetuning die Leistung eines Transfer Learnings ohne Finetuning meist deutlich übertrifft.
Grundvoraussetzung für den erfolgreichen Einsatz der Features auf neuen Bilddaten ist die Verwendung eines besonders großen und vielfältigen Datensatzes im Pretraining. Um eine gute Leistung zu erreichen, sollte die ursprüngliche Aufgabe im Pretraining der Zielaufgabe darüber hinaus möglichst ähnlich sein. Abschnitt 3.2 geht auf diesen Aspekt näher ein.

3.2 Übertragbarkeit von Gewichten bzw. Features

Um die beeindruckenden Eigenschaften von Neuronalen Netzen zu erklären, wurde in [Yosinski et al., 2014] genauer untersucht, wie gut gelernte Features eines tiefen Faltungsnetzes auf neue Aufgaben übertragen werden können. Dazu wurden bereits gelernte Features auf Zielaufgaben angewendet, die verschieden starke Ähnlichkeit mit der ursprünglichen Aufgabe besaßen. Dabei wurde festgestellt, dass insbesondere die ersten Schichten von Faltungsnetzen auch bei unterschiedlichen Aufgaben fast identische Filter ausprägen. Diese sind weitgehend unabhängig von den trainierten Daten und reagieren auf einfache Strukturen wie Ecken, Kreise oder Kanten. Ähnliches wurde ebenfalls in [Kim et al., 2017] und [Zeiler and Fergus, 2014] berichtet. Wie in Abschnitt 4.3.4 beschrieben wird, handelt es sich bei dem Zieldatensatz in dieser

Arbeit ebenfalls um einen, der sich stark vom Ursprungsdatensatz unterscheidet. Daher wird in dieser Arbeit überprüft, wie stark die im Pretraining gelernten Features durch das Finetuning verändert werden. Abschnitt 4.7.2 geht darauf näher ein.
Wie weiterhin in [Yosinski et al., 2014] und [Zeiler and Fergus, 2014] festgestellt wurde, sind die gelernten Features umso spezifischer, je tiefer die betrachtete Schicht in einem Netz liegt. Darüber hinaus wurde in [Yosinski et al., 2014] und [Donahue et al., 2014] beobachtet, dass die Übertragbarkeit gelernter Features von der Ähnlichkeit der ursprünglichen Aufgabe des Netzes zu der neuen Aufgabe abhängt. Innerhalb von [Yosinski et al., 2014] wurde beispielsweise die Klassifikation verschiedener Tiere als eher ähnlich bezeichnet, sowie die Unterscheidung zwischen Tieren und Wohnungseinrichtung als sehr unähnlich. Abbildung 3.1 zeigt dazu eine Visualisierung der Performance des Transfer Learnings bei ähnlichen und unähnlichen Aufgaben im Vergleich vor und nach dem Finetuning. Es ist zu erkennen, dass bei unähnlichen Aufgaben das Finetuning geringere Vorteile bringt, jedoch immer noch hilfreich ist. Darüber hinaus ist in Abbildung 3.1 sichtbar, dass ein Netz davon profitiert, dass möglichst viele Schichten während des Transfer Learnings übernommen werden. Im Unterschied zu [Yosinski et al., 2014] werden in dieser Arbeit stets alle Faltungsschichten direkt übernommen. Jedoch wird, wie in Abschnitt 4.7.5 beschrieben, auch untersucht, ob es möglich ist, die Gewichte der vollverschalteten Schichten zu übernehmen. Die Einteilung in ähnliche und unähnliche Aufgaben bzw. Daten ist in [Yosinski et al., 2014] rein qualitativ vorgenommen worden. In dieser Masterarbeit wird darüber hinaus, wie in Abschnitt 5.3.1 beschrieben, diese Aussage mit Ergebnissen untermauert.
Weiterhin hat sich in [Yosinski et al., 2014] ergeben, dass selbst bei sehr unterschiedlichen Aufgaben Transfer Learning die Leistung im Vergleich zu völlig zufällig initialisierten Gewichten verbessert und die Generalisierungsfähigkeit steigern kann. Auf Basis dieser Erkenntnisse wird in [Yosinski et al., 2014] abschließend empfohlen, dass ein Transfer Learning, falls möglich, immer zufällig initialisierten Gewichten vorzuziehen ist. Um diese pauschale Empfehlung zu überprüfen, wird in dieser Arbeit ebenfalls ein Training mit zufällig initialisierten Gewichten als Baseline durchgeführt.
Die Abhängigkeit zwischen Übertragbarkeit der Features und der Ähnlichkeit

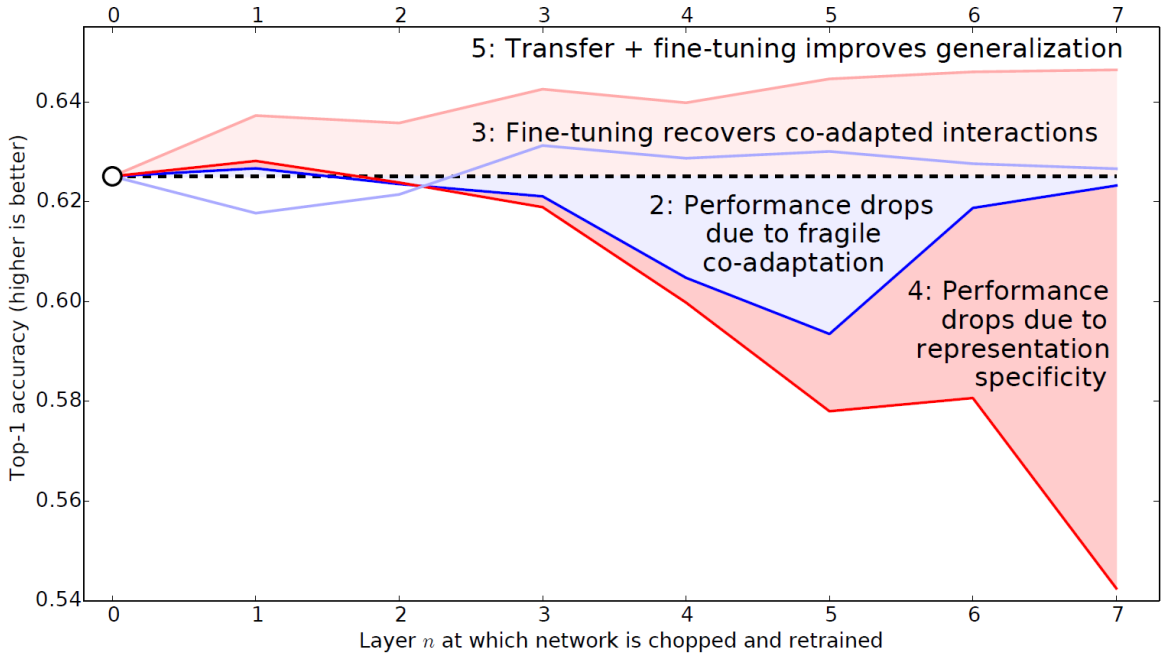


Abbildung 3.1: Visualisierung der Performance des Transfer Learnings bei ähnlichen und unähnlichen Aufgaben im Vergleich vor und nach dem Finetuning
Zu sehen ist eine Visualisierung der Genauigkeit nach dem Transfer Learning. Die blaue Kurve repräsentiert dabei ein Netz, dessen Zielklassifikationsaufgabe sich stark von der Ursprungsaufgabe unterscheidet. Die rote Kurve repräsentiert ein Netz, dessen Ziel- und Ursprungsaufgaben sehr ähnlich sind. Aufgetragen wird die Accuracy vor (dunkle Kurven) und nach dem Finetuninig (helle Kurven). Weiterhin ist zu beachten, dass die Accuracy über der Nummer der Schicht, bis zu der die Gewichte übernommen wurden, aufgetragen wird. Ein Punkt bei Schicht Drei bedeutet, dass von dem ursprünglichen Netz die Gewichte bis zur dritten Faltungsschicht übernommen wurden und der Rest zufällig initialisiert wurde. Dementsprechend ist ein Punkt bei Null eine komplette Zufallsinitialisierung. Weiterhin ist zu erwähnen, dass abseits des Finetunings die zufällig initialisierten Gewichte in allen Fällen trainiert wurden.

zwischen Ursprungs- und Zielaufgabe konnte abseits der Bildverarbeitung auch für Aufgaben des Natural Language Processings (NLP) beobachtet werden. In [Mou et al., 2016] wurde dazu die Übertragbarkeit von NLP-Features untersucht. Innerhalb von zwei Experimenten mit unterschiedlichen Zieldatensätzen konnte ein weitgehend analoges Verhalten zu der Übertragung von Bild-Features festgestellt werden. Die

NLP-Features sind demnach ebenfalls davon abhängig, wie ähnlich Ursprungs- und Zielaufgabe sind. Unterscheiden sich diese in einem semantischen Sinn sehr stark, so fällt die Performancesteigerung eher gering aus. Weiterhin wurde in [Mou et al., 2016] festgestellt, dass es sinnvoll ist, keine Gewichte während des Finetunings einzufrieren, sondern das ganze Netz lernen zu lassen. Um zu überprüfen, ob es pauschal richtig ist, stets alle Schichten eines Netzes während des Finetunings lernen zu lassen, werden in dieser Arbeit eine größere Menge an Trainings durchgeführt. Innerhalb dieser Trainings werden, wie in Abschnitt 4.5.3 beschrieben, unterschiedlich viele Schichten der Netze eingefroren.

3.3 Analyse der Features vor- und nach dem Finetuning

Um die Eigenschaften des Transfer Learnings vor und nach dem Finetuning besser zu verstehen, wurden diese in [Kim et al., 2017] tiefergehend analysiert. Dazu wurde ein Convolutional Neural Net (CNN), welches auf ImageNet-Daten gelernt wurde, auf eine optische Materialprüfung angewendet bzw. optimiert. Verwendet wurde dazu ein Netz der VGG16-Architektur mit circa 138 Millionen Gewichten und lediglich 6900 Bildbeispielen innerhalb des Trainingsdatensatzes. Aus der Kombination einer großen Anzahl von Gewichten und eines relativ kleinen Zieldatensatzes resultiert die Gefahr, dass das Netz bei einem zufällig initialisierten Training sehr schnell die Daten auswendig lernt und kaum generalisiert. Um diesem Problem zu begegnen, soll Transfer Learning zur Leistungssteigerung und Verbesserung der Generalisierungsfähigkeit eingesetzt werden. Abseits der Materialprüfung ist das Problem eines zu kleinen Zieldatensatzes in vielen Fachbereichen sehr akut. Insbesondere im medizinischen Bereich ist es oft unmöglich, ein CNN mit zufällig initialisierten Gewichten zu trainieren. Der Einsatz von Transfer Learning hat sich dadurch zu einer Art Standardvorgehen entwickelt. In [Cheplygina et al., 2018], einer Übersichtsstudie, werden mehrere Dutzend Arbeiten aufgelistet, welche Transfer Learning einsetzen, um dem Problem eines zu kleinen Datensatzes zu begegnen. Die Zielaufgaben sind alle in einem medizinischen Kontext angesiedelt.

In [Kim et al., 2017] wurde darüber hinaus untersucht, wie ein Transfer Learning mit und ohne Finetuning abschneidet. Ohne konnte das CNN auf der Zielaufgabe eine Genauigkeit von ca. 79% erreichen und selbst mit lediglich einer Epoche Finetuning wurden 99% Genauigkeit erreicht. Eine ähnliche Steigerung der Performance, mit vergleichsweise wenigen Trainingsepochen, konnte ebenfalls in [Zhang et al., 2018] beobachtet werden und wird gleichermaßen in dieser Arbeit erwartet.
Ein interessanter Effekt, der in [Kim et al., 2017] beobachtet wurde, ist, dass durch dieses geringe Finetuning das CNN in erster Linie die hinteren Schichten angepasst hat. In Abbildung 3.2 ist gut zu erkennen, dass sich die Gewichte durch das Finetuning in den vorderen Schichten prozentual nur wenig ändern. In den hinteren Schichten tritt jedoch eine deutlich größere Änderung auf. Wie bereits in Abschnitt 3.2 erwähnt, werden in den vorderen Schichten eines CNNs Features ausgeprägt, welche sehr allgemeingültig sind und dementsprechend auch bei einer Übertragung kaum Anpassung

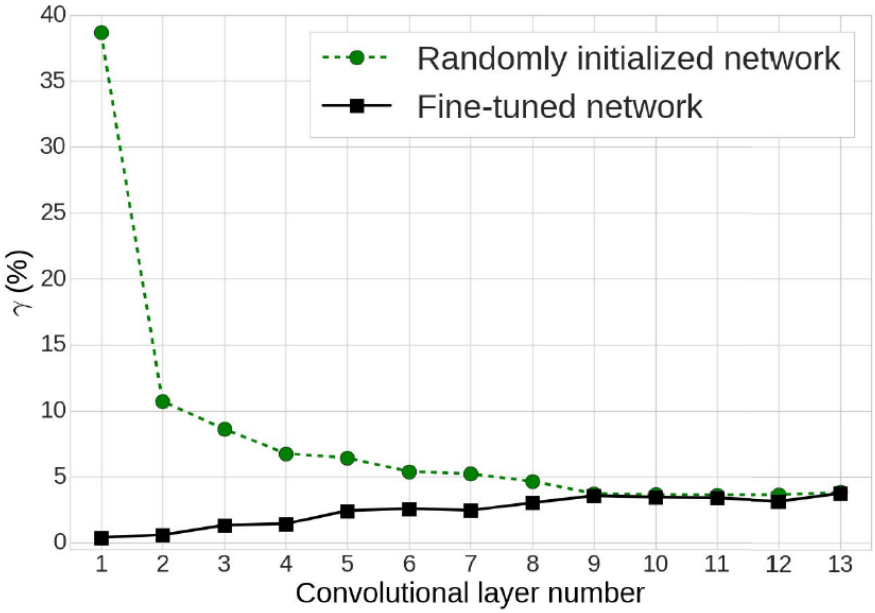


Abbildung 3.2: Absolute Gewichtsdifferenz vor und nach Finetuning
Zu sehen ist die l2-Norm der Gewichte des Netzes vom Zustand vor dem Finetuning zu dem danach (schwarz) und die Gewichte eines zufällig initialisierten Netzes zu dem Zustand nach einem Training (grün).

benötigen. Das beobachtete Verhalten ist dementsprechend naheliegend. Um über die Erkenntnisse aus [Kim et al., 2017] hinaus zu gehen, wird in dieser Arbeit untersucht, ob es sich dabei um ein allgemeingültiges Verhalten handelt oder um ein für die VGG-Architektur spezifisches. Dazu wird, wie in Abschnitt 4.7.2 beschrieben, ein ähnliches Experiment durchgeführt.

Weiterhin konnte in [Kim et al., 2017] beobachtet werden, dass das geringe Finetuning zu einer Featureselektion führt. Abbildung 3.3 zeigt dazu ein Testsample, welches zum einen von dem CNN ohne Finetuning und zum anderen von dem CNN mit Finetuning verarbeitet wurde. Visualisiert wird dabei jeweils die gleiche Schicht der CNNs. Wie gut zu erkennen ist, verwendet das CNN nach dem Finetuning deutlich weniger Features um das Bild zu beschreiben, weshalb der Großteil der Aktivierungen der Featuremaps nahe Null (schwarz) ist. Andere Arbeiten, in denen Ähnliches beobachtet werden konnte, wurden in [Kim et al., 2017] jedoch nicht genannt. Es ist daher ungewiss, ob es sich dabei lediglich um ein Verhalten handelt, welches aus den Daten resultiert. Um das zu überprüfen, wird in dieser Arbeit, wie in Abschnitt 4.7.4 beschrieben, ein analoges Experiment durchgeführt. Dabei gilt es zu evaluieren, ob das

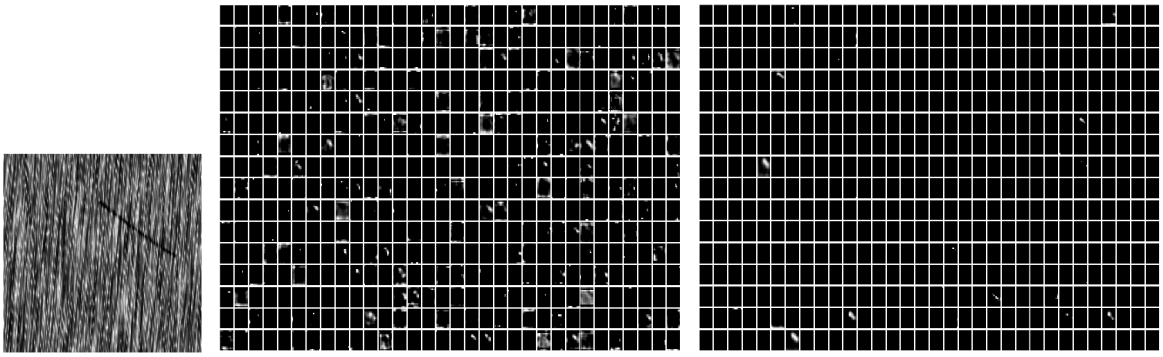


Abbildung 3.3: Vergleich der Aktivierung zweier Faltungsschichten
Zu sehen ist links ein Bildbeispiel, welches vor und nach dem Finetuning des CNNs verarbeitet wurde. Visualisiert wurde die Aktivierung der letztes Faltungsschicht des Netzes vor dem Fintuning (Mitte) und nach dem Finetuning (rechts). Es ist leicht zu erkennen, dass nach dem Finetuning eine deutlich geringere Aktivierung vorliegt (mehr schwarz). Viele Featuremaps sind sozusagen tot, wodurch das Finetuning einer Featureselektion entspricht.

gleiche Verhalten zu finden ist und ob es für die VGG-Architektur oder bestimmte Bildbeispiele spezifisch ist.

3.4 Zweistufiges Transfer Learning

In [Ng et al., 2015] wird ein Neuronales Netz darauf trainiert, Emotionen anhand eines Bildausschnittes des Gesichts zu bestimmen. Es wurde ein Transfer Learning verwendet, wodurch die Accuracy um ca. 16% im Vergleich zur verwendeten Baseline gesteigert werden konnte. Als Besonderheit ist zu erwähnen, dass das Finetuning in zwei Schritten durchgeführt wurde. Insgesamt stehen innerhalb der Arbeit drei verschiedene Zieldatensätze und der ImageNet-Datensatz für das Pretraining zur Verfügung. Da es lediglich das Ziel ist, auf einem der drei Zieldatensätze eine gute Leistung zu erreichen, ergeben sich verschiedene mögliche Kombinationen, um die anderen für das Finetuning zu verwenden.
Die drei Zieldatensätze (EmotiW [Dhall et al., 2016], FER28 [Langner et al., 2010], FER32 [Langner et al., 2010]) wurden, wie in Abb. 3.4 zu sehen, zu verschiedenen Zieldatensätzen zusammengefasst und in einem zweiten Finetuningschritt auf den eigentlichen Zieldatensatz EmotiW final optimiert. Innerhalb der Arbeit wird auch untersucht, wie sich ein Finetuning mit lediglich einem Schritt, also direkt auf dem eigentlichen Zieldatensatz, verhält. Zwar konnten die Netze, welche ein Finetuning in zwei Schritten durchlaufen haben, nicht konstant besser sein als die Netze mit einem Schritt, allerdings zeichnet sich eine Tendenz zugunsten des zweistufigen Finetunings ab.
Die Möglichkeit eines zweistufigen Finetunings besteht ebenfalls für den Zieldatensatz dieser Arbeit. Dieser unterteilt sich in einen großen und einen kleinen Datensatz, wobei es sich bei dem kleineren um eine Teilmenge des großen handelt. Um die Trainingsdauer zu verringern und die Leistung zu verbessern, könnte der kleinere Datensatz für ein zweistufiges Pretraining genutzt werden. Da vorhergehende Experimente bereits einen enormen Rechenaufwand verursacht haben, ist die Durchführung aus zeitlichen Gründen jedoch nicht möglich gewesen.

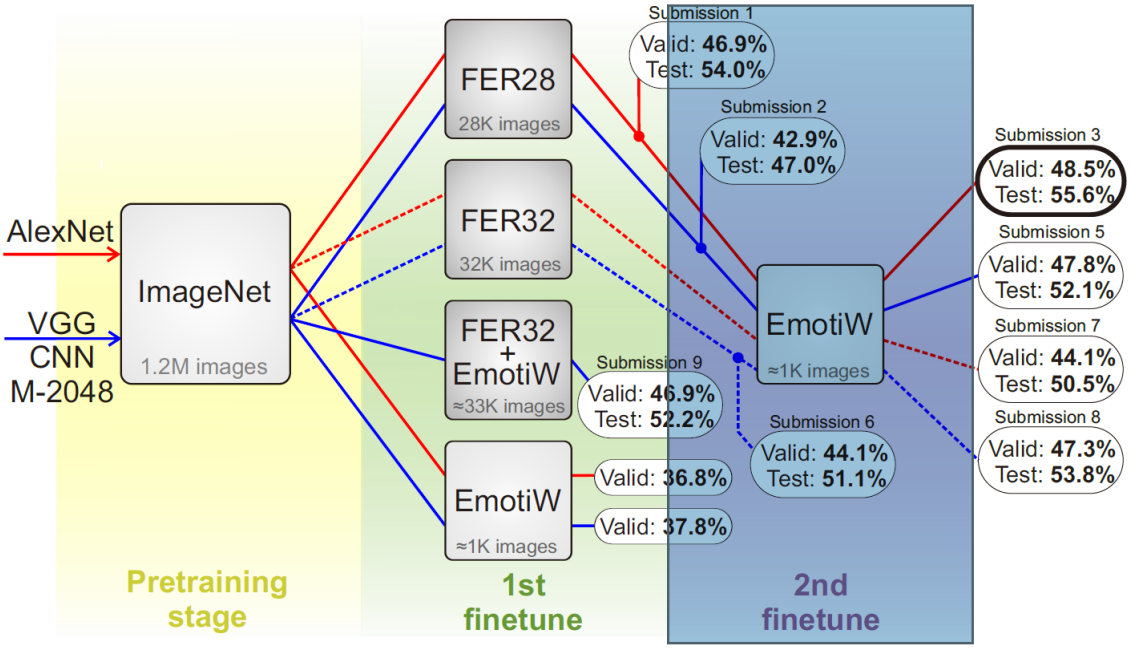


Abbildung 3.4: Zweistufiges Transfer Learning

Zu sehen ist eine Abbildung verschiedener Kombinationen eines zweistufigen Transfer Learnings. Es wurden ein VGG (blau) und ein AlexNet (rot) mit verschiedenen Daten gelernt und für die Teilnahme an dem "Emotion Recognition in the Wild" Wettbewerb von 2015 eingereicht. Für jede Kombination ist die Leistung auf dem Validierungsdatensatz und ggf. dem unbekannten Testdatensatz der Einreichung angegeben. Es ist gut zu erkennen, dass die Trainings mit mehr Daten, also mehreren Datensätzen, meistens besser abschneiden konnten, als das Training mit lediglich einem Datensatz.

3.5 Transfer Learning für die Erkennung von Straßenschäden

In [Gopalakrishnan et al., 2017] wird, dieser Arbeit sehr ähnelnd, ein tiefes Faltungsnetz der VGG16-Architektur verwendet, um Schäden auf Bildern des Straßenbelages zu erkennen. Anders als in dieser Masterarbeit wird in [Gopalakrishnan et al., 2017] jedoch lediglich eine Architektur untersucht und der Fokus liegt auf dem Vergleich unterschiedlicher Klassifikationsverfahren.

Dazu wurde, ähnlich wie in [Razavian et al., 2014], ein auf ImageNet-Daten vortrainiertes Netz ohne Finetuning als Feature-Generator verwendet. Die ermittelten Features für die Straßendaten werden anschließend verwendet, um verschiedene Klassifikatoren, wie unter anderem eine Support-Vector-Machine und ein einschichtiges Neuronales Netz, zu lernen. Der Ansatz mit einem einschichtigen Neuronalen Netz ist in ähnlicher Form ebenfalls in dieser Arbeit wiederzufinden. Im Gegensatz zu [Gopalakrishnan et al., 2017] wird in dieser Arbeit jedoch eine VGG19-Architektur und ein dreischichtiger Klassifikator verwendet. Darüber hinaus wird außerdem mit einem Finetuning gearbeitet. Nichtsdestotrotz konnten in [Gopalakrishnan et al., 2017], wie auch in [Razavian et al., 2014], ohne ein Finetuning bereits bemerkenswerte Leistungen erzielt werden.
Eine weitere Arbeit, in der Transfer Learning für die Erkennung von Straßenschäden eingesetzt wurde, ist [Zhang et al., 2018]. Dort wurde per Transfer Learning eine am AlexNet angelehnte Architektur trainiert. Im Gegensatz zu dem Vorgehen in dieser Masterarbeit wird in [Zhang et al., 2018] jedoch nach dem Neuronalen Netz eine weitere Verarbeitung vorgenommen, um vorhandene Risse exakt zu segmentieren. Nichtsdestotrotz konnte die Accuracy in [Zhang et al., 2018] durch ein Finetuning von 89% auf 94% gesteigert werden. Ähnliche Vorgehensweisen zur Detektion von Schäden auf Beton- oder Asphaltoberflächen per Transfer Learning sind in [Gopalakrishnan et al., 2018], einer Fortsetzung von [Gopalakrishnan et al., 2017], [Kim et al., 2018] und [An et al., 2018] umgesetzt.

3.6 Beispiele weiterer Anwendungen von TL

Wie in den Abschnitten 3.1 bis 3.3 bereits festgestellt wurde, ist es allgemein anerkannt, dass ein Transfer Learning uneingeschränkt sinnvoll ist. Voraussetzung ist jedoch, dass zu der verwendeten Architektur auch bereits ein vortrainiertes Netz existiert. Darüber hinaus kann es auch sinnvoll sein, erst gezielt ein Pretraining durchzuführen, um die gelernten Gewichte für ein Transfer Learning zu verwenden. In [Ahmed et al., 2015] wird vor dem eigentlichen Training ein gezieltes Pretraining eingesetzt. Gegenstand von [Ahmed et al., 2015] ist das Training eines Neuronalen

Netzes zur Wiedererkennung von Personen. Das Netz erhält dazu je ein Bild zweier Personen und soll auf Basis dieser entscheiden, ob es sich um die gleiche Person handelt. Einer der eingesetzten Datensätze enthält dabei nur eine kleine Menge von 1940 positiven Bildbeispielen. Durch ein Pretraining konnte, wie auch schon in den zuvor genannten Transfer-Learning-Arbeiten, Overfitting deutlich reduziert und eine bessere Generalisierung erreicht werden. Eine ähnliche Steigerung der Generalisierungsfähigkeit wird in dieser Arbeit ebenfalls durch das Transfer Learning erhofft.

Transfer Learning eignet sich jedoch nicht nur zur reinen Steigerung der Leistung eines Neuronalen Netzes. Anders als in dieser Arbeit gibt es auch viele weitere Einsatzmöglichkeiten, die von der Übertragung zuvor gelernter Gewichte profitieren. Beispielsweise wurde in [Shi et al., 2017] Transfer Learning verwendet, um die Zuverlässigkeit eines Proposal Generators zu verbessern. Der Proposal Generator hat das Ziel, aus einem Bild bestimmte Regionen auszuwählen, welche möglicherweise Objekte enthalten. Solche Techniken kommen zum Einsatz, falls innerhalb eines Trainingsdatensatzes nur bekannt ist, welche Objekte auf dem Bild enthalten sind und nicht, wo diese sich befinden - sogenannte schwach gelabelte Bilddaten (bzw. ”weakly labeled”). Ein vortrainiertes Netz wird dabei durch ein Finetuning angepasst, sodass dieses die Bildkandidaten des Generators gewichten kann. Darüber hinaus gibt es noch viele weitere Beispiele, wie Transfer Learning eingesetzt werden kann.

3.7 Fazit

Unabhängig von den konkreten Datensätzen zeichnet sich in allen Arbeiten ab, dass ein Transfer Learning generell immer sinnvoll ist. Die einzige Voraussetzung dabei ist die Notwendigkeit eines vortrainierten Netzes in der Architektur, welche auch für die Zielaufgabe verwendet werden soll. Selbst wenn kein Netz mit der entsprechenden Architektur existiert, kann ein Pretraining sinnvoll sein. Dies ist in der Regel dann der Fall, wenn der Zieldatensatz zu klein ist, um ein Training mit zufällig initialisierten Gewichten ohne starkes Overfitting durchzuführen. Abschließend lässt sich sagen, dass falls ein vortrainiertes Netz in der benötigten Architektur vorliegt, Transfer Learning nur Vorteile bietet und grundsätzlich in Erwägung gezogen werden sollte. Es ermöglichen sich sowohl Leistungssteigerungen der Klassifikation als auch eine Verbesserung der Generalisierungsfähigkeit, bei gleichzeitig wenigen bekannten Nachteilen.

4 Transfer Learning für die Erkennung von Straßenschäden

Wie bereits in Kapitel 3 beschrieben, besitzt Transfer Learning insbesondere dann Vorteile, wenn vergleichsweise wenig Bildbeispiele für ein Klassifikationsproblem vorliegen. Der German Asphalt Pavement Distress Datensatz (GAPs, s. Abschnitt 4.3), der in dieser Arbeit als Zieldatensatz verwendet wird, ist mit seinen 50000 Trainingsbeispielen eben von dieser Problematik betroffen. Nichtsdestotrotz ist es wünschenswert, dass die Leistung des Netzes nicht zu stark unter dem Mangel an Daten leidet. Daher bietet sich eine Untersuchung an, ob Transfer Learning in diesem Fall effektiv eingesetzt werden kann. Für das Pretraining wird der weitverbreitete ImageNet-Datensatz (s. Abschnitt 4.3) verwendet. Die zwei Hauptunterschiede der beiden Datensätze sind zum einen, dass der GAPs-Datensatz aus Schwarzweißbildern, anstatt Farbbildern, besteht, und zum anderen, dass lediglich in zwei Klassen (”Schaden” und ”Kein Schaden”), anstatt in 1000, unterteilt wird. Ziel der Arbeit ist es, die Leistung eines Neuronalen Netzes, welches per Transfer Learning trainiert wurde, im Vergleich zu einem Training ”from scratch” zu verdeutlichen. Erwartet wird dabei eine Leistungssteigerung und ein deutlich schnelleres Konvergieren des Trainings. Darüber hinaus wird erwartet, dass Effekte in Bezug auf die Gewichte des Netzes beobachtet werden können. Abschnitt 4.7 befasst sich umfassend mit den erwarteten Beobachtungen. Abbildung 4.1 skizziert den Ablauf der wichtigsten experimentellen Leistungen dieser Arbeit.

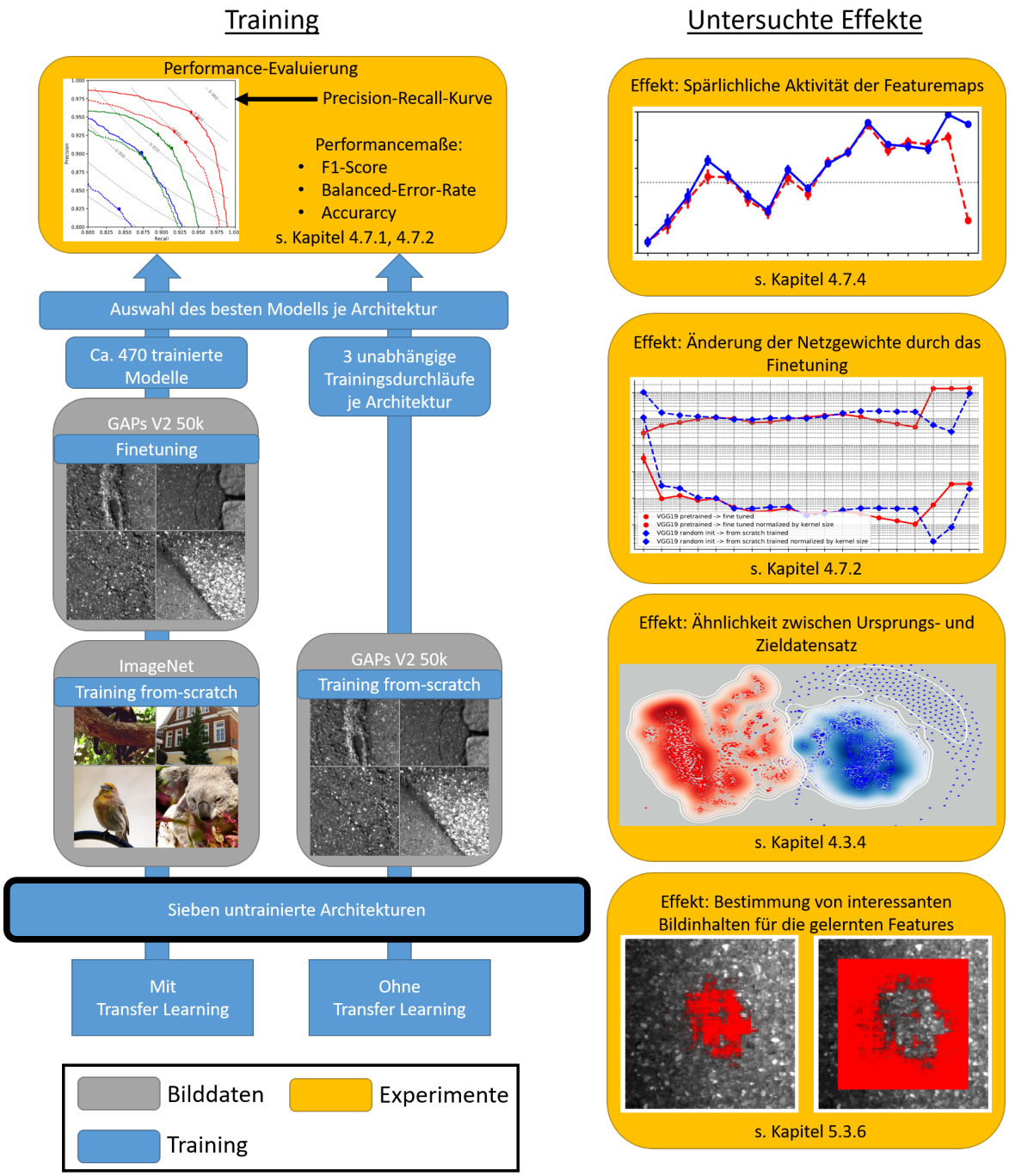


Abbildung 4.1: Überblick über das Vorgehen in dieser Masterarbeit

Zu sehen ist der schematische Ablauf der geplanten Trainings und Experimente dieser Masterarbeit.

4.1 Überblick

Wie in Abschnitt 2.2 bereits beschrieben, besteht die Möglichkeit, das Transfer Learning unterschiedlich zu gestalten. Darauf wird in Abschnitt 4.2 eingegangen. Da das Pretraining der verwendeten Architekturen auf dem ImageNet-Datensatz durchgeführt wurde, besitzen die Netze Gewichte für einen Input mit drei Kanälen. Um die Netze für die Schwarzweißbilder des GAPs-Datensatzes anzupassen, wird wie in Abschnitt 4.4.1 beschrieben vorgegangen. Weitere Abweichungen zu den originalen Netzarchitekturen werden in Abschnitt 4.6 diskutiert. Abschnitt 4.4 geht darauf aufbauend auf die Vor- bzw. Nachverarbeitung der Ein- und Ausgabe der Netze ein. In Abschnitt 4.5 wird auf das detaillierte Trainingsvorgehen sowie Ziele eingegangen und Abschnitt 4.3 gibt einen Einblick in die verwendeten Datensätze und ihre Unterschiede. Zuletzt werden in Abschnitt 4.7 etwaige Effekte besprochen, welche von den Experimenten erwartet werden.

4.2 Transfer Learning für die Erkennung von Straßenschäden

In Bezug auf das Transfer Learning gibt es, wie in Abschnitt 2.2 beschrieben, die Möglichkeit, nicht alle Schichten des vortrainierten Netzes zu übernehmen. Entsprechend den Resultaten in [Yosinski et al., 2014] werden jedoch alle direkt übertragbaren Gewichte, das heißt die aus den Faltungsschichten, immer vollständig übernommen. Etwaige vollverschaltete Schichten werden für das Finetuning zufällig initialisiert. Zwar ist es möglich, vollverschaltete Schichten, welche vor der Softmax-Ausgabe liegen, direkt zu übernehmen, jedoch wurde dies, wie auch in [Yosinski et al., 2014] oder [Kim et al., 2017], nicht durchgeführt. Darüber hinaus wäre dies lediglich für das AlexNet und VGG19 möglich gewesen, da nur diese beiden mehr als eine vollverschaltete Schicht besitzen. Des Weiteren werden viele Resultate dieser Arbeit mit den Ergebnissen aus [Kim et al., 2017] verglichen, wo auch ein VGG19 eingesetzt wird und ebenfalls keine vollverschalteten Schichten übernommen wurden. Zugunsten der Vergleichbarkeit

wurde in dieser Arbeit daher auf die Übernahme verzichtet. Aufgrund der großen Ähnlichkeit der AlexNet-Architektur mit dem VGG19 wurde aus Vergleichbarkeitsgründen bei dem AlexNet ebenfalls darauf verzichtet.
Nichtsdestotrotz bildet Vorversuch 5.1.3 einen Sonderfall. Bei diesem Vorversuch sollte vor dem Finetuning ein Klassifikationsneuron, welches für eine Klassifikation der Straßendaten am besten geeignet ist, ermittelt und ebenfalls übernommen werden. In Vorversuch 5.1.3 war es jedoch nicht möglich, solch ein Neuron zu ermitteln, weshalb es nicht sinnvoll war, die Gewichte vollständig zu übernehmen.
In Abschnitt 2.2 wurde ebenfalls beschrieben, dass für das Finetuning eine reduzierte Lernrate verwendet werden sollte. Da alle Experimente, die ein Finetuning umfassen, jedoch ohnehin mit verschiedenen Lernraten durchgeführt werden, spielt dieser Einfluss keine Rolle.

4.3 Datensätze

4.3.1 ImageNet

Die ImageNet-Datenbank [Deng et al., 2009] ist über die Zeit zu einem der wichtigsten Referenzdatensätze innerhalb der Computer Vision geworden. Es werden circa 14 Mio. Bildbeispiele in über 21000 sogenannte Synsets unterteilt, welche eine große Bandbreite aller erdenklichen Objekte abdecken. Die Kategorisierung der Bildbeispiele geht dabei auf das WordNet [Kilgarriff, 2000] zurück, welches alle Einträge baumförmig strukturiert. Die Trainingsbeispiele, welche für das Pretraining verwendet wurden, sind jedoch nur eine Teilmenge des gesamten Datensatzes. Es handelt sich dabei um den Trainingsdatensatz, welcher für die ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [Russakovsky et al., 2015] verwendet wurde. Dieser besitzt etwa 1,2 Mio. Trainingsbeispiele für 1000 Klassen.

4.3.2 German Asphalt Pavement Distress Dataset - GAPs

Der German Asphalt Pavement Distress (GAPs) Datensatz [Eisenbach et al., 2017] ist der erste frei verfügbare Datensatz für Schäden in Straßenbelägen und wurde 2017 durch das Fachgebiet für Neuroinformatik und kognitive Robotik der Technischen Universität Ilmenau veröffentlicht.
Um den vormals hohen Anteil manueller Arbeit in dem Evaluierungsprozess von Straßenbelägen zu verringern, wurde der GAPs-Datensatz erstellt. Ziel der Erstellung war es, einerseits einen ausreichend großen Datensatz zu realisieren, um selbst sehr tiefe Neuronale Netze effektiv zu trainieren, und andererseits auch eine Vergleichbarkeit zwischen wissenschaftlichen Arbeiten zu ermöglichen. Insbesondere die Vergleichbarkeit verschiedener Arbeiten ermöglicht einen Benchmark für Verfahren aus diesem Fachbereich. Abbildung 4.2 zeigt ein Beispiel des Datensatzes und wozu ein Neuronales Netz auf Basis von diesem verwendet werden könnte. Insgesamt besteht der GAPs-Datensatz aus 1969 Full-HD 8-Bit-Graustufenbildern, dich sich in 1418 Trainingsbilder, 51 Validierungsbilder und 500 Testbilder unterteilen. Die Full-HD Bilder werden für das Training in kleinere Bildausschnitte (Patches) unterteilt, welche der Inputgröße des Neuronalen Netzes entsprechen. Die Trainingsbilder stammen jeweils von zwei verschiedenen Bundesstraßen, welche sich allgemein in einem schlechten Zustand befanden. Die Validierungbilder wurden von den gleichen Bundesstraßen entnommen, jedoch von einem anderen Abschnitt. Für die Testbilder wurde eine dritte Bundesstraße als Quelle verwendet, welche sich allgemein in einem besseren Zustand befand. Alle Bilder wurden im Sommer 2015 an einem warmen, trockenen Tag aufgenommen. Innerhalb dieser Arbeit wird der GAPs 50k in Version 2.0 verwendet, welcher aus einer Teilmenge des gesamten Datensatzes besteht. Abbildung 4.3 gibt eine Übersicht über die Verteilung der Trainingssamples. Zu beachten ist, dass es sich um einen unbalancierten Datensatz handelt, welcher 60% Samples ohne Schaden und 40% Samples mit Schaden enthält. Als Neuerung in der Version 2.0 wurde der Datensatz um weitere 500 Testbilder ergänzt, womit nun 1000 Testbilder zu Verfügung stehen. Wie bereits in Abschnitt 4.5.4 beschrieben, wird eine Vierteilung des Datensatzes verwendet. Analog zu dem Vorgehen in [Ng, 2016] wurde

der Validierungs-Test-Datensatz aus der Menge der Testdaten erstellt.

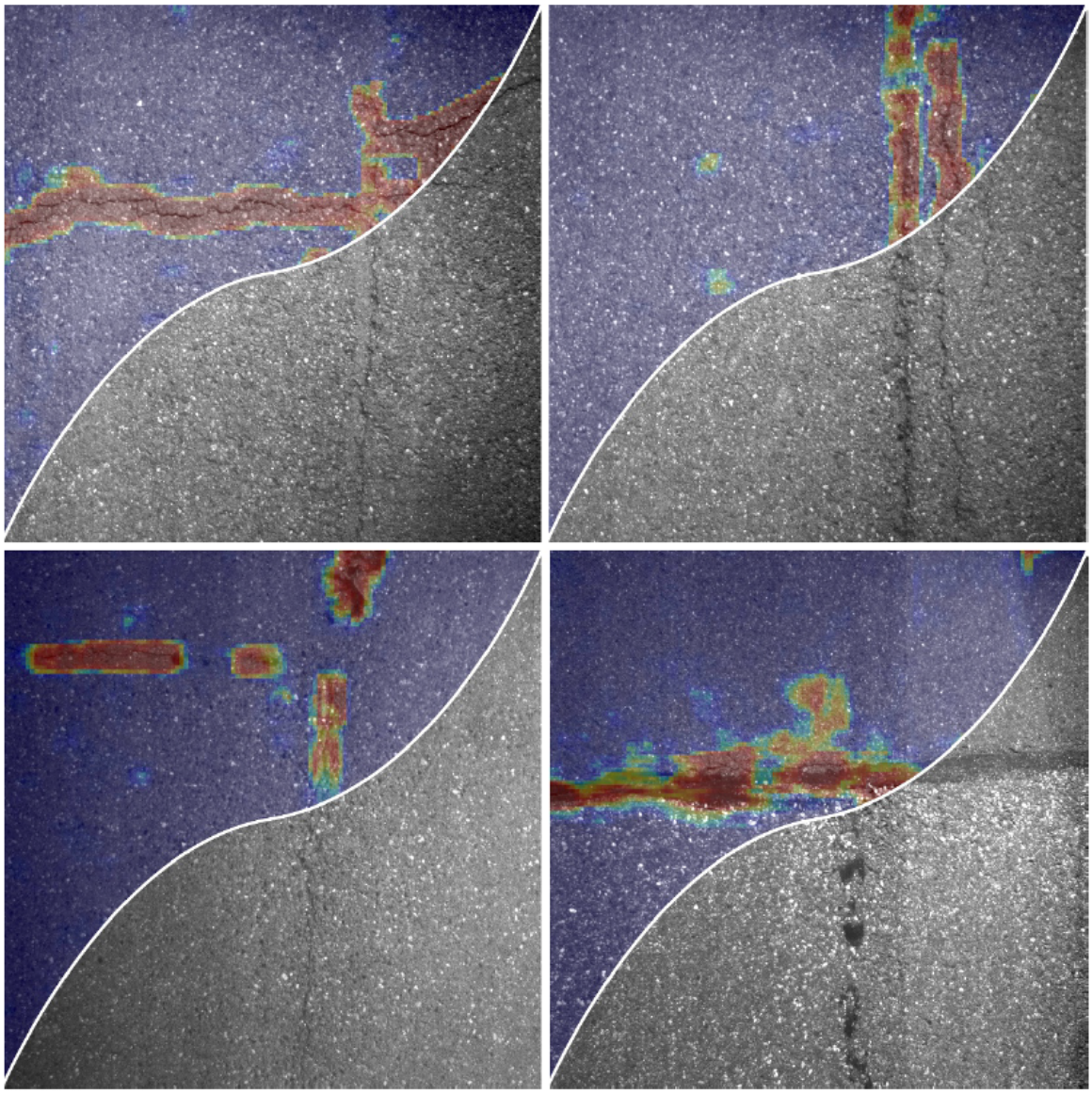


Abbildung 4.2: Bildbeispiele des GAPs-Datensatzes
Zu sehen sind vier Bildbeispiele des GAPs-Datensatzes mit verschieden stark ausgeprägten Straßenschäden. Jedes Beispiel ist überlagert mit der Netzausgabe eines Faltungsnetzes, wie es in [Eisenbach et al., 2017] vorgestellt wurde. Blau symbolisiert dabei eine geringe Wahrscheinlichkeit eines Straßenschadens und rot eine hohe. Bildquelle: [Eisenbach et al., 2017]

4.3.3 Zusammenfassung

Abbildung 4.3 zeigt eine exemplarische Gegenüberstellung der beiden verwendeten Datensätze. Außerdem wird die Anzahl der Samples in den Datensätzen genauer aufgeschlüsselt. Des Weiteren beinhaltet der GAPs-Datensatz, im Gegensatz zum ImageNet-Datensatz, ausschließlich Schwarzweißbilder. Es ist daher notwendig, die Netzeingabe bzw. die Gewichte der Architekturen, wie in Abschnitt 4.4 erläutert, vorzubereiten.

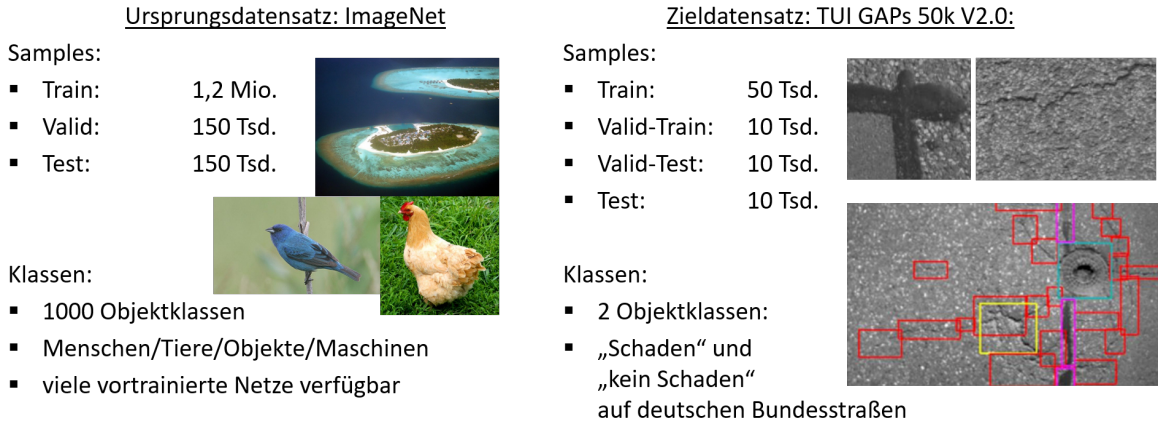


Abbildung 4.3: Überblick über den ImageNet und GAPs-Datensatz
Zu sehen ist ein Überblick über die verwendeten Datensätze. Außerdem werden exemplarisch zufällige Beispiele der Datensätze gezeigt und die Anzahl der Samples aufgeschlüsselt.
Zufällig aus URL-Liste gezogene ImageNet-Bildbeispiele (links):
http://image-net.org/download-imageurls
GAPs-50k-V2-Bildquelle (rechts): [Eisenbach et al., 2017]

4.3.4 Ähnlichkeit der Datensäze

Die Übertragbarkeit der Features ist abhängig von der Ähnlichkeit zwischen Ursprungs- und Zieldatensatz, wie bereits in Abschnitt 3.2 angesprochen wurde. Zumindest qualitativ lässt sich schnell die Aussage treffen, dass die Bilddaten des GAPs-Datensatzes sich hinsichtlich ihrer Art deutlich von denen des ImageNet-Datensatzes unterscheiden.

Der GAPs-Datensatz besteht größtenteils aus sehr hoch strukturierten homogenen Strukturen, auf welchen die Objekte, also die Straßenschäden, abgebildet sind. Der ImageNet-Datensatz hingegen besitzt ein deutlich größeres Spektrum. Durch 1000 verschiedene Objektklassen entsteht eine große Bandbreite an verschiedensten Strukturen, welche sowohl hoch als auch niedrig strukturierte Bildbeispiele umfassen. Abbildung 4.5 zeigt jeweils zehn komplett zufällig gezogene Bildbeispiele aus den beiden Datensätzen und soll einen Eindruck über die Art der Datensätze liefern. Um diesen qualitativen Eindruck zu untermauern, wurden weiterführende Untersuchungen zur Trennbarkeit der beiden Datensätze in den jeweiligen Feature-Räumen der vortrainierten Modelle durchgeführt. Dazu wurden zufällig gezogene Bildbeispiele sowohl des GAPs- als auch des ImageNet-Datensatzes durch die Netze verarbeitet und die Ausgaben der letzten Faltungsschicht gespeichert. Die resultierenden hochdimensionalen Featuremaps beider Datensätze wurden vereint und durch eine PCA (s. Abschnitt 2.4.1) bzw. t-SNE (s. Abschnitt 2.4.2) auf zwei Dimensionen reduziert. Im Falle von sehr unterschiedlichen Daten sollten die Datenpunkte beider Datensätze in der zweidimensionalen Repräsentation trennbare Cluster ergeben. Das in Abschnitt 5.3.1 beschriebene Experiment präsentiert die Ergebnisse. Der konkrete Ablauf wird in Pseudocode 4.4 erläutert. Um einen möglichen Einfluss der Inputkodierung, wie in Tabelle 4.1 aufgeführt, der verschiedenen Architekturen auszuschließen, wurden die Daten für jede Architektur entsprechend vorverarbeitet.

Eingaben

1 $V(x)$ // *Model vor Finetuning*

2 $X = \{x_0, \ldots, x_N\}$ // *N verschiedene GAPs-Bildbeispiel*

3 $Z = \{z_0, \ldots, z_N\}$ // *N verschiedene ImageNet-Bildbeispiel*

Algorithmus

4 `Für die letzte Faltungsschicht in` V;

5 $out = \{V(x_0), \ldots, V(x_N), V(z_0), \ldots, V(z_N)\}$; // $\forall x \in X, \forall z \in Z$

// *out hat die Form:* $(2 * N) \times W \times H \times C$

6 $out = reshape(out, ((2 * N) * (W * H * C))$;

7 $out_{pca} = pca(out)$; // *Implementierung aus scikit-learn [Pedregosa et al., 2011]*

8 $out_{tsne} = tsne(out)$; // *Implementierung aus scikit-learn [Pedregosa et al., 2011]*

// out_{pca} *und* out_{tsne} *haben die Form:* $(2 * N) \times 2$

// *für jeden Datenpunkt wurde ein* X *und* Y *im zweidimensionalen ermittelt*

Rückgabe

9 out_{pca} // *zweidimensionale PCA-Datenpunkte*

10 out_{tsne} // *zweidimensionale t-SNE-Datenpunkte*

Abbildung 4.4: Pseudocode der Berechnung der PCA- und t-SNE-Darstellung

Zu sehen ist der Ablauf der Berechnung der zweidimensionalen Darstellung der GAPs- und ImageNet-Daten mittels PCA und t-SNE. Das beschriebene Vorgehen wird für alle Architekturen durchgeführt. Die für die Datenpunkte ermittelten zweidimensionalen Repräsentationen werden farblich getrennt als normales Diagramm und Dichtediagramm wiedergegeben. Abbildung 5.10 und 5.9 zeigen Beispiele zur ResNet50-Architektur.

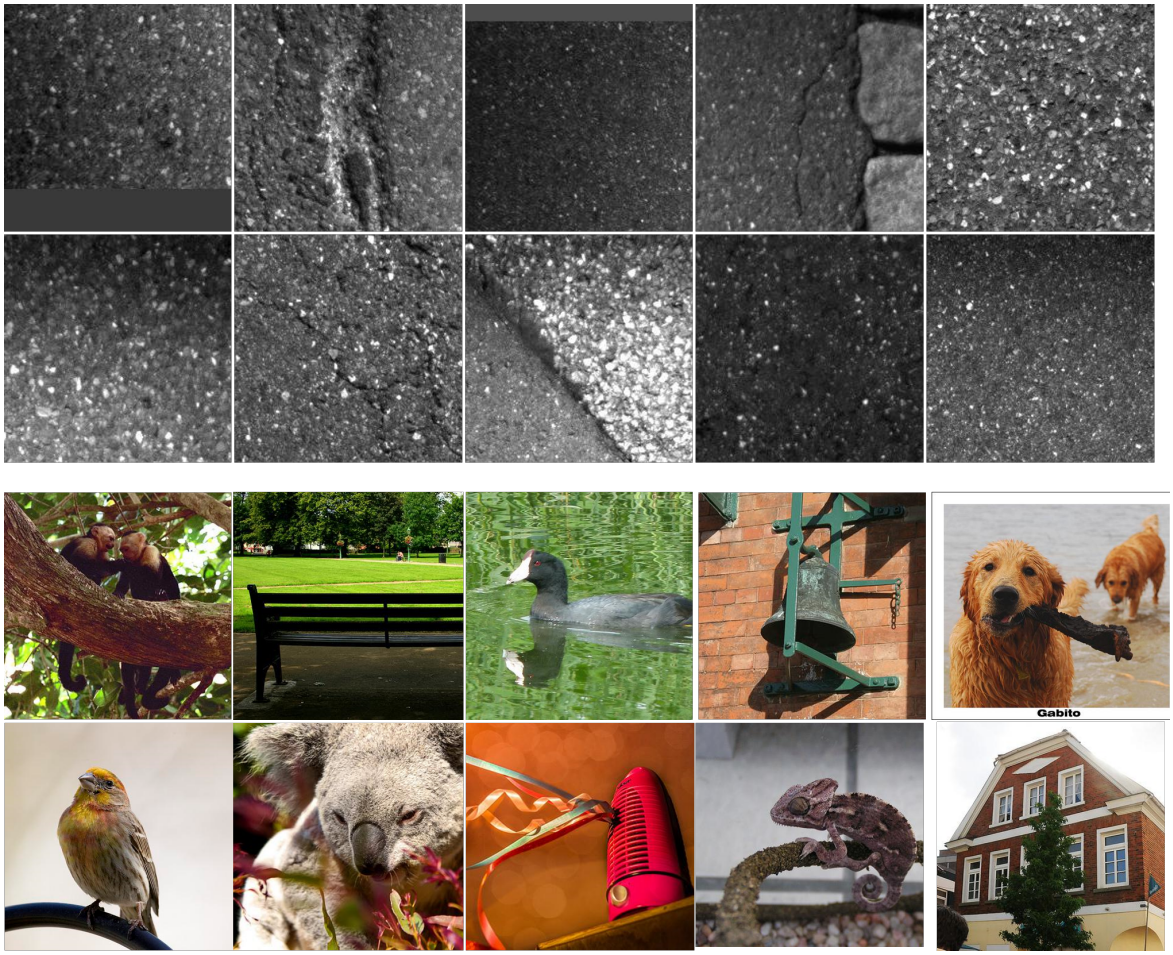


Abbildung 4.5: Vergleich des ImageNet- und GAPs-Datensatzes

Zu sehen sind jeweils zehn zufällig gezogene Bildbeispiele der Datensätze. Wie zuvor in Abschnitt 3.2 erläutert, ist die Übertragbarkeit gelernter Features abhängig von der Ähnlichkeit zwischen Ursprungs- und Zieldatensatz. Wie gut zu erkennen ist, sind die hochstrukturierten GAPs-Bilder nur schwer mit den ImageNet-Bildern vergleichbar.

Zufällig aus URL-Liste gezogene ImageNet-Bildbeispiele (unten): http://image-net.org/download-imageurls

GAPs-50k-V2-Bildquelle (oben): https://www.tu-ilmenau.de/neurob/data-sets-code/gaps/

4.4 Ein- und Ausgabe des Netzes

Für das Finetuning werden dem Netz Trainingsbeispiele präsentiert, welche jeweils einen einzelnen Kanal besitzen und auf einen Wertebereich von $[-1, +1]$ normiert sind. Um die vortrainierten Netze auf den Input mit lediglich einem Kanal anzupassen, wurde die jeweils erste Schicht der Netze wie in Abschnitt 4.4.1 beschrieben transformiert. Darüber hinaus wird keine weitere Data Augmentation oder Ähnliches durchgeführt. Das Netz bildet die Inputbilder auf die zwei Objektklassen "kein Schaden" (Klasse 0) und "Schaden" (Klasse 1) ab. Da die Klassifikation der ImageNet-Daten 1000 Klassen beinhaltet hat, werden die Netze für das Finetuning dahingehend verändert, dass die Ausgabeschicht lediglich auf zwei Neuronen reduziert wurde: jeweils eins pro Klasse des GAPs-Datensatzes. Die beiden Neuronen werden, wie in Abschnitt 4.2 bereits beschrieben, zufällig initialisiert.

4.4.1 Transformation: 3D Farbbilder zu 1D Schwarzweißbilder

Alle verwendeten Architekturen wurden im Pretraining mit Farbbildern des ImageNet-Datensatzes trainiert. Die erste Faltungsschicht besitzt dementsprechend einen Kernel, welcher Gewichte für drei Eingabekanäle enthält. Für das Finetuning wird die jeweils erste Faltungsschicht daher ebenfalls ein dreikanäliges Farbbild erwarten. Da der GAPs-Datensatz jedoch aus Schwarzweißbildern besteht, muss zunächst eine Anpassung durchgeführt werden.
Mathematisch wird durch die Faltung eine Multiplikation berechnet, durch welche die Pixelwerte des Eingabebildes durch den Kernel gewichtet aufsummiert werden. Die Ausgabe ergibt sich somit aus $w^T * x + b$, wobei w den Faltungskernel, x die Eingabe und b einen Bias repräsentiert. Grundlegend ergeben sich zwei mathematisch identische Herangehensweisen, die in Abbildung 4.6 verdeutlicht werden. Entweder wird das Schwarzweißbild dreifach hintereinander gelegt, um somit drei Kanäle nachzuahmen, oder die Gewichte des Kernels werden für die drei Kanäle aufsummiert. Mathematisch sind beide Methoden identisch und führen zu dem gleichen Ergebnis. In der Praxis ist die zuletzt genannte Variante jedoch die sinnvollere, da somit nur ein Drittel des

Speichers für ein Trainingsbeispiel benötigt wird. Selbst bei einem überschaubaren Trainingsdatensatz mit 50.000 Beispielen kann der theoretische Speicherbedarf, sofern alle Trainingsdaten im Speicher verfügbar sein sollen, von etwa 28GB auf 9GB reduziert werden.

Für die Umsetzung innerhalb dieser Arbeit wurde daher das zuletzt genannte Vorgehen gewählt. Die verwendeten Architekturen wurden daher dahingehend verändert, dass die Gewichte des Kernels der jeweils ersten Faltungsschicht über die Kanäle hinweg aufsummiert wurden.

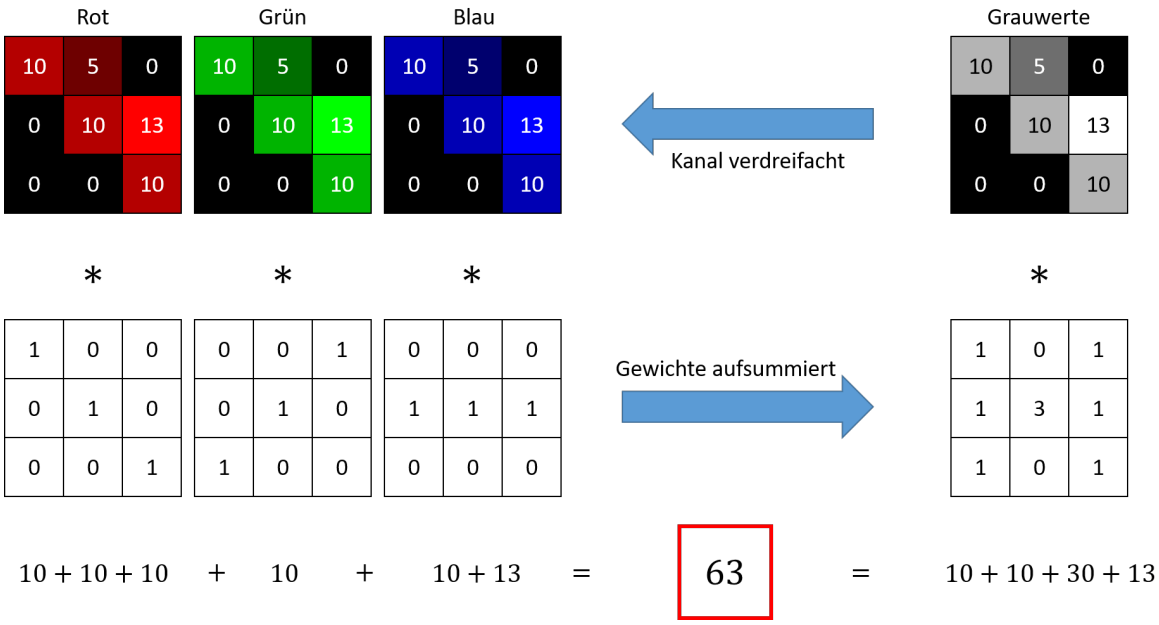


Abbildung 4.6: Gewichtstransformation von 3D zu 1D

Zu sehen ist eine Darstellung der beiden Möglichkeiten, eine dreidimensionale Inputschicht eines Netzes und die zugehörigen Gewichte in eine eindimensionale Variante umzuwandeln. Wie in Abschnitt 4.4.1 ausführlich beschrieben, gibt es zwei Möglichkeiten, den Input eines Netzes von drei Dimensionen auf eine zu reduzieren. Es können entweder die Faltungskernel der ersten Schicht aufaddiert (unten) oder es kann der Schwarzweißinput dreifach hintereinander eingegeben werden (oben). Mathematisch wird, wie zu erkennen ist, eine identische Operation ausgeführt.

4.4.2 Transformation: Korrektur der Inputskodierung

Wie Tabelle 4.1 zeigt, wurden das AlexNet, ResNet50 und VGG19 im Pretraining mit einer sich zum Zieldatensatz unterscheidenden Inputkodierung trainiert. Im Gegensatz zu der trainierten Inputkodierung der drei genannten Architekturen sind die Daten des GAPs-Datensatzes auf den Bereich $-1 \cdots +1$ normiert. Dieser Unterschied hätte durch eine Normalisierung der Gewichte, analog zur Inputkodierung, kompensiert werden können.
Wie in Abschnitt 2.3.3 beschrieben, wird durch den Einsatz von Batch Normalization die Abhängigkeit der Ausgabekodierung zwischen den Schichten reduziert. Die Anpassung der Gewichte des ResNet50 hätte somit maßgeblich nur die erste Faltungsschicht betroffen. Die Ausgabekodierung aller folgenden Schichten werden durch die Batch Normalization reguliert. Für das AlexNet und VGG19 hätten jedoch die Gewichte der gesamten Netze angepasst werden müssen. Der Einfachheit halber wurde auf diesen Schritt verzichtet. Der Vorversuch in Abschnitt 5.1.1 zeigt, dass die unterschiedliche Inputkodierung keinen nennenswerten Einfluss auf das grundlegende Verhalten des Transfer Learnings besitzt.

4.5 Training

Der Kern dieser Arbeit stellt das Training der sieben Netzarchitekturen, wie in Abschnitt 4.5.3 beschrieben, mit verschiedenen Konfigurationen auf dem GAPs-50k-Datensatz in Version 2 dar. Im Folgenden soll ein Überblick über den Trainingsverlauf und die Durchführung gegeben werden.

4.5.1 Pretraining

Zu allen verwendeten Architekturen stehen vortrainierte Gewichte zur Verfügung, welche auf Basis des ImageNet-Datensatzes trainiert wurden. Die Gewichte und der Code für das ResNet50, InceptionNet V3, MobileNet, VGG19 und XceptionNet konnten direkt aus den Keras Applications [Chollet et al., 2015] verwendet werden.
Die Architekturen für das AlexNet und das SE-ResNet50 mussten zunächst portiert

bzw. erstellt werden. Für die Erstellung des SE-ResNet50 wurde auf die normale ResNet50 Architektur zurückgegriffen. Entsprechend der Arbeit nach [Hu et al., 2017] wurden die SE-Blöcke ergänzt. Die Gewichte für das SE-ResNet50 wurden aus den Tensorpack Modellen [Wu et al., 2016] portiert.
Sowohl der Code als auch die Gewichte für das AlexNet wurden von [Heuritech, 2017] portiert. Tabelle 4.1 gibt eine Übersicht über die verwendete Inputkodierung während des Pretrainings.

Architektur	**Mittelwertfrei in Bezug auf den**	**Kanalreihenfolge**	**Skalierung**
AlexNet	ImageNet-Datensatz*	RGB	$0 \ldots 255$
InceptionNetV3	Pixelwert von 127.5	RGB	$-1 \ldots 1$
ResNet50	ImageNet-Datensatz*	BGR	$0 \ldots 255$
SE-ResNet50	Pixelwert von 127.5	BGR	$-1 \ldots 1$
MobileNet	Pixelwert von 127.5	RGB	$-1 \ldots 1$
Xception	Pixelwert von 127.5	RGB	$-1 \ldots 1$
VGG19	ImageNet-Datensatz*	BGR	$0 \ldots 255$

Tabelle 4.1: Überblick der Inputkodierung der Architekturen
Zu sehen ist die Auflistung der Inputkodierung aller Architekturen. Zwar ist die Kanalreihenfolge während des Pretrainings angegeben, spielt durch die Aufsummierung von 3D zu 1D jedoch keine Rolle.
**Mittelwerte des ImageNet-Datensatzes für die Kanäle in der Reihenfolge BRG sind: 103.939, 116.779, 123.68*

4.5.2 Framework, Optimierer und Performancemaße

Für das Training der Neuronalen Netze wurde das Keras Framework [Chollet et al., 2015] mit Tensorflow Backend [Abadi et al., 2015] und Python 2.7 gewählt. Darüber hinaus wurde als Optimierer durchgehend Stochastic Gradient Decent (SGD [Rumelhart et al., 1986]) in Kombination mit dem kategorischen Cross-Entropy-Loss verwendet.

Zur Bewertung des Trainings werden der F1-Score, die BER, die Accurarcy und das Precision-Recall-Diagramm auf Validierungs-, Validierungs-Test- und Testdatensatz berechnet. Die drei genannten Evaluierungsdatensätze werden in Abschnitt 4.5.4 im Detail erläutert.

4.5.3 Durchführung

Transfer Learning

Grundlegend wurden alle sieben Netze jeweils 25 Epochen mit je 6 verschiedenen Lernraten und je 6 verschiedenen Abstufungen trainiert. In jeder dieser Abstufungen sind unterschiedlich viele Schichten für das Training eingefroren. Sinnvollerweise werden lediglich die Schichten eingefroren, welche nicht zufällig initialisiert werden. Die Abstufungen wurden in 20% Schritten durchgeführt, womit sich die Stufen 0%, 20%, 40%, 60%, 80% und 100% ergeben. Außerdem werden die Schichten immer ausgehend von dem Netzanfang eingefroren. Insgesamt werden somit 252 Trainingsdurchläufe mit je 25 Epochen durchgeführt. Nach Bedarf wurden allerdings weitere Trainings mit anderen Lernraten ergänzt. Insbesondere für das VGG19 haben sich die anfangs festgelegten Stufen der Lernrate als ungenügend herausgestellt. Abschnitt 5.2.1 geht darauf näher ein und präsentiert die Ergebnisse.

From-scratch

Als Referenz wurde zusätzlich zum Transfer-Learning-Ansatz noch ein Training from-scratch durchgeführt. Dafür wurden zu jeder Architektur jeweils drei Trainingsdurchläufe mit je 250 Epochen absolviert. Das mehrfache Training soll sicherstellen, dass eine angemessene Zuverlässigkeit der Ergebnisse vorliegt. Falls die Abweichungen zwischen den drei Durchläufen nur gering ausfallen, ist davon auszugehen, dass die Ergebnisse des Trainings repräsentativ sind. Abschnitt 5.2.1 geht im Detail auf die Ergebnisse ein.

4.5.4 Vierteilung des Trainingsdatensatzes

Entsprechend dem Vorgehen von [Ng, 2016] ist der Trainingsdatensatz in vier Teile untergliedert. Diese werden im Folgenden Trainigs- ("train"), Validierungs- ("valid"), Validierungs-Test- ("valid test") und Testdatensatz ("test") genannt. Während der Zweck des Trainigsdatensatzes intuitiv verständlich ist, folgt eine kurze, zweckmäßige Einteilung der restlichen drei Teile:

- Validierungsdatensatz:
  Bildet das Abbruchkriterium des Trainings und wird verwendet, um festzulegen, wann das Netz nichts mehr lernt und das Training abgebrochen werden kann.

- Validierungs-Test-Datensatz:
  Spiegelt die Generalisierungsfähigkeit des Netzes während des Trainings wider und wird verwendet, um die beste Epoche eines Trainings auszuwählen.

- Testdatensatz:
  Wird verwendet, um die Leistung der ausgewählten Trainingsepoche des Netzes auf völlig unbekannten Daten zu evaluieren.

4.6 Abweichungen der Architekturen

4.6.1 XceptionNet und InceptionNet V3

Sowohl das XceptionNet als auch das InceptionNet V3 wurden auf Bildausschnitten mit einer Größe von 299×299 Pixeln vortrainiert. Das Finetuning wird der Einfachheit halber und zur besseren Vergleichbarkeit jedoch für alle Architekturen auf einer Bildgröße von 224×224 durchgeführt. Vorversuch 5.1.2 zeigt, dass die Leistung des Netzes trotz kleinerer Trainingsdaten nur unwesentlich negativ beeinflusst wird.

4.6.2 AlexNet

Das AlexNet besitzt in seiner ursprünglichen Version nach [Krizhevsky et al., 2012] eine Local Response Normalization (LRN). Dieser Normalisierungsschritt ist auf eine bestimmte Region der Featuremaps beschränkt und normiert deren Aktivierungen über die Kanäle hinweg. Trotz des damaligen großen Erfolgs des AlexNet konnte sich die Local Response Normalization nicht durchsetzen und hat nachfolgend kaum Anwendung gefunden. Der Einfachheit halber wurde für diese Arbeit, wie beispielsweise auch in [Zhang et al., 2018] und [Shin et al., 2016], auf den Einsatz der LRN verzichtet.

4.6.3 MobileNet

Das MobileNet unterscheidet sich von der ursprünglichen Version hauptsächlich hinsichtlich der letzten Klassifikationsschichten. In der originalen Arbeit von [Howard et al., 2017] wurde das Netz durch eine vollverschaltete Schicht ohne Dropout abgeschlossen. Die Keras Implementierung nutzt jedoch eine Dropoutschicht mit einer Dropout-Rate von 0.001 und anschließender Faltungsschicht als Klassifikator. Während die Faltungsschicht lediglich eine Umformulierung ist und in diesem Fall mathematisch identische Ergebnisse liefert, verursacht Dropout jedoch eine leichte Veränderung der Architektur.
Darüber hinaus wurde der Parameter α für alle Experimente konstant gleich Eins gewählt. Es wurde also immer die volle, ursprüngliche Kapazität des Netzes verwendet.

4.7 Erwartete Effekte

Wie eingangs bereits erläutert, werden neben der reinen Leistungssteigerung durch das Transfer Learning auch weitere Effekte erwartet. Ziel dieser Masterarbeit ist nicht nur die Anwendung von Transfer Learning, sondern auch die Überprüfung dieser Effekte. Die erwarteten Beobachtungen gehen maßgeblich auf die Arbeit von [Kim et al., 2017] zurück und werden nachfolgend im Einzelnen erklärt.

4.7.1 Verbesserte Generalisierungsfähigkeit

Wie in Kapitel 3 beschrieben, legen verschiedene Arbeiten die Tatsache nahe, dass Transfer Learning einen positiven Einfluss auf die Generalisierungsfähigkeit hat. Diese Verbesserung würde sich direkt aus der Leistung des Netzes auf dem Test- bzw. Validierungs-Test-Datensatz ablesen lassen. Als Referenz bzw. Baseline wird jeweils ein identisches Netz mit zufällig initialisierten Gewichten (from-scratch) trainiert. Erwartungsgemäß sollte die Leistung des Netzes auf Basis des Transfer Learnings über der des anderen Netzes liegen. Das in Abschnitt 5.2.1 beschriebene Experiment liefert dazu einen detaillierten Überblick über alle Trainings. Zur Bewertung der Leistung wurde der F1-Score, die Balanced-Error-Rate (BER), die Accurarcy (ACC) und die Precision-Recall-Kurve (PR-Kurve) verwendet und wie folgt berechnet:

$$ACC = \frac{tp + tn}{P + N} \qquad | \; fp/tp = false/true\ positives \tag{4.1}$$

$$BER = \frac{1}{2}(\frac{fn}{P} + \frac{fp}{N}) \qquad | \; fn/tn = false/true\ negatives \tag{4.2}$$

$$Precision = \frac{tp}{tp + fp} \qquad | \; P = positives = fp + tp \tag{4.3}$$

$$TPR = Recall = \frac{tp}{P} \qquad | \; N = negatives = fn + tn \tag{4.4}$$

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4.5}$$

4.7.2 Schnelles Training bzw. nur geringe Anpassung der Gewichte

Aufgrund der Initialisierung des Netzes in einem Zustand, der deutlich näher an einem guten Optimum ist, müssen die Gewichte betragsweise weniger stark geändert werden als bei einer zufälligen Initialisierung. Insbesondere die vorderen Schichten, welche, wie in Abschnitt 3.2 beschrieben, eher allgemeinere Features (Kanten-, Ecken-Features, usw.) ausprägen, müssen kaum angepasst werden. Daraus folgt, analog zu den Beobachtungen in [Kim et al., 2017], die Erwartung, dass die Gewichte der vorderen Schichten im Betrag weniger stark verändert werden als die der hinteren Schichten. Überprüft wird dies durch die Bildung der l2-Norm zwischen den Gewichten vor und nach dem Finetuning. Das in Abschnitt 5.3.2 beschriebene Experiment geht darauf näher ein. Das Vorgehen zur Berechnung der l2-Norm kann dem Pseudocode 4.7 entnommen werden.
Darüber hinaus sollte sich durch die Initialisierung in der Nähe eines guten Optimums, selbst mit einer geringeren Lernrate, schnell eine hohe Leistung des Klassifikators einstellen. Um diese Erwartung zu überprüfen, wird der Trainingsverlauf während des Finetunings mit dem Verlauf des from-scratch-Trainings verglichen. Insbesondere während der ersten Epochen sollte das Finetuning deutlich besser abschneiden als die Netze, welche von einer zufälligen Initialisierung ausgehend trainiert werden. Das in Abschnitt 5.2.1 beschriebene Experiment liefert dazu einen detaillierten Überblick über alle Trainings. Zur Bewertung der Trainingsgeschwindigkeit wird die Anzahl der benötigten Trainingsepochen verwendet.

4.7.3 Feature-Selektion statt neuer Features lernen

In [Kim et al., 2017] wurde die Wirkung des Finetunings auf die Gewichte eines Netzes als eine Art Feature-Selektion beschrieben. Dies hat zufolge, dass insbesondere die vorderen Schichten eines Netzes keine neuen Features lernen, sondern eher vorhandene optimieren und unnötige Features ausblenden. Voraussetzung dafür ist, dass schon während des Pretrainings die benötigten Features gelernt wurden. Konkret bedeutet das für die Trainingsbilder des Ursprungsdatensatzes, dass diese mehr Varianz oder

Eingaben

1 $W_{vorfinetuning} = \{(w_0), \ldots, (w_{LV})\}$ // *LV Faltungsschichten vor Finetuning*

2 $W_{nachfinetuning} = \{(w_0), \ldots, (w_{LN})\}$ // *LN Faltungsschichten nach Finetuning*

// *w Gewichte des Faltungskernels der Form* $H \times W \times C$

Initialisierung

3 $d_{c,l} \leftarrow 0$; // *für alle* $c \in [0, \ldots, C]$ *und* $l \in [0, \ldots, LV]$

Algorithmus

4 `Für jede Faltungsschicht` $l \in L$;

5 `Für jeden Kanal` $c \in C$;

6 $d_{c,l} = \frac{|w_{c,l,nachfinetuning} - w_{c,l,vorfinetuning}|}{|w_{c,l,vorfinetuning}|}$;

7 `Wenn normierte Distanz berechnet wird, dann;`

8 $d_{c,l} = d_{c,l}/anzahl\_gewichte(w_{c,l,nachfinetuning})$;

Rückgabe

9 $d_{c,l}$ // *Gewichtsnorm vor und nach dem Finetuning*

// *für jede Faltungsschicht und jeden Kanal*

Abbildung 4.7: Pseudocode der Berechnung der Gewichtsdifferenzen

Zu sehen ist der Ablauf der Berechnung der Gewichtsdifferenzen der Faltungskernel vor und nach dem Finetuning, in Pseudocode. Die Differenz bzw. l2-Norm wird für jeden Kernel einer Schicht getrennt berechnet, wodurch bei der Visualisierung ebenfalls ein Konfidenzintervall ermittelt werden kann. Das beschriebene Vorgehen wird auf alle Architekturen und auch auf die Baseline gleich angewendet.

Diversität, im Sinne von verschiedenen Bildstrukturen, als der Zieldatensatz besitzen müssen. Meist wird Transfer Learning mit dem Ziel eingesetzt, die Leistung eines Klassifikators auf besonders kleinen Zieldatensätzen zu verbessern. In diesen Fällen besitzt der Ursprungsdatensatz im Vergleich zum Zieldatensatz meist automatisch eine höhere Diversität. Insbesondere auch dadurch, dass für das Pretraining in vielen Fällen der ImageNet-Datensatz verwendet wird.

Im Falle einer Feature-Selektion sollten dementsprechend zwei Beobachtungen gemacht

werden können:

- Die Gewichte des Netzes, insbesondere die der vorderen Schichten, sollten durch das Finetuning nur wenig angepasst werden. Um das zu überprüfen, wird die Gewichtsdifferenz der Gewichte vor und nach dem Finetuning betrachtet. Die Berechnung der Differenz wird in Abschnitt 4.7.2 und dem Pseudocode 4.7 beschrieben. Die Ergebnisse werden in Abschnitt 5.3.2 präsentiert.

- Die resultierenden Featuremaps innerhalb des Netzes und insbesondere die der hinteren Schichten sollten vergleichsweise spärlich besetzt sein. Um das zu überprüfen, wird die Spärlichkeit jeder Schicht, vor und nach dem Finetuning, in Bezug auf einige zufällige Bildbeispiele gemessen. Die Berechnung der Spärlichkeit wird in Abschnitt 4.7.4 und Pseudocode 4.8 erläutert. Die Ergebnisse werden in Abschnitt 5.3.3 präsentiert.

4.7.4 Spärlich besetzte Featuremaps

Innerhalb der Untersuchungen in [Kim et al., 2017] wird weiterhin berichtet, dass durch die Feature-Selektion, wie auch schon in Abschnitt 4.7.3 erwähnt, die Featuremaps nach dem Finetuning deutlich spärlicher belegt sind. Die Spärlichkeit, im Folgenden auch Sparsity genannt, beschreibt dabei das Verhältnis zwischen den Werten einer Featuremap, die gleich Null sind, und der Anzahl der Werte insgesamt. Mit anderen Worten beschreibt die Spärlichkeit dementsprechend, wie viel Prozent einer Featuremap gleich Null oder nahe Null sind. Die Berechnung der Spärlichkeit wird im Pseudocode 4.8 näher erläutert.
Die Featuremaps, welche durch das Finetuning so verändert werden, dass sie generell keine Aktivierung mehr aufweisen, werden als tot bezeichnet. Die toten Features können aus dem Finetuning resultieren, wenn z.B. Farb-Features oder Features, welche nur auf sehr niedrig strukturierte Flächen reagieren, umgelernt werden. Zur Untersuchung der Spärlichkeit werden konkret zwei Teilaspekte untersucht:

- **Der Verlauf der Sparsity über alle Schichten des Netzes.** Um zu überprüfen, ob eine Feature-Selektion durchgeführt wird, also die Spärlichkeit über viele

Schichten hinweg generell steigt, wird der Verlauf über alle Schichten des Netzes, vor und nach dem Finetuning, gemessen. Da die Aktivierung abhängig vom Input des Netzes ist, werden mehrere zufällige Beispiele mit und ohne Schaden verwendet. Der Ablauf der Berechnung wird im Pseudocode 4.8 näher erläutert.

- **Die Aktivierung der letzten Featuremap vor der Klassifikation.** Um qualitativ bewerten zu können, ob eine Feature-Selektion stattgefunden hat, wird die Aktivierung der letzten Faltungsschicht in Bezug auf einige zufällige Bildbeispiele visualisiert. Darüber hinaus lässt die Visualisierung ebenfalls Rückschlüsse auf die Inhalte zu, welche eine besonders hohe Aktivierung der Features hervorrufen. Abbildung 5.16 zeigt ein Beispiel. Die Berechnung der Visualisierung ist im Pseudocode 4.9 näher erläutert.

Die Ergebnisse der beiden Teilaspekte aus Abschnitt 5.3.3 können allerdings lediglich mit den Ergebnissen aus [Kim et al., 2017] verglichen werden. Trotz intensiver Suche konnte abgesehen von [Kim et al., 2017] keine Arbeit gefunden werden, in der vergleichbare Untersuchungen in Bezug auf die Spärlichkeit der Featuremaps durchgeführt wurden.

Eingaben

1 $V(x)$ // *Model vor Finetuning*

2 $N(x)$ // *Model nach Finetuning*

3 $X = \{x_0, \ldots, x_N\}$ // *N verschiedene Bildbeispiel mit und ohne Schaden*

4 τ // *Threshold ab dem die Aktivierung als Null angenommen wird*

Initialisierung

5 $sparcity_{v,i,l_v,c} \leftarrow 0$; // *Sparsity vor dem Finetuning*

6 $sparcity_{n,i,l_n,c} \leftarrow 0$; // *Sparsity nach dem Finetuning*

// *für jedes Bildbeispiel $i \in out$ und jeden Kanal $c \in C$ einer Faltungsschicht l_v bzw. l_n*

Algorithmus

7 `Für jede Faltungsschicht` $l_v \in V$ `und` $l_n \in N$;

8 $out_v = \{V(x_0), \ldots, V(x_N)\}$; // $\forall x \in X$

9 $out_n = \{N(x_0), \ldots, N(x_N)\}$; // $\forall x \in X$

// *out_v und out_n haben die Form: Bildbeispiele $\times W \times H \times C$, wobei C für die verschiedenen Kanäle der jeweiligen Faltungsschicht steht*

10 `Für jedes Bildbeispiel` $i \in out$ `und jeden Kanal` $c \in C$;

11 $out_{v,i,c}, out_{n,i,c} = normalize(out_{v,i,c}, out_{n,i,c})$; // *normalisieren$(x, y)$ wie in*
// *Pseudocode 4.10 beschrieben*

12 $out_{v,i,c} = threshold(out_{v,i,c}, \tau)$; // *Alle Aktivierungen kleiner τ*
// *werden zu Null gesetzt*

13 $out_{n,i,c} = threshold(out_{n,i,c}, \tau)$;

14 $sparcity_{v,i,l_v,c} = \dfrac{count\_zeros(out_{v,i,c})}{cout\_elements(out_{v,i,c})}$;

15 $sparcity_{n,i,l_n,c} = \dfrac{count\_\_zeros(out_{n,i,c})}{cout\_elements(out_{n,i,c})}$;

Rückgabe

16 $sparcity_{v,i,l_v,c}$ // *Sparsity vor dem Finetuning*

17 $sparcity_{n,i,l_n,c}$ // *Sparsity nach dem Finetuning*

// *für jedes Bildbeispiel $i \in out$ und jeden Kanal $c \in C$ der Faltungsschicht l*

Abbildung 4.8: Pseudocode der Berechnung des Verlaufs der Spärlichkeit über ein ganzes Netz

Zu sehen ist der Ablauf der Berechnung der Spärlichkeit über ein Netz vor und nach dem Finetuning in Pseudocode.

Eingaben

```
1   V(x)                                   // Model vor Finetuning
2   N(x)                                   // Model nach Finetuning
3   X = {x_0, ..., x_N}                    // N verschiedene Bildbeispiel mit und ohne Schaden
```

Algorithmus

```
4   Für einzig die letzte Faltungsschicht l_v ∈ V und l_n ∈ N;
5      out_v = {V(x_0), ..., V(x_N)};                                   // ∀x ∈ X
6      out_n = {N(x_0), ..., N(x_N)};                                   // ∀x ∈ X
                                   // out_v und out_n haben die Form: Bildbeispiele × W × H × C
                                   // C steht für die verschiedenen Kanäle der jeweiligen Faltungsschicht
7      Für jedes Bildbeispiel i ∈ out und jeden Kanal c ∈ C;
8         out_{v,i,c}, out_{n,i,c} = normalize(out_{v,i,c}, out_{n,i,c});   // normalize(x, y) wie in
                                                                   // Pseudocode 4.10 beschrieben

9   Für jedes Bildbeispiel i ∈ out;
10     Ein weißes Bild erstellen;
11     Für jeden Kanal c ∈ C;
12        a_v = out_{v,i,c};                          // Aktivierung des Kanals c der letzten
13        a_n = out_{n,i,c};                          // Faltungsschicht vor bzw. nach dem Finetuning
                                                      // a_v und a_n haben die Form W × H
14        Aktivierungsblöcke dem weißen Bild mit Abstand hinzufügen;
15        Bild speichern;
```

Abbildung 4.9: Pseudocode der Berechnung der Visualisierung der Aktivierung der letzten Faltungsschicht

Zu sehen ist der Ablauf der Erstellung der Feature-Aktivitätsbilder der jeweils letzten Faltungsschicht. Das beschriebene Vorgehen wird auf alle Architekturen angewendet. Abbildung 5.16 zeigt ein Beispiel, wie eines der generierten Bilder aussieht.

Eingaben

1 $out_{v,i,c}$ // *Aktivierung vor Finetuning*

2 $out_{n,i,c}$ // *Aktivierung nach Finetuning*

// *Jeweils für jedes Bildbeispiel* $i \in out$ *und jeden Kanal* $c \in C$

Algorithmus

3 `Für jedes Bildbeispiel` $i \in out$ `und jeden Kanal` $c \in C$;

4 $a_v = out_{v,i,c}$; // *Aktivierung* a_v *eines einzelnen Kanals vor dem Finetuning*

5 $a_n = out_{n,i,c}$; // *Aktivierung* a_n *eines einzelnen Kanals nach dem Finetuning*

// a_v *und* a_n *haben die Form* $W \times H$

6 $a_{min} = \{min(a_{v,0,0}, a_{n,0,0}), \ldots, min(a_{v,W,H}, a_{n,W,H})\}$; // *Für* $w \in W$ *und* $h \in H$

// *Minimum wird elementweise für einen Kanal gebildet*

7 $out_{v,i,c} = out_{v,i,c} - a_{min}$;

8 $out_{n,i,c} = out_{n,i,c} - a_{min}$;

9 $a_{max} = max(max(out_{v,i,c}), max(out_{n,i,c}))$; // *Maximum wird über beide*

// *korrespondierenden Kanäle* hinweg gebildet*

10 $out_{v,i,c} = out_{v,i,c}/a_{max}$;

11 $out_{n,i,c} = out_{n,i,c}/a_{max}$;

Rückgabe

12 $out_{v,i,c}$ // *Normalisierte Aktivierung vor dem Finetuning*

13 $out_{n,i,c}$ // *Normalisierte Aktivierung nach dem Finetuning*

// *für jedes Bildbeispiel* $i \in out$ *und jeden Kanal* $c \in C$

Abbildung 4.10: Pseudocode der Normalisierung von Featuremaps

Zu sehen ist der Ablauf der Normalisierung von Featuremaps, welche vor und nach dem Finetuning berechnet wurden. Für die Normalisierung wird zunächst um das elementweise Minimum zweier korrespondierender Featuremaps verschoben und anschließend um das Maximum geteilt. Korrespondierende Featuremaps bezieht sich dabei auf die Reihenfolge innerhalb einer Schicht. Die Featuremap, welche durch den ersten Kernel einer Schicht vor dem Finetuning erzeugt wurde, korrespondiert mit der Featuremap, welche durch den ersten Kernel einer Schicht nach dem Finetuning erzeugt wurde.

4.7.5 Übernahme möglichst vieler Gewichte

Entsprechend der Erkenntnis aus [Yosinski et al., 2014], dass möglichst viele Gewichte übernommen werden sollen, stellt sich die Frage, ob Leistungsvorteile ebenfalls aus der Übernahme der vollverschalteten Schichten entstehen. Im Detail ist dabei lediglich die Übernahme der letzten Ausgabeschicht problematisch. Da die Netze auf dem ImageNet-Datensatz vortrainiert wurden, besitzen sie 1000 Ausgabeneuronen. Da der GAPs-Datensatz jedoch nur zwei Ausgabeneuronen besitzt, muss eine Auswahl getroffen werden.

Dazu wird aus allen 1000 Ausgabeneuronen des ImageNet Klassifikators jeweils das Neuron gesucht, durch das die GAPs-Daten möglichst gut klassifiziert werden können. Im Detail wird dafür der Fisher-Score verwendet, um zunächst zu bewerten, welches Neuron die GAPs-Daten überhaupt linear trennbar kodiert. Das Vorgehen zur Berechnung des Fisher-Scores wird in Pseudocode 4.11 im Detail erläutert.

Aufbauend darauf können die beiden Neuronen für das Transfer Learning übernommen werden, welche die Daten am besten trennen. Falls keine zwei Neuronen existieren, ist es auch möglich, ein Neuron zu verwenden, dessen Ausgabe zu kopieren und für die zweite Klasse zu invertieren. Somit würde sich je ein Neuron ergeben, welches bei jeweils einer der beiden Klassen hoch bzw. niedrig aktiviert ist. Dabei muss es nicht der Fall sein, dass das Neuron, welches die Ausgabe für die beiden Objektklassen am besten trennt, auch das Neuron ist, das insgesamt am höchsten aktiviert ist. In einem Vorversuch, wie in Abschnitt 5.1.3 beschrieben, wird über den Fisher Score bewertet, wie gut jedes einzelne Neuron die Klassen trennt. Um einen möglichen Einfluss der Inputkodierung, wie in Tabelle 4.1 aufgeführt, der verschiedenen Architekturen auszuschließen, wurden die GAPs-Daten für jede Architektur entsprechend vorverarbeitet.

Eingaben

1 $V(x)$ // *Model vor Finetuning*

2 $D = \{d_0, \ldots, d_N\}$ // *N verschiedene Trainingsbeispiele des GAPs-Datensatzes*

3 $Y = \{y_0, \ldots, y_N\}$ // *Labels der Trainingsbeispiele*

Algorithmus

Bestimmung des Fisher-Scores: //

4 $out = \{V(d_0), \ldots, V(d_N)\}$; // *Netzausgabe für alle Daten berechnen*

// *out hat die Form:* $Trainingsbeispiele \times 1000 Klassen$

5 $fs = get\_fisher\_score(out, Y)$; // *Implementierung aus [Li et al., 2017b]*

// *bzw. [Li et al., 2017a]*

Transfer der besten Neuronen: //

6 `Zwei Neuronen mit dem besten Fisher-Score auswählen;`

7 `Letzte Klassifikationsschicht von` V `durch diese beiden Neuronen ersetzen;`

Rückgabe

8 fs // *Je ein Fisher-Score für jedes der 1000 Ausgabeneuronen*

9 $V_{gaps}(x)$ // *Model für die GAPs-Klassifikation vor dem Finetuning*

Abbildung 4.11: Pseudocode der Berechnung des Fisher-Scores

Zu sehen ist der Ablauf der Berechnung des Fisher-Scores und die darauf basierende Auswahl der geeignetsten Neuronen. Das beschriebene Verfahren wird auf jede Architektur angewendet.

4.8 Zusammenfassung

Kern der Arbeit ist zum einen die Bewertung der Leistungssteigerung durch Transfer Learning für die Erkennung von Straßenschäden und zum anderen die Analyse, inwiefern ein Transfer Learning die Gewichte und Ausgaben eines Netzes beeinflusst. Dazu werden verschiedene Netzarchitekturen per Transfer Learning trainiert und über viele verschiedene Trainings hinweg miteinander verglichen. Neben der reinen Leistungssteigerung werden darüber hinaus verschiedene Beobachtungen erwartet, welche durch das Finetuning hervorgerufen werden. Dazu zählen:

- bessere Generalisierungsfähigkeit im Vergleich zu einem Training from-scratch
- schnell konvergierendes Training
- Feature-Selektion statt dem Neu-Lernen von Features
- eine Leistungssteigerung, umso mehr Gewichte übernommen werden

Experimente 5

Wie zuvor in Kapitel 4 erläutert wurde, ist das Ziel dieser Arbeit nicht nur die Ermittelung der Effektivität von Transfer Learning für die Detektion von Straßenschäden. Darüber hinaus sollen außerdem, wie in Abschnitt 4.7 beschrieben, verschiedene Effekte beobachtet werden. Um diese beiden Aspekte zu untersuchen, wurden verschiedene Experimente durchgeführt. Voraussetzungen für diese Experimente, wie z.B. der Einfluss der Inputkodierung oder verschiedener Inputgrößen, welche zunächst geklärt werden müssen, sind in Abschnitt 5.1 aufgeführt. Abschnitt 5.2.1 gibt im Anschluss die ermittelte Leistung der verschiedenen Architekturen auf Basis des Transfer Learnings und der Baseline wieder. Die folgenden Abschnitte 5.3.1 bis 5.3.6 gehen auf die verschiedenen Experimente zur Untersuchung der erwarteten Effekte ein. Abschließend wird in Abschnitt 5.4 eine Zusammenfassung gegeben.

5.1 Vorversuche

Wie zuvor in Abschnitt 4.5 erläutert wurde, stammen die Architekturen aus unterschiedlichen Quellen und es wurden für das Pretraining unterschiedliche Einstellungen verwendet. Innerhalb dieser Arbeit wurde jedoch für alle Transfer-Learning-Trainings eine einheitliche Inputgröße von 224×224 und die für den GAPs-Datensatz angedachte Inputkodierung zwischen $-1 \cdots +1$ gewählt. Um einen möglichen Einfluss der Abweichungen von der im Pretraining verwendeten Inputkodierung auszuschließen, wird dieser und der Aspekt der Inputgröße in Abschnitt 5.1.1 und 5.1.2 untersucht.
Darüber hinaus wurde angedacht, zu untersuchen, ob die Übernahme möglichst vie-

ler Gewichte, d.h. auch die der vollverschalteten Schichten, einen positiven Einfluss auf das Transfer Learning hat. Für eine vollständige Übernahme müssen zwei Ausgabeneuronen der Klassifikationsschicht ausgewählt werden. Abschnitt 5.1.2 versucht Ausgabeneuronen des ImageNet-Klassifikators zu ermitteln, welche sich für die Übernahme eignen.

5.1.1 Auswirkung einer größeren Inputkodierung

Wie zuvor in Abschnitt 4.5 erläutert wurde, sind das AlexNet, ResNet50 und VGG19 mit einer Inputskalierung von $0 \ldots 255$ vortrainiert worden. Im Gegensatz dazu wird innerhalb dieser Arbeit einheitlich die für den GAPs-Datensatz angedachte Inputkodierung zwischen $-1 \cdots +1$ gewählt. Um einen Einfluss der geänderten Inputkodierung zu untersuchen, wurde für diese drei Architekturen exemplarisch ein Finetuning mit einer Inputskalierung von $0 \ldots 255$ durchgeführt.

Ergebnis

Tabelle 5.1 vergleicht die Leistung zwischen beiden Inputkodierungen auf Basis des F1-Scores. Es ist gut zu erkennen, dass der Leistungsunterschied zwischen beiden Trainings gering ausfällt. Darüber hinaus fällt der Unterschied zugunsten des Trainings mit der Inputkodierung von $-1 \cdots +1$ aus.

Diskussion

Die Ergebnisse zeigen, dass trotz einer anderen Inputkodierung kein entscheidender Leistungsunterschied festgestellt werden kann. Es wird vermutet, dass der Einfluss der Inputkodierung durch das Finetuning kompensiert wird. Um diesen Aspekt zu bewerten, kann die Höhe der Änderung der Gewichte durch das Finetuning betrachtet werden. Die Gewichtsänderungen der verwendeten Architekturen werden in Abschnitt 5.3.2 detailliert untersucht.
Vergleicht man die Höhe der Gewichtsänderung durch das Training from-scratch und Finetuning, fällt auf, dass in beiden Fällen die Gewichte ähnlich stark angepasst werden. Da die Änderung bereits für ein Training from-scratch ausreichend ist, wird er-

Architektur	Val-Test-F1-Score für Inputkodierung $[-1 \cdots +1]$	Val-Test-F1-Score für Inputkodierung $[0 \ldots 255]$	Lernrate
AlexNet	**0.890**	0.850	0.0005
VGG19	**0.915**	0.900	0.000355
ResNet50	**0.905**	0.896	0.001

Tabelle 5.1: Gegenüberstellung der Performance mit unterschiedlicher Inputkodierung
Zu sehen ist die Performance der drei Architekturen, die im Pretraining mit einer Inputkodierung von $0 \ldots 255$ gelernt wurden. Die präsentierte Leistungen haben sich jeweils aus einem Finetuning mit den GAPs-Daten ergeben, welche einmal original, im Bereich $-1 \cdots +1$ und einmal im Bereich $0 \ldots 255$ normiert wurden. Es ist gut zu erkennen, dass sich trotz unterschiedlicher Inputkodierung kein entscheidender Leistungsunterschied ergibt.

wartet, dass die Stärke der Änderung ebenfalls ausreichend ist, um während des Finetunings die unterschiedliche Inputkodierung auszugleichen.
Darüber hinaus ist zu erkennen, dass die Differenz zwischen beiden Inputkodierungen im Fall des ResNet50 am geringsten ausfällt. Es wird vermutet, dass dies auf den Einsatz von Batch Normalization im ResNet50 zurückzuführen ist. Durch die Batch Normalization wird die Abhängigkeit von der Ein- bzw. Ausgabekodierung zwischen den einzelnen Schichten im Netz verringert. Die veränderte Inputkodierung im Finetuning mit den GAPs-Daten betrifft somit hauptsächlich die erste Faltungsschicht, welche sich vor der ersten Batch-Normalization-Schicht befindet.

5.1.2 Auswirkung eines kleineren Inputs

Wie zuvor erläutert, wurde sowohl das XceptionNet als auch das InceptionNetV3 mit einer Auflösung von 299×299 vortrainiert. Innerhalb dieser Arbeit wurden alle Netze jedoch auf dem gleichen Datensatz mit einer einheitlichen Bildgröße von 224×224 trainiert. In einem Vorversuch wurde deswegen untersucht, ob die kleinere Bildgröße einen Einfluss auf die Performance hat. Dazu wurden die Ergebnisse aus dem Training,

mit der in dieser Arbeit üblichen Inputgröße von 224×224, mit einem Training mit der ursprünglichen Inputgröße von 299×299 verglichen. Zu beiden Architekturen wurden daher je sechs Trainings mit unterschiedlich vielen eingefrorenen Schichten durchgeführt. Für diese insgesamt zwölf Trainings wurden die Bildbeispiele um ein Zeropadding ergänzt, sodass die Netze Bilder im Zielformat von 299×299 erhielten. Das Zeropadding wurde gleichmäßig am Rand der Bildbeispiele ergänzt, sodass das eigentliche Straßenbild zentral in dem 299×299 Bild lag. Zu beachten ist dabei, dass durch das Zeropadding keinerlei Information hinzugefügt wird, sodass beiden Netzen die gleiche Information zur Verfügung steht und fair verglichen wird. Abbildung 5.1 zeigt die Ergebnisse und eine Gegenüberstellung des Trainings ohne Zeropadding.

Ergebnis

Wie in Abbildung 5.1 zu sehen, fällt der Unterschied verschwindend gering aus. Lediglich beim InceptionNet und auch nur ohne eingefrorene Schichten konnte festgestellt werden, dass durch die geringere Inputgröße mehr Trainingsepochen nötig sind, um die Bestleistung zu erreichen. Darüber hinaus begünstigt die höhere Inputgröße die Stabilität des Trainings, falls alle Faltungsschichten eingefroren sind. Die Performance konnte dadurch jedoch nicht ausschlaggebend gesteigert werden.

Diskussion

Die Ergebnisse zeigen, dass die kleinere Inputgröße keine gravierende Auswirkung auf die Performance hat. Eine Einschränkung ist jedoch, dass dies nur für eine bestimmte Lernrate ermittelt wurde. Die gewählte Lernrate ist dabei die gleiche, die auch in dem jeweils besten Trainingsdurchlauf der beiden Architekturen, in Abschnitt 5.2.3, verwendet wurde. Für diesen Fall kann ein negativer Einfluss auf die Performance abgelehnt werden. Darüber hinaus ist zu erwarten, dass der Unterschied bei anderen Lernraten wahrscheinlich sehr gering ausfällt. Wegen des potentiell großen Mehraufwands für die Untersuchung weiterer Lernraten wurde aus zeitlichen Gründen darauf verzichtet.

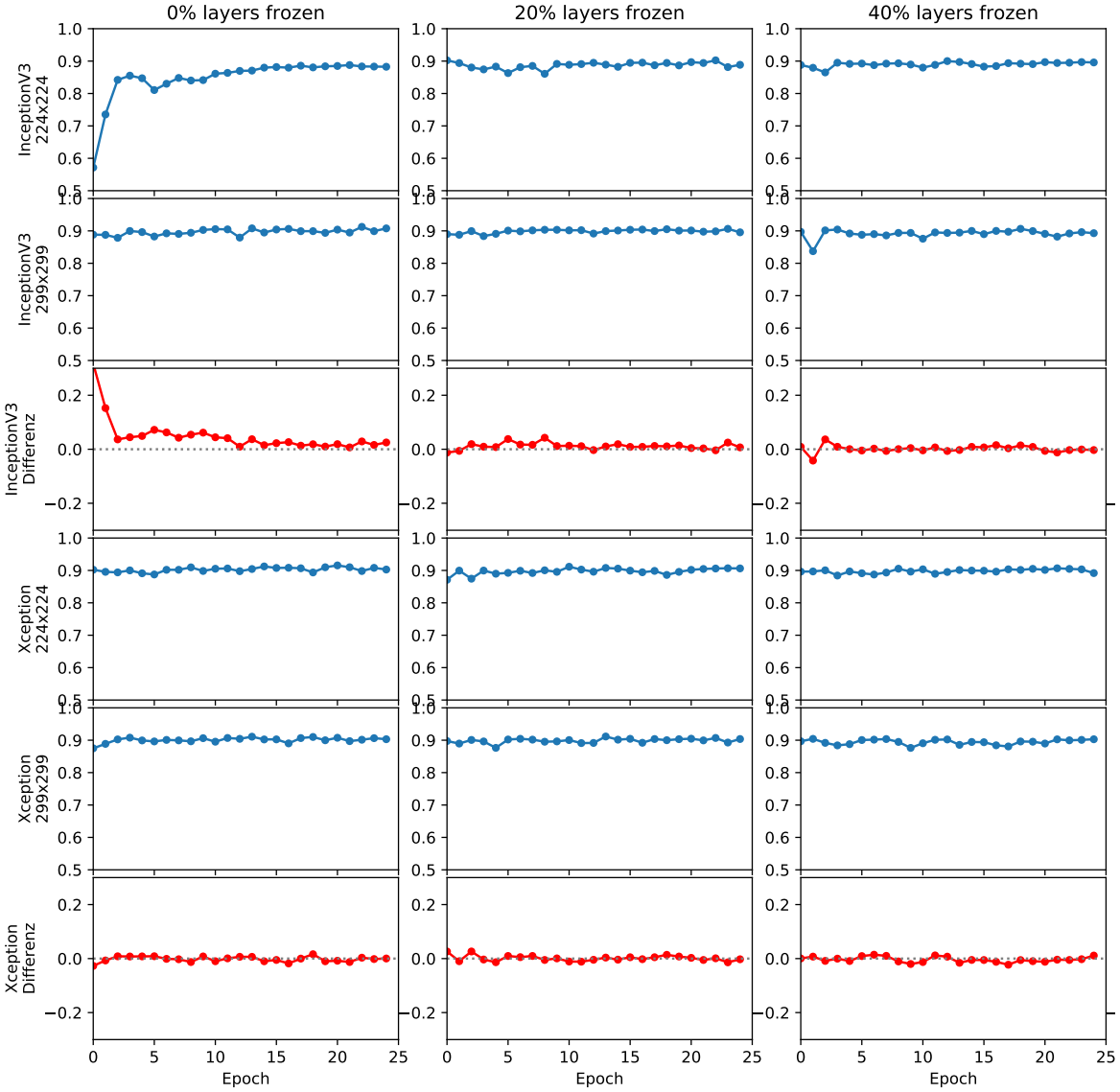


Abbildung 5.1: Performance mit Patchgröße 224×224 vs. 299×299 mit Zeropadding

Zu sehen ist der Verlauf des Validation-Test-F1-Scores für das InceptionNetV3 (blau, obere Hälfte) und das XceptionNet (blau, untere Hälfte) jeweils mit und ohne Zeropadding. Wie an der Differenz der Kurven mit und ohne Zeropadding (rot) gut zu erkennen ist, führt die geringere Inputgröße zu keinen nennenswerten Performanceeinbußen. Lediglich das Training des InceptionNets ohne eingefrorene Schichten verläuft geringfügig schneller.

5.1.3 Lineare Trennbarkeit der ImageNet-Klassifikatorausgabe

Wie in Abschnitt 4.7.5 beschrieben, ist es möglich, ein Transfer Learning durchzuführen und sämtliche Gewichte, selbst die der vollverschalteten Schichten, zu übernehmen. Voraussetzung dafür ist allerdings, dass zumindest ein Neuron in dem originalen, vortrainierten Netz existiert, dessen Ausgabe für beide Klassen trennbar ist. Im Detail bedeutet dies, dass der ganze GAPs-Trainingsdatensatz durch das vortrainierte Netz (inkl. ImageNet-Klassifikator) verarbeitet und überprüft wird, ob sich eines der 1000 Ausgabeneuronen für die Trennung eignet. Um die Eignung eines Neurons zu ermitteln, wird für jedes Neuron der Fisher-Score, wie in Abschnitt 4.7.5 beschrieben, mit den Daten der beiden GAPs-Klassen berechnet. Die besten Neuronen, entsprechend dem Fisher-Score, werden in einem Violinendiagramm, in Abbildung 5.2, visualisiert. Falls ein Neuron die beiden GAPs-Klassen gut trennt, was unabhängig von der Höhe der Aktivierung sein kann, so sollten die Clustermittelpunkte auf der Aktivierungs-Achse deutlich versetzt sein. Für die Ermittlung des Fisher-Scores wurden die GAPs-Daten in die jeweilige Inputkodierung transformiert, welche auch im Pretraining der entsprechenden Architektur genutzt wurde. Die ursprüngliche Inputkodierung aller Architekturen ist in Abschnitt 4.5.1 aufgelistet.

Ergebnis

Wie Abbildung 5.2 für das VGG19 zeigt, ist mit keinem Neuron eine zufriedenstellende Trennung der beiden Klassen möglich. Die Verteilung der übrigen Architekturen verhält sich ähnlich und ist im Anhang in Abschnitt A.1 zu finden.
Eine Übernahme der Gewichte der letzten vollverschalteten Schicht ist unter diesen Umständen für keine der verwendeten Architekturen sinnvoll. Da das VGG19 und AlexNet zwei vollverschaltete Schichten besitzen, wäre für diese Architekturen eine Übernahme der vorhergehenden vollverschalteten Schicht möglich. Aus Vergleichbarkeitsgründen wurde darauf jedoch, wie in Abschnitt 4.2 beschrieben, verzichtet. Aufgrund begrenzter Ressourcen war es nicht möglich, alle Daten für die Berechnung zu verwenden, weswegen zufällig 20000 der insgesamt 50000 GAPs-Trainingsbeispiele ausgewählt wurden.

Diskussion

Wie bereits beschrieben, konnten für keine Architektur ImageNet-Ausgabeneuronen ermittelt werden, die sich für eine Übernahme eignen. Insgesamt weist auch dies darauf hin, dass es sich bei den ImageNet- und GAPs-Daten um sehr unterschiedliche Daten handelt, die durch die ImageNet-Features nur schwer getrennt werden können.

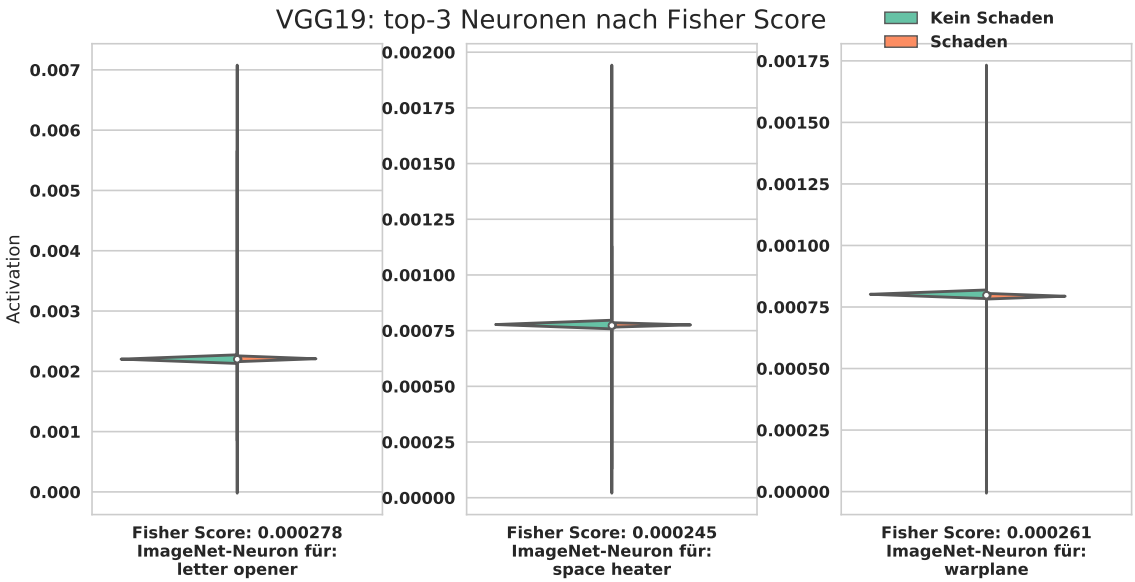


Abbildung 5.2: Ausgabeverteilung des VGG19

Zu sehen ist die Verteilung der Ausgabe für den GAPs Datensatz, der fünf besten Neuronen, entsprechend dem Fisher-Score. Es ist deutlich zu erkennen, dass keines der Neuronen dafür geeignet ist, die beiden Klassen voneinander zu trennen.

5.2 Trainingsergebnisse

Das erste Experiment bildet die Basis dieser Arbeit. Es wurden 7 Architekturen mit unterschiedlichen Lernraten und in jeweils sechs verschiedenen Stufen der Einfrierung trainiert. Insgesamt wurden 426 Trainingsdurchläufe mit jeweils 25 Epochen durchgeführt. Für jede Epoche wurden jeweils auf dem Validierungs- und Validierungs-Test-Datensatz der F1-Score, die Balanced Error Rate (BER), die Accurarcy (ACC) und die Precision-Recall-Kurve berechnet.
Der Übersicht halber wurde darauf verzichtet, sämtliche Diagramme aufzuführen und es werden stattdessen nur die wesentlichen gezeigt. Eine vollständige Auflistung aller Diagramme ist im Anhang ab Abschnitt A.2 zu finden. Auf Basis der Trainingsergebnisse wurden verschiedene Auswertungen durchgeführt, deren Ergebnisse im Nachfolgenden präsentiert werden.

5.2.1 Teil 1: Auswahl und Leistung der Baseline

Versuchsaufbau

Um eine Vergleichbarkeit zu ermöglichen, wurden zusätzlich zu jeder Architektur jeweils drei Trainingsdurchläufe mit zufällig initialisierten Gewichten, also ohne Transfer Learning, durchgeführt - nachfolgend auch Baseline genannt. Abbildung 5.3 zeigt den Trainingsverlauf jeweils dreier Baseline-Trainingsdurchläufe. Es ist gut zu erkennen, dass die Performance der Baseline zwischen allen Durchläufen nur minimal schwankt. Für alle nachfolgenden Vergleiche wird daher lediglich die beste Epoche, nach Validierungs-Test-F1-Score, verwendet. Zu jeder dieser sieben besten Epochen der Baseline wird, analog zu Teil 3, die PR-Kurve auf Validierungs-, Validierungs-Test- und Testdatensatz ermittelt. Abbildung 5.6 präsentiert beispielhaft die Ergebnisse für das InceptionNet.

Ergebnisse

Abbildung 5.3 zeigt die drei durchgeführten Trainingsdurchläufe der Baseline für das MobileNet und XceptionNet. Wie gut zu erkennen ist, besitzen diese durch die zufällige Initialisierung geringfügige Abweichungen. Insgesamt ist die Performance der drei Trainings jedoch vergleichbar. Eine vollständige Auflistung aller Verläufe ist im Abschnitt A.4 zu finden. Für die weiteren Auswertungen wird einfachheitshalber lediglich der Trainingsdurchlauf mit dem besten Validierungs-Test-F1-Score betrachtet. Diese besten Netze der Baseline wurden für eine bessere Vergleichbarkeit ebenfalls in den PR-Kurven aus Teil 3 abgetragen.

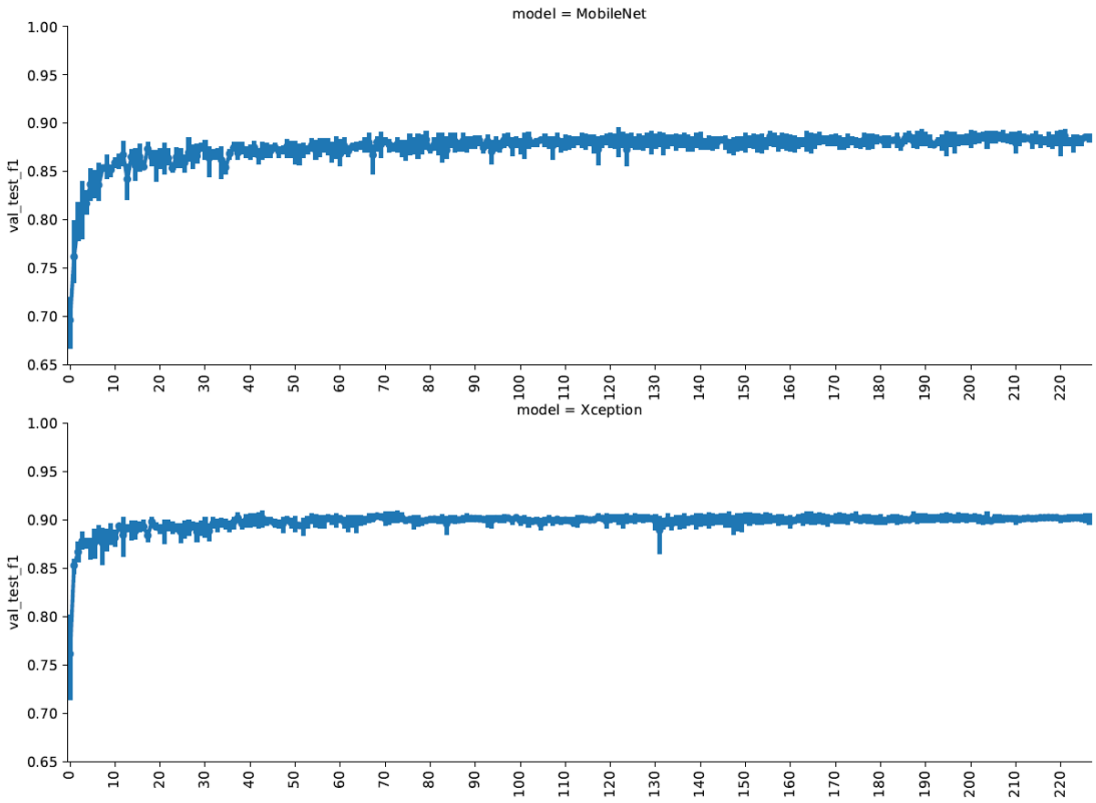


Abbildung 5.3: Trainingsverlauf der Baseline

Zu sehen ist der Verlauf von jeweils drei Trainingsdurchläufen der Baseline. Wie zu erkennen ist, schwankt die Leistung anfangs zunächst stärker, pendelt sich jedoch nach einigen Epochen ein. Unabhängig von der Architektur kann ein ähnlicher Verlauf festgestellt werden. Die Bestleistung aller drei Durchläufe unterscheidet sich lediglich minimal.

Diskussion

Wie an den Trainingsverläufen in Abbildung 5.3 zu erkennen ist, fällt der Einfluss verschiedener zufälliger Initialisierungen relativ gering aus. Zwar sind zu Beginn des Trainings noch stärkere Schwankungen möglich, doch sobald ein Plateau erreicht wurde, sind die Abweichungen nur noch gering vorhanden.

5.2.2 Teil 2: Auswahl der besten Epoche eines jeden Trainings

Versuchsaufbau

Aus allen 426 Trainings wurde die jeweils beste Epoche anhand des F1-Scores auf dem Validierungs-Test-Datensatz ermittelt. Zur Visualisierung der großen Menge an Ergebnissen wurde eine Heatmap, s. Abbildung 5.4, gewählt. In der Heatmap können die verschiedenen Konfigurationen aus Architektur, Lernrate und eingefrorenen Schichten direkt miteinander verglichen werden.

Ergebnisse

Abbildung 5.4 zeigt die Evaluierung der jeweils besten Epoche aller Trainingsdurchläufe auf Basis des F1-Scores. Es ist deutlich zu erkennen, dass sowohl eine zu hohe oder zu niedrige Lernrate als auch eine zu hohe Anzahl eingefrorener Schichten zu einer schlechten Leistung führen. Bei genauerer Betrachtung der Heatmap ohne Skalierung (s. Anhang A.6) fällt auf, dass die Netze mit sehr schlechter Leistung gegen einen Accuracy-Wert von $0,6$ bzw. F1-Score von $0,571$ konvergieren. Durch den 60-zu-40 unbalancierten Datensatz entspricht dies der Wahrscheinlichkeit bei zufälligem Raten. Darüber hinaus ist insbesondere in der Heatmap des Validierungs-Datensatzes (s. Anhang A.4) deutlich zu erkennen, dass eine erhöhte Lernrate in Kombination mit einer gewissen Menge eingefrorener Schichten ebenfalls zu guten Ergebnissen führen kann. Des Weiteren ist zu erkennen, dass sowohl das AlexNet als auch das SE-ResNet50 vergleichsweise schlecht abgeschnitten haben. Bei diesen Netzen ist nicht nur die Anzahl der gut abschneidenden Konfigurationen sehr beschränkt, sondern auch die Performance insgesamt.

Diskussion

Wie mithilfe der Heatmap in Abbildung 5.4 gut zu erkennen ist, verhält sich das Training wie erwartet. Friert man zu viele Schichten ein, bricht die Klassifikationsleistung ein oder der Klassifikator rät nur noch. Wird die Lernrate zu hoch oder zu gering gewählt, verläuft das Training ebenfalls ungünstig. Weitere aufgenommene

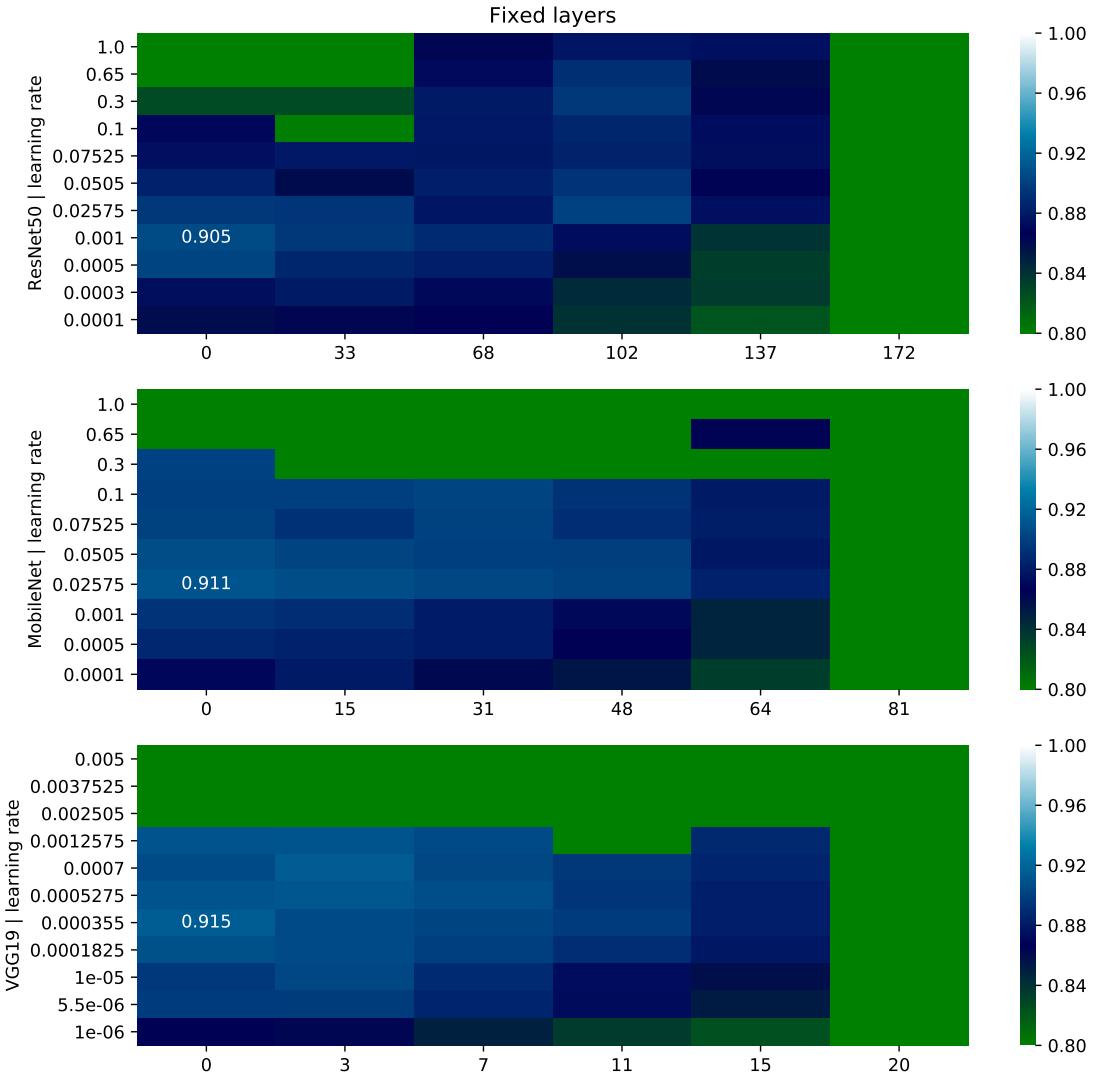


Abbildung 5.4: Performance-Heatmap des Validation-Test-F1-Scores

Zu sehen ist die Höhe des F1-Scores auf dem Validation-Test-Datensatz der jeweils besten Epoche eines Trainingsdurchlaufes. Grün symbolisiert eine schlechte Leistung und blau bzw. weiß eine sehr gute. Darüber hinaus ist zu jeder Architektur die Konfiguration aus Lernrate und eingefrorenen Schichten hervorgehoben, welche die Bestleistung über alle Konfigurationen auf dem F1-Score besitzt.

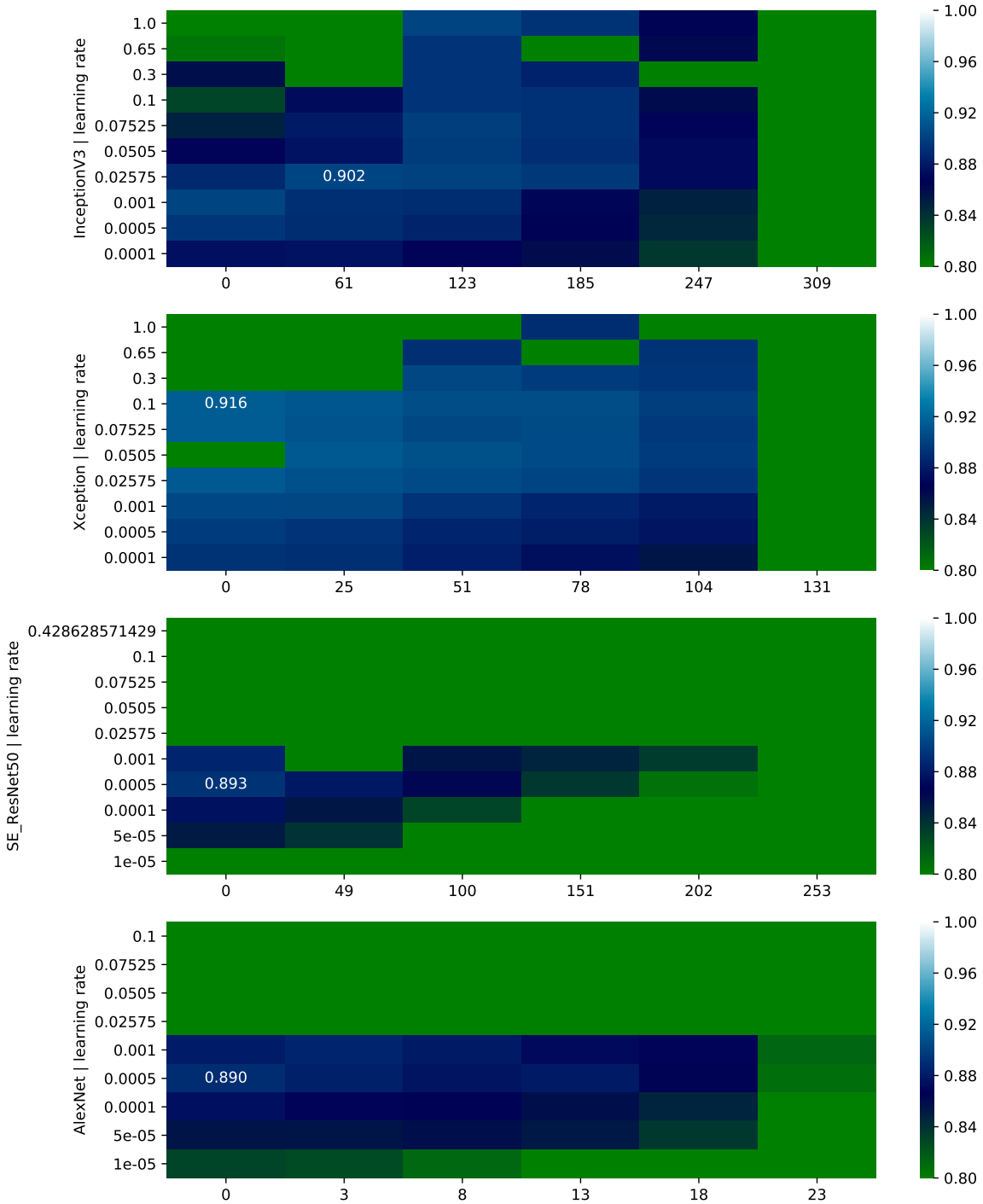


Abbildung 5.5: Performance-Heatmap des Validation-Test-F1-Scores

Zu sehen ist die Höhe des F1-Scores auf dem Validation-Test-Datensatz der jeweils besten Epoche eines Trainingsdurchlaufes. Die Farbkodierung und Hervorhebungen sind analog zu Abbildung 5.4.

Beobachtungen sind im Folgenden erläutert.

Ausreißer im Training:
Die Heatmap in Abbildung 5.4 zeigt einige Ausreißer. Diese Ausreißer sind Konfigurationen aus Lernrate und eingefrorenen Schichten, die zu einer sehr schlechten Leistung führen, wohingegen direkt umliegende Konfigurationen deutlich besser abschneiden. Das ist beispielsweise in der Heatmap des InceptionNets gut zu erkennen. Eine mögliche Ursache ist die Abfolge, in der die Schichten eingefroren wurden. Bei der Abfolge wurde keine Nebenläufigkeit im Netz beachtet. So kann es sein, dass durch das Einfrieren der Schichten in prozentualen Sprüngen Teile eines Inception-Moduls ungünstig eingefroren werden.
Eine weitere Ursache könnte sein, dass z.B. beim XceptionNet oder MobileNet nur die räumliche Faltung einer Separable Convolution oder beim SE-ResNet50 nur ein Teil des SE-Blocks eingefroren wurden. Die Problematik der Ausreißer ist für die in dieser Arbeit durchgeführten Experimente jedoch nicht entscheidend, da, außer beim InceptionNet, die beste Leistung stets in einer Konfiguration erreicht wurde, in der keine Schichten eingefroren waren. Des Weiteren sind die Ausreißer auch im Fall des InceptionNets unproblematisch, da die umliegenden Konfigurationen der Ausreißer schlechter abschneiden als die Bestleistung des InceptionNets. Es ist daher zu erwarten, dass auch die Konfiguration des Ausreißers wahrscheinlich keine bessere Leistung erzielt hätte.

Schlechte Leistung bei vollständigem Einfrieren der Faltungsschichten:
Auf Basis des schrittweisen Einfrierens der Schichten ist nicht erkennbar, dass das Finetuning ausschließlich eine reine Feature-Selektion bewirkt. Es wurde angenommen, dass der ImageNet-Datensatz eine sehr hohe Diversität besitzt und im Pretraining auch Features gelernt wurden, welche sich daher für die Kodierung von Straßendaten eignen. Sofern diese Annahme erfüllt wäre, hätten die Netze, selbst bei vollständig eingefrorenen Faltungsschichten, besser abschneiden können. Dies ist jedoch nicht der Fall, weshalb es sich um ein weiteres Indiz dafür handelt, dass die Klassifikation der GAPs-Daten sich von der ursprünglichen Aufgabe stark unterscheidet. Ähnliches

legen die Ergebnisse zur Trennbarkeit beider Datensätze aus Abschnitt 5.3.1 nahe. In [Kim et al., 2017] konnte ein VGG16, ebenfalls bei stark unterschiedlichem Zieldatensatz und vollständig eingefrorenen Faltungsschichten, dennoch eine Accuracy von 80% bis 85% erreichen. Offen bleibt in [Kim et al., 2017] jedoch, auf welchem Teil des verwendeten Zieldatensatzes die genannte Accuracy ermittelt wurde. Betrachtet man beispielsweise den Validierungs-Datensatz, so konnten bis auf das SE-ResNet50 alle Architekturen eine Accuracy von circa 80% erreichen. Das ResNet50, MobileNet und XceptionNet erreichten sogar ein wenig mehr. Zwischen Validierungs- und Trainingsdatensatz existiert jedoch eine Korrelation, wodurch mit einem guten Abschneiden zu rechnen ist. Abbildung A.4 zeigt zum Vergleich die Heatmap auf Basis der Validierungs-Accuracy. Abschnitt 5.3.5 geht auf den zuvor genannten Effekt der Feature-Selektion weiter ein.

Keine allgemeingültige Wahl der Lernrate möglich:
Noch entscheidender als die Option, manche Schichten einzufrieren, ist die richtige Wahl der Lernrate. Als grober Richtwert ist es üblich, das Finetuning mit einer Lernrate in Höhe eines Zehntels der im Pretraining verwendeten Lernrate zu beginnen. Leider konnten die im Pretraining verwendeten Lernraten nicht ermittelt werden. Geht man jedoch von dem gebräuchlichen Wert von 0.01 aus, ist anhand der Heatmap zu erkennen, dass ein Zehntel dieses Wertes nicht pauschal zur besten Performance führt. Viel mehr scheint die geeignetste Lernrate abhängig von der Architektur und der Anzahl eingefrorener Schichten zu sein.

Einfrieren der Schichten:
Bemerkenswert ist, dass beim MobileNet, VGG19, InceptionNetV3 und XceptionNet bis zu 60% der Schichten eingefroren werden können, ohne entscheidend an Performance einzubüßen. Auch bei dem ResNet50 existiert eine Konfiguration mit 60% eingefrorenen Schichten, welche kaum schlechtere Ergebnisse erzielt. Da das AlexNet kein solches Verhalten zeigt, legt dies die Vermutung nahe, dass die verwendeten Netze für das Klassifikationsproblem vergleichsweise zu groß sind.

5.2.3 Teil 3: Auswahl der besten Epoche einer Architektur

Versuchsaufbau

Aus den 426 besten Trainingsepochen aus Teil 1 wurde anhand des Validation-Test-F1-Scores zu jeder der sieben Architekturen die insgesamt beste Epoche ausgewählt. Jede dieser sieben Epochen stellte somit die beste Konfiguration aus Lernrate und eingefrorenen Schichten einer bestimmten Architektur dar. Für alle sieben ermittelten Klassifikatoren wurde darüber hinaus eine Auswertung auf dem Test-Datensatz durchgeführt. Abbildung 5.6 präsentiert beispielhaft die Ergebnisse für das InceptionNetV3. Eine vollständige Auflistung aller Diagramme ist im Anhang ab Abschnitt A.3 zu finden.

Ergebnisse

Abbildung 5.6 zeigt die PR-Kurve der besten Epoche aller Trainings des InceptionNetsV3. Es ist gut zu erkennen, wie positiv das Transfer Learning auf die Generalisierungsfähigkeit gewirkt hat. Durch das Transfer Learning konnte das Netz auf dem Test-Datensatz sogar besser abschneiden als auf dem Validierungs-Test-Datensatz. Mit dieser bemerkenswerten Leistung ist es unter allen Architekturen die mit dem besten F1-Score auf den Testdaten. In absteigender Reihenfolge folgen VGG19, ResNet50, MobileNet, Xception, AlexNet und SE-ResNet50. Eine ausführliche Aufstellung aller PR-Kurven kann Abschnitt A.3 entnommen werden. Tabelle 5.2 gibt einen Überblick über die Performance auf dem Test-Datensatz.
Vergleicht man die Leistung auf dem Test-F1-Score des Transfer Learnings mit der Baseline, fällt insbesondere die Leistungssteigerung durch Transfer Learning bei der InceptionNet-, MobileNet- und ResNet50-Architektur auf. Die Leistung beim VGG19 und SE-ResNet50 konnte moderat gesteigert werden und die Leistung des XceptionNet und AlexNet fiel annähernd identisch aus. Es ist dabei insbesondere zu erwähnen, dass der Leistungszuwachs durch Transfer Learning auf dem Test-Datensatz deutlich höher ausfiel als auf den anderen beiden Datensätzen. Der folgende Teil 4 präsentiert weitere Ergebnisse aus dem Vergleich zwischen Transfer Learning und Baseline.

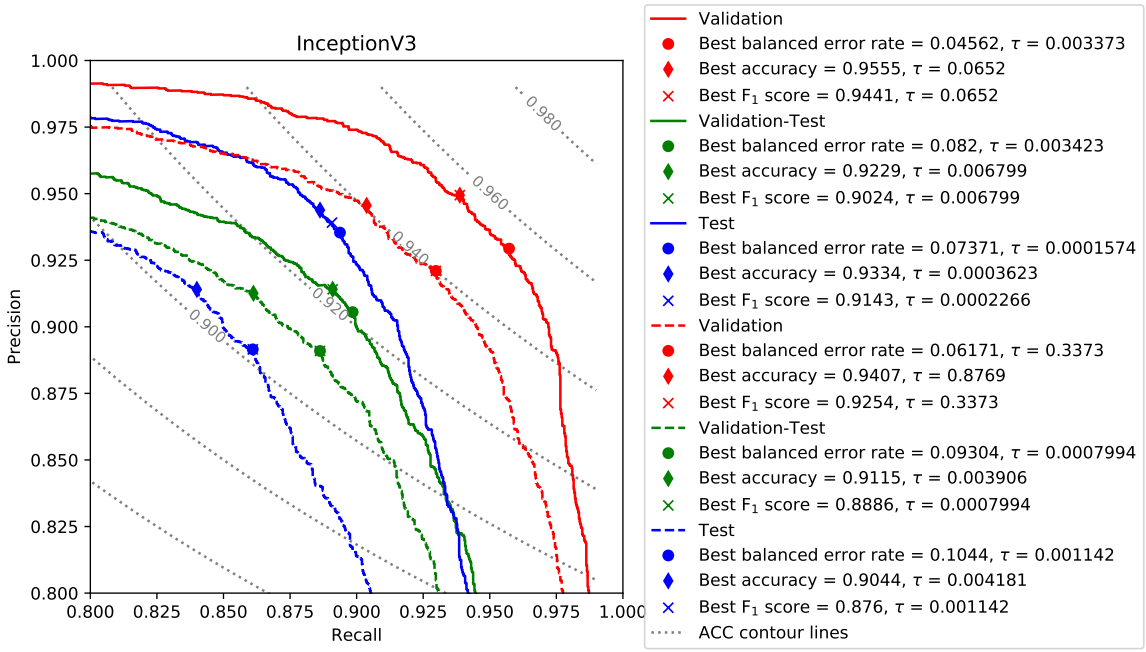


Abbildung 5.6: PR-Kurve: InceptionNet

Zu sehen ist die PR-Kurve aus dem Transfer Learning des InceptionNetV3 gegenüber der Baseline (gestrichelt). Insbesondere der große Performancegewinn auf dem Test-Datensatz ist auffällig.

Architektur	Test-ACC	Test-F1-Score	Test-BER
AlexNet	0.8946	0.8604	0.1181
InceptionNetV3	**0.9334**	**0.9143**	**0.07371**
ResNet50	0.9184	0.895	0.08942
SE-ResNet50	0.884	0.8566	0.1186
MobileNet	0.9109	0.8868	0.09546
Xception	0.9106	0.8842	0.09788
VGG19	0.9286	0.9065	0.0815

Tabelle 5.2: Performancevergleich aller Architekturen

Zu sehen ist die Auflistung der besten Balanced Error Rate, Accuracy und des F1-Scores auf dem Test-Datensatz einer jeden Architektur.

Diskussion

Wie bereits zuvor in diesem Abschnitt erläutert, konnte durch das Transfer Learning insbesondere die Leistung auf dem Test-Datensatz gesteigert werden. Außerdem sind die InceptionNet-, MobileNet- und ResNet50-Architektur durch den Leistungszugewinn durch Transfer Learning aufgefallen. Weitere aufgenommene Beobachtungen sind im Folgenden erläutert.

Kein erkennbarer Zusammenhang zwischen Architektur und Effektivität des Transfer Learnings:
Sowohl die Ergebnisse der PR-Kurven als auch die in Abbildung 5.7 gezeigten Ergebnisse sprechen insbesondere durch die häufig große Leistungssteigerung auf den Test-Daten deutlich für Transfer Learning. Unklar bleibt jedoch der genaue Zusammenhang zwischen der Architektur und der Effektivität des Transfer Learnings. Eine mögliche Erklärung für die vergleichsweise schlechte Leistung des SE-ResNet50 liegt in den gelernten Gewichten der SE-Blöcke. Wie Abbildung A.23 zeigt, müssen die Gewichte innerhalb des SE-Blocks stark umgelernt werden. Für das AlexNet könnte eine Erklärung sein, dass es nicht tief genug ist, um möglichst universelle Features auszuprägen. Außer hinsichtlich ihrer Tiefe unterscheidet sich die Architektur des AlexNet und die des VGG19 kaum.
Die Hintergründe, warum das XceptionNet schlecht abschneidet, bleiben jedoch unklar. Es besitzt keine Besonderheiten, die nicht auch im ResNet50 bzw. im MobileNet eingesetzt werden. Möglicherweise ist es die Kombination aus Res-Blöcken und Separable Convolutions, die zu dem vergleichsweise schlechten Ergebnis führen. Eine weitere Möglichkeit hätte grundsätzlich die Inputskalierung darstellen können. Es wäre denkbar, dass Features, welche mit anderer Inputskalierung gelernt wurden, durch die Xception-Architektur nicht gut weiterverwendet werden können. Für das Pretraining des XceptionNets wurde jedoch eine Vorverarbeitung gewählt, welche die Daten ebenfalls zwischen -1 und $+1$ normiert, weshalb dieser Einfluss ausgeschlossen werden kann. Ein weiterer Grund hätte die geänderte Inputgröße sein können. Diese wurde für das Finetuning von 299×299 auf 224×224 reduziert, allerdings konnte im

Vorversuch in Abschnitt 5.1.2 kein davon ausgehender Einfluss festgestellt werden.
Ein weiteres Indiz für den schlechten Transfer der vom XceptionNet gelernten Features ist die hohe Änderung der Gewichte während des Finetunings. Diese wurden größtenteils zehnfach stärker verändert als die im Training from-scratch gelernten Features. Abschnitt 5.3.2 geht auf diesen Aspekt näher ein.
Nichtsdestotrotz erzielt das XceptionNet im F1-Score auf dem Test-Datensatz eine gute Leistung. Lediglich die Differenz zwischen Transfer Learning und Baseline ist gering.

Einfrieren von Faltungsschichten kann Vorteile haben:
Sehr bemerkenswert ist, dass das InceptionNetV3 als einzige Architektur die beste Generalisierungsfähigkeit bei einer Konfiguration mit 20% eingefrorenen Schichten besitzt. Da die beste Epoche auf den Validierungsdaten jedoch weiterhin auf einer Konfiguration ohne eingefrorene Schichten liegt, lässt sich folgern, dass die kleine Menge eingefrorener Schichten positiv auf die Generalisierungsfähigkeit wirkt. Offen bleibt jedoch, ob dieser positive Effekt auf die Inception-Architektur beschränkt ist oder generell auftritt. Möglicherweise sind die gewählten 20% Schritte zu groß und gute Konfigurationen der anderen Architekturen wurden übersprungen. Aus zeitlichen Gründen konnte jedoch kein Training in feineren Abstufungen durchgeführt werden.
Nichtsdestotrotz fällt der Performanceunterschied zwischen der Konfiguration mit 20% eingefrorenen Schichten und der ohne Einfrierung mit 0.0006 im Test-F1-Score sehr gering aus. Dennoch kann zumindest für die Inception-Architektur gefolgert werden, dass es nicht pauschal am besten ist, stets alle Schichten zu trainieren.
Für das MobileNet, VGG19 und XceptionNet können in der Heatmap ebenfalls Konfigurationen mit eingefrorenen Schichten beobachtet werden, welche annähernd identische Performance besitzen wie die ohne Einfrierung. Eine feinere Abstufung beim Einfrieren der Schichten wäre für diese Architekturen ebenfalls sehr interessant.

Teil 4: Vergleich Transfer Learning zur Baseline

Versuchsaufbau

Wie zuvor erwähnt, ist ein Ziel von Transfer Learning, die Generalisierungsfähigkeit zu steigern. Dazu werden die Ergebnisse aus Teil 2 und Teil 3 verglichen. Es wird die Differenz der Bestleistung des F1-Scores zwischen dem Transfer Learning und der Baseline gebildet. Auf diese Weise wurde der Performanceunterschied auf dem Validierungs-Test und Test-Datensatz ermittelt. Abbildung 5.7 präsentiert die Ergebnisse.

Ergebnisse

Abbildung 5.7 unterstreicht erneut den positiven Effekt von Transfer Learning auf die Generalisierungsfähigkeit. Es ist jeweils die Differenz des F1-Scores auf den beiden Datensätzen aufgetragen. Wie deutlich zu erkennen ist, konnte außer beim AlexNet die Leistung auf beiden Datensätzen gesteigert werden. Insbesondere auf dem Test-Datensatz ist durch Transfer Learning eine erhebliche Steigerung möglich.

Diskussion

Vergleicht man die Leistung der Baseline mit dem Transfer Learning auf den PR-Kurven und die Ergebnisse aus Abbildung 5.7, ist der Erfolg von Transfer Learning deutlich zu erkennen. Bei keiner der Architekturen konnte auf dem Test-Datensatz ein negativer Einfluss durch Transfer Learning festgestellt werden. Die Aussage, dass ein Transfer Learning grundsätzlich zu empfehlen ist, sofern eine der Standardarchitekturen eingesetzt wird, wie sie beispielsweise in [Yosinski et al., 2014] getroffen wurde, kann daher unterstützt werden.

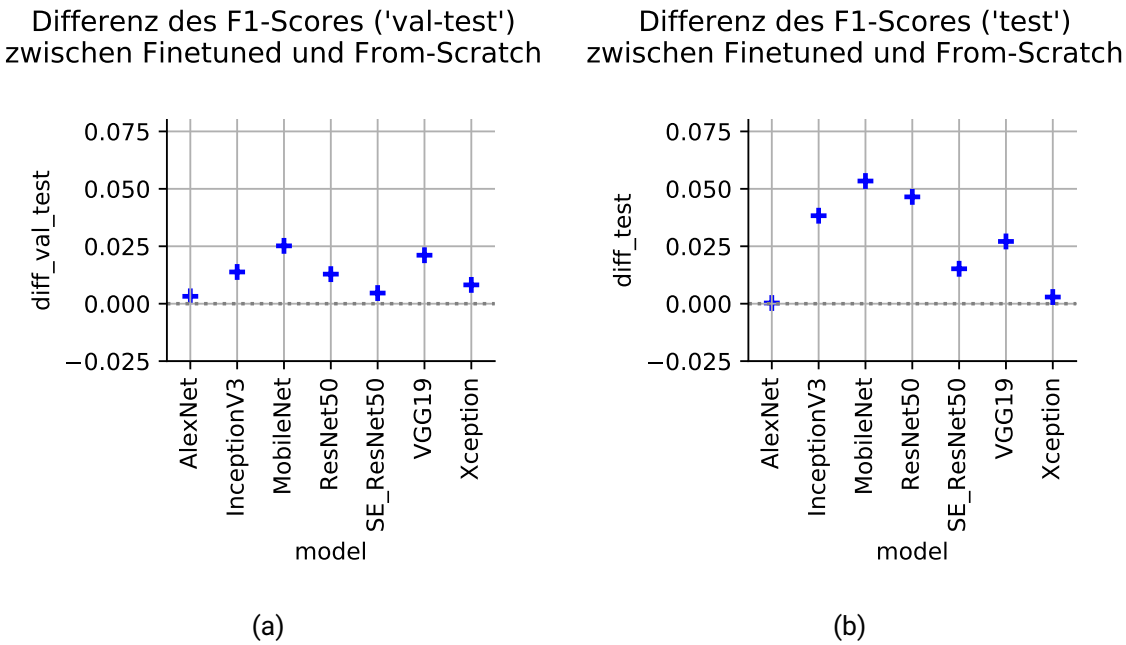


Abbildung 5.7: Performancevergleich: Transfer Learning zur Baseline

Zu sehen ist die Differenz des F1-Scores auf verschiedenen Datensätzen. Es ist gut zu erkennen, dass der Performancegewinn auf dem Validierungs-Test-Datensatz zunächst geringer ausfällt. Auf dem Test-Datensatz kann Transfer Learning im Vergleich jedoch erheblich besser abschneiden. Lediglich das AlexNet und das XceptionNet bilden dabei eine Ausnahme, dort ist die Leistung annähernd identisch mit der Baseline.

Teil 5: Bewertung der Trainingsdauer (Epochenanzahl)

Versuchsaufbau

Zur Untersuchung, ob Transfer Learning die Trainingsdauer positiv beeinflusst, wird insbesondere die Performance der ersten Epochen betrachtet. Abbildung 5.8 zeigt den Trainingsverlauf mit Transfer Learning und dagegen die ersten 25 Epochen der Baseline, für das InceptionNet und das SE-ResNet50.

Ergebnisse

Im direkten Vergleich der ersten 25 Trainingsepochen zwischen Transfer Learning und Baseline fällt auf, dass die meisten Netze schon nach lediglich einer Epoche Training annähernd Bestleistung erreichen. Bis auf das SE-ResNet50 und das AlexNet können alle Architekturen nach nur einer Epoche zumindest die Bestleistung der Baseline erreichen bzw. übertreffen. Nichtsdestotrotz erreichen auch diese beiden Architekturen schneller ein Plateau als das Baselinetraining. Abbildung 5.8 zeigt den Verlauf des SE-ResNet50 und den des InceptionNetV3. Eine vollständige Auflistung aller Verläufe kann Abschnitt A.5 entnommen werden.

Diskussion

Wie dem Ergebnis aus Abbildung 5.8 entnommen werden kann, ist die Trainingsdauer durch Transfer Learning deutlich reduziert worden. Unabhängig von der Wiederverwendbarkeit der im Pretraining gelernten Features, welche unter anderem in Abschnitt 5.3.1 weiter diskutiert wird, wirkt die Übernahme als eine gut gewählte Initialisierung des Netzes. Wie bereits in Abschnitt 4.7.2 erläutert wird, kann durch diese gute Initialisierung deutlich schneller ein Optimum gefunden werden. Wie die Ergebnisse aus Abschnitt 5.2.3 zeigen, wird aufgrund dieser guten Initialisierung sogar ein besseres Optimum gefunden als bei einem Training from-scratch.

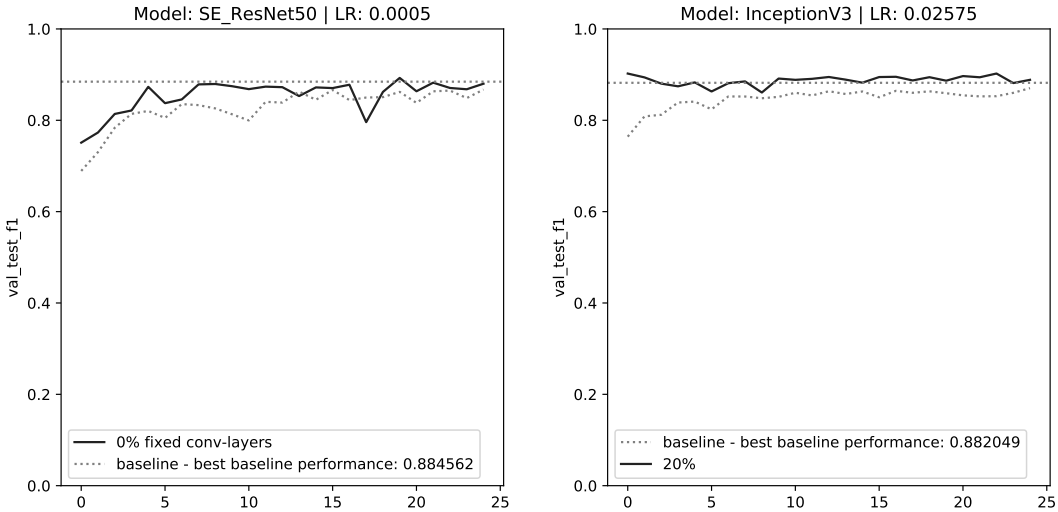


Abbildung 5.8: Trainingsverlauf mit Transfer Learning und Baseline

Zu sehen ist der Verlauf der ersten 25 Trainingsepochen des Transfer Learnings und der Baseline (gestrichelt) für das SE-ResNet50 (links) und das InceptionNetV3 (rechts). Die horizontale Linie repräsentiert dabei den Bestwert der Baseline. Das SE-ResNet50 ist dabei die Architektur, bei welcher der Geschwindigkeitsvorteil für das Training am geringsten ausfiel. Dennoch ist deutlich zu erkennen, dass das Transfer Learning gegenüber der Baseline, in Bezug auf die Trainingsdauer, besser abschneidet. Das Training des InceptionNetV3 entspricht hingegen dem besten Fall. Dort konnte bereits durch eine Epoche Training die Bestleistung der Baseline übertroffen werden.

5.3 Untersuchte Effekte

Wie zuvor in Abschnitt 4.8 beschrieben, ist das Ziel dieser Masterarbeit nicht nur die Evaluierung der reinen Performance des Transfer Learnings, es wird außerdem erwartet, dass mehrere Effekte beobachtet werden. Die erwarteten Effekte des Transfer Learnings sind:

- bessere Generalisierungsfähigkeit im Vergleich zu einem Training from-scratch
- schnell konvergierendes Training
- Feature-Selektion statt dem Neu-Lernen von Features, s. Abschnitt 5.3.3
- eine Leistungssteigerung, umso mehr Gewichte übernommen werden

Die Verbesserung der Generalisierungsfähigkeit durch das Transfer Learning im Vergleich zu einem Training from-scratch wurde bereits in Abschnitt 5.2.3 diskutiert. Ebenso wurde die für das Training notwendige Dauer bereits in Abschnitt 5.2.3 untersucht.
Der Effekt der Feature-Selektion beschreibt im Vergleich zu den zwei zuvor genannten Effekten einen komplexeren Zusammenhang. Im folgenden Abschnitt 5.3.3 wird dazu in zwei Teilexperimenten die Spärlichkeit der Featuremaps über alle Schichten der Netze insgesamt und, davon getrennt, die Spärlichkeit der Featuremaps der letzten Faltungsschicht untersucht. Darüber hinaus wurde in Abschnitt 5.3.2 untersucht, wie stark die aus dem Pretraining vorhandenen Features während des Finetunings verändert wurden. In dem Fall, dass sowohl die Spärlichkeit der Featuremaps steigt als auch die Features nach dem Finetuning nur wenig geändert werden, kann von einer Feature-Selektion gesprochen werden. Wie zuvor in Abschnitt 4.3.4 erläutert bzw. in [Yosinski et al., 2014] und [Mou et al., 2016] beobachtet wurde, ist die Übertragbarkeit der Features aus dem Pretraining abhängig von der Ähnlichkeit zwischen Ursprungs- und Zielaufgabe. Dementsprechend wird die Erwartung nahegelegt, dass falls die Aufgaben ähnlich sein sollten, die Netze auch ohne Finetuning, also mit vollständig eingefrorenen Faltungsschichten, eine gute Leistung erreichen werden. Wie in Abschnitt 5.2.2 festgestellt wurde, ist dies jedoch nicht der Fall. Um diese Beobachtung zu untermauern,

wird die Ähnlichkeit zwischen Ursprungs- und Zielaufgabe in Abschnitt 5.3.1 näher untersucht.

5.3.1 Trennbarkeit der Datensätze

Versuchsaufbau

Wie in [Yosinski et al., 2014] und [Mou et al., 2016] ermittelt wurde, ist die Übertragbarkeit der Features abhängig von der Ähnlichkeit der verwendeten Datensätze.
Sind sich Ziel- und Ursprungsdaten sehr ähnlich, ist auch die Übertragbarkeit der im Pretraining gelernten Features hoch. Um die Ähnlichkeit der beiden verwendeten Datensätze bewerten zu können, wurde auf Basis der auf ImageNet vortrainierten Architekturen ein Vergleich, wie in Abschnitt 4.3.4 beschrieben, erstellt.
Durch die Netze wurde jeweils der GAPs- und ImageNet-Validierungsdatensatz verarbeitet und die Ausgabe der letzten Faltungsschicht entnommen. Die Ausgabe der Netze entspricht den Featuremaps, welche von den vollverschalteten Schichten für die Klassifikation verwendet werden. Betrachtet man die Featuremaps als Vektor, so spannen diese einen Raum auf, in den jeder Input hinein übertragen wird. Vergleicht man zwei ähnliche Inputs, werden diese ähnliche gelernte Features ansprechen und somit auch an eine ähnliche Stelle in diesem Vektorraum übertragen. Im Umkehrschluss sollten unähnliche Inputs im Vektorraum auch weiter voneinander entfernt sein. Aus dieser Überlegung heraus kann versucht werden, den GAPs- und ImageNet-Datensatz als Cluster in diesem Raum zu trennen. Falls die Datensätze unähnlich sind, sollten sie daher auch in diesem Vektorraum separierbar sein. Für die Interpretation, ob entsprechende Cluster vorliegen, werden PCA und t-SNE verwendet, um den Vektorraum auf zwei Dimensionen zu transformieren.
Darüber hinaus wird in einem zweiten Schritt versucht, die Trennbarkeit beider Datensätze auch in einer der früheren Schichten nachzuweisen. Falls die im Pretraining gelernten Features gänzlich ungeeignet sein sollten, um die GAPs-Daten zu verarbeiten, sollten sich die beiden Datensätze auch in diesem Fall gut trennen lassen.
Aufgrund begrenzter Ressourcen und Zeit war es jedoch nicht möglich, alle Daten für die Berechnung der PCA- und t-SNE-Darstellung zu verwenden. Als Basis für die

PCA-Repräsentation der letzten Faltungsschicht dienten zufällige 10000 GAPs- und 10000 ImageNet-Beispiele aus den jeweiligen Validierungsdatensätzen. Für die t-SNE-Darstellung der letzten Faltungsschicht wurden jeweils 5000 zufällige Beispiele ausgewählt. Für die Berechnung der PCA- und t-SNE-Darstellungen der vorderen Schichten des VGG19 bzw. auch der anderen Netze musste noch stärker ausgewählt werden. Tabelle 5.3 zeigt eine Aufschlüsselung, wie viele Bildbeispiele für die Berechnung der verschiedenen Schichten des VGG19 verwendet werden konnten.

VGG19-Schicht	**PCA-Bildbeispiele je für ImageNet und GAPs**	**t-SNE-Bildbeispiele je für ImageNet und GAPs**
Schicht 22-17	5000	2500
Schicht 16-12	3250	1500
Schicht 11- 8	1600	900
Schicht 7	3000	1500
Schicht 6- 4	800	400
Schicht 3- 1	450	250

Tabelle 5.3: Anzahl zufällig ausgewählter Bildbeispiele für die PCA- bzw. t-SNE-Berechnung
Zu sehen ist die Aufschlüsselung, wie viele Bildbeispiele je aus den beiden Datensätzen verwendet wurden, um die PCA bzw. t-SNE zu berechnen. Die limitierende Größe für die Berechnung ist der begrenzte Arbeitsspeicher. Es ist gut zu erkennen, wie der Speicherbedarf pro Bildbeispiel in Richtung des Anfangs, also der Netzeingabe, steigt. Zurückzuführen ist der steigende Speicherbedarf auf die Größe der Featuremaps, die durch ein Bildbeispiel erzeugt werden, welche zu Beginn des Netzes noch deutlich größer sind als in der Nähe der Netzausgabe.

Ergebnis

Abbildung 5.10 zeigt die Visualisierung der ImageNet- und GAPs-Features innerhalb des zweidimensionalen PCA-Unterraums des ResNet50. Im Vergleich dazu zeigt Abbildung 5.9 eine durch t-SNE ermittelte Repräsentation. Wie in Abschnitt 2.4.2 be-

schrieben, gibt die t-SNE Visualisierung keine Auskunft über die Clustergrößen oder Entfernungen. Sie zeigt jedoch, analog zur PCA, dass es in dem hochdimensionalen Featureraum zwei Cluster gibt, welche eindeutig voneinander trennbar sind. Zu beachten ist, dass für die Berechnung der Featuremaps die Netze in der Konfiguration vor dem Finetuning verwendet wurden. Die Daten wurden daher entsprechend der im Pretraining verwendeten Inputkodierung vorverarbeitet.

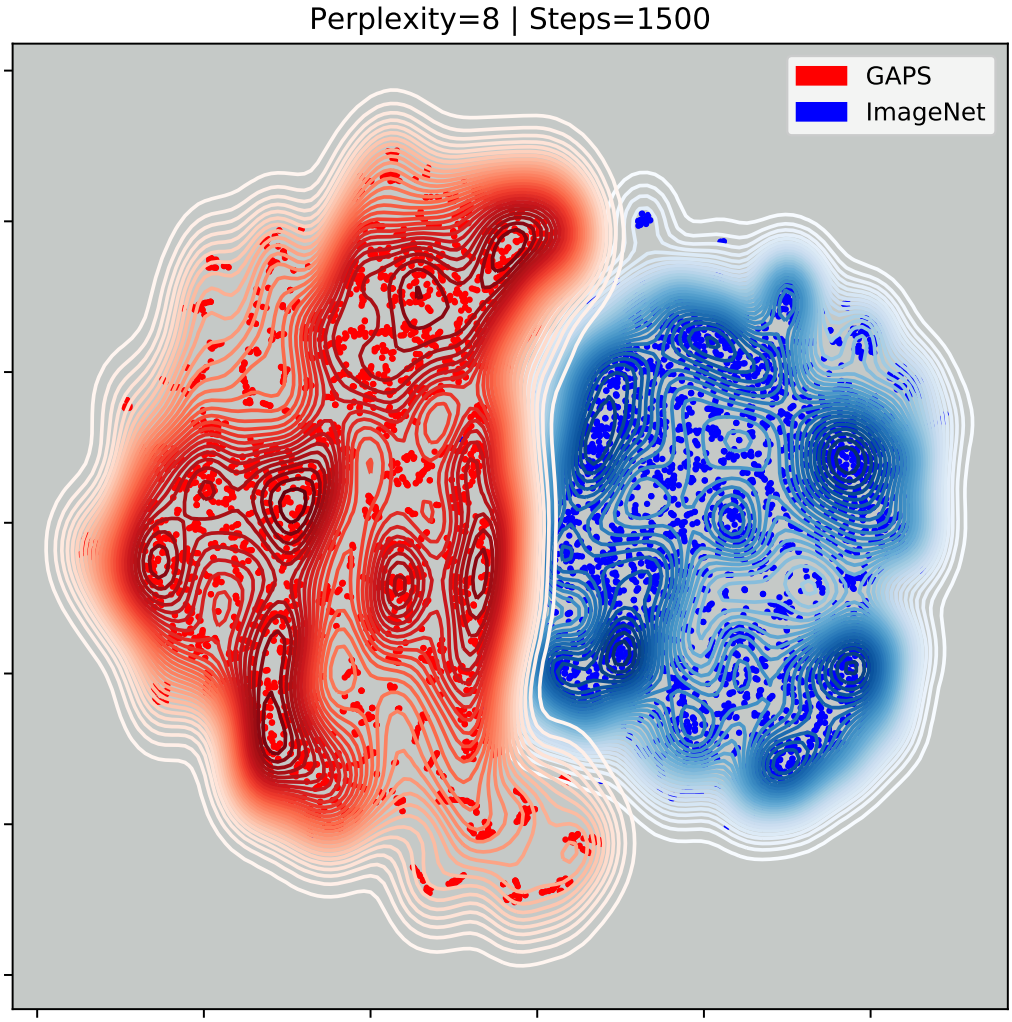


Abbildung 5.9: t-SNE-Darstellung der ResNet50-Features

Zu sehen ist eine zweidimensionale Dichtedarstellung der Features der GAPs- und ImageNet-Daten auf Basis des ResNet50. Es ist deutlich zu erkennen, dass beide Datensätze Cluster bilden, die sich nur am Rand leicht überlappen.

Sowohl die Abbildung der t-SNE-Repräsentation (s. Abbildung 5.9) als auch die der PCA (s. Abbildung 5.10) sind jeweils repräsentativ für fast alle Architekturen. Die GAPs-Daten bilden meist einen deutlich kleineren Cluster, der sich außerdem gut von dem der ImageNet-Daten trennen lässt. Bei der t-SNE-Darstellung lassen sich die Features sogar noch zuverlässiger separieren. Lediglich das SE-ResNet50 bildet eine Ausnahme. Weder in der t-SNE- noch in der PCA-Repräsentation lassen sich die GAPs- von den ImageNet-Daten separieren. Abbildung 5.11 zeigt die PCA-Darstellung des SE-ResNet50. Anhand der angegebenen, normierten Eigenwerte ist zu erkennen, dass

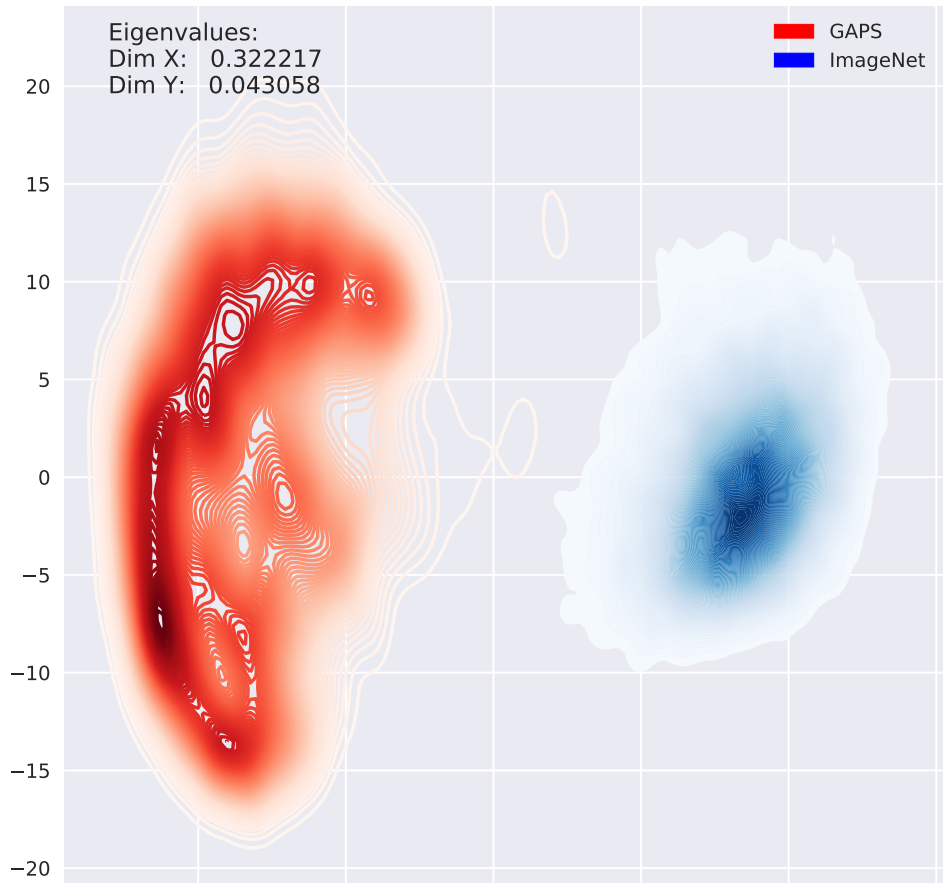


Abbildung 5.10: PCA-Darstellung der ResNet50-Features

Zu sehen ist eine zweidimensionale Dichtedarstellung der Features der GAPs- und ImageNet-Daten auf Basis des ResNet50. Es ist deutlich zu erkennen, dass beide Cluster problemlos trennbar sind. Darüber hinaus sind die auf Eins normierten Eigenwerte der ermittelten Eigenvektoren angegeben.

sich sämtliche Information bzw. Varianz fast ausschließlich entlang eines Eigenvektors befindet. Eine ausführliche Aufstellung aller Diagramme ist im Anhang in Abschnitt A.6 zu finden.

Betrachtet man anschließend die PCA- und t-SNE-Darstellung der Datensätze in früheren Schichten, fällt die Trennung schon deutlich schwieriger. Abbildung 5.12 zeigt die PCA-Darstellung der ersten Schichten des VGG19. Insbesondere die Verteilung nach den ersten Faltungsschichten ist repräsentativ für fast alle Architekturen. Während die ImageNet-Daten einen eher flächigen Cluster bilden, scheinen die GAPs-Daten entlang einer Achse verteilt zu sein, welche den ImageNet-Cluster seitlich schneidet. Lediglich

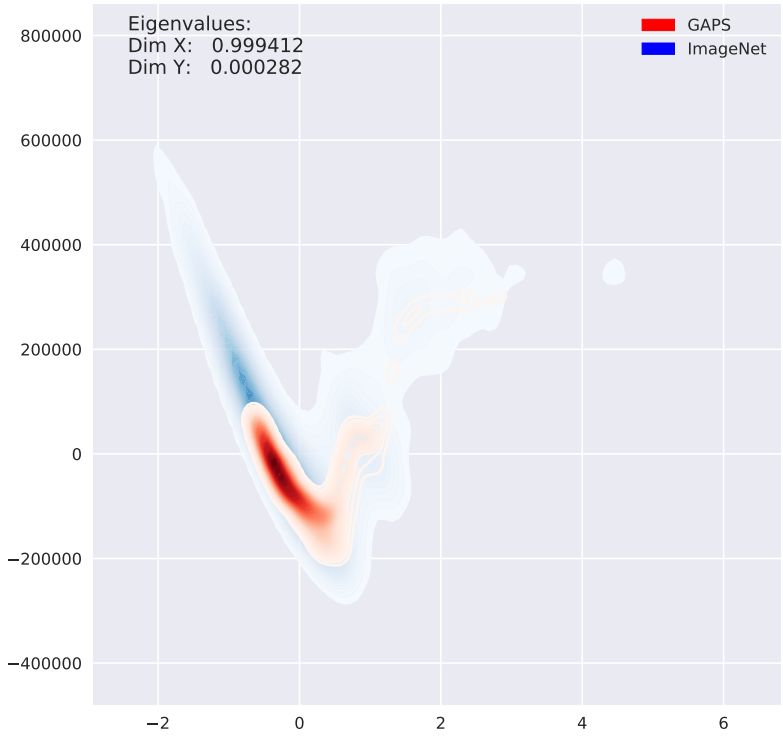


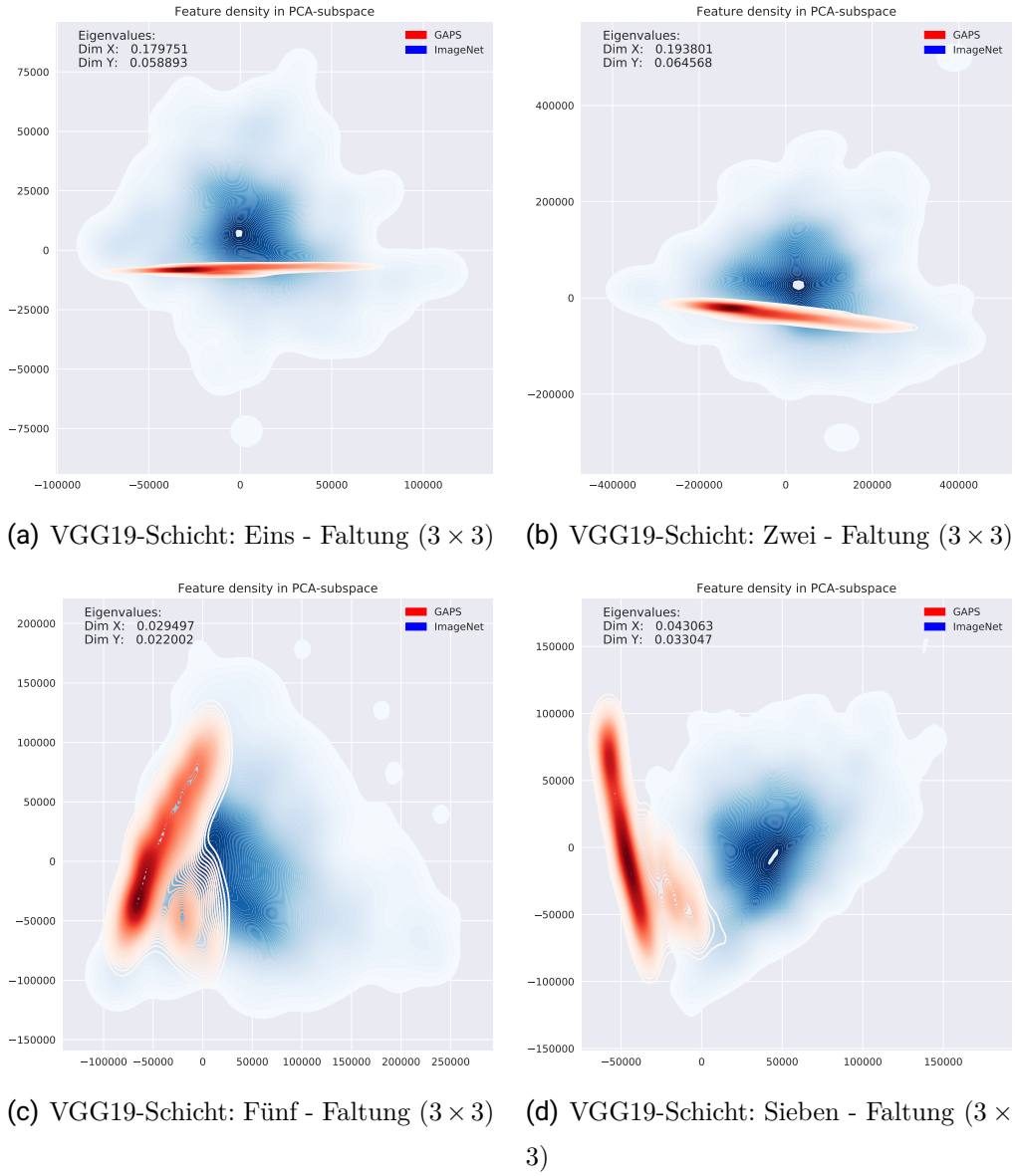
Abbildung 5.11: PCA-Darstellung der SE-ResNet50-Features

Zu sehen ist eine zweidimensionale Dichtedarstellung der Features der GAPs- und ImageNet-Daten auf Basis des SE-ResNet50. Darüber hinaus sind die auf Eins normierten Eigenwerte der ermittelten Eigenvektoren angegeben. Wie gut zu erkennen ist, überlappen sich beide Cluster, weshalb keine Trennung möglich ist. Auffällig ist jedoch, dass annähernd die gesamte Information entlang des ersten Eigenvektors vorliegt. Abschnitt 5.3.1 geht auf diese Beobachtung näher ein.

das ResNet50 bildet die beiden Datensätze so ab, dass sich beide Cluster auch in der ersten Faltungsschicht vollständig überlappen. Abbildung 5.13 zeigt ein Beispiel. Die durch die t-SNE gefundene Darstellung zeigt eine ähnliche Darstellung, weshalb auf die Abbildung der t-SNE-Repräsentationen verzichtet wurde.

Diskussion

Analog zu den Ergebnissen in [Kim et al., 2017] konnte festgestellt werden, dass sich die GAPs- und ImageNet-Daten meistens eindeutig durch die zuvor gelernten ImageNet-Features trennen lassen. Es lässt sich dementsprechend ein deutlicher Unterschied zwischen Ursprungs- und Zielaufgabe schlussfolgern.
Einzige Ausnahme für die Trennbarkeit ist das SE-ResNet50. Abbildung 5.11 zeigt, dass sich die Clusterzentren beider Architekturen in der PCA-Darstellung überlappen. Das gleiche Ergebnis kann auch in der t-SNE Darstellung vorgefunden werden. Mit Hinblick auf die Resultate aus dem Vorversuch in Abschnitt 5.1.3 ist dieses Resultat für das SE-ResNet50 wenig verwunderlich. Da die Netzausgabe des SE-ResNet50 annähernd keine Varianz besitzt, enthält der erste Eigenvektor der PCA-Darstellung ebenfalls 99,99% der Information. Es wird davon ausgegangen, dass wahrscheinlich die SE-Blöcke des Netzes für die lineare Verteilung der Features verantwortlich sind.
Betrachtet man die Ergebnisse der früheren Schichten des VGG19 in Abbildung 5.12, so fällt auf, dass erst nach etwa einem Drittel des Netzes, in Schicht Sieben, eine klare Trennung der beiden Datensätze möglich ist. Nichtsdestotrotz ist für alle Architekturen, außer für das ResNet50, selbst in der ersten Schicht ein deutlicher Unterschied in der Verteilung der Features beider Datensätze festzustellen. Da solch ein Unterschied selbst in der ersten Schicht der meisten Architekturen festgestellt werden kann, handelt es sich dabei um ein weiteres Indiz dafür, dass es sich bei den verwendeten Datensätzen um sehr unterschiedliche handelt.
Obwohl frühe Schichten eines Faltungsnetzes sehr allgemeine Features, wie z.B. Ecken- und Kanten-Features, ausprägen, befindet sich der Großteil der Features der GAPs-Daten lediglich auf einer Geraden. Diese lineare Verteilung spricht dafür, dass der GAPs-Datensatz nur eine vergleichsweise geringe Menge der Features der ersten



(a) VGG19-Schicht: Eins - Faltung (3×3)

(b) VGG19-Schicht: Zwei - Faltung (3×3)

(c) VGG19-Schicht: Fünf - Faltung (3×3)

(d) VGG19-Schicht: Sieben - Faltung (3×3)

Abbildung 5.12: PCA-Darstellung unterschiedlicher Schichten des VGG19

Zu sehen ist eine zweidimensionale Dichtedarstellung der Features der GAPs- und ImageNet-Daten auf Basis unterschiedlicher Schichten des VGG19. Darüber hinaus sind die auf Eins normierten Eigenwerte der ermittelten Eigenvektoren angegeben. Es ist gut zu erkennen, dass die GAPs- und ImageNet-Features erst ab Schicht Sieben eindeutig trennbar sind. Nichtsdestotrotz fällt auch in früheren Schichten die lineare Verteilung der GAPs-Features auf. Die Art der Verteilung der GAPs-Features lässt trotz Überlappung darauf schließen, dass eine deutlich geringere Menge der vortrainierten Features angesprochen wurde als von den ImageNet-Daten.

Schicht anspricht: ein weiteres Indiz für den Unterschied beider Datensätze.

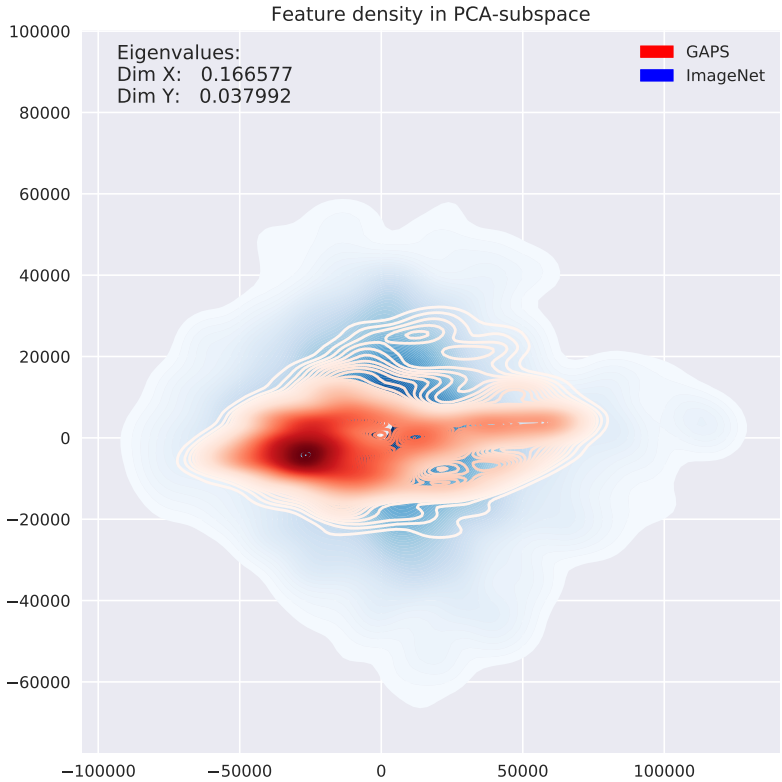


Abbildung 5.13: PCA-Darstellung der Features der ersten Schicht der ResNet50
Zu sehen ist eine zweidimensionale Dichtedarstellung der Features der GAPs- und ImageNet-Daten auf Basis der ersten Schicht des ResNet50. Darüber hinaus sind die auf Eins normierten Eigenwerte der ermittelten Eigenvektoren angegeben. Es ist gut zu erkennen, dass die GAPs- und ImageNet-Features durch die erste Faltungsschicht nicht trennbar kodiert werden.

5.3.2 Betragliche Änderung der Gewichte

Versuchsaufbau

Zur Bewertung des Effektes des Finetunings auf die bereits gelernten Features wird die l2-Norm der Gewichte vor und nach dem Finetuning wie folgt gebildet:

$$d_{n,l} = \frac{|w_{n,l,neu} - w_{n,l,alt}|}{|w_{n,l,alt}|} \quad | \; \forall Neuronen : n \quad \forall Schichten : l \tag{5.1}$$

Wie auch in [Kim et al., 2017] wird davon ausgegangen, dass falls die Abweichung vor und nach dem Finetuning gering ausfällt, nur eine geringe Anpassung der Features vorgenommen wurde. Features mit einer Änderung nahe Null sollten somit noch auf die gleichen Strukturen reagieren wie vor dem Finetuning. Darüber hinaus ist ebenfalls ein normierter Verlauf der l2-Norm abgebildet, welcher die Änderung in Relation zur Anzahl der Gewichte pro Filter setzt. Eine beispielhafte Darstellung für das VGG19 ist in Abbildung 5.14 zu sehen und die ausführliche Auflistung aller Diagramme ist im Anhang in Abschnitt A.7 zu finden.

Ergebnis

Abbildung 5.14 zeigt den Verlauf der Änderung der Gewichte über die Schichten des VGG19 für das Transfer Learning und die Baseline. Das Verhalten der Netze ist, sofern man den normierten Verlauf betrachtet, für alle Architekturen ähnlich: Die erste Schicht wird stärker angepasst, wie auch beim VGG19. Anschließend folgt eine moderate Abnahme der Änderung bis hin zu den Klassifikationsschichten. Dort steigt die Änderung erneut stark an. Weiterhin ist zu erkennen, dass, außer in den ersten Faltungsschichten, kaum Varianz in der Änderung vorliegt. Das führt zu der Erkenntnis, dass im Schnitt alle Filter betragsmäßig gleich stark verändert wurden. Eine weitere Regelmäßigkeit fällt bei Faltungsschichten mit einem 1×1 Filter auf, wie sie beispielsweise im MobileNet (s. Abschnitt A.7) vorkommen. Dort ist gut zu erkennen, dass diese Schichten verhältnismäßig deutlich stärker verändert werden als Faltungsschichten mit größeren Kernel. Wie bereits in Abschnitt 2.3.3 beschrieben, gewichten die 1×1 Korrelationen zwischen den Featuremaps. Die hohe Anpassung der Gewichte der 1×1 Filter ist daher auf eine Umgewichtung der Featuremaps zurückzuführen.

Diskussion

Wie bereits in Abschnitt 5.3.2 beschrieben, lässt sich der Verlauf der Gewichtsdifferenzen in drei Phasen unterteilen. Der Verlauf beginnt mit einer starken Gewichtsänderung in der ersten bzw. den ersten Schichten. Diese Änderung ist auf die Anpassung der Features auf den einkanaligen Input zurückzuführen. Im Pretraining mit den ImageNet-Daten wurden Farbbilder verwendet, welche dementsprechend auch Farbfeatures hervorgebracht haben, welche für den GAPs-Datensatz nicht benötigt werden. Der zweite starke Anstieg der Gewichtsänderung findet in den vollverschalteten Schichten statt. Diese werden auch für das Transfer Learning zufällig initialisiert und müssen dementsprechend auch stark angepasst werden.
Das dritte Merkmal ist der kontinuierlich leichte Abfall der normierten Gewichtsän-

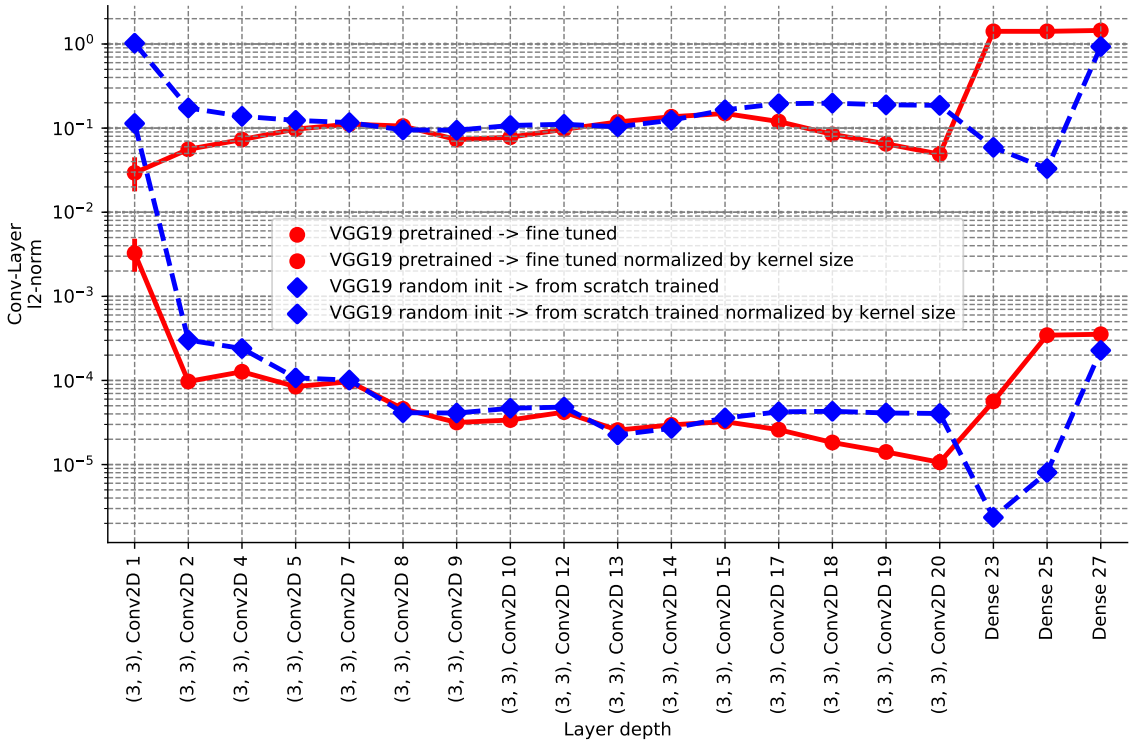


Abbildung 5.14: Gewichtsänderung des VGG19
Zu sehen ist der Verlauf der Änderung der Gewichte über die verschiedenen Schichten des VGG19. Besonders auffällig sind die hohen Änderungen am Anfang und am Ende des Netzes.

derung bzw. der leichte Anstieg der unnormierten. In einem analogen Experiment in [Kim et al., 2017] wurde für das VGG16 ein ähnlicher unnormierter Verlauf beobachtet. Abbildung 5.15 zeigt den Verlauf der Gewichtsänderung des VGG16 aus [Kim et al., 2017].

Die Schlussfolgerung in [Kim et al., 2017] ist, dass die vorderen Schichten des VGG16 ähnliche Features extrahieren, wie es schon im ImageNet-Datensatz der Fall war. Mit dem Wissen des normierten Verlaufs kann jedoch gesagt werden, dass dieser Anstieg lediglich auf die steigende Anzahl Gewichte je Schicht zurückzuführen ist. Darüber hinaus existiert an dieser Stelle ein leichter Widerspruch. In [Kim et al., 2017] wurde ebenfalls festgestellt, dass Ursprungs- und Zielaufgabe sehr verschieden sind. Einerseits wird aufgrund des großen Unterschieds argumentiert, dass die gelernten Features schlecht übertragbar sind, und andererseits wird in Bezug auf die Sparsity beschrieben, dass die Features sehr ähnliche Inhalte extrahieren.

Unabhängig davon konnte keine konkrete Ursache für das beobachtete Verhalten der

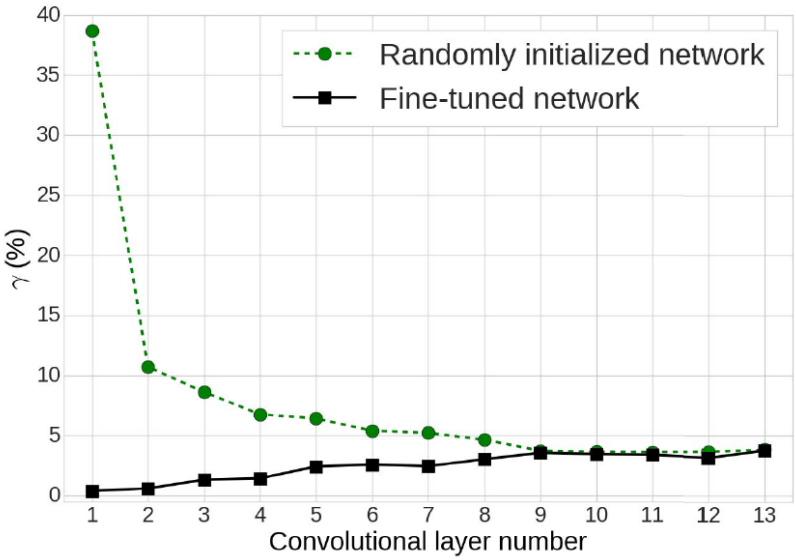


Abbildung 5.15: Gewichtsänderung des VGG16

Zu sehen ist die Gewichtsänderung des VGG16 aus [Kim et al., 2017]. Es ist gut zu erkennen, dass sich das zufällig initialisierte Netz (grün) insbesondere in den vorderen Schichten deutlich stärker anpasst. Das Netz aus dem Finetuning führt nach [Kim et al., 2017] hingegen nur geringe Anpassungen zur Featureselektion durch.

abfallenden Gewichtsänderungen bestimmt werden. Eine mögliche Erklärung ist, dass wider Erwarten die sehr allgemeinen Features der ersten Schichten ungeeignet sind, um die GAPs-Daten zu klassifizieren.
Betrachtet man den Verlauf des Trainings from-scratch, fällt auf, dass die Gewichte im Training ähnlich stark wie die im Finetuning verändert wurden. Im Falle des InceptionNetV3 und XceptionNet wurden die Gewichte durch das Finetuning sogar deutlich stärker geändert. Da insbesondere das InceptionNet im Test-F1-Score eine besonders gute Leistung erzielt hat, spricht die vergleichsweise große Gewichtsänderung gegen eine reine Feature-Selektion. Ähnliches wird ebenfalls durch die Ergebnisse aus Abschnitt 5.3.1 nahegelegt. Die unterschiedliche Verteilung der GAPs- und ImageNet-Features, selbst in den ersten Schichten der Netze, lässt darauf schließen, dass die GAPs-Daten einen deutlich kleineren Teil der vortrainierten Features ansprechen als die ImageNet-Daten.
Darüber hinaus könnte die im from-scratch Training und Finetuning gleichermaßen starke Änderung ein Zeichen dafür sein, dass die Netze zu groß sind und selbst die zufällig initialisierten Faltungsschichten teilweise ausreichend sind, um den GAPs-Datensatz zu klassifizieren. Sowohl im Training from-scratch als auch im Finetuning liegt die Änderung pro Gewicht bei meist unter 1%. Eine Möglichkeit dies weiter zu untersuchen, wäre ein Netz zu trainieren, und, im Gegensatz zu dem bisherigen Vorgehen, Faltungsschichten am Ende des Netzes für das Training einzufrieren. Falls selbst mit zufällig initialisierten, eingefrorenen Schichten ein Trainingserfolg erzielt werden kann, könnten die Architekturen für das gegebene Problem zu groß sein. Aus zeitlichen Gründen konnte dieser Ansatz jedoch nicht weiter verfolgt werden.
Eine weitere Beobachtung ist, dass der Verlauf des ResNet50, XceptionNet, MobileNet und SE-ResNet50 stark durch Peaks geprägt ist. Für das SE-ResNet50, welches den unruhigsten Verlauf besitzt, lassen sich die meisten Peaks auf das Umlernen der Gewichtung in den SE-Blöcken zurückführen. Alle vier besitzen unter anderem durch Seperable-Convolutions auch 1×1 Faltungen, welche ebenfalls als Peak erkennbar sind. Analog zu den SE-Blöcken ist es auch ihre Aufgabe, Korrelationen über die Kanäle hinweg zu bewerten. Aufgrund des stark unterschiedlichen Datensatzes ist es naheliegend, dass diese Bewertung der Kanäle neu gelernt werden muss. Weit weniger stark

können die Peaks bei den 1×1 Faltungen des InceptionNets beobachtet werden. Scheinbar interessant wäre es außerdem, die betragliche Änderung zwischen dem Training from-scratch und Finetuning zu bilden. Dadurch wäre es möglich, eine Aussage über die Ähnlichkeit der Features zu erhalten, welche im Finetuning und im Training from-scratch gelernt wurden. Problematisch ist jedoch, dass es zwar möglich ist, dass beide Netze sehr ähnliche Features ausprägen, diese in ihrer Reihenfolge innerhalb einer Schicht jedoch gänzlich anders angeordnet sein können. Die fehlende Information über die Anordnung würde zu einem kombinatorischen Problem führen, weshalb dieser Ansatz nicht weiter verfolgt wurde.

5.3.3 Spärliche Featuremaps

Wie in [Kim et al., 2017] beschrieben wurde, sollte das Transfer Learning eine Art Feature-Selektion innerhalb des Netzes bewirken. Um diese Eigenschaft zu überprüfen, wird die Aktivität der Featuremaps vor und nach dem Finetuning betrachtet. Die gesamte Untersuchung wurde exemplarisch für zehn Bildbeispiele (fünf "Kein Schaden", fünf "Schaden") zu jeder Architektur durchgeführt und ist in zwei Abschnitte unterteilt. Im ersten Teil wird die Aktivierung der letzten Featuremap, welche zur Klassifikation genutzt wird, qualitativ, in Bezug auf eine Feature-Selektion, bewertet. Anschließend wird im zweiten Teil die Sparsity aller Schichten der Architekturen betrachtet. In Abschnitt 5.3.3 wird schon vorab eine Zusammenfassung der Erkenntnisse aus beiden Teilen gegeben.

Zusammenfassung

- Netze lernen Features, welche Schadstellen finden, und erzeugen auch an der korrespondierenden Stelle der Featuremap eine hohe Aktivität.
- Bei der Erkennung von schadfreien Bildern ist in erster Linie die Bildmitte relevant.
- Es konnte kein eindeutiger Hinweis gefunden werden, dass eine reine Feature-Selektion, wie beispielsweise in [Kim et al., 2017], durchgeführt wurde.

- Der Verlauf der Sparsity ist abhängig von der Architektur.
- In Richtung der Ausgabeschichten der Netze konnte insgesamt ein Anstieg der Sparsity beobachtet werden. Die Höhe der Sparsity ist jedoch abhängig von der Architektur.
- Die Ergebnisse aus [Kim et al., 2017] konnten bedingt reproduziert werden. Die in [Kim et al., 2017] gezogenen Schlussfolgerungen sind jedoch nicht übertragbar.

5.3.4 Spärliche Featuremaps: Teil 1 - Visualisierung der letzten Featuremap

Versuchsaufbau

Im ersten Schritt wird die Ausgabe der jeweils letzten Schicht vor den Klassifikationsschichten eines Netzes für ein gegebenes Bildbeispiel entnommen und, wie in Pseudocode 4.10 beschrieben, normiert. Die normierte Aktivierung der Featuremaps wird anschließend blockweise als Bild visualisiert. Das detaillierte Vorgehen wird in Abschnitt 4.7.4 erläutert. Abbildung 5.19 zeigt die Visualisierung der normierten Ausgabe des InceptionNetV3 vor und nach dem Finetuning. Weiße Pixel einer Featuremap symbolisieren eine hohe und dunkle Pixel eine geringe Aktivität.

Ergebnis

Im Gegensatz zu den bisherigen Ergebnissen ist es bezüglich der Featureaktivität deutlich schwieriger, Gemeinsamkeiten in den Ergebnissen zu entdecken. Betrachtet man zunächst nur Samples ohne Schaden, dann lassen sich jedoch drei Gruppen bilden. Zur ersten Gruppe gehören das XceptionNet, das MobileNet und das ResNet50. Alle drei Netze zeichnen sich dadurch aus, dass sie Samples ohne Schaden durch Features kodieren, welche sehr spärlich belegt sind. Falls auf einer Featuremap eine Aktivierung vorliegt, dann bildet diese lediglich einen einzelnen, mittig zentrierten Hotspot. Darüber hinaus kann festgestellt werden, dass das Finetuning, außer beim ResNet50, zu einer Reduzierung der Aktivität geführt hat. Abbildung 5.16 zeigt ein Beispiel des MobileNets.

Zur zweiten Gruppe gehören das SE-ResNet50 und das InceptionNet. Diese prägen ebenfalls Features aus, welche bei Samples ohne Schaden einen mittigen Hotspot ermitteln. Darüber hinaus zeigen beide Architekturen jedoch auch verschiedene andere Muster in den Aktivitäten. Allgemein generiert das SE-ResNet50 bei Samples ohne Schaden jedoch eher Featuremaps, welche gleichmäßig hoch aktiviert oder komplett inaktiv sind. Bei dem InceptionNetV3 sind hingegen eher einzelne Hotspots zu erken-

nen. Abbildung 5.19 zeigt ein Beispiel des InceptionNets.
Zur dritten Gruppe gehören das VGG19 und das AlexNet, welche beide durch eine besonders hohe Inaktivität auffallen. Weiterhin ist zu erkennen, dass das AlexNet ebenfalls mittige Hotspots zeigt. Abbildung 5.18 zeigt ein Beispiel des VGG19.

Betrachtet man anschließend Samples mit einem Straßenschaden, lassen sich das ResNet50, das MobileNet, das AlexNet und das VGG19 gut zusammenfassen. Die Features der vier Architekturen weisen bei Samples mit Schaden hauptsächlich Aktivität entlang der Kontur des Straßenschadens auf und sind ansonsten kaum oder gar nicht aktiviert. "Entlang der Kontur" bezieht sich dabei auf den korrespondierenden Ort einer Featuremap in Bezug auf den Ort des Straßenschadens im Eingabebild, welcher

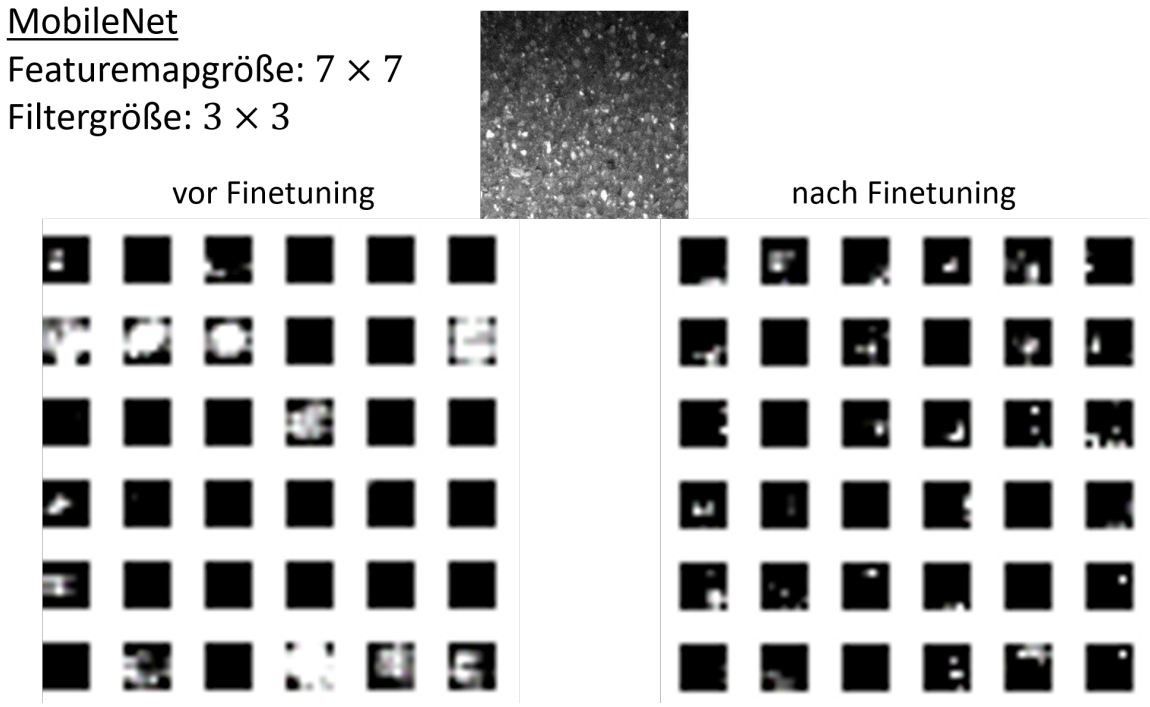


Abbildung 5.16: Featuremap-Visualisierung eines Samples ohne Schaden des MobileNet
Zu sehen ist ein repräsentativer Ausschnitt der Visualisierung der Aktivierung vor der Klassifikationsschicht des MobileNet. Es ist gut zu erkennen, dass die Sparsity für das gegebene Sample ohne Schaden nach dem Finetuning zugenommen hat.

die Aktivität auf der Featuremap hervorruft. Insgesamt konnte beobachtet werden, dass die Spärlichkeit durch das Finetuning leicht erhöht wurde. Abbildung 5.17 zeigt ein Beispiel des MobileNets.

Für das XceptionNet, InceptionNet und SE-ResNet50 lassen sich wenige Gemeinsamkeiten ableiten. Zwar besitzen alle drei Features, welche denen der vier anderen Architekturen teilweise ähnlich sind, allerdings weisen sie auch Unterschiede auf. Das XceptionNet fällt zum einen dadurch auf, dass unabhängig von der Position der Schadstelle auf dem Bildbeispiel in erster Linie eine hohe Aktivierung der oberen Pixel der Featuremaps auftreten. Im Gegensatz dazu scheint die Aktivierung der Featuremaps des InceptionNets stark davon abhängig zu sein, wie das Bildbeispiel ursprünglich aussah. Bildet der Straßenschaden auf dem Eingabebild beispielsweise eine kreuzförmige Struktur, so ist in den Featuremaps häufig ebenfalls eine kreuzförmige Struktur zu beobachten. Auffällig ist jedoch die abgegrenzte Verteilung der aktivierten Featuremaps

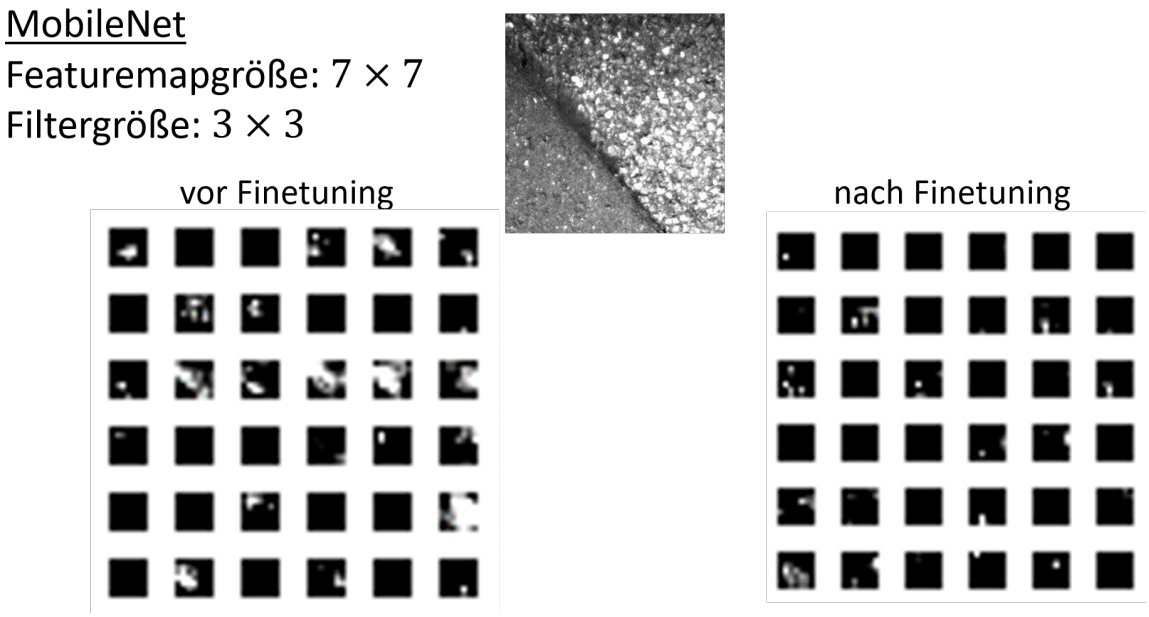


Abbildung 5.17: Featuremap-Visualisierung eines Samples mit Schaden des MobileNet

Zu sehen ist ein repräsentativer Ausschnitt der Visualisierung der Aktivierung vor der Klassifikationsschicht des MobileNet. Es ist gut zu erkennen, dass die Sparsity für das gegebene Sample mit Schaden nach dem Finetuning zugenommen hat.

des InceptionNetV3. Bildbeispiele mit Schadstelle führen beispielsweise hauptsächlich zu einer Aktivierung der Featuremaps des kurzen Faltungs- und Pooling-Branches. Abbildung 5.19 zeigt ein Beispiel zu dem InceptionNet. Das SE-ResNet50 fällt dagegen hauptsächlich durch die große Mischung verschieden aktivierter Featuremaps auf. Vergleicht man abschließend die Aktivierungen vor und nach dem Finetuning mit Hinblick auf eine Art Feature-Selektion, so kann keine allgemeingültige Aussage für alle Architekturen getroffen werden. Grundlegend existiert eher die Tendenz, dass nach dem Finetuning eine höhere Sparsity vorliegt, wie z.B. beim VGG19. Jedoch ist es beispielsweise beim ResNet50 der Fall, dass nach dem Finetuning keine Veränderung der Sparsity beobachtet werden konnte.

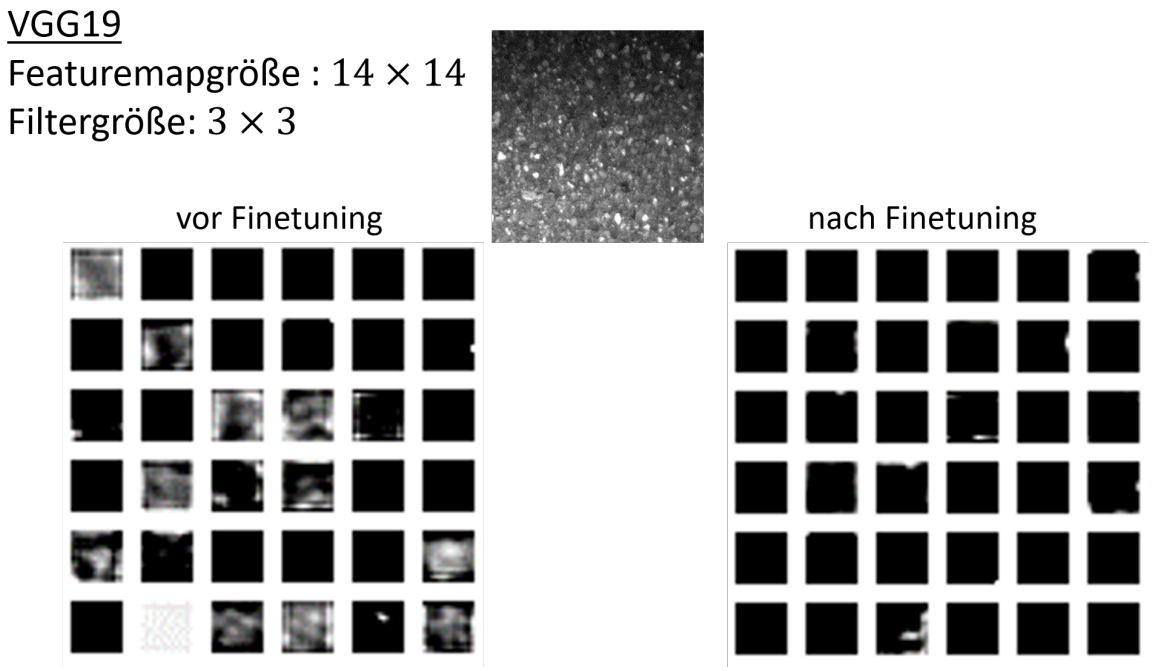


Abbildung 5.18: Featuremap-Visualisierung eines Samples ohne Schaden des VGG19

Zu sehen ist ein repräsentativer Ausschnitt der Visualisierung der Aktivierung vor den Klassifikationsschichten des VGG19. Es ist gut zu erkennen, dass die Sparsity für das gegebene Sample ohne Schaden nach dem Finetuning erheblich zugenommen hat.

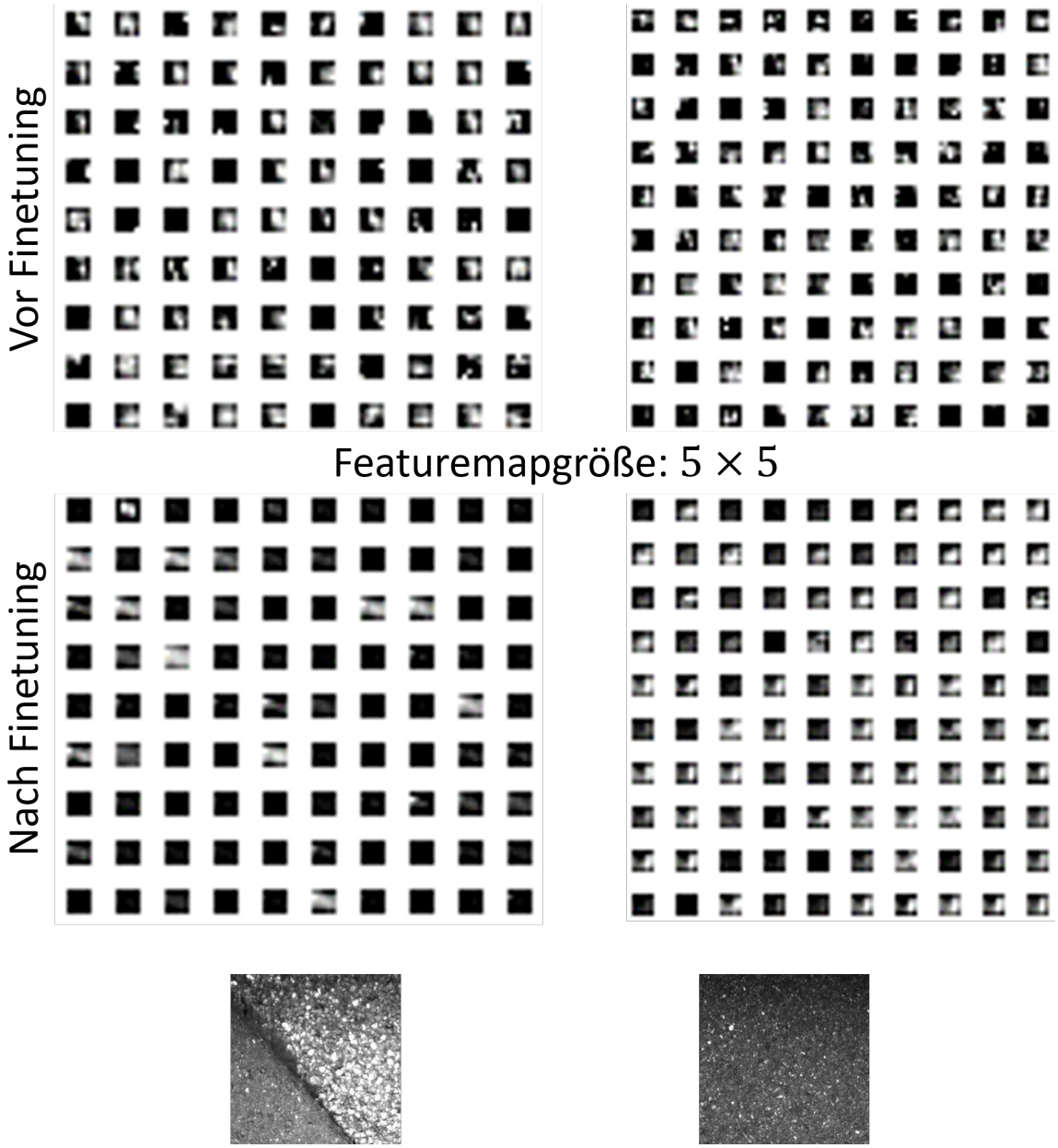


Abbildung 5.19: Featuremap-Visualisierung des InceptionNetV3

Zu sehen ist ein repräsentativer Ausschnitt der Visualisierung der Aktivierung vor den Klassifikationsschichten des InceptionNetV3. Für ein Sample mit Schaden ist gut zu erkennen, wie die Sparsity deutlich gestiegen ist. Im Gegensatz dazu ist die Aktivierung im Falle eines Samples ohne Schaden deutlich höher.

Diskussion

Abhängig von der Architektur tritt nach dem Finetuning die erwartete Reduktion der Aktivierung unterschiedlich stark auf. Insgesamt wurde jedoch festgestellt, dass die Sparsity eher steigt als fällt. Wie bereits in Abschnitt 5.3.4 beschrieben, konnte bei Beispielen ohne Schaden meist eine mittige Aktivierung beobachtet werden. Zurückzuführen ist dies auf das Vorgehen, wie die Bildausschnitte aus den Straßenbildern gewonnen wurden, und ist explizit gewünscht. Während des Ausschneidens der 224×224 Pixel großen Bildbeispiele aus den FullHD-Bildern wurde für die Bildbeispiele ohne Schaden lediglich darauf geachtet, dass die zentrale Region von 64×64 Pixeln keinen Schaden enthält. Umliegende Regionen können auch in Beispielen, welche als schadenfrei gelabelt sind, Schäden enthalten. Durch dieses Vorgehen wird für die Klassifikation der Samples ohne Schaden eine Fokussierung auf das Bildzentrum gelegt. Ziel ist es, damit die Klassifikationseigenschaften für den Einsatz in einer Segmentierung, wie in Abbildung 4.2 zu sehen, zu optimieren.
Beispiele mit Schaden hingegen führen eher, wie in Abschnitt 5.3.4 beschrieben, zu einer hohen Aktivierung entlang der Konturen des Schadens. Da die meisten Netze Ausgabefeaturemaps von gerade einmal 5×5 oder 7×7 Pixeln besitzen, ist die Interpretation des Ortes der Aktivierung jedoch nicht eindeutig. Beim AlexNet und VGG19, welche eine Größe von 12×12 bzw. 14×14 Pixeln besitzen, ist der Zusammenhang jedoch deutlicher zu erkennen.
Des Weiteren ist in den Featuremaps des InceptionNets häufig eine stärkere Aktivierung im unteren Pooling-Teil der Visualisierung (s. Abbildung 5.20) zu erkennen. Hervorgerufen wird dies durch den Aufbau des InceptionNets. Da die verschiedenen Branches unterschiedliche Filter und Filtergrößen besitzen, sprechen diese auch verschiedene Strukturen an. Abbildung 5.20 zeigt eine Einteilung der Featuremap zu den entsprechenden Branches.

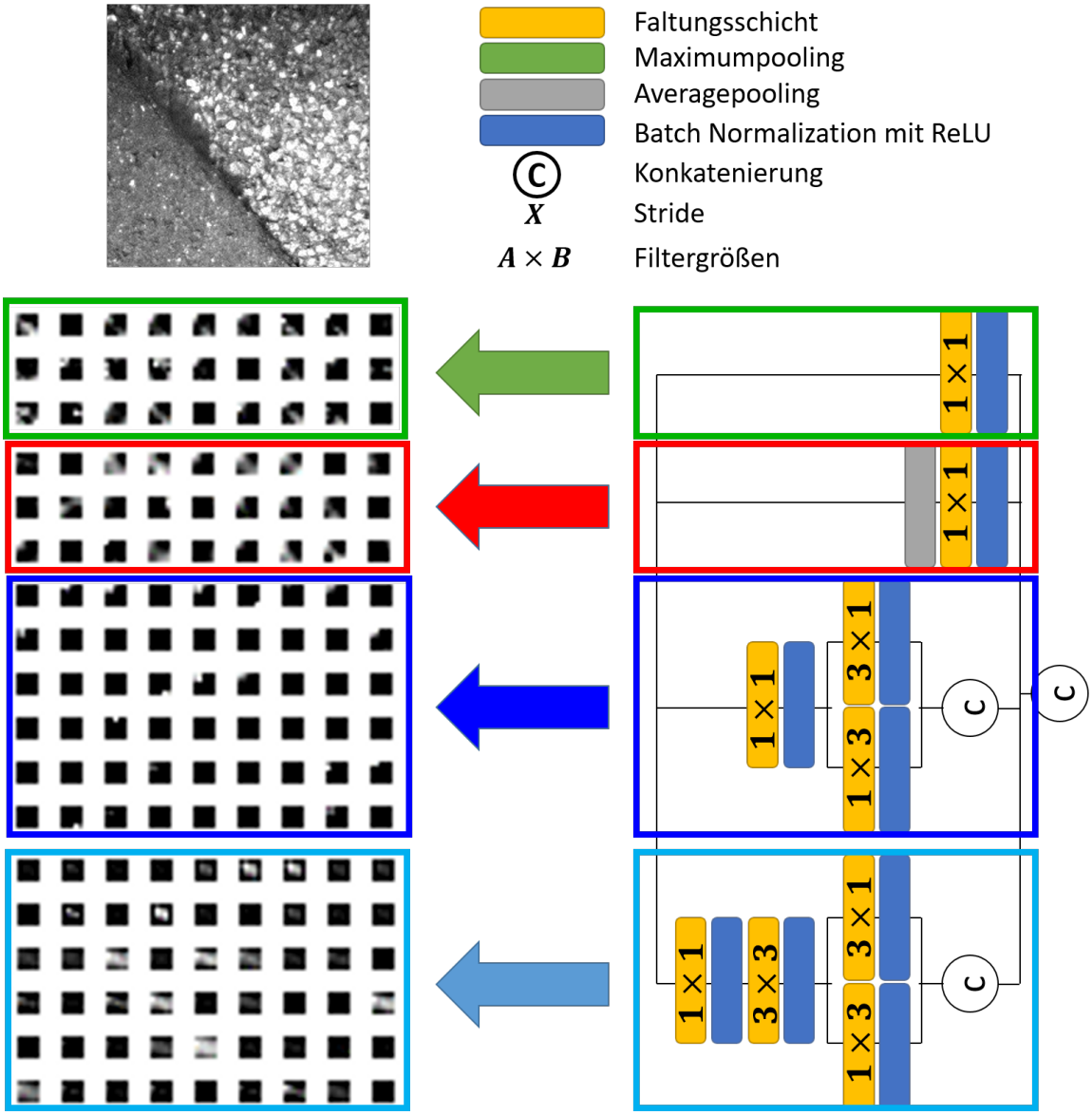


Abbildung 5.20: Aufteilung der Featuremap des InceptionNetV3

Zu sehen ist der Ursprung der verschiedenen Abschnitte der konkatenierten Featuremap des InceptionNetV3. Bei vielen Beispielen konnte insbesondere im Abschnitt des Pooling-Branches eine hohe Aktivität beobachtet werden.

5.3.5 Spärliche Featuremaps: Teil 2 - Sparsity-Diagramm

Versuchsaufbau

Im zweiten Schritt wird für jede Schicht des Netzes die Sparsity berechnet. Die Sparsity repräsentiert den prozentualen Anteil von Aktivitäten nahe Null eines jeden Filters innerhalb einer Schicht. Abschnitt 5.3.5 zeigt ein Diagramm des Verlaufs der Sparsity jeder einzelnen Schicht des VGG19. Vor der Berechnung der Sparsity werden die Aktivitäten, wie detailliert in Abschnitt 4.7.4 beschrieben wird, auf $0 \ldots 1$ normiert. Zu jeder normierten Featuremap wird ein Sparsitywert berechnet. Es entstehen pro Schicht so viele Sparsitywerte wie diese Schicht Neuronen besitzt. Abgebildet wird der Mittelwert der Sparsitywerte pro Schicht sowie das dazugehörige Konfidenzintervall bei einem Konfidenzniveau von 95%. Zu beachten ist, dass ein Threshold angewendet wurde, ab welchem die Aktivität als "Null" gewertet wird. Dieser kann dem jeweiligen Diagramm entnommen werden.

Ergebnis

Vergleichbar mit den Ergebnissen im vorherigen Abschnitt 5.3.4 ist es bei der Sparsity schwierig, Gemeinsamkeiten zwischen den Architekturen abzuleiten. Nichtsdestotrotz lassen sich gewisse Zusammenhänge zwischen der Architektur und dem Verlauf der Sparsity ermitteln.
Als erstes fallen das AlexNet und das VGG19 durch einen recht ähnlichen Verlauf auf. Beide Netze starten mit einer tendenziell geringeren Sparsity in der Inputschicht und steigern sich im Verlauf. Insbesondere im Vergleich zu den anderen Architekturen besitzen diese beiden eine sehr hohe Sparsity, welche kontinuierlich durch das Netz verläuft. Vergleicht man Samples mit Schaden mit Samples ohne Schaden, so fällt auf, dass die Netze bei Bildern ohne Straßenschäden eine generell leicht niedrigere Sparsity besitzen. Abbildung 5.23 zeigt ein Beispiel des VGG19.
Als nächstes können das ResNet50 und das XceptionNet zusammengefasst werden. Beide besitzen allgemein eine sehr geringe Sparsity, welche sich im Verlauf der Netze

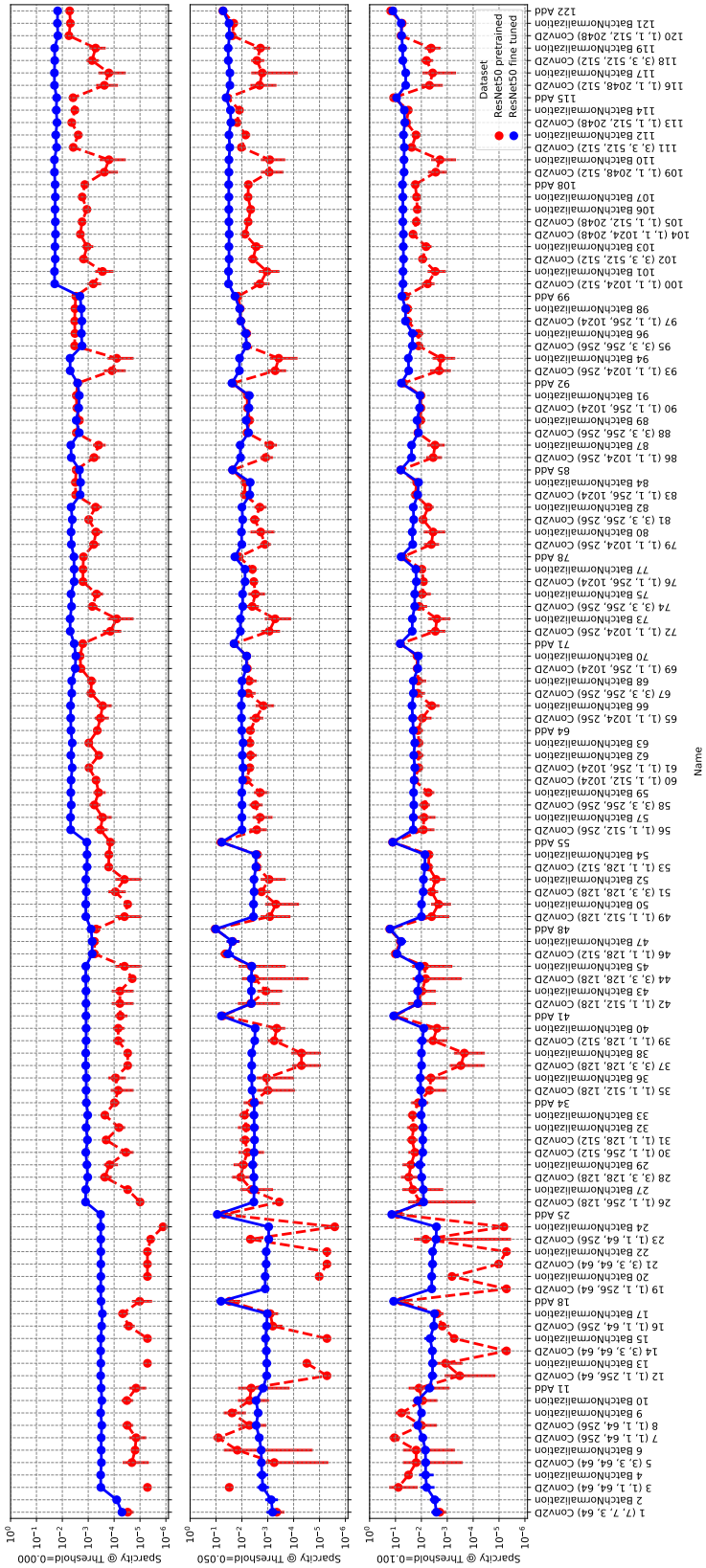


Abbildung 5.21: Sparsity-Verlauf des ResNet50

Zu sehen ist der Verlauf der Sparsity des ResNet50 vor und nach dem Finetuning, für verschiedene Thresholds. Insbesondere bei einem Threshold von Null ist gut der leichte abgestufte Anstieg der Sparsity zu erkennen. Nichtsdestotrotz besitzt das ResNet50 selbst in den letzten Schichten noch eine relativ geringe Sparsity. Das Bildbeispiel, welches zur Berechnung der Sparsity verwendet wurde, ist das gleiche, welches auch in Abbildung 5.22 verwendet wurde. Der Übersicht halber wurde an dieser Stelle auf die Darstellung verzichtet.

leicht steigert, jedoch auch in der letzten Schicht noch vergleichsweise gering ausfällt. Darüber hinaus ist der Verlauf sehr gleichmäßig, ohne große Sprünge oder Auffälligkeiten. Vergleicht man die Sparsity von Bildbeispielen mit Schaden mit Beispielen ohne Schaden, so lassen sich ebenfalls keine großen Veränderungen feststellen. Abbildung 5.21 zeigt ein Beispiel des ResNet50.

Im Kontrast dazu besitzen das SE-ResNet50 und das MobileNet einen sehr markanten Verlauf. Beide besitzen über weite Teile der Netze eine sehr geringe Sparsity, welche von einzelnen Peaks durchbrochen wird. Zurückzuführen sind diese Peaks auf die Besonderheiten der Architekturen. Das SE-ResNet50 besitzt die SE-Module, deren vollverschaltete Schichten eine enorm hohe Sparsity haben und so durch einzelne Peaks erkennbar sind. Der Ursprung der Peaks des MobileNets liegt im Gegensatz dazu in den ReLu-Aktivierungsschichten, welche negative Aktivierungen zu Null abbilden. Während das MobileNet durch einen höheren Threshold nur vergleichsweise wenig beeinflusst wird, ist gut zu erkennen, dass große Teile der Aktivierung des SE-ResNet50 sehr gering ausfallen. Ein nur leicht höherer Threshold führt dadurch direkt zu einem sprunghaften Anstieg der Sparsity über die hintere Hälfte des Netzes. Beim Vergleich von Beispielen mit und ohne Schaden ist erneut keine Besonderheit zu entdecken. Abbildung 5.22 zeigt einen repräsentativen Ausschnitt des SE-ResNet50 und den Verlauf des MobileNet. Abbildung A.27 zeigt den vollständigen Verlauf des SE-ResNet50.

Das InceptionNetV3 verhält sich über weite Teile analog zu dem SE-ResNet50 oder MobileNet. Insbesondere anfangs besitzt es eine sehr geringe Sparsity, welche sich im Verlauf steigert und durch einzelne Peaks unterbrochen wird. Bei genauerer Betrachtung fällt auf, dass die Peaks nach den Batch-Normalization-Schichten auftreten. Der Ursprung ist jedoch nicht die Batch Normalization, sondern die ReLu-Aktivierungsschichten, welche auf diese folgen. Zugunsten einer besseren Übersichtlichkeit wurden die ReLu-Aktivierungsschichten hier jedoch nicht dargestellt. Beim Vergleich von Beispielen mit und ohne Schaden fällt auf, dass beide Arten zu einer sehr hohen Sparsity vor der Klassifikationsschicht führen. Insgesamt kann bei Beispielen ohne Straßenschaden jedoch eine allgemein geringere Sparsity beobachtet werden als bei Samples mit Schaden. Abbildung 5.25 zeigt ein Beispiel des InceptionNets.

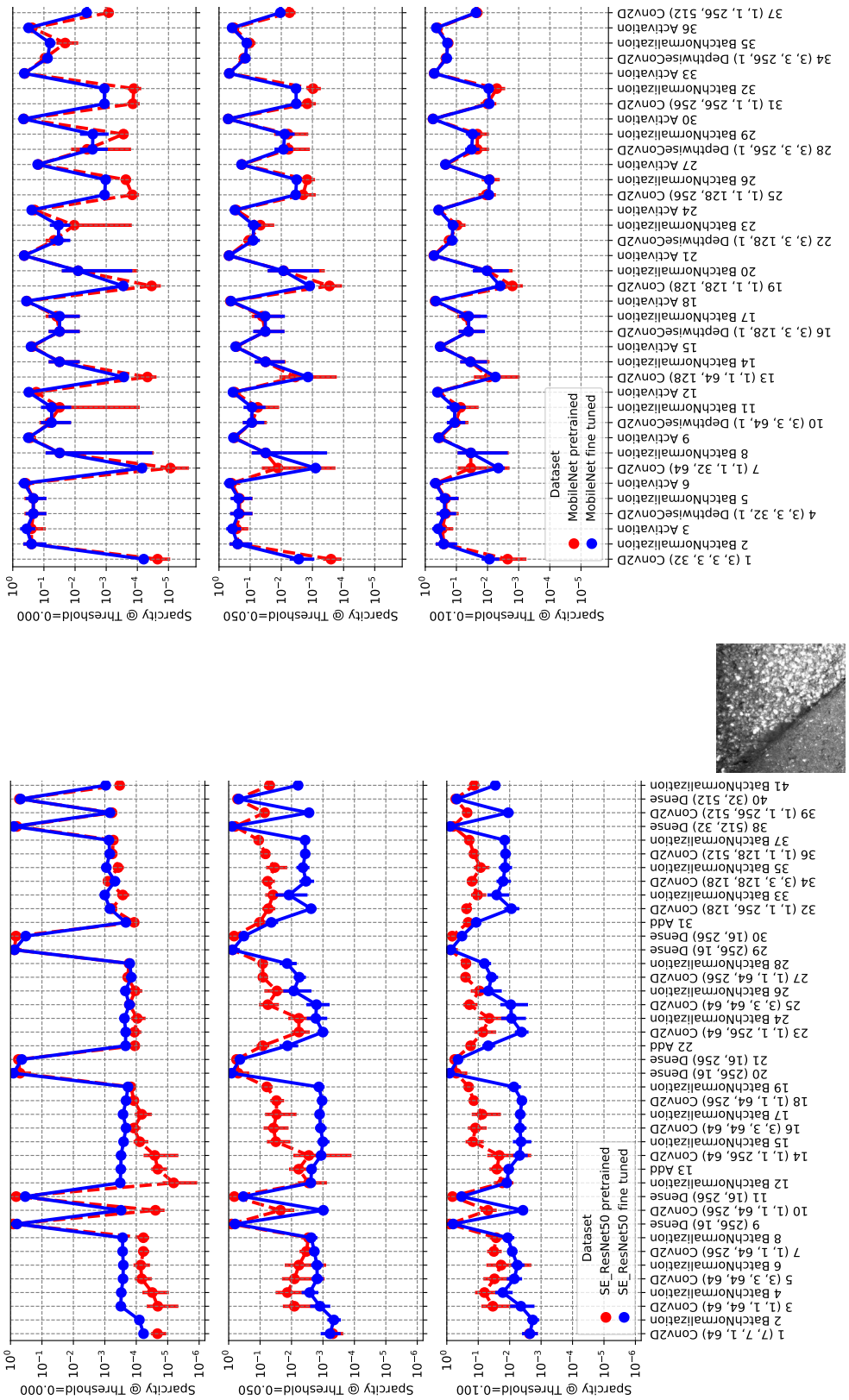


Abbildung 5.22: Gegenüberstellung des Sparsity-Verlaufs des SE-ResNet50 und MobileNet

Zu sehen ist der Sparsity-Verlauf über die ersten Schichten des SE-ResNet50 (links) und MobileNet (rechts). Für die Berechnung wurde das mittig abgebildete Bildbeispiel verwendet. Beide besitzen über weite Teile des Netzes eine sehr geringe Sparsity. Wie gut zu erkennen ist, steigt die Sparsity des SE-ResNet50 durch einen höheren Threshold deutlich stärker als beim MobileNet.

Diskussion

Wie aufgrund der Ergebnisse aus Abschnitt 5.3.4 bereits vermutet wurde, nimmt die Sparsity auch über den gesamten Verlauf des Netzes tendenziell eher zu als ab. Beim Vergleich der Sparsity-Diagramme ergibt sich ebenfalls eine deutliche Tendenz dazu, dass die Sparsity durch das Finetuning steigt. Jedoch scheint die Betrachtung der Sparsity für komplexere Architekturen als das AlexNet bzw. das VGG19 nur bedingt hilfreich.

Im Gegensatz zum AlexNet und VGG19 ist die Interpretation anderer Architekturen schwieriger. Durch die ReLu-Aktivierungsfunktion, SE-Module und räumlichen Faltungen der Separable Convolution steigt die Sparsity teilweise deutlich. Skip-Connections, Inception-Module und 1×1 Faltungen der Separable Convolution lassen die Sparsity hingegen teilweise stark sinken. Der Wechsel aus räumlicher Faltung, ReLu-Aktivierungsfunktion und 1×1 Faltung führen bei dem MobileNet zu einem sägezahnförmigen Verlauf. Abgesehen vom InceptionNet und MobileNet besitzen diese komplexeren Architekturen jedoch selbst in der letzten Faltungsschicht eine Sparsity von unter 5%. Dies bedeutet jedoch nicht, dass alle Features der Netze auch notwendig sind. Abbildung 5.24 zeigt beispielhaft für das XceptionNet, dass viele Features auf gleiche Strukturen reagieren und daher eine hohe Redundanz innerhalb des Netzes vorliegt. Diese hohe Redundanz ist ein weiteres Indiz dafür, dass die komplexeren Architekturen für die gegebene Problemstellung zu groß sind.

Ein weiterer Einfluss der SE-Module ist, dass durch die stetige Multiplikation mit Gewichtungen kleiner Eins nur eine recht geringe Aktivierung vorliegt, wodurch sich ein höherer Threshold relativ stark auswirkt.

Analog zu dem Sparsity-Verlauf in [Kim et al., 2017] (s. Abbildung 5.26) steigt die Sparsity des VGG19 auch auf fast 100% an. Ein deutlicher Unterschied liegt jedoch im Verlauf über die vorderen Schichten. Im Gegensatz zum VGG16 in [Kim et al., 2017] erfolgt der Anstieg nicht langsam, sondern sprunghaft. Die wahrscheinliche Ursache dafür liegt in der leicht unterschiedlichen Berechnung der Sparsity, auf die im Folgen-

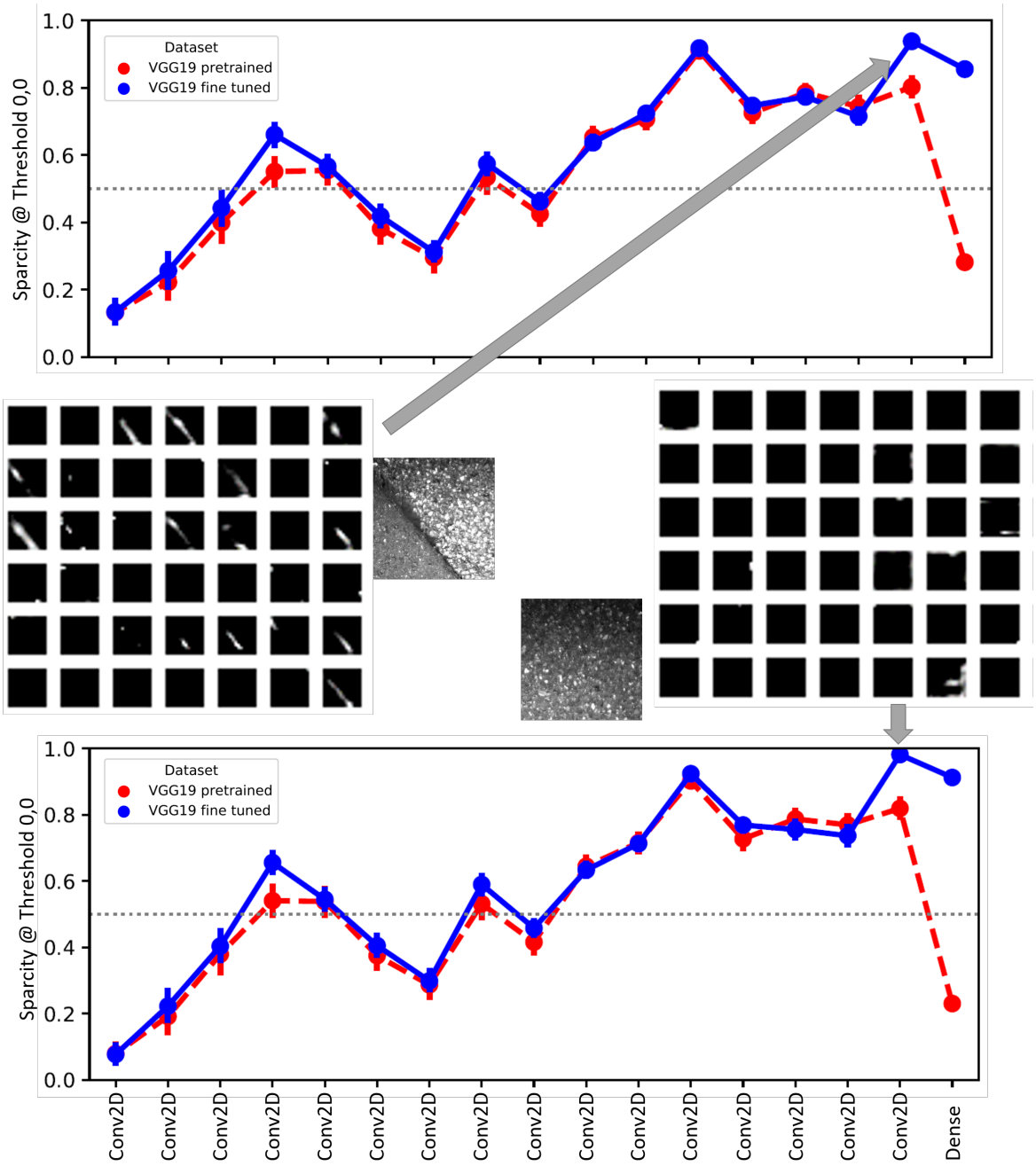


Abbildung 5.23: Sparsity-Überblick über das VGG19

Zu sehen ist ein Überblick über den Verlauf der Sparsity zweier Bildbeispiele des VGG19. Des Weiteren ist die Visualisierung der letzten Faltungsschicht abgebildet. Wie gut zu erkennen ist, kann sich bereits ein geringer Unterschied in der Sparsity deutlich auf die Featuremaps auswirken.

den noch eingegangen wird. Darüber hinaus ist leider nicht bekannt, ob in [Kim et al., 2017] ein Threshold verwendet wurde. Des Weiteren fällt auf, dass der Verlauf des AlexNet dem des VGG19 sehr ähnlich ist, weshalb dieser für einfache Architekturen charakteristisch zu sein scheint.

Ein für ResNet-Architekturen charakteristischer Verlauf ist eine konstant niedrige Sparsity, welche über den Verlauf des Netzes langsam, aber stufenweise, steigt. Dies kommt durch die Skip-Connection zustande. Es wird stetig die Aktivierung des vorherigen Moduls hinzu addiert, was dazu führt, dass eine gewisse Aktivierung von Beginn an bis zum Ende des Netzes durchgereicht wird. Wie Abbildung 5.21 repräsentativ zeigt, tritt der stufenweise Anstieg zu Beginn eines Residual-Blocks auf. Innerhalb eines Residual-Blocks bleibt die Sparsity jedoch annähernd konstant. Betrachtet man allerdings den Verlauf mit einem höheren Threshold von 5% oder 10%, ebenfalls in Abbildung 5.21 zu sehen, treten in den Additionsschichten vermehrt Spikes auf. Diese

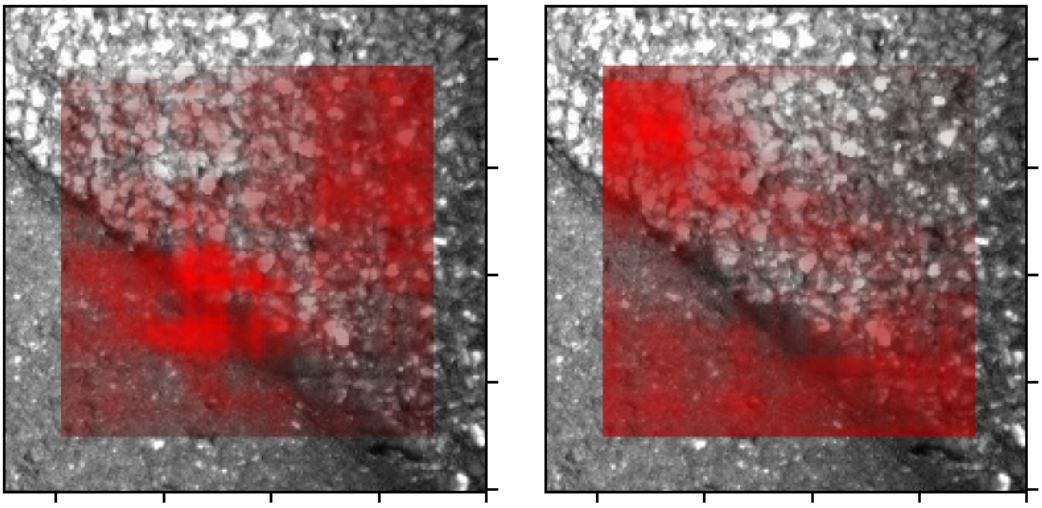


Abbildung 5.24: Feature-Visualisierung des XceptionNet

Zu sehen sind die Feature-Attention-Diagramme von zwei beispielhaften Features der letzten Seperable-Convolution-Schicht des XceptionNet. Beide Diagramme wurden wie in Abschnitt 5.3.6 beschrieben gewonnen. Die beiden abgebildeten Feature-Typen sind repräsentativ für annähernd sämtliche 2048 Features dieser Schicht. Das Interesse der Features in der vorhergehenden bzw. nachfolgenden Schicht ist, ähnlich wie in dieser, auf nur wenige Feature-Typen beschränkt.

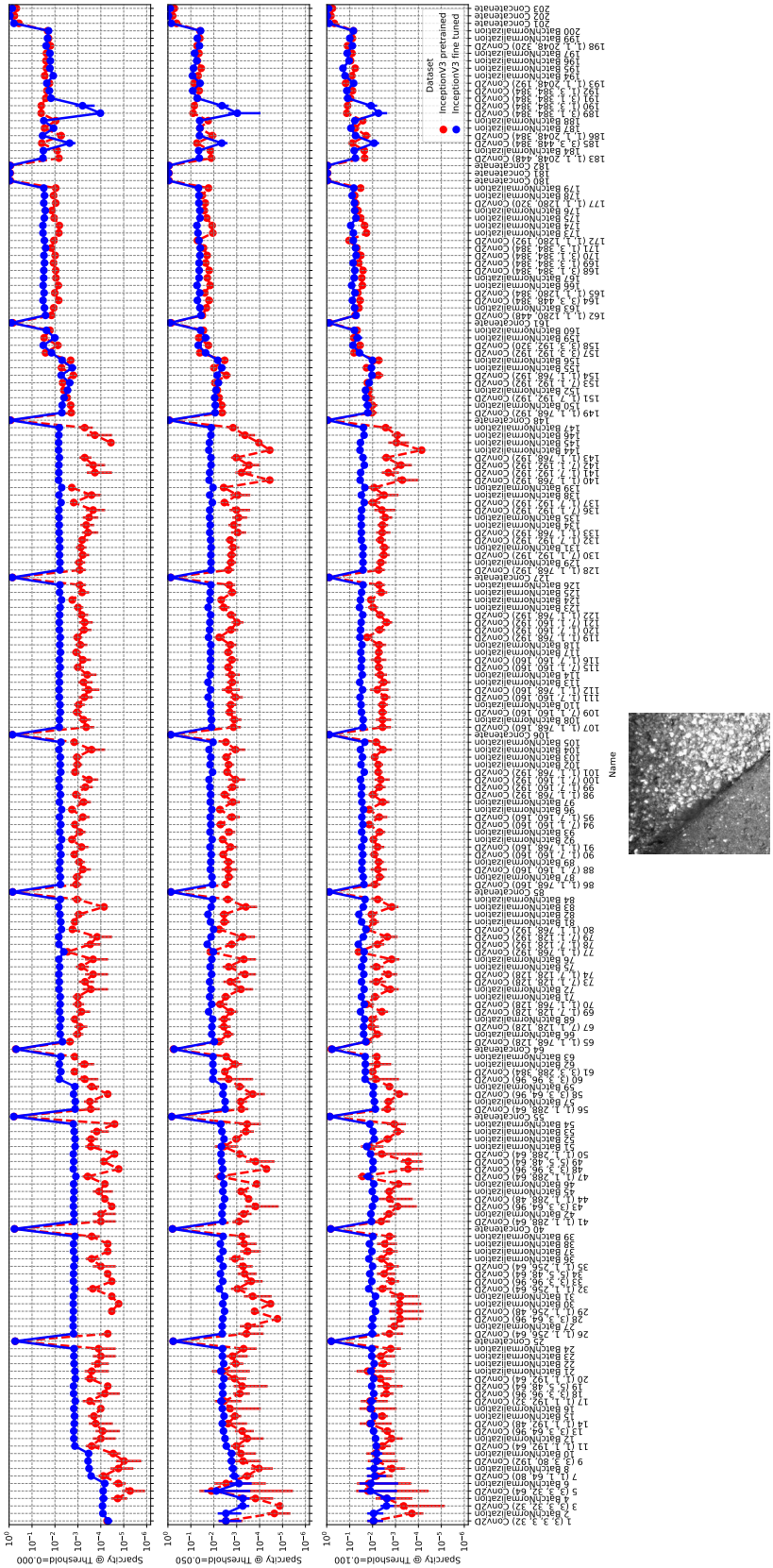


Abbildung 5.25: Sparsity-Verlauf des InceptionNetV3

Zu sehen ist der Verlauf der Sparsity des InceptionNetV3 vor und nach dem Finetuning, für verschiedene Thresholds und das gegebene Bildbeispiel. Es ist gut zu erkennen, dass der Verlauf durch Peaks in den Konkatenierungsschichten durchbrochen wird. Zurückzuführen sind die Peaks auf die ReLu-Aktivierungsfunktionen, welche direkt vor der Konkatenierung angewendet werden. Darüber hinaus kann nur ein mäßiger Anstieg der Sparsity beobachtet werden.

Spikes sind auf eine negative Aktivierung zurückzuführen, welche durch die ReLu-Aktivierungsfunktion abgeschnitten werden. Das InceptionNet verhält sich bis auf das letzte Inception-Modul ähnlich, besitzt jedoch keine Skip-Connections. Die kurzen Branches der Inception-Module scheinen sich ähnlich auszuwirken.
Wird abschließend der Verlauf der Sparsity vor und nach dem Finetuning verglichen, kann kein eindeutiger Hinweis auf eine reine Feature-Selektion, wie in [Kim et al., 2017], festgestellt werden. Überraschenderweise fällt auf, dass der Verlauf über weite Teile der Netze identisch ist und sich auch in den letzten Schichten nicht entscheidend voneinander abhebt.
In [Kim et al., 2017] konnte dagegen ein anderes Verhalten beobachtet werden, wie in Abbildung 5.26 zu erkennen ist. Ursache dafür ist die leicht unterschiedliche Be-

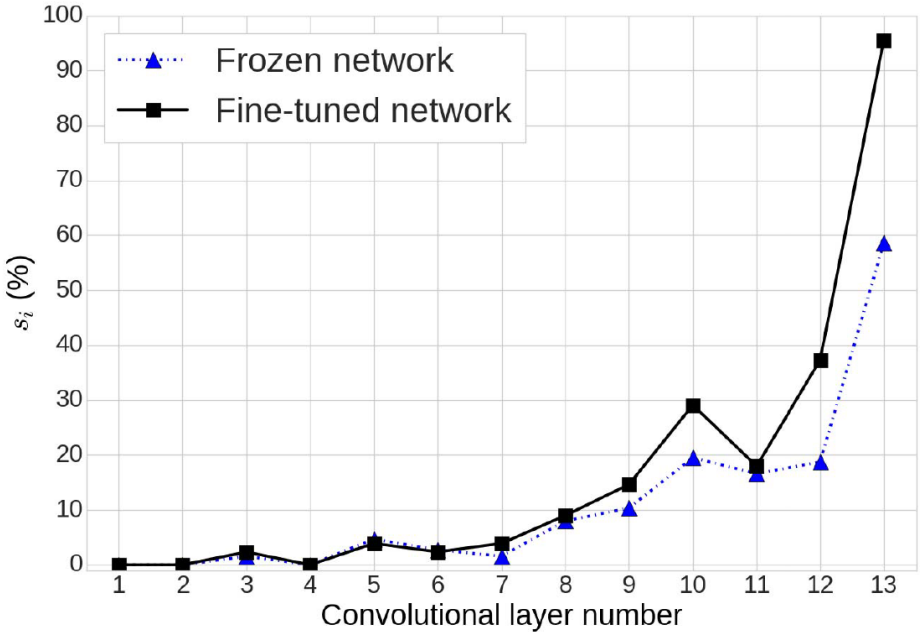


Abbildung 5.26: Sparsity des VGG16

Zu sehen ist der Verlauf der Sparsity für jede Schicht des VGG16 aus [Kim et al., 2017]. Zu beachten ist das unterschiedliche Vorgehen zur Berechnung der Sparsity im Vergleich zu dieser Arbeit. Für die Berechnung werden lediglich vollständig nicht-aktivierte Featuremaps gezählt. Eine Sparsity von 50% bedeutet daher, dass die Hälfte aller Featuremaps vollständig inaktiv sind und die andere Hälfte mindestens einen Pixel enthält, welcher größer Null ist.

rechnung der Sparsity. In [Kim et al., 2017] wird die Sparsity anhand vollständig toter Kanäle bestimmt. Besitzt ein Kanal daher lediglich einen einzigen, minimal aktivierten Pixel, so wird dieser als "nicht tot" gewertet. Wird die Sparsity auf diese Weise für die Featuremap des Beispiels ohne Schaden in Abbildung 5.23 berechnet, würde sich eine ähnliche Sparsity wie in [Kim et al., 2017] ergeben.
Würde man die Sparsity auf diese Weise jedoch auf einem Beispiel mit Schaden berechnen, wie in Abbildung 5.23 zu sehen, würde die Sparsity deutlich geringer ausfallen. Folglich ist die zu beobachtende Sparsity abhängig von dem gewählten Bildbeispiel und es bleibt die Aussage bestehen, dass das Finetuning keine reine Feature-Selektion bewirkt hat. Es wäre lediglich möglich, dass eine Selektion für die Objektklasse ohne Schaden durchgeführt wurde. Zur Überprüfung müssten jedoch weitere Experimente unternommen werden, welche aus zeitlichen Gründen nicht möglich waren. Es könnte ebenfalls sein, dass das Netz komplett neue Features gelernt hat und bei dem Training lediglich mehr Features für die Objektklasse mit Schaden als für die ohne Schaden gelernt wurden.

5.3.6 Feature Attention

Eine nähere Betrachtung der Features soll Aufschluss darüber geben, was das Netz gelernt hat. Es wird dabei analog zu der Verdeckungsmethode nach [Zeiler and Fergus, 2014] vorgegangen.
Dem Netz werden eine Reihe Beispiele präsentiert und es wird untersucht, welche der gezeigten Bildregionen zu einer hohen Aktivierung der einzelnen Neuronen geführt haben. Damit wird untersucht, auf welche Bildinhalte die gelernten Features empfindlich sind. Darüber hinaus kann betrachtet werden, wie viele Features das Netz ausgeprägt hat, um jeweils Klasse "Schaden" bzw. "Kein Schaden" zu detektieren. Besonders interessant ist, ob die Detektion der Straßenschäden auch von den zugehörigen Defekten (Risse, Schlaglöcher, usw.) in der Straße hervorgerufen wird. Im optimalen Fall wird es Features geben, welche genau auf diese Bildinhalte empfindlich sind. Denkbar ist jedoch auch, dass das Netz andere Bildinhalte findet, welche für eine Klassifikation verwendet werden können. Diese könnten sich z.B. durch einen Bias ergeben, beispielsweise falls auf allen Trainingsbildern mit Schaden eine bestimmte Sorte Asphalt verwendet wurde.
Für die Durchführung werden zehn Bildbeispiele (fünf "Kein Schaden", fünf "Schaden") jeder Architektur präsentiert und es wird das "Interesse" jedes Neurons als Overlay über das Eingabebild gelegt. Abbildung 5.28 zeigt ein Beispiel für verschiedene Samples der letzten Schicht des InceptionNetV3 vor dem Klassifikator. Eine Auflistung weiterer ausgewählter Diagramme ist im Anhang in Abschnitt A.10 zu finden.

Ergebnis

Wie in Abschnitt 3.2 bereits erläutert, prägen Faltungsnetze insbesondere in den ersten Schichten relativ allgemeingültige Filter aus, welche auf Strukturen wie Ecken, Kanten oder verschiedene Helligkeiten reagieren. Dieses Verhalten konnte für alle Architekturen gleichermaßen festgestellt werden. Abbildung 5.29 zeigt annähernd identische Filter, welche in allen Architekturen in den ersten Faltungsschichten beobachtet werden konnten. Für das MobileNet ergeben sich, wie in Abbildung 5.29 zu sehen, in der ersten Faltungsschicht beispielsweise 17 Filter, welche auf den hellen oberen Bereich

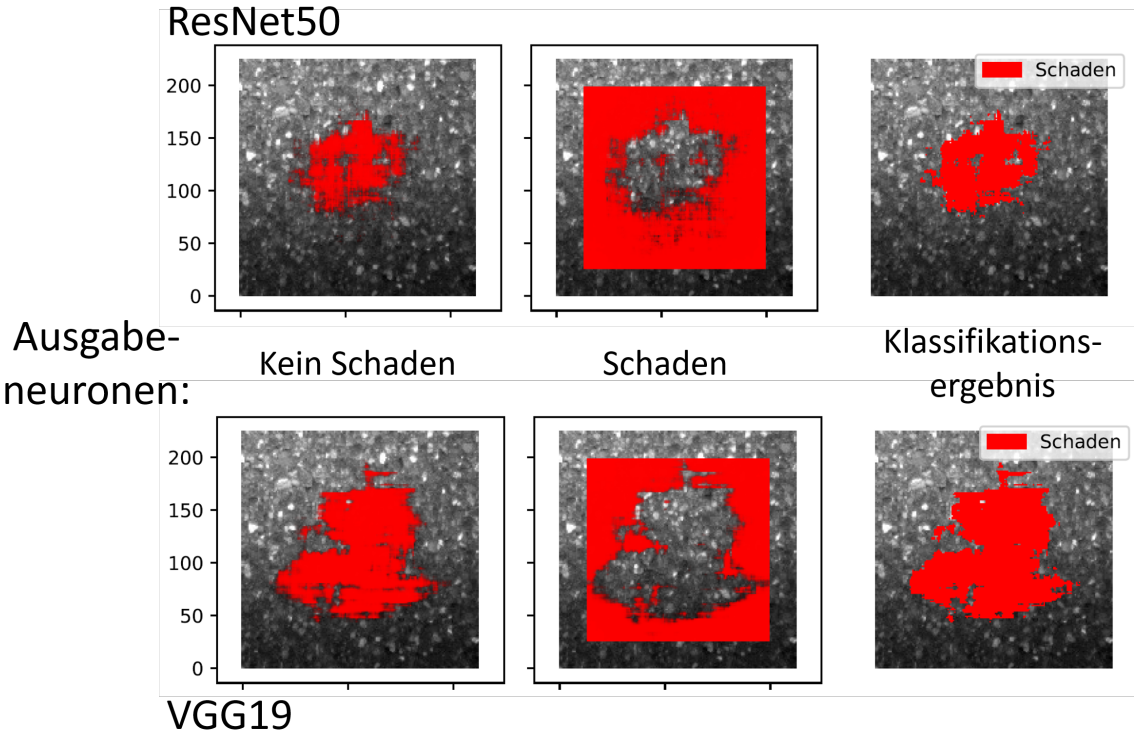


Abbildung 5.27: Feature-Visualisierung der Klassifikationsschicht

Zu sehen ist die Visualisierung des Featureinteresses der Klassifikationsschicht des ResNet50 und des VGG19 auf einem Sample ohne Schaden. Es ist deutlich zu erkennen, dass für beide Netze insbesondere die Mitte eines Samples ohne Schaden relevant ist, damit diese korrekt klassifiziert werden. Die Ausgabe der Neuronen links und in der Mitte repräsentieren das Interesse der Softmax-Ausgabe. Die rechten Ausgaben zeigen hingegen wie das Bildbeispiel, bei gegebener Verdeckung, klassifiziert worden ist. Wie gut zu erkennen ist, konzentriert sich das Ausgabeneuron für die Klasse "Kein Schaden" besonders auf die Bildmitte, wohingegen das Neuron für die Klasse "Schaden" mehr auf die äußeren Regionen fokussiert ist. Wird eine Region in der Mitte des Bildes nun verdeckt, resultiert dies in einem ausreichend großen Aktivierungsverlust für das Kein-Schaden-Neuron, sodass das Bild als Schaden klassifiziert wird. Ein ähnliches Verhalten ist bei den meisten Samples ohne Schaden und bei allen Architekturen zu beobachten.

empfindlich sind, und 15 Filter, die auf den dunklen unteren reagieren. Ein ähnliches Verhältnis ergibt sich auch für die anderen Architekturen.
Betrachtet man hingegen die Netzausgabe, so fällt insbesondere auf, dass bei Beispielen ohne Schaden die Bildmitte eine wichtige Rolle spielt. Ist die Bildmitte verdeckt, neigen viele Klassifikatoren dazu, das Bild fälschlicherweise als schadhaftes Beispiel zu klassifizieren. Abbildung 5.27 zeigt ein Beispiel zu zwei Architekturen, das Verhalten ist jedoch bei allen anderen ebenfalls erkennbar. Zurückzuführen ist dieses Verhalten, wie bereits in Abschnitt 5.3.4 erläutert wurde, darauf, dass nur für eine zentrale Region von 64×64 Pixeln garantiert ist, dass es dort keinen Schaden gibt. Umliegende Regionen können auch in Beispielen, welche als schadenfrei gelabelt sind, Schäden enthalten.

Diskussion

Wie bereits in Abschnitt 5.3.6 beschrieben, wurden die Erwartungen erfüllt: Frühere Schichten reagieren auf einfache Strukturen, wie Helligkeit, Ecken und Kanten. Spätere Schichten reagieren hingegen eher auf konkrete Bildinhalte, wie Risse oder Schlaglöcher, oder eben auf das Fehlen dieser Inhalte. Aus zeitlichen Gründen war jedoch keine Detailanalyse dazu möglich, welche Features auf welche Regionen ansprechen. Nützlich wäre diese Betrachtung, um herauszufinden, wie viele Features auf "Schaden" und wie viele auf "Kein Schaden" reagieren.
Wie schon in Abschnitt 5.3.4 erläutert wurde, zeigt auch Abbildung 5.30, dass alle Netze sehr aufmerksam auf die Mitte der Beispiele ohne Schaden reagieren. Dem InceptionNet gelingt es, die Beispiele ohne Schaden selbst mit Verdeckung stets richtig zu klassifizieren, weshalb dort keine Veränderung zu erkennen ist. Für die anderen Netze gilt: Falls die Mitte verdeckt ist, neigen sie zu einer Falschklassifikation. Dies spricht dafür, dass falls ein Sample keinen Schaden besitzt, diese Information hauptsächlich aus der Bildmitte abgeleitet wird. Ähnliches konnte schon in Abschnitt 5.3.4 beobachtet werden und ist auf die Art zurückzuführen, wie die Bildbeispiele aus den FullHD-Bildern ausgeschnitten wurden. In Abschnitt 5.3.4 wird dieser Effekt näher beschrieben.

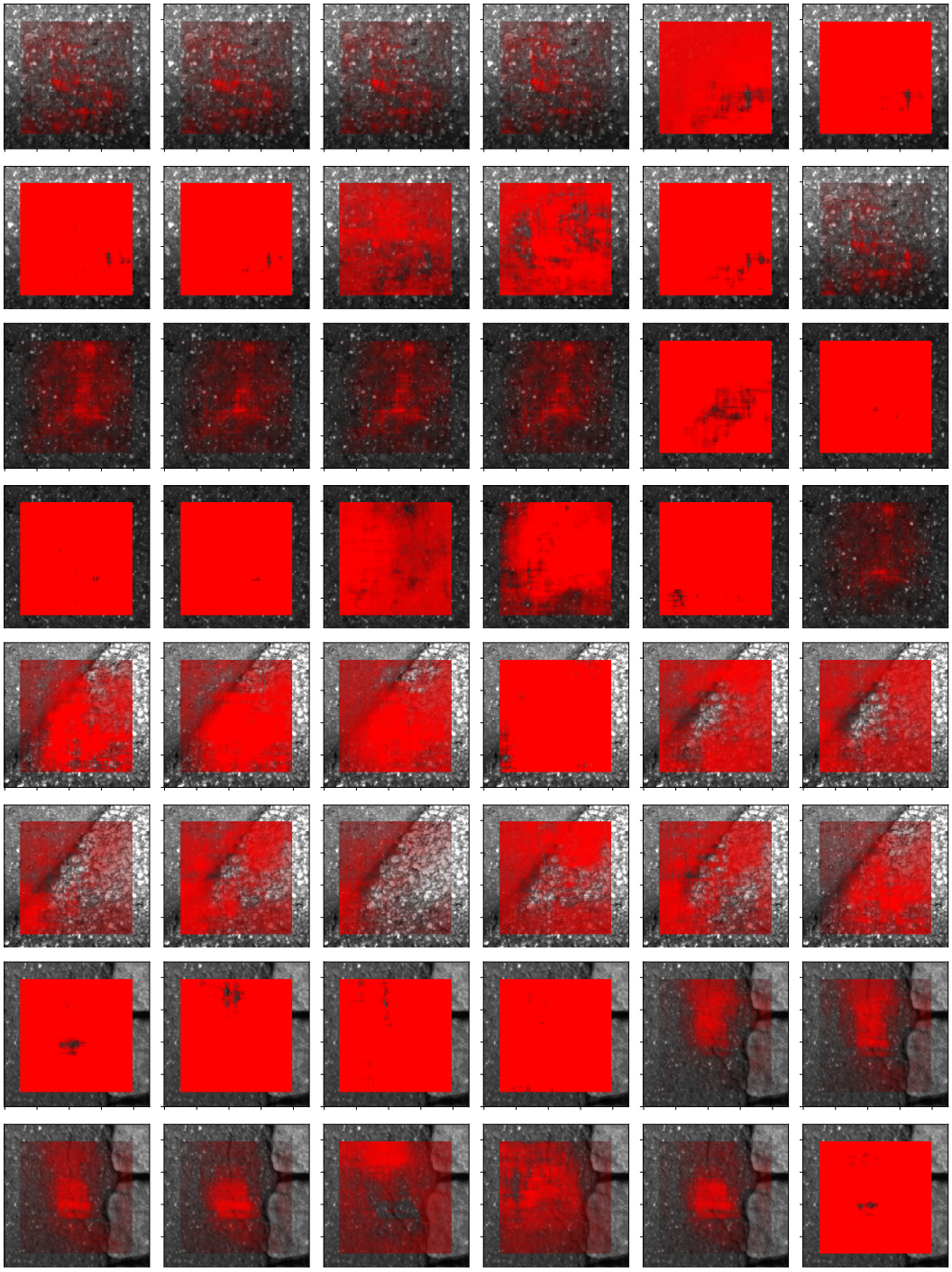


Abbildung 5.28: Feature-Visualisierung des InceptionNetV3

Zu sehen ist das Interesse einiger ausgewählter Filter der letzten Konkatenierungsschicht des InceptionNets für verschiedene Bildbeispiele. Zu beachten ist, dass für alle Beispiele jeweils die gleichen Features gezeigt werden. Es ist deutlich zu erkennen, dass Beispiele mit und ohne Schaden genau entgegengesetzte Filter ansprechen.

Darüber hinaus konnte festegestellt werden, dass viele Filter scheinbar keine bestimmten Vorlieben haben und auf fast jede Stelle des Bildes reagieren. Die Straßendaten sind generell hoch strukturiert und die Verdeckung ist eine strukturlose Maske. Es liegt also nahe, dass diese Features auf die Struktur der Bilder reagieren.
Problematisch ist die Anwendung jedoch bei den Bildbeispielen mit Schaden, wie es Abbildung A.28 zeigt. Für die Verdeckung wurde ein Block mit einer Größe von 51×51 Pixel gewählt. Da die Straßenschäden meist große Teile des gesamten Bildes einnehmen, konnte die Aktivierung eines Neurons durch die Verdeckung häufig kaum verändert werden. Um eine Veränderung zu beobachten, hätte eine Maske gewählt werden müssen, welche annähernd so groß ist wie das Bild selbst. Praktikabler wäre daher eine für jedes Beispiel mit Schaden angepasste Maske, was aus zeitlichen Gründen nicht möglich gewesen ist.

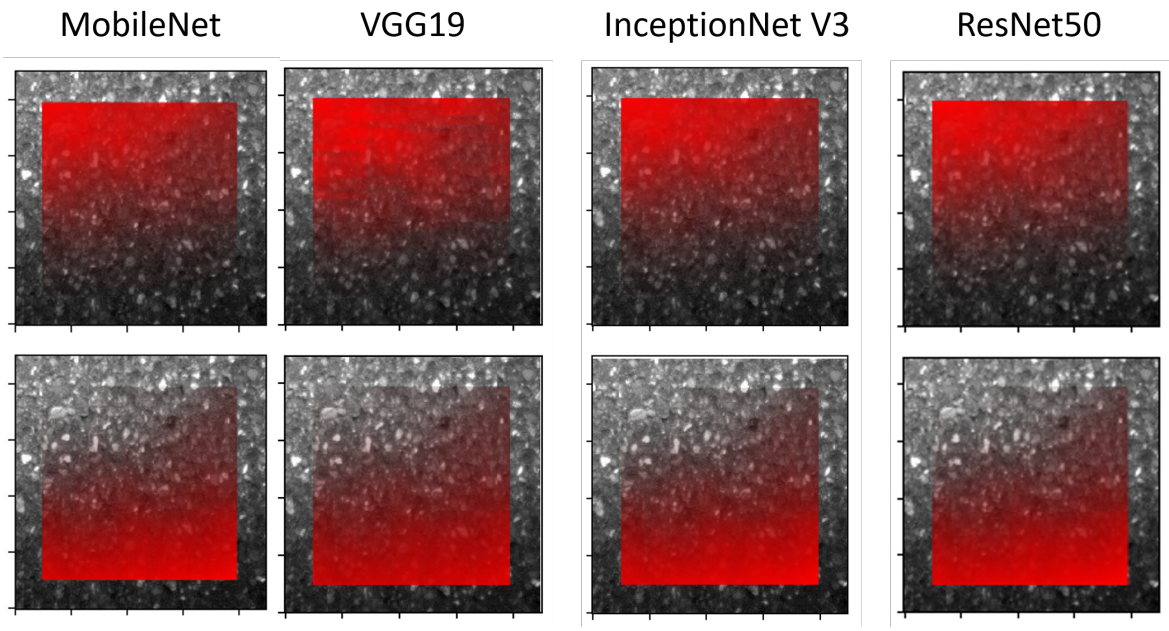


Abbildung 5.29: Feature-Visualisierung der ersten Faltungsschicht
Zu sehen sind die Visualisierungen fast identischer Features, welche in der ersten Schicht des jeweiligen Netzes beobachtet werden konnten. Die gezeigten Features sind insbesondere für die erste Faltungsschicht unabhängig von der Architektur sehr repräsentativ. Der Großteil der Features in den ersten Schichten weist ein ähnliches Verhalten auf.

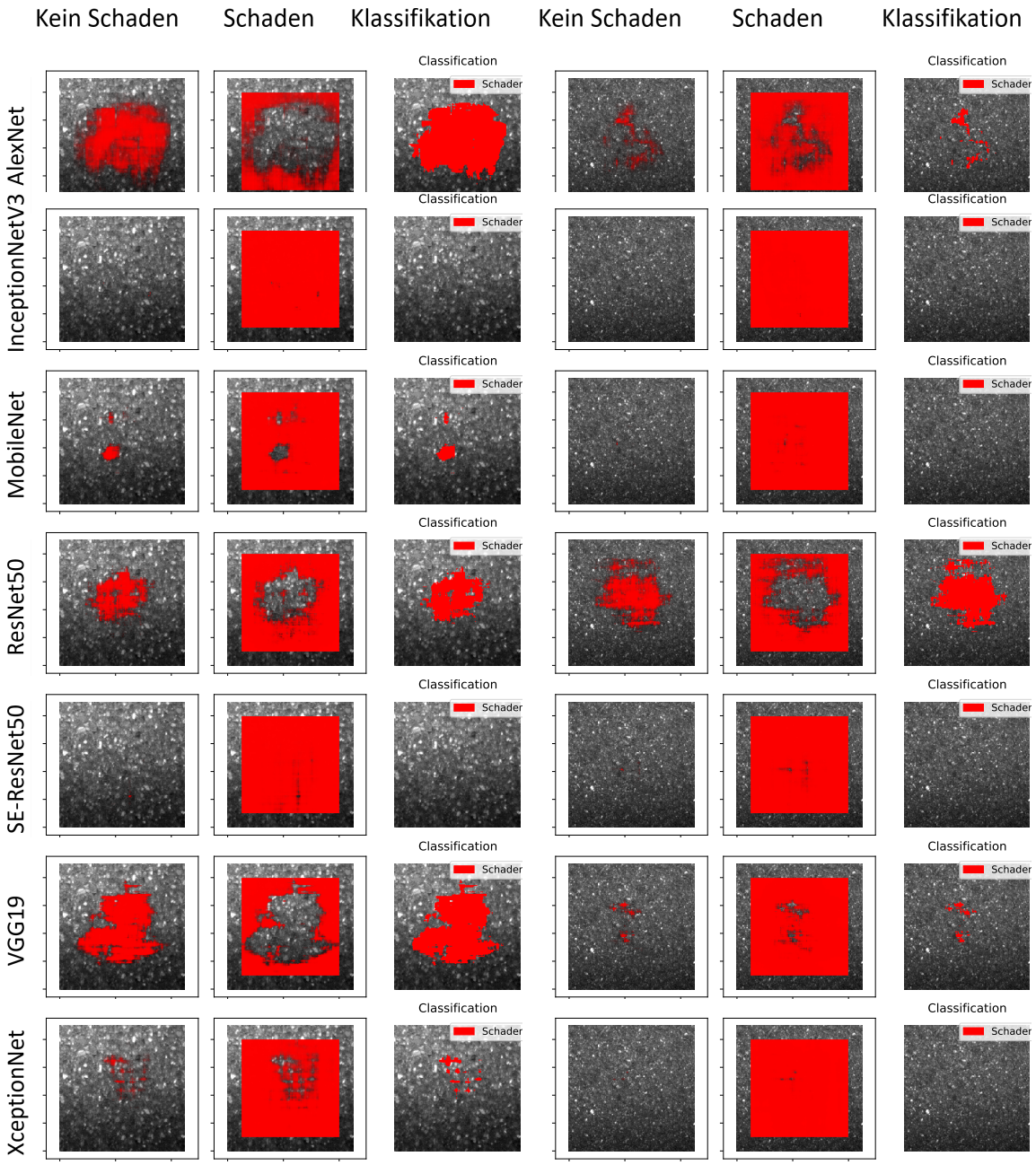


Abbildung 5.30: Visualisierung der Feature-Attention zweier Negativbeispiele
Zu sehen ist die Feature-Attention der beiden Ausgabeneuronen (links und mitte) und die resultierende Klassifikation anhand des stärker aktivierten Neurons (rechts).

5.4 Zusammenfassung

Wie zuvor in Kapitel 4 beschrieben, ist es das Ziel dieser Arbeit, neben der reinen Evaluierung der Leistung von Transfer Learning auch die in Abschnitt 4.8 zusammengefassten Effekte zu bewerten. Auf Basis der Ergebnisse dieses Kapitels lassen sich die Erkenntnisse zur Leistung und den beobachteten Effekten des Transfer Learnings in Bezug auf Straßenschäden wie folgt zusammenfassen:

- Die Generalisierungsfähigkeit konnte für einige Architekturen deutlich gesteigert werden (s. Abschnitt 5.2.3).
- Die Trainingszeit für das Finetuning ist mit etwa einer Epoche bzw. wenigen Epochen außerordentlich kurz (s. Abschnitt 5.2.3).
- Unter Umständen ist es hilfreich, einen kleinen Teil des Netzwerkes für das Finetuning einzufrieren. Dies wurde für die InceptionNet V3 Architektur bei 20% eingefrorenen Schichten beobachtet (s. Abschnitt 5.2.3).
- Es wurde festgestellt, dass es sich beim GAPs und ImageNet-Datensatz um sowohl optisch als auch in Bezug auf die erzeugten Features sehr unterschiedliche Datensätze handelt (s. Abschnitt 5.3.1).
- Die betragsmäßige Änderung der Gewichte durch das Finetuning fällt meist sehr gering aus (s. Abschnitt 5.3.2).
  - Für alle Architekturen beschränkt sich der größte Teil der Änderung auf die ersten und letzten Schichten des Netzes.
  - 1×1 Faltungen wurden durch eine Umgewichtung der Features verhältnismäßig stärker verändert.
- Es ist keine allgemeingültige Aussage in Bezug auf die Sparsity der Aktivierungen der Architekturen möglich (s. Abschnitt 5.3.4 und 5.3.5).
  - Wie hoch die Sparsity ausfällt, ist sehr stark davon abhängig, welche Architektur verwendet wird.

  - Allgemein besitzen das AlexNet und VGG19 die höchste Sparsity und das XceptionNet, ResNet50, SE-ResNet50 und InceptionNetV3 eine besonders geringe. Die Sparsity-Verläufe der drei zuletzt genannten sind jedoch durch Spikes geprägt.

- Die Prämisse, dass Netze insbesondere in den vorderen Schichten sehr allgemeingültige Features ausprägen, konnte bestätigt werden (s. Abschnitt 5.3.6).
  - Auch die Features, welche zur Klassifikation verwendet werden, sind architekturübergreifend relativ ähnlich.

Fazit

6

Transfer Learning konnte auf die Detektion von Straßenschäden erfolgreich angewendet werden. Es konnte gezeigt werden, dass durch ein Finetuning die Generalisierungsfähigkeit gesteigert wird. Die Höhe der Steigerung ist abhängig von der gewählten Architektur. Ein negativer Einfluss konnte jedoch bei keiner Architektur festgestellt werden. Insgesamt konnten sich das MobileNet, ResNet50 und InceptionNetV3 gegenüber dem AlexNet und XceptionNet hinsichtlich der Effektivität des Transfer Learnings durchsetzen.

Dieser Erfolg konnte trotz großem Unterschied zwischen Ursprungs- und Zielaufgabe festgestellt werden. Darüber hinaus wurde beobachtet, dass das Einfrieren weniger Schichten ebenfalls der Generalisierungsfähigkeit zugute kommen kann. Insbesondere die Inception-Architektur konnte in diesem Aspekt überzeugen und setzte sich dadurch mit einem Vorsprung von mindestens 0,0078 im Test-F1-Score gegenüber den anderen Architekturen durch.

Die erwartete Feature-Selektion konnte nicht beobachtet werden. Die Ergebnisse aus Kapitel 5.3.2 und 5.3.4 sprechen eher gegen eine Feature-Selektion. Betrachtet man die Gewichtsänderung vor und nach dem Finetuning, wurde überraschenderweise festgestellt, dass die Gewichte der vorderen Schichten stärker verändert wurden als die der hinteren.

Abschließend kann der Konsens des State-of-the-Art bestätigt werden. Sofern eine Standardarchitektur eingesetzt wird, ist der Einsatz von Transfer Learning sehr empfehlenswert und bietet kaum Nachteile. Folgende wesentliche Punkte haben sich aus den Experimenten in Kapitel 5 außerdem ergeben:

- Die Trainingszeit für das Finetuning ist mit etwa einer Epoche bzw. wenigen Epochen außerordentlich kurz.
- Unter Umständen ist es hilfreich, einen kleinen Teil des Netzwerkes für das Finetuning einzufrieren. Dies wurde für die InceptionNet V3 Architektur bei 20% eingefrorenen Schichten beobachtet.
- Die Prämisse, dass Netze insbesondere in den vorderen Schichten sehr allgemeingültige Features ausprägen, konnte bestätigt werden.
- Die betragsmäßige Änderung der Gewichte durch das Finetuning fällt meist sehr gering aus.
  - Für alle Architekturen beschränkt sich der größte Teil der Änderung auf die ersten und letzten Schichten des Netzes.
  - 1×1 Faltungen wurden durch eine Umgewichtung der Features verhältnismäßig stärker verändert.

6.1 Ausblick

Wie in den entsprechenden Abschnitten der Diskussion bereits festgestellt, ergeben sich für folgende Aspekte Möglichkeiten, weitere Untersuchungen anzuschließen:

- Eine detailliertere Analyse des Einflusses der Datenvorverarbeitung auf das Finetuning.
- Ein Vergleich der Gewichtsänderungen mit verschiedenen Konfigurationen an eingefrorenen Schichten.
- Eine genauere Evaluierung des Nutzens eingefrorener Schichten während des Transfer Learnings durch feinere Abstufungen des Einfrierens.

Weiterführend bieten sich jedoch auch neue Betrachtungen an. Wie im State-of-the-Art bereits erwähnt, wurde in [Ng et al., 2015] ein zweistufiges Transfer Learning durchgeführt, mit dem bessere Ergebnisse erzielt wurden. Interessant wäre daher ein Pretraining auf ImageNet- und GAPs-50k- mit anschließendem Finetuning auf dem vollständigen GAPs-Datensatz.
Eine weitere sehr interessante Betrachtung wäre die Berechnung der Information-Plane nach dem Vorbild von [Shwartz-Ziv and Tishby, 2017]. Dort wurde die Mutual Information für jede Schicht eines Netzes in Bezug auf den Input und Output berechnet. Ziel ist es, mit der Mutual Information zu jeder Schicht ein Maß für den Informationsgehalt zu ermitteln. In [Shwartz-Ziv and Tishby, 2017] wird das Training eines Neuronalen Netzes in zwei Phasen eingeteilt: eine, in der primär Features gelernt werden, um den Loss während des Trainings zu minimieren, und eine sog. ”representation compression phase”, in der gelernte Features generalisiert werden. Sollte diese Unterteilung zutreffend sein, sollte bei einer Feature-Selektion während des Finetunings eher das Verhalten der ”representation compression phase” beobachtet werden können.

Anhang: Grafiken und Diagramme

A

Im Folgenden sind die vollständigen Abbildungen und Auflistungen der Materialien der einzelnen Experimente zu finden.

Abschnitt A.1: Ausgabeverteilung der fünf besten Neuronen entsprechend dem Fisher-Score. Ergänzungen zu Abschnitt 5.1.3.

Abschnitt A.2: Heatmap der Ergebnisse der Validierungs-Accuracy und des Validierungs-Test-F1-Scores. Ergänzungen zu Abschnitt 5.2.2.

Abschnitt A.3: Vollständige Auflistung aller PR-Kurven mit Baseline. Ergänzungen zu Abschnitt 5.2.3.

Abschnitt A.4: Trainingsverlauf der drei unabhängigen Baseline-Trainings. Ergänzungen zu Abschnitt 5.2.1.

Abschnitt A.5: Vergleich des Trainingsverlaufs mit der Baseline. Ergänzungen zu Abschnitt 5.2.3.

Abschnitt A.6: Vollständige Auflistung aller PCA- und t-SNE Darstellungen. Ergänzungen zu Abschnitt 5.3.1.

Abschnitt A.7: Vollständige Auflistung der Gewichtsänderungen aller Architekturen. Ergänzungen zu Abschnitt 5.3.2.

Abschnitt A.8: Grafische Gegenüberstellung aller verwendeter Architekturen. Ergänzungen zu Abschnitt 2.3.

Abschnitt A.9: Vollständiger Verlauf der Sparsity des SE-ResNet50 zu einem gegebenen Bildbeispiel. Ergänzung zu Abschnitt 5.3.5.

Abschnitt A.10: Visualisierung der Ausgabeschicht mittels Feature-Attention für Positiv- und Negativbeispiele. Ergänzungen zu Abschnitt 5.3.6.

A.1 Fisher-Scores

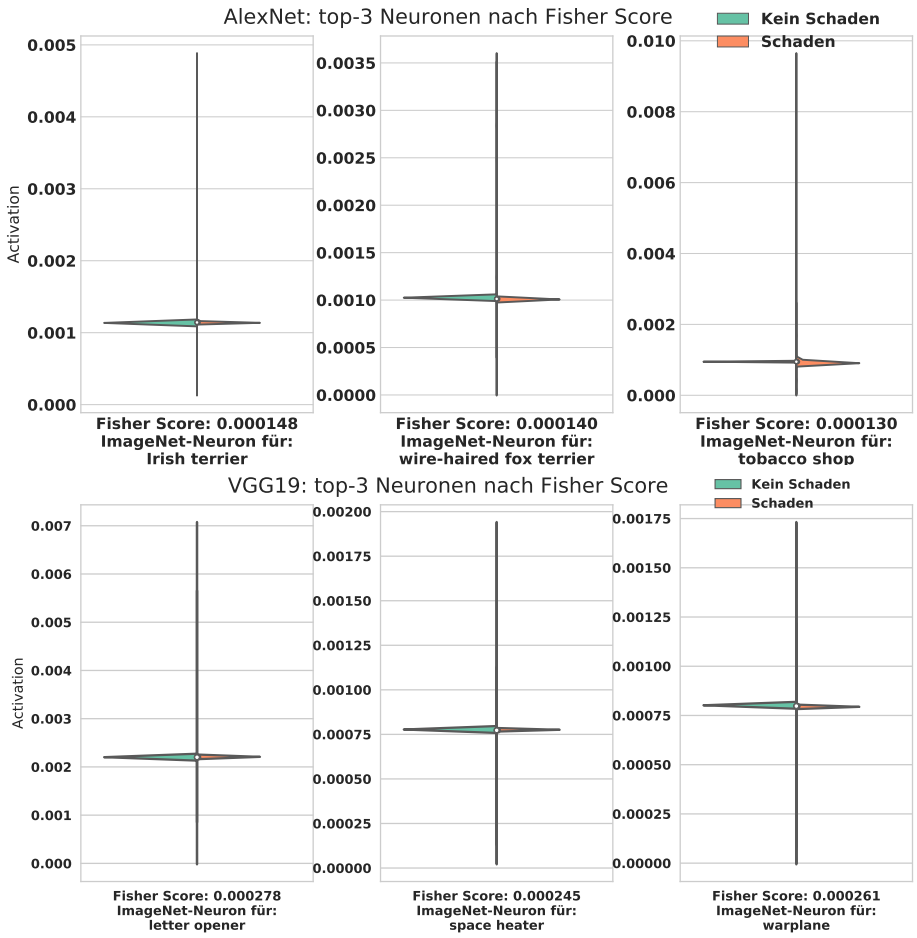


Abbildung A.1: Ausgabeverteilung der fünf besten Neuronen

Zu sehen ist die Ausgabeverteilung der fünf besten Neuronen entsprechend dem Fisher-Score. Abgebildet sind das AlexNet (oben) und VGG19 (unten).

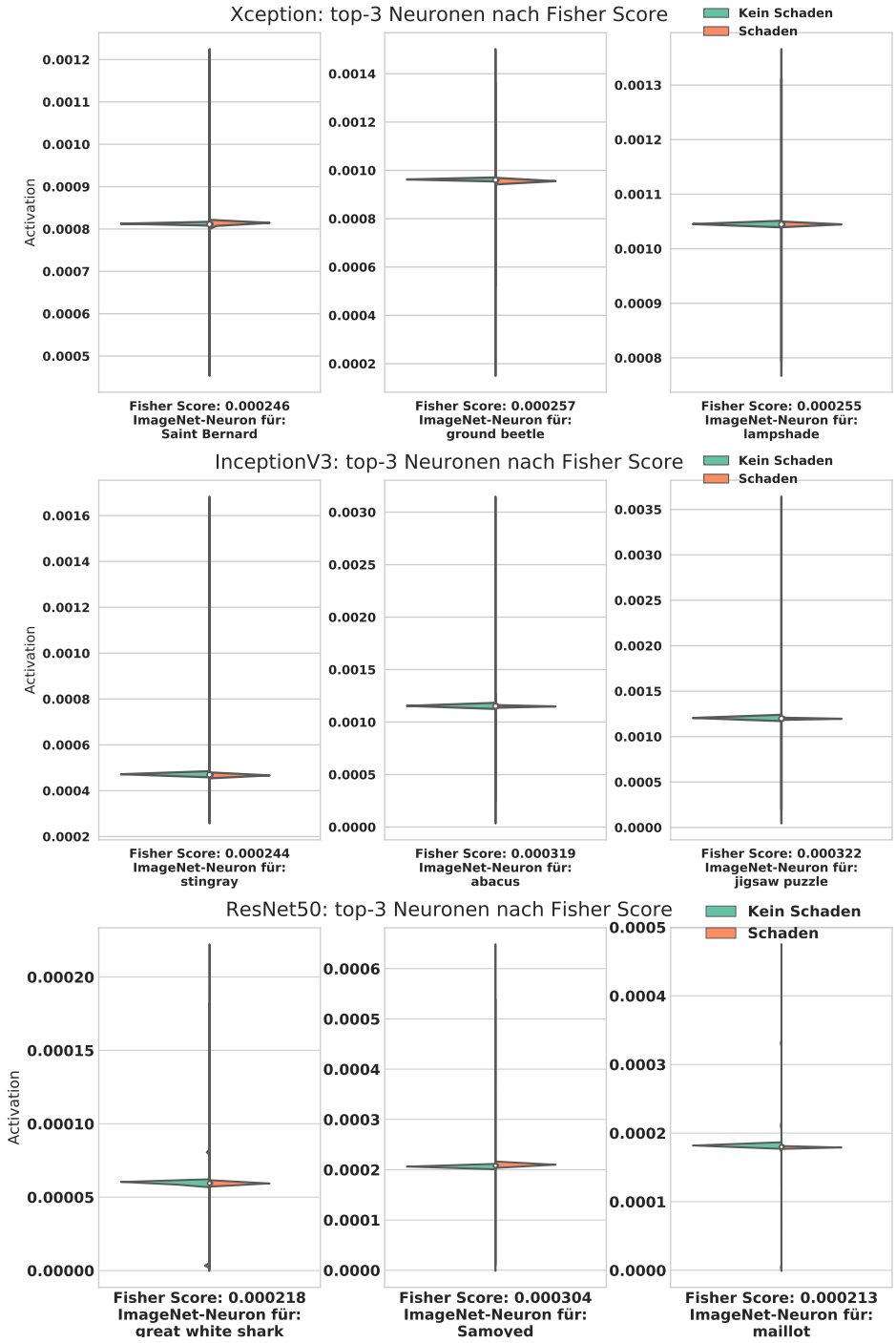


Abbildung A.2: Ausgabeverteilung der fünf besten Neuronen

Zu sehen ist die Ausgabeverteilung der fünf besten Neuronen entsprechend dem Fisher-Score. Abgebildet sind das XceptionNet (oben), InceptionNetV3 (mittig) und ResNet50 (unten).

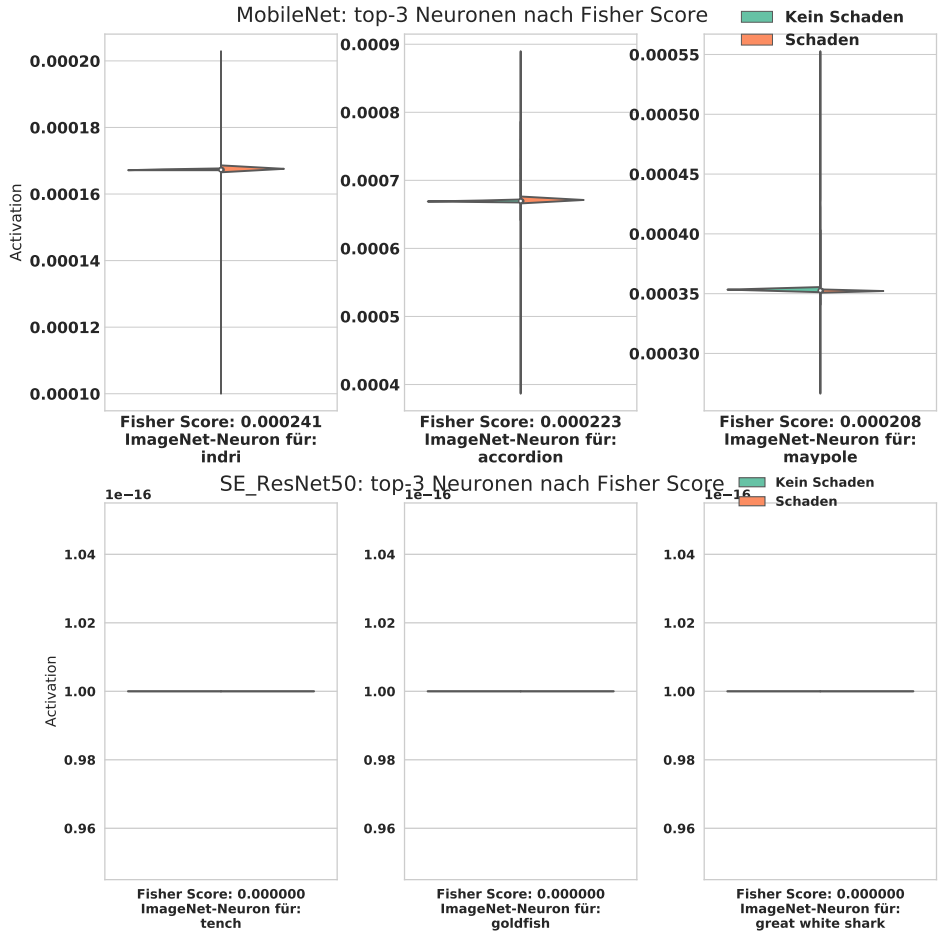


Abbildung A.3: Ausgabeverteilung der fünf besten Neuronen

Zu sehen ist die Ausgabeverteilung der fünf besten Neuronen entsprechend dem Fisher-Score. Abgebildet ist MobileNet (oben) und SE-ResNet50 (unten). Im Fall des SE-ResNet50 ist der Fisher-Score für alle Ausgabeneuronen gleich Null, da für alle Bildbeispiele, unabhängig ob mit oder ohne Schaden, stets das gleiche Neuron mit einer Ausgabe von glatt Eins aktiviert gewesen ist.

A.2 Heatmap

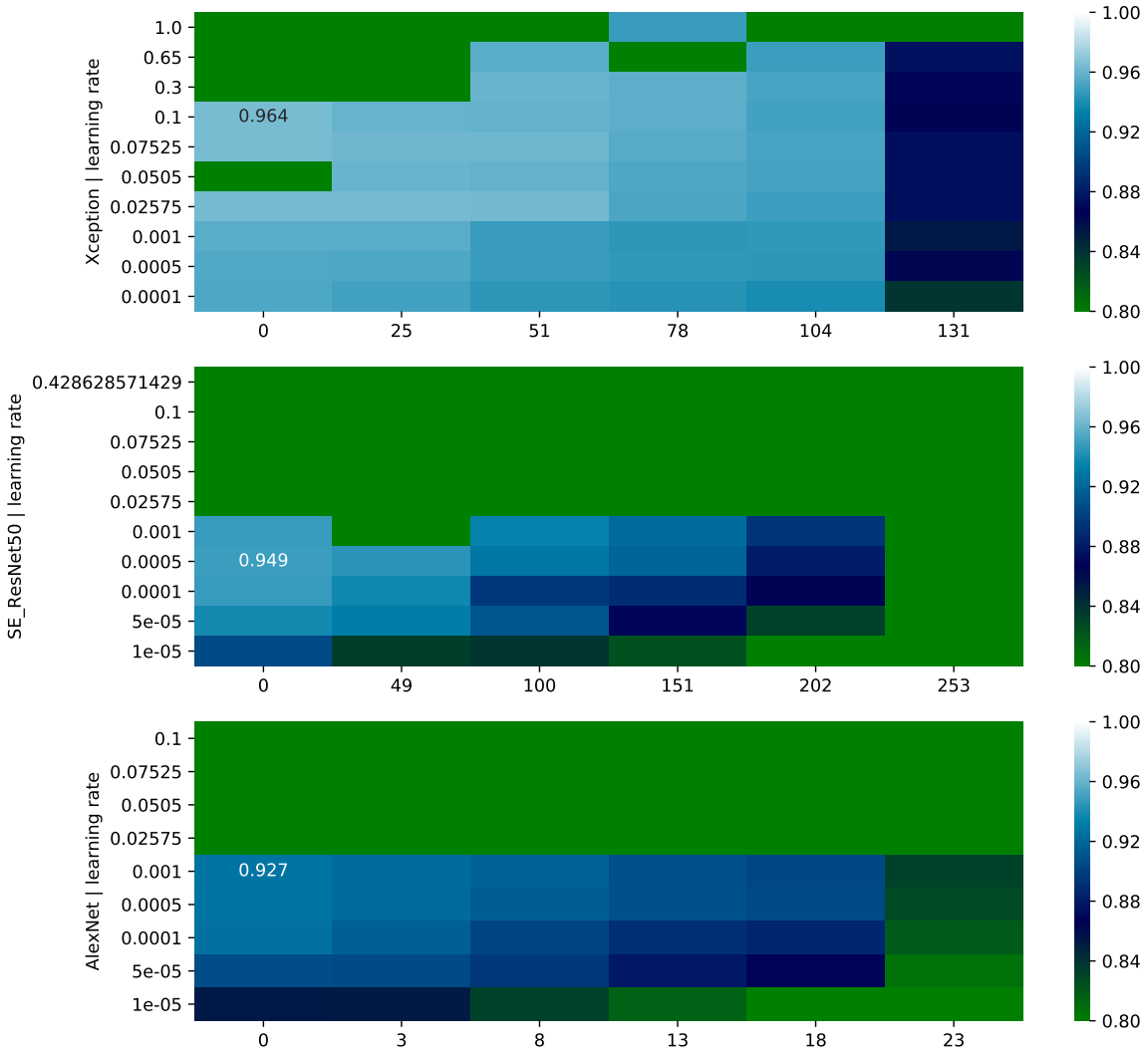


Abbildung A.4: Performance-Heatmap der Validation-Accuracy

Zu sehen ist die Höhe der Accuracy auf dem Validation-Datensatz der jeweils besten Epoche eines Trainingsdurchlaufes. Grün symbolisiert eine schlechte Leistung und blau bzw. weiß eine sehr gute. Darüber hinaus ist zu jeder Architektur die Konfiguration aus Lernrate und eingefrorenen Schichten hervorgehoben, welche die Bestleistung über alle Konfigurationen auf der Accuracy besitzt.

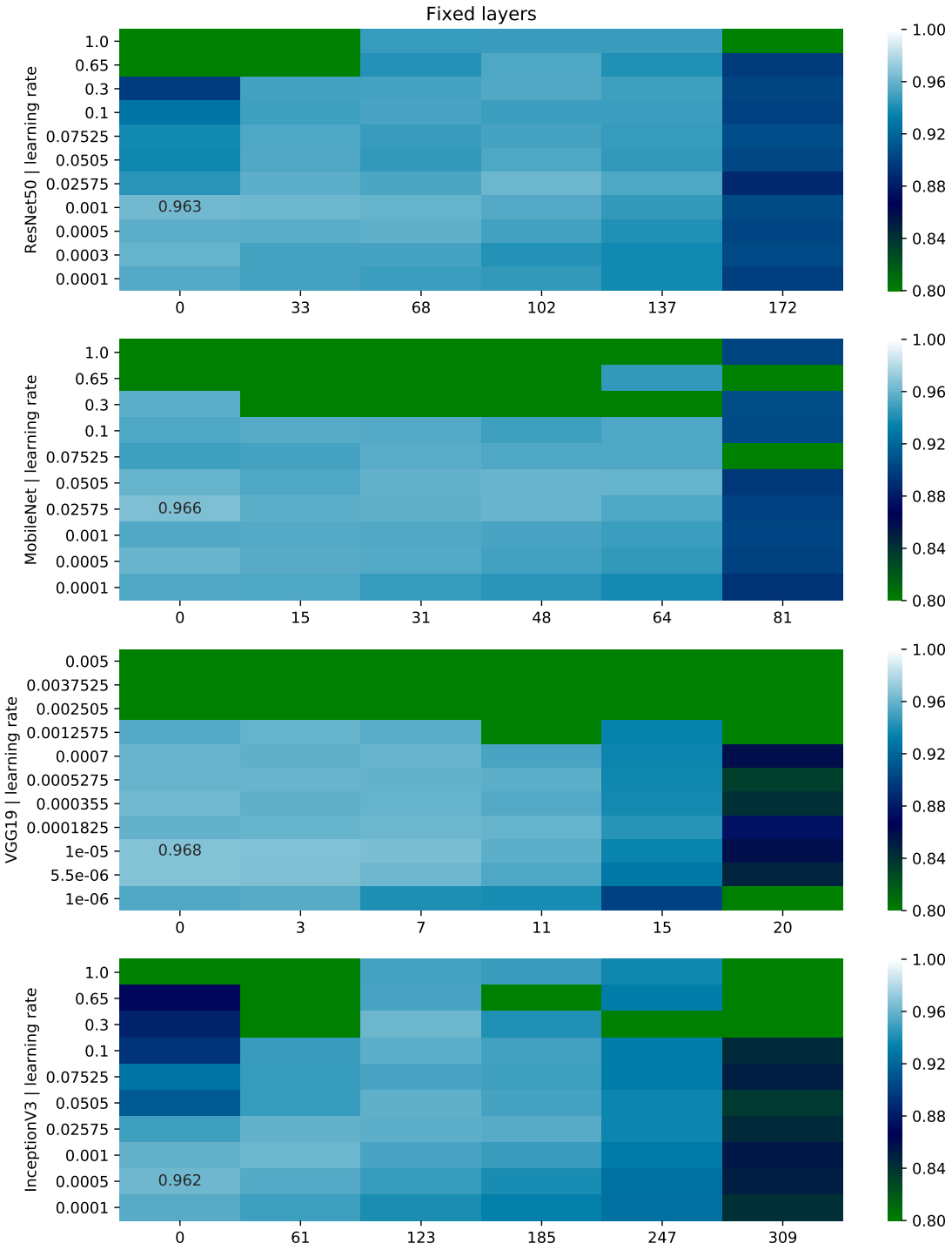


Abbildung A.5: Performance-Heatmap der Validation-Accuracy

Zu sehen ist die Höhe der Accuracy auf dem Validation-Datensatz der jeweils besten Epoche eines Trainingsdurchlaufes. Die Farbkodierung und Hervorhebungen sind analog zu Abbildung A.4.

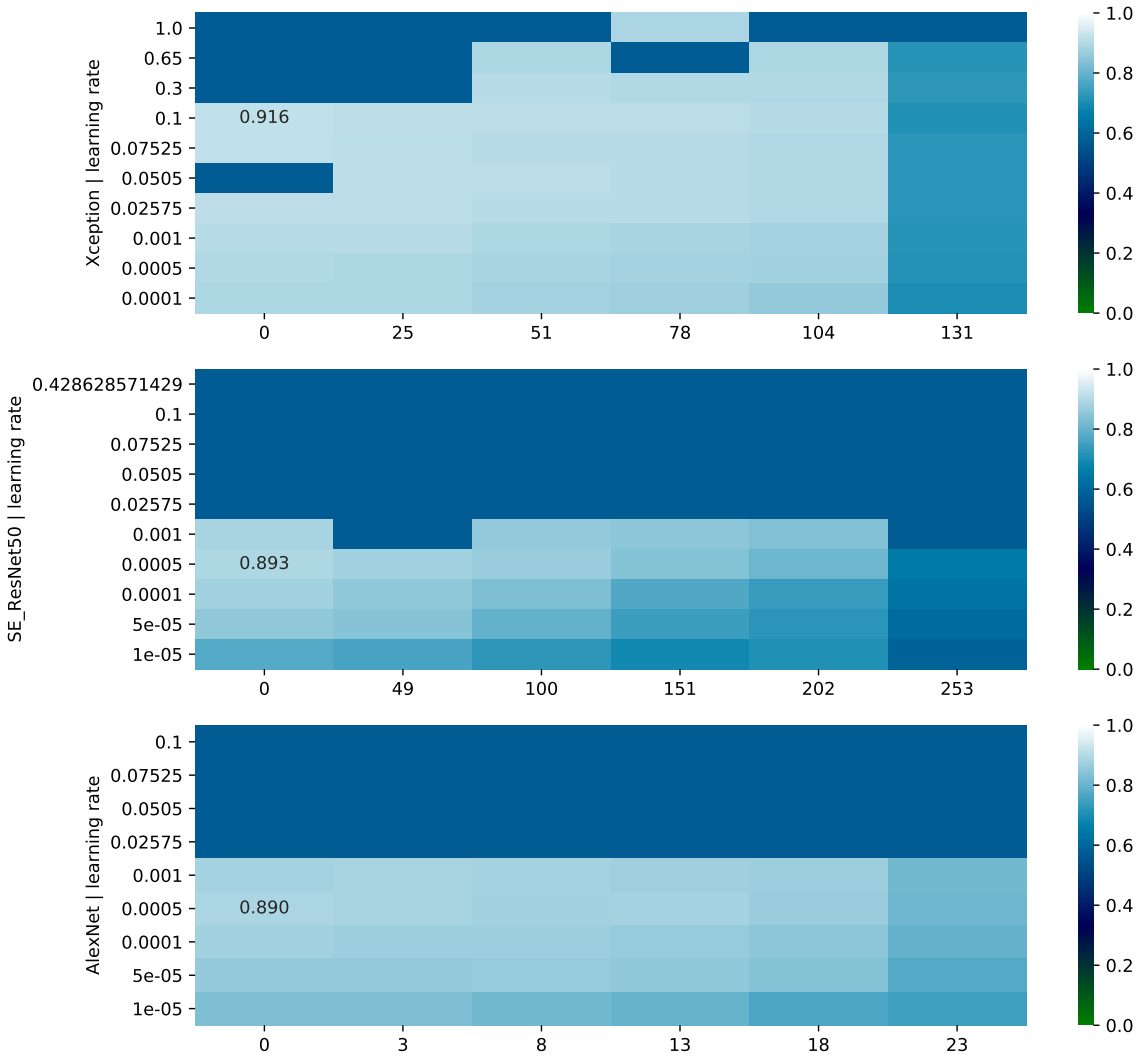


Abbildung A.6: Performance-Heatmap des Validation-Test-F1-Scores

Zu sehen ist die Höhe des F1-Scores auf dem Validation-Test-Datensatz der jeweils besten Epoche eines Trainingsdurchlaufes. Grün symbolisiert eine schlechte Leistung und blau bzw. weiß eine sehr gute. Darüber hinaus ist zu jeder Architektur die Konfiguration aus Lernrate und eingefrorenen Schichten hervorgehoben, welche die Bestleistung über alle Konfigurationen auf dem F1-Score besitzt.

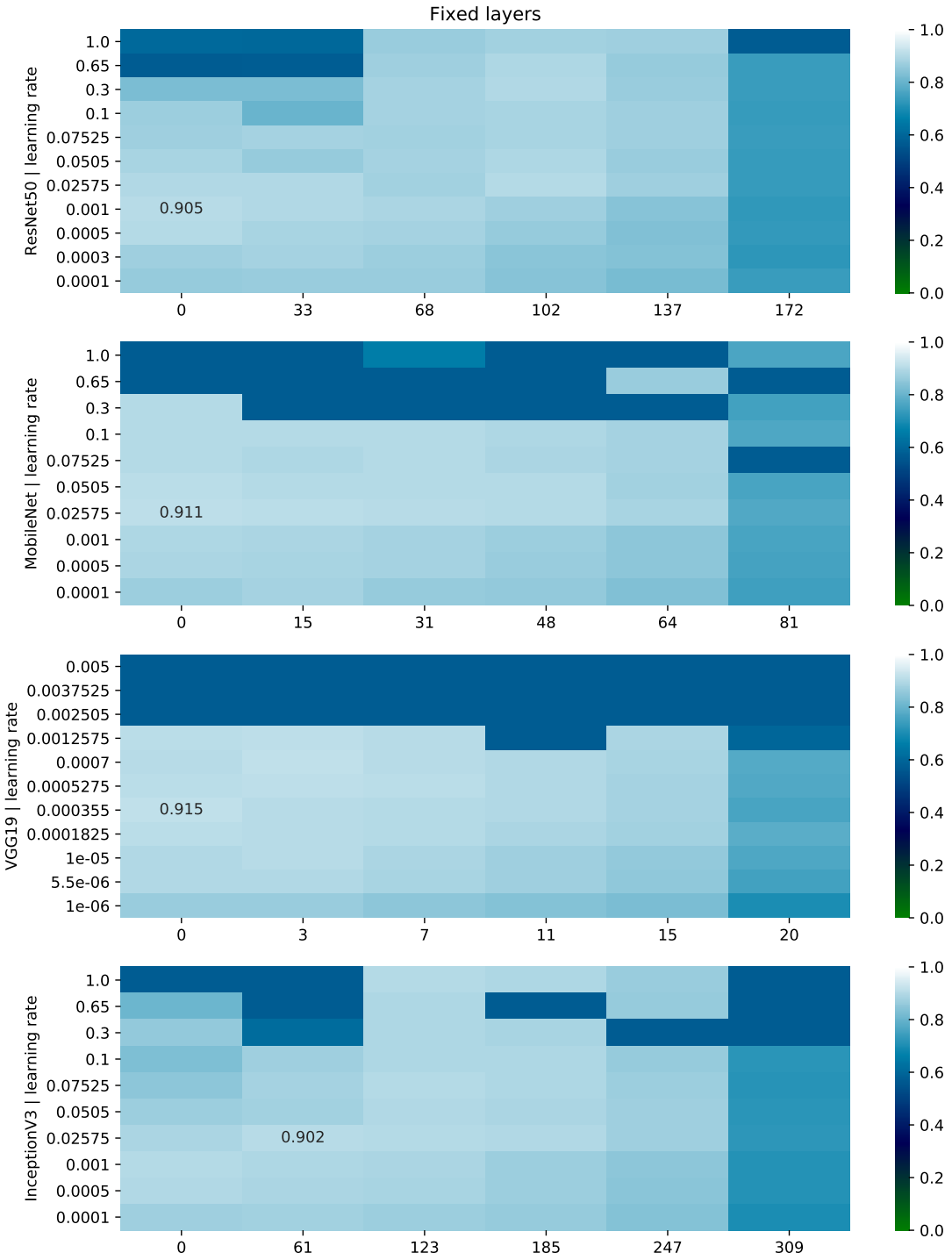


Abbildung A.7: Performance-Heatmap des Validation-Test-F1-Scores

Zu sehen ist die Höhe der F1-Scores auf dem Validation-Test-Datensatz der jeweils besten Epoche eines Trainingsdurchlaufes. Die Farbkodierung und Hervorhebungen sind analog zu Abbildung A.6.

A.3 PR-Kurven

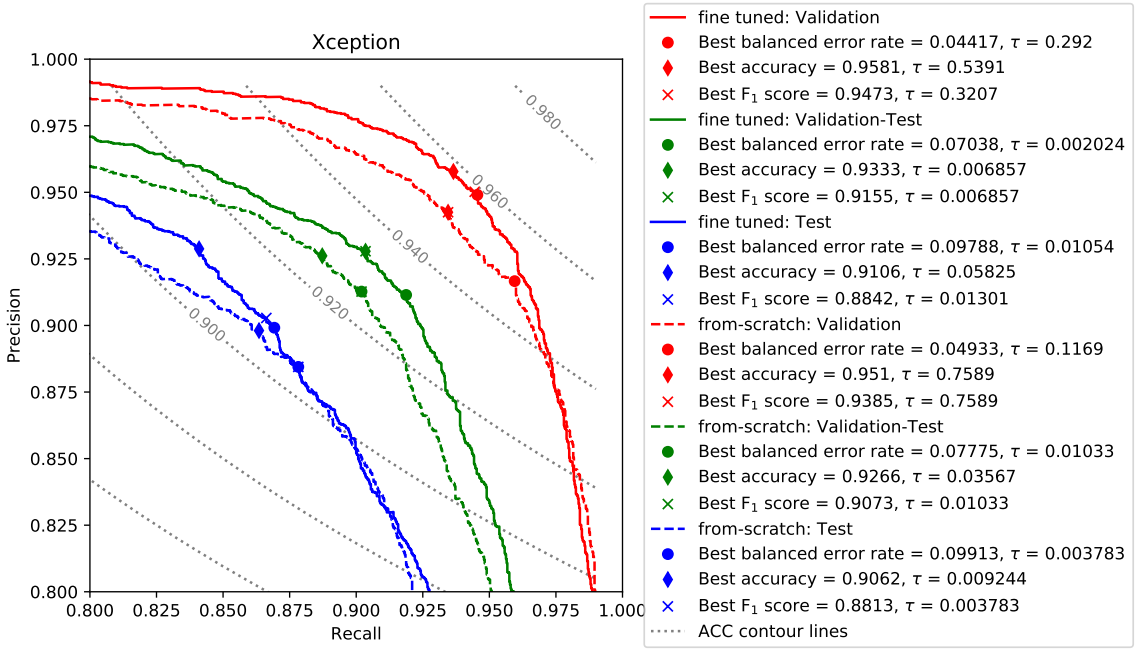
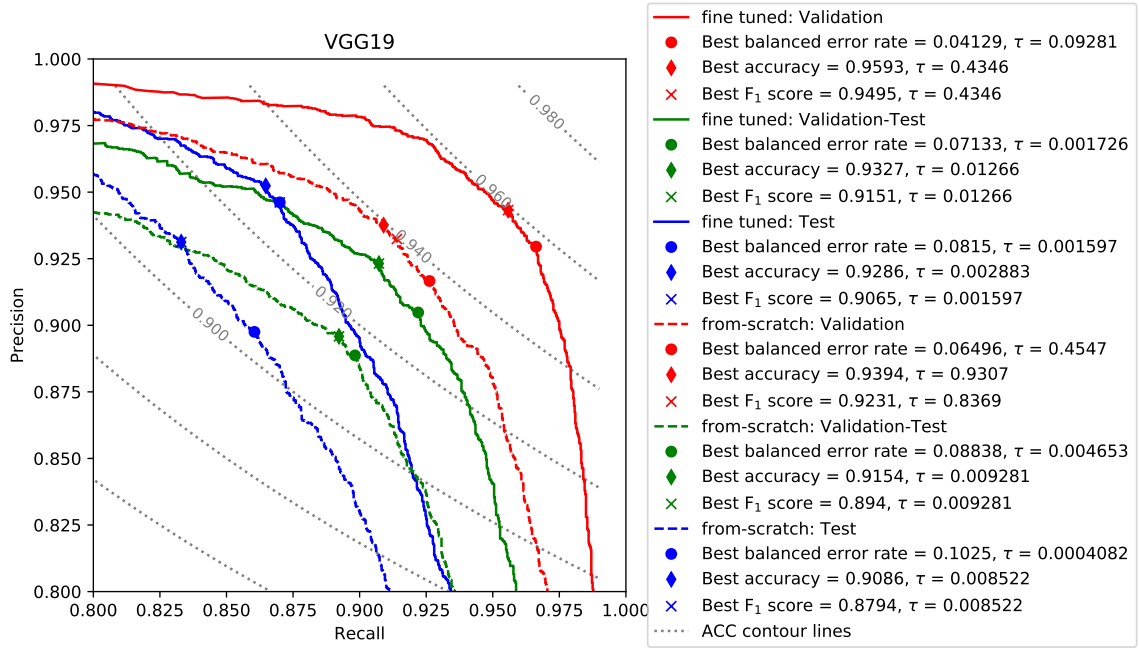


Abbildung A.8: PR-Kurve des XceptionNet

Zu sehen ist der Verlauf verschiedener Performancemaße in verschiedenen Farben. Der Verlauf der Baseline ist gestrichelt eingezeichnet.



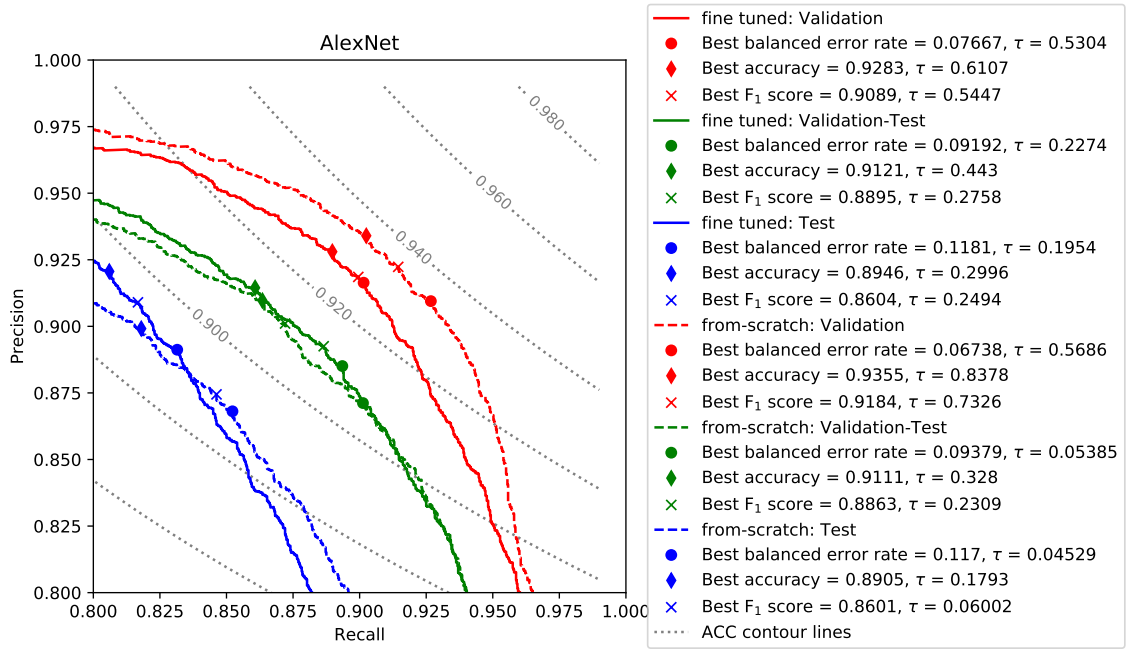


Abbildung A.9: PR-Kurve des VGG19 und AlexNet

Zu sehen ist der Verlauf verschiedener Performancemaße in verschiedenen Farben. Der Verlauf der Baseline ist gestrichelt eingezeichnet.

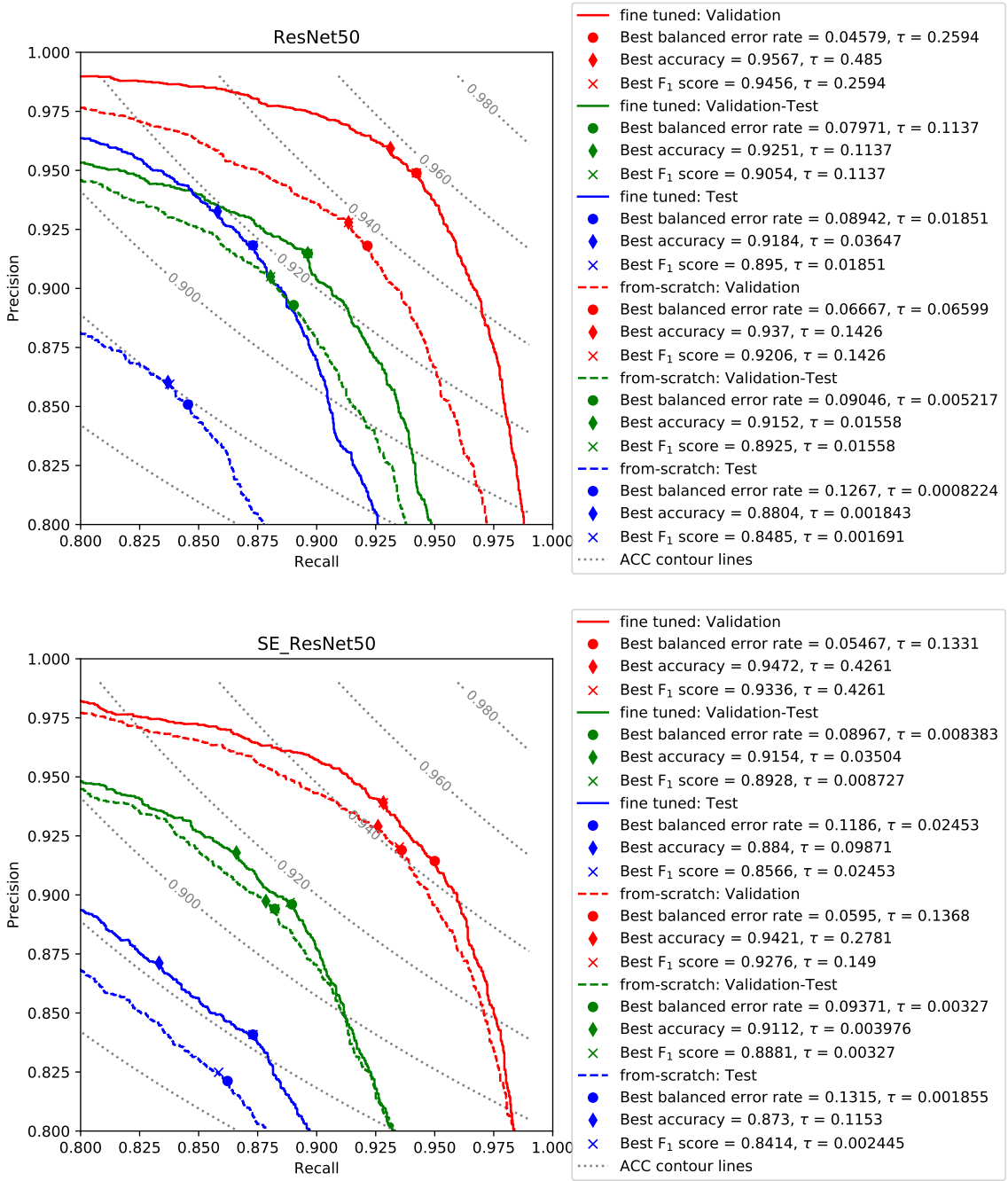
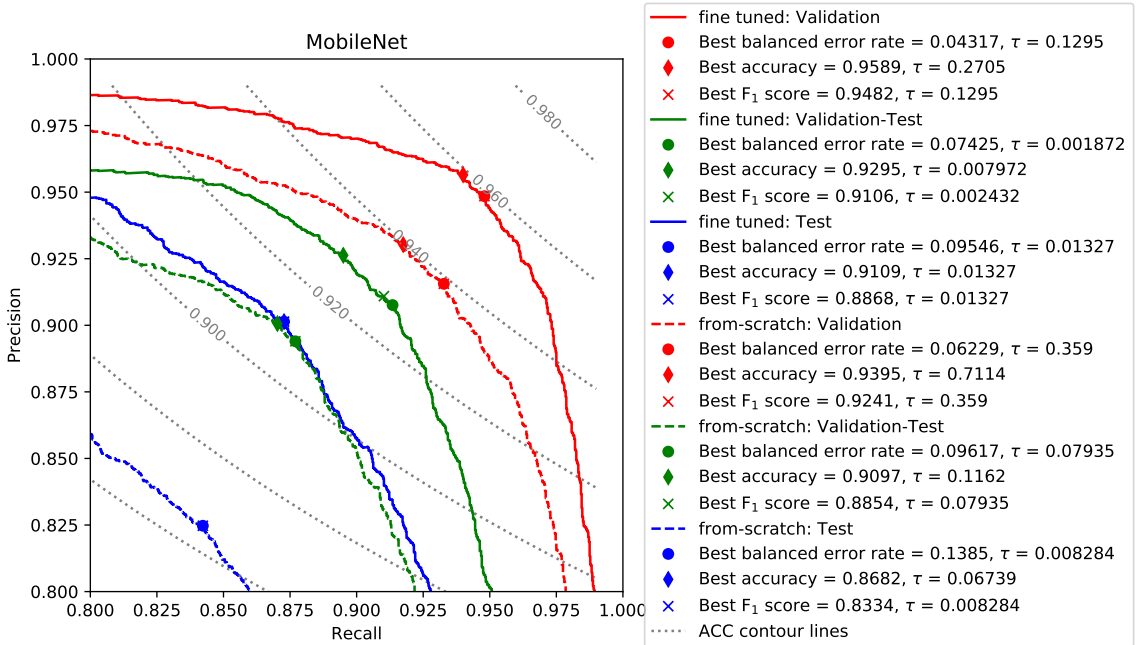


Abbildung A.10: PR-Kurve des ResNet50 und SE-ResNet50

Zu sehen ist der Verlauf verschiedener Performancemaße in verschiedenen Farben. Der Verlauf der Baseline ist gestrichelt eingezeichnet.



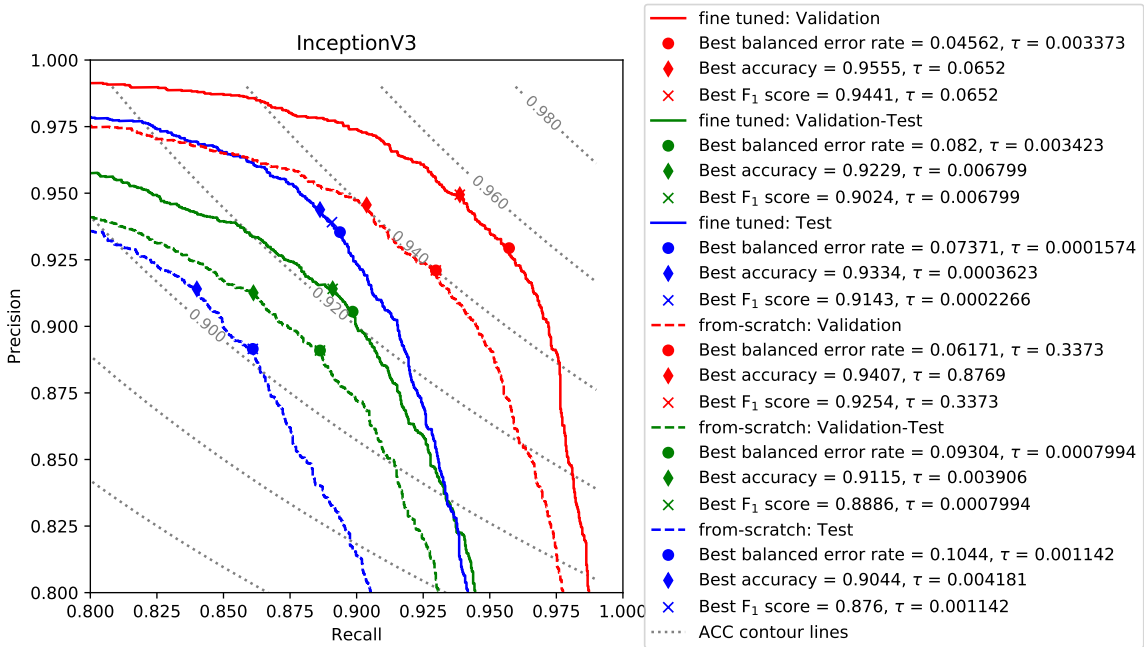


Abbildung A.11: PR-Kurve des MobileNet und InceptionNetV3

Zu sehen ist der Verlauf verschiedener Performancemaße in verschiedenen Farben. Der Verlauf der Baseline ist gestrichelt eingezeichnet.

A.4 Vergleich der Baseline

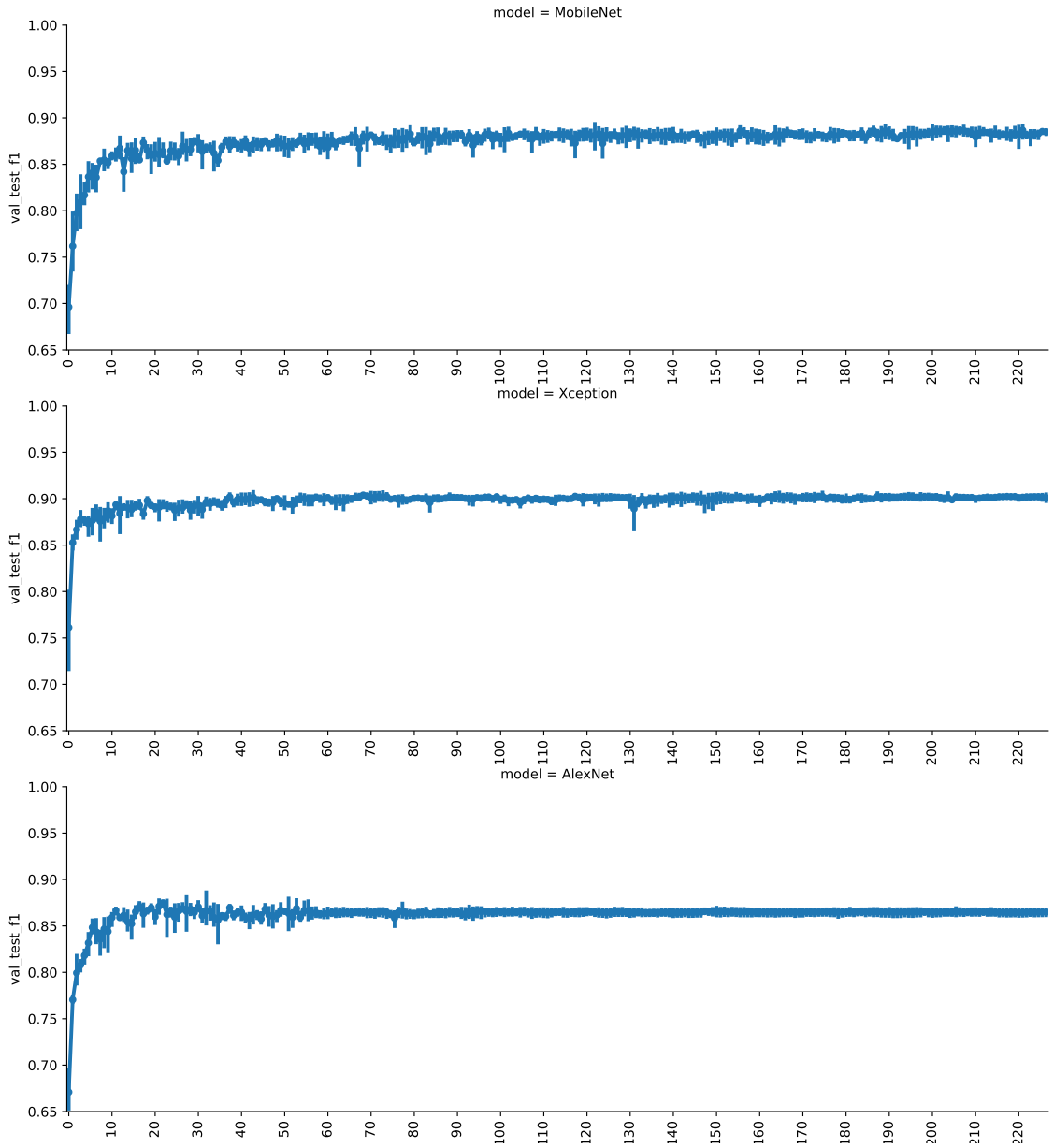


Abbildung A.12: Trainingsverlauf der Baseline

Zu sehen ist der mittlere Validation-Test-F1-Score und das 95%-Konfidenzintervall der Baseline. Abgebildet ist das MobileNet (oben), XceptionNet (mittig) und AlexNet (unten). Es ist gut zu erkennen, dass sich die Bestleistung aller drei Durchläufe lediglich minimal voneinander unterscheidet.

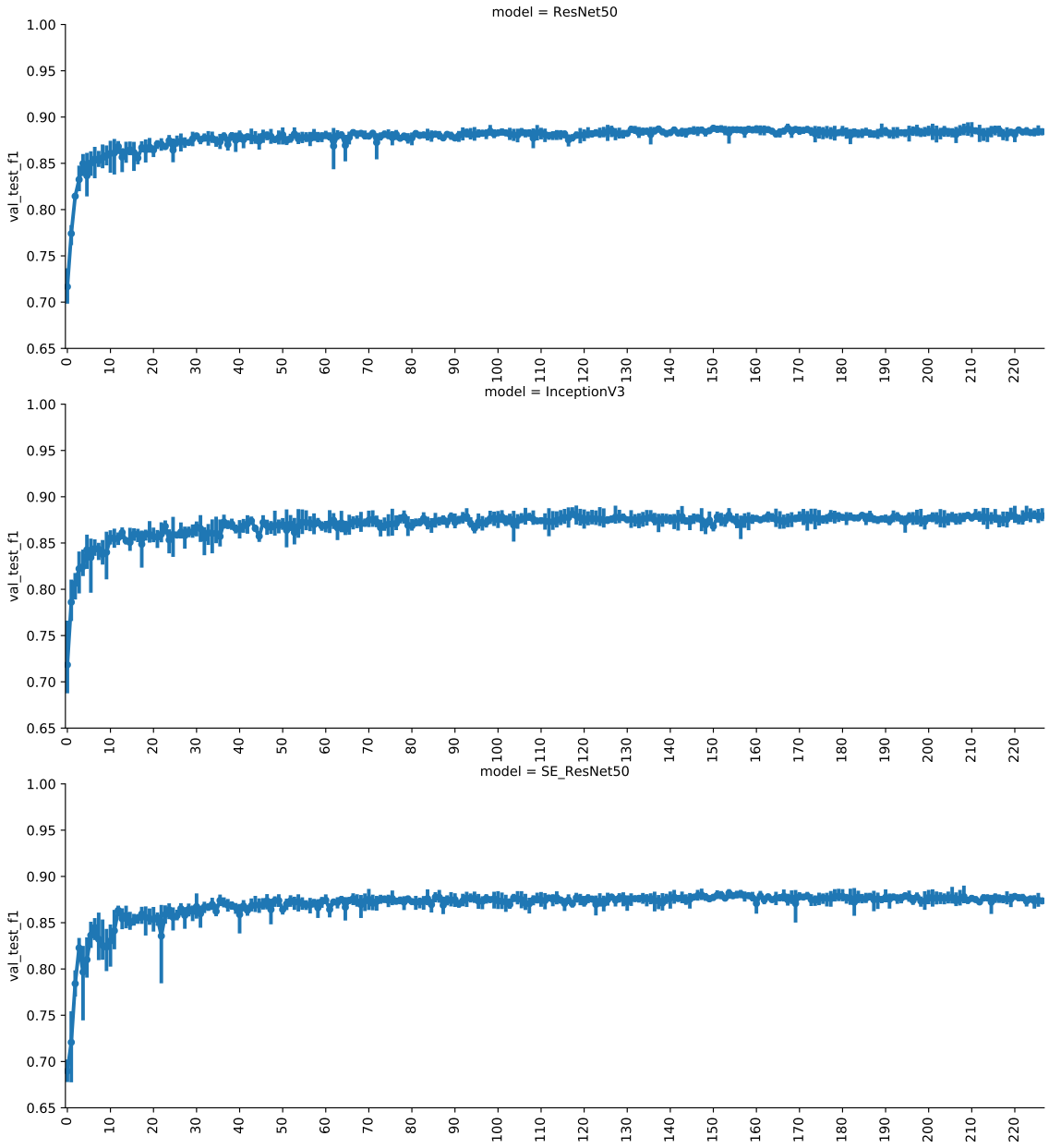


Abbildung A.13: Trainingsverlauf der Baseline

Zu sehen ist der mittlere Validation-Test-F1-Score und das 95%-Konfidenzintervall der Baseline. Abgebildet ist das ResNet50 (oben), InceptionNetV3 (mittig) und SE-ResNet50 (unten). Es ist gut zu erkennen, dass sich die Bestleistung aller drei Durchläufe lediglich minimal voneinander unterscheidet.

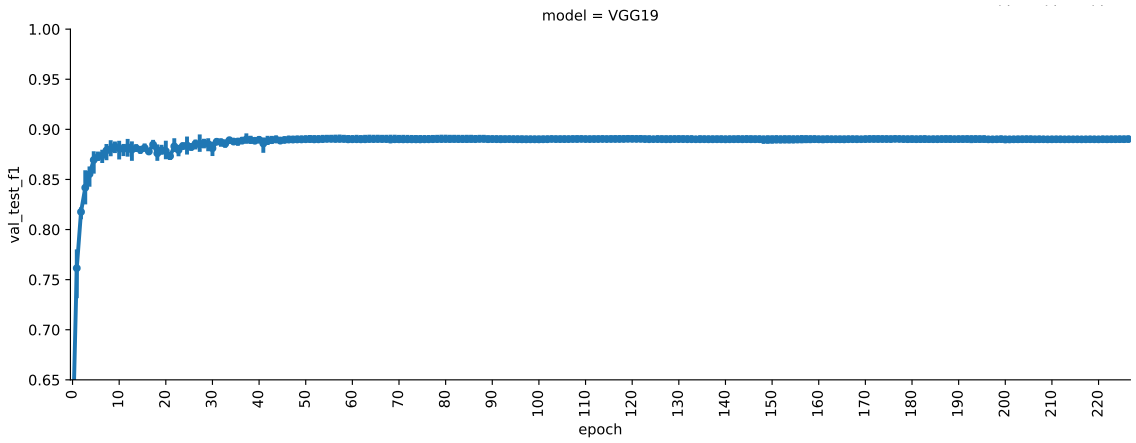
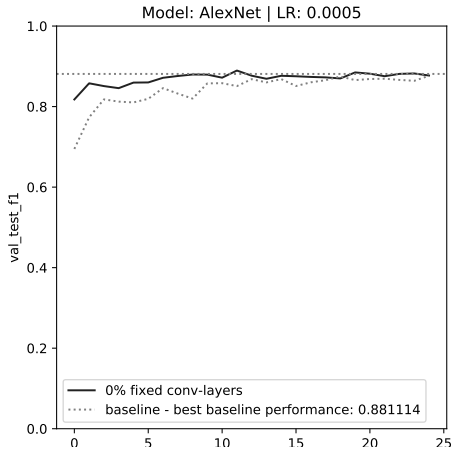


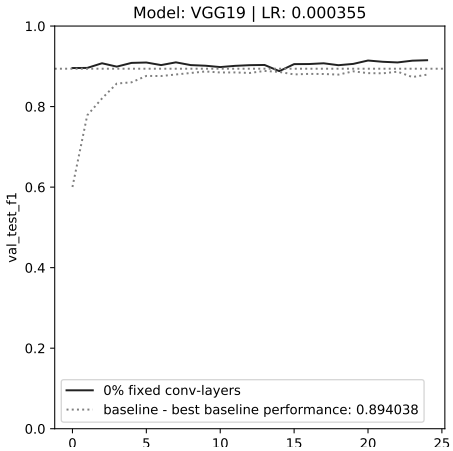
Abbildung A.14: Trainingsverlauf der Baseline

Zu sehen ist der mittlere Validation-Test-F1-Score und das 95%-Konfidenzintervall der Baseline. Abgebildet ist das VGG19. Es ist gut zu erkennen, dass sich die Bestleistung aller drei Durchläufe lediglich minimal voneinander unterscheidet.

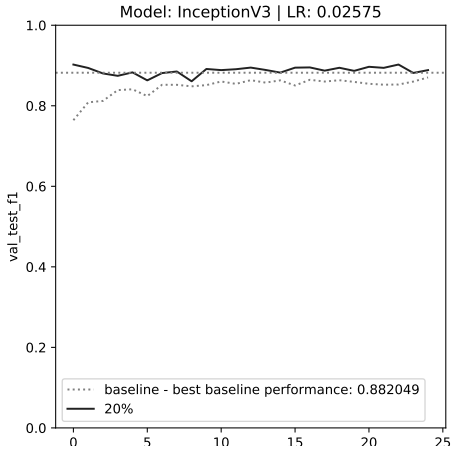
A.5 Vergleich des Trainingsverlaufs



(a)



(b)



(c)

Abbildung A.15: Trainingsverlauf mit Transfer Learning und Baseline
Zu sehen ist der Verlauf der ersten 25 Trainingsepochen des Transfer Learnings und der Baseline (gestrichelt) für das AlexNet (a), VGG19 (b) und InceptionNetV3 (c). Die horizontale Linie repräsentiert dabei den Bestwert der Baseline. Es ist deutlich zu erkennen, dass das Transfer Learning gegenüber der Baseline, in Bezug auf die Trainingsdauer, besser abschneidet. Das Training des InceptionNetV3 entspricht dem besten Fall. Dort konnte bereits durch eine Epoche Training die Bestleistung der Baseline übertroffen werden.

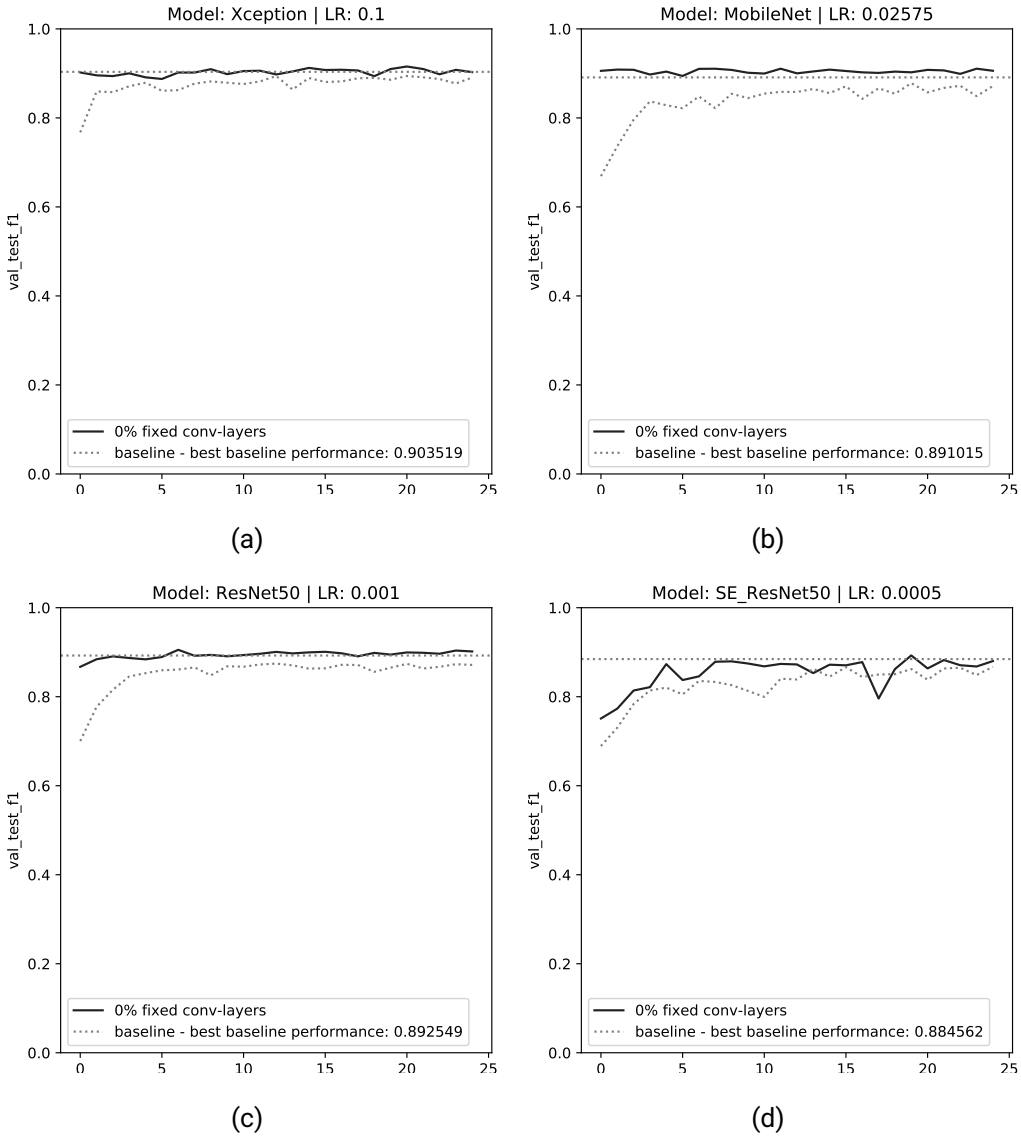
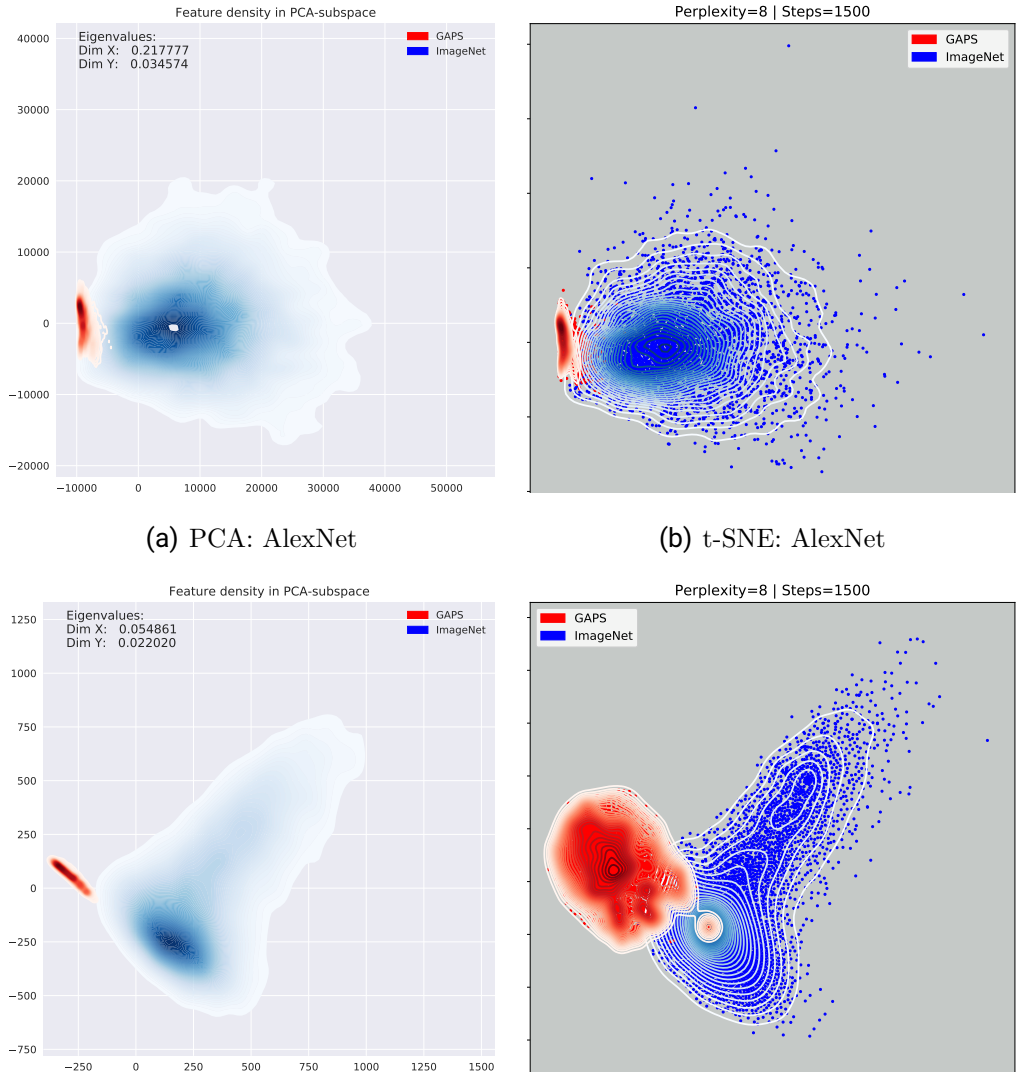


Abbildung A.16: Trainingsverlauf mit Transfer Learning und Baseline
Zu sehen ist der Verlauf der ersten 25 Trainingsepochen des Transfer Learnings und der Baseline (gestrichelt) für das XceptionNet (a), MobileNet (b) , ResNet50 (c) und SE-ResNet50 (d). Die horizontale Linie repräsentiert dabei den Bestwert der Baseline. Es ist deutlich zu erkennen, dass das Transfer Learning gegenüber der Baseline, in Bezug auf die Trainingsdauer, besser abschneidet. Das SE-ResNet50 ist dabei die Architektur, bei welcher der Geschwindigkeitsvorteil für das Training am geringsten ausfiel.

A.6 Trennbarkeit der Datensätze



(a) PCA: AlexNet

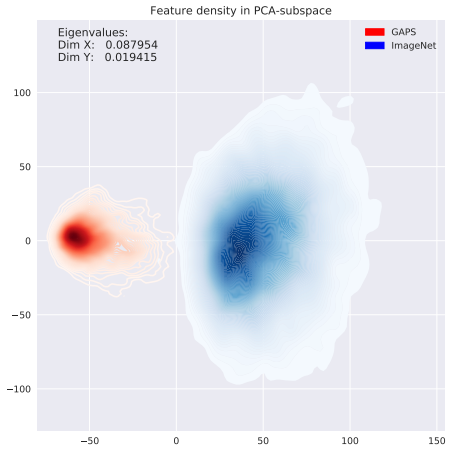
(b) t-SNE: AlexNet

(c) PCA: VGG19

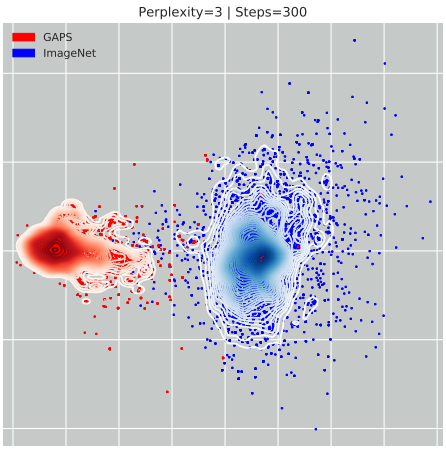
(d) t-SNE: VGG19

Abbildung A.17: PCA- und t-SNE-Darstellung der ImageNet-Feature

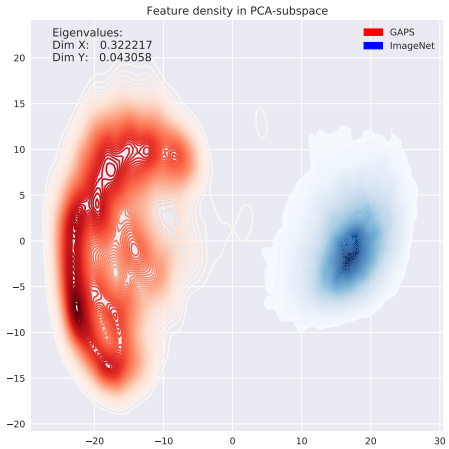
Zu sehen ist die PCA- (links) und t-SNE-Darstellung (rechts) der ImageNet- und GAPs-Daten, aller verwendeter Architekturen. Die Visualisierungen wurden mithilfe der Featuremaps erstellt, welche durch die im Pretraining gelernten ImageNet-Feature erzeugt wurden. Es ist gut zu erkennen, wie ähnlich sich die Darstellungen meist sind. Wie in Abschnitt 2.4.2 beschrieben, gibt die t-SNE-Darstellung jedoch keine zuverlässige Aussage über die Nähe bzw. Größe der ermittelten Cluster.



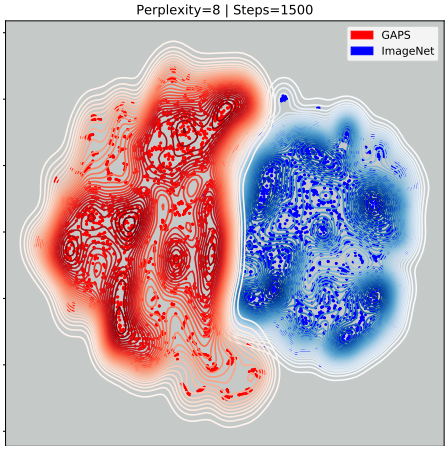
(a) PCA: InceptionV3



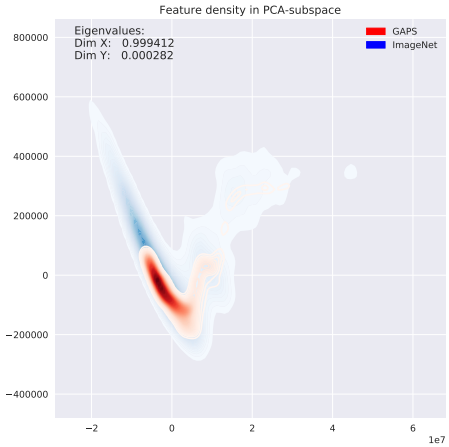
(b) t-SNE: InceptionV3



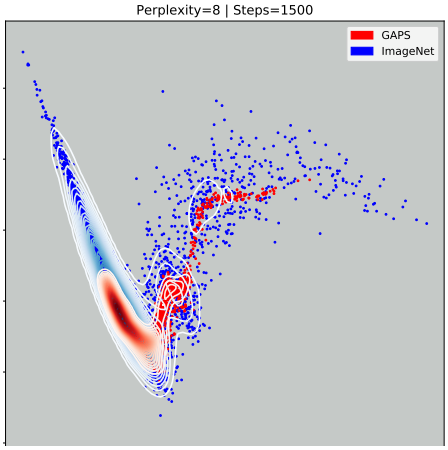
(c) PCA: ResNet50



(d) t-SNE: ResNet50



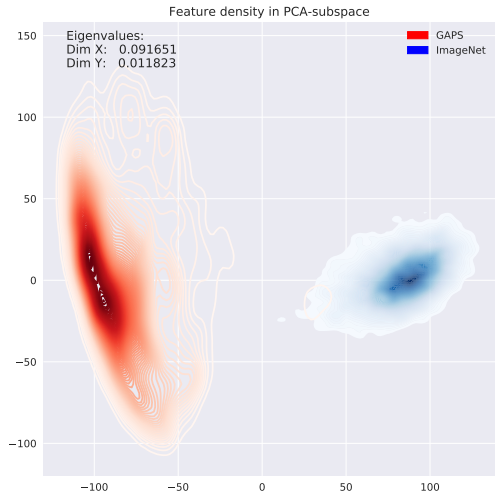
(e) PCA: SE-ResNet50



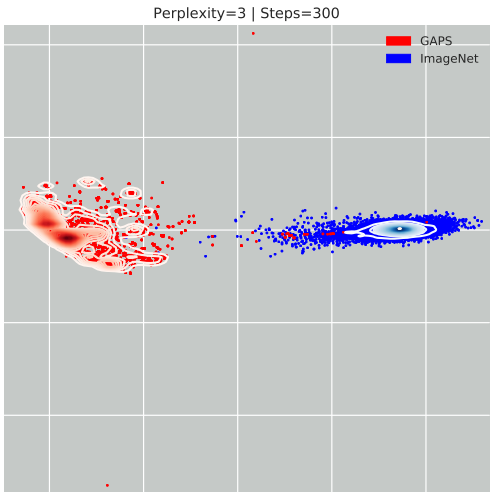
(f) t-SNE: SE-ResNet50

Abbildung A.18: PCA- und t-SNE-Darstellung der ImageNet-Feature

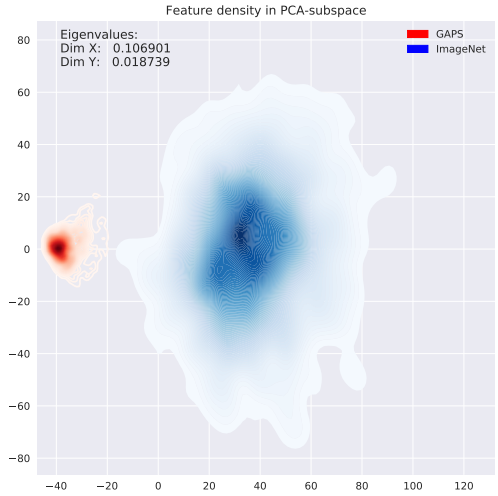
Anmerkungen s. Abbildung A.17.



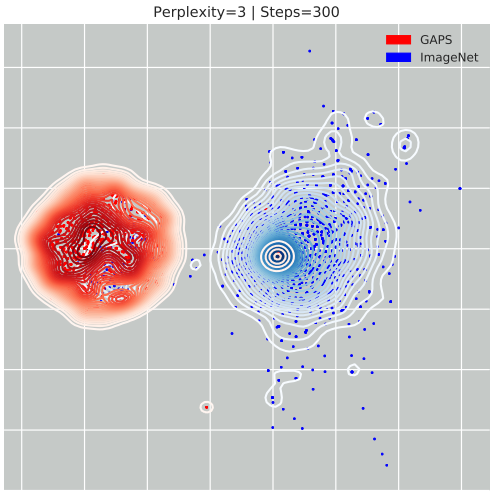
(a) PCA: MobileNet



(b) t-SNE: MobileNet



(c) PCA: XceptionNet



(d) t-SNE: XceptionNet

Abbildung A.19: PCA- und t-SNE-Darstellung der ImageNet-Feature

Anmerkungen s. Abbildung A.17.

A.7 Betragliche Änderung der Gewichte

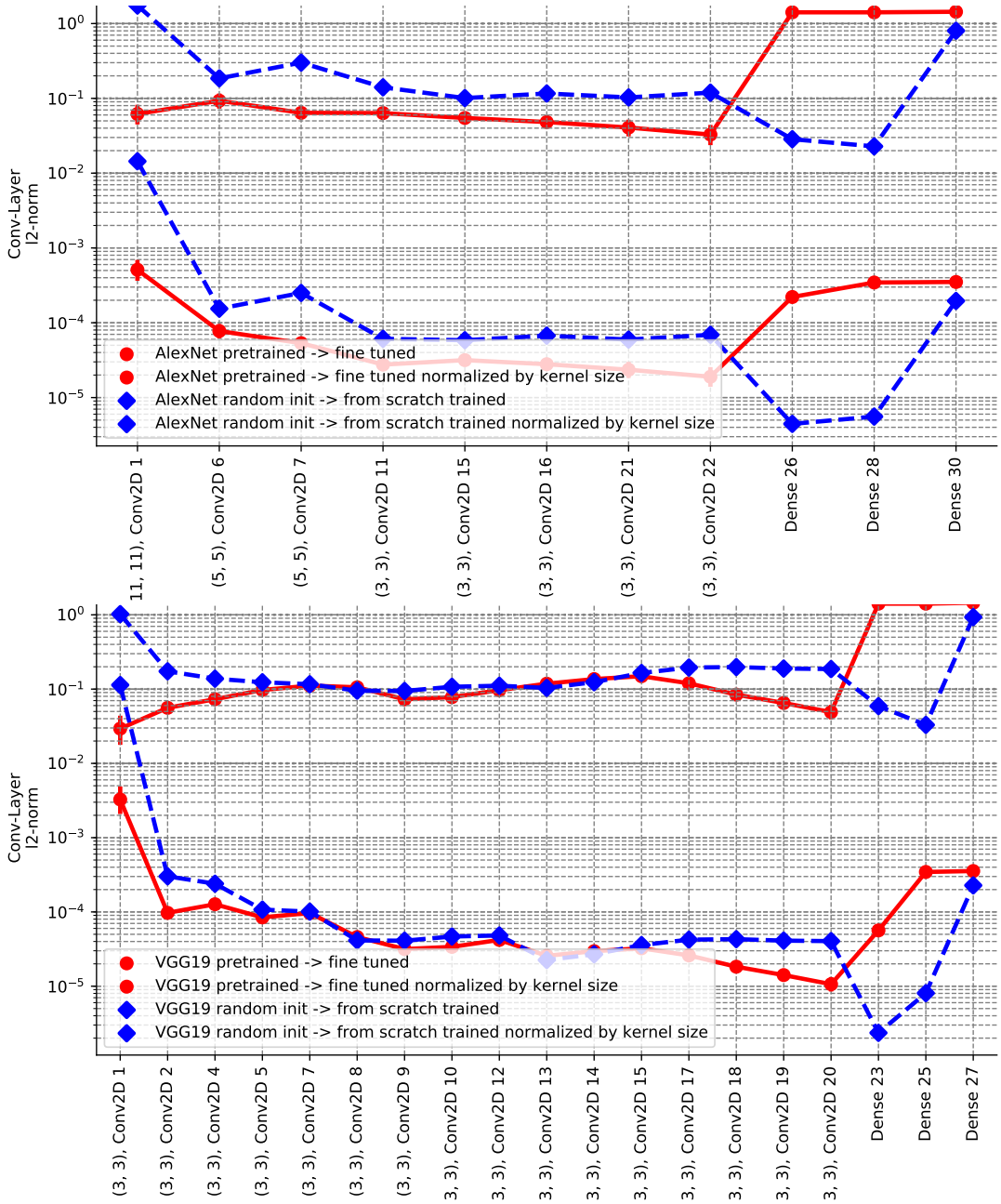


Abbildung A.20: Gewichtsänderung: AlexNet und VGG19

Zu sehen ist die schichtweise Änderung der Gewichte des AlexNet (oben) und VGG19 (unten), logarithmisch abgetragen. Die Berechnungsvorschrift findet sich in Abschnitt 5.3.2.

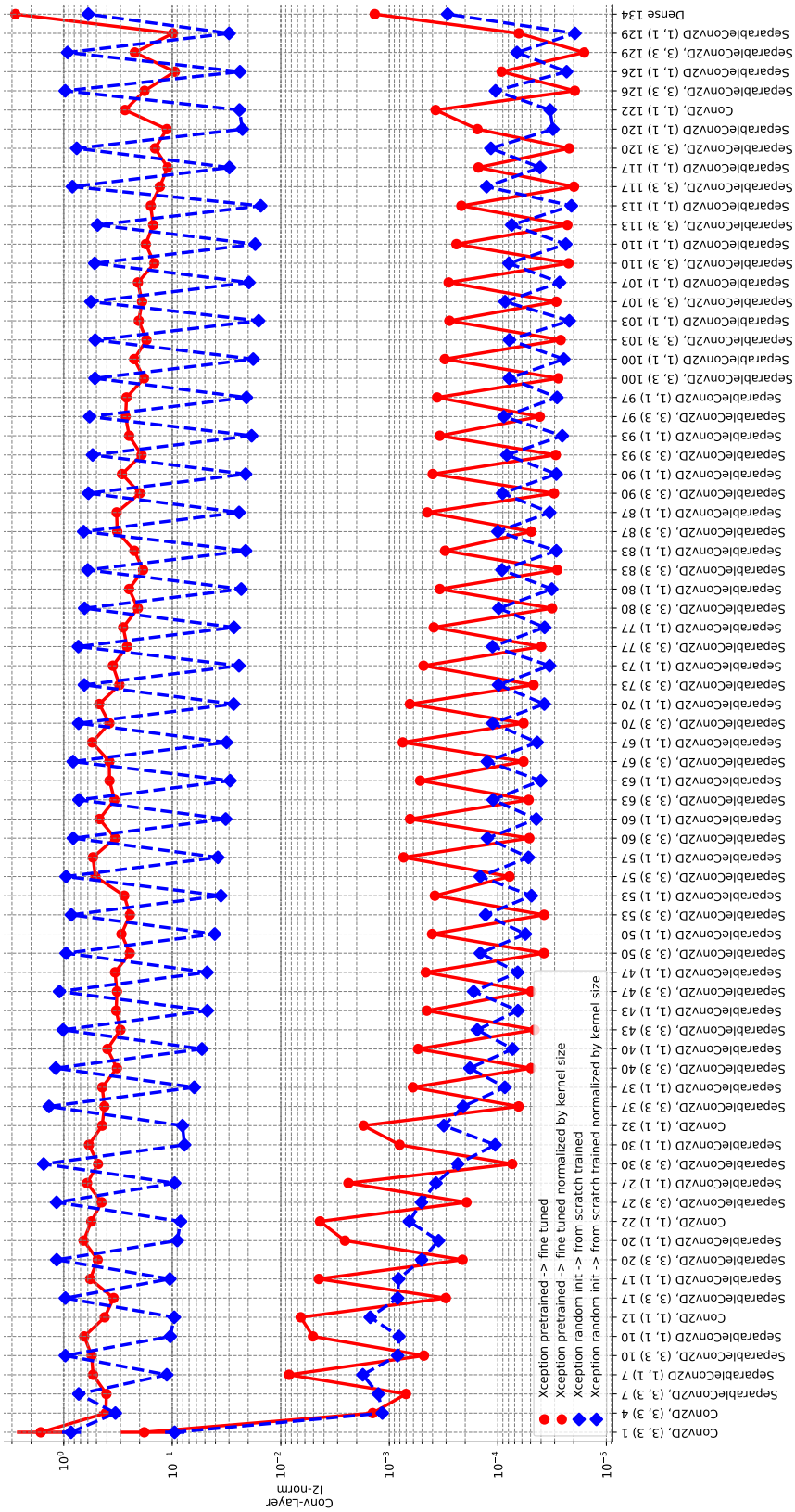


Abbildung A.21: Gewichtsänderung: XceptionNet

Zu sehen ist die schichtweise Änderung der Gewichte, logarithmisch abgetragen. Die Berechnungsvorschrift findet sich in Abschnitt 5.3.2.

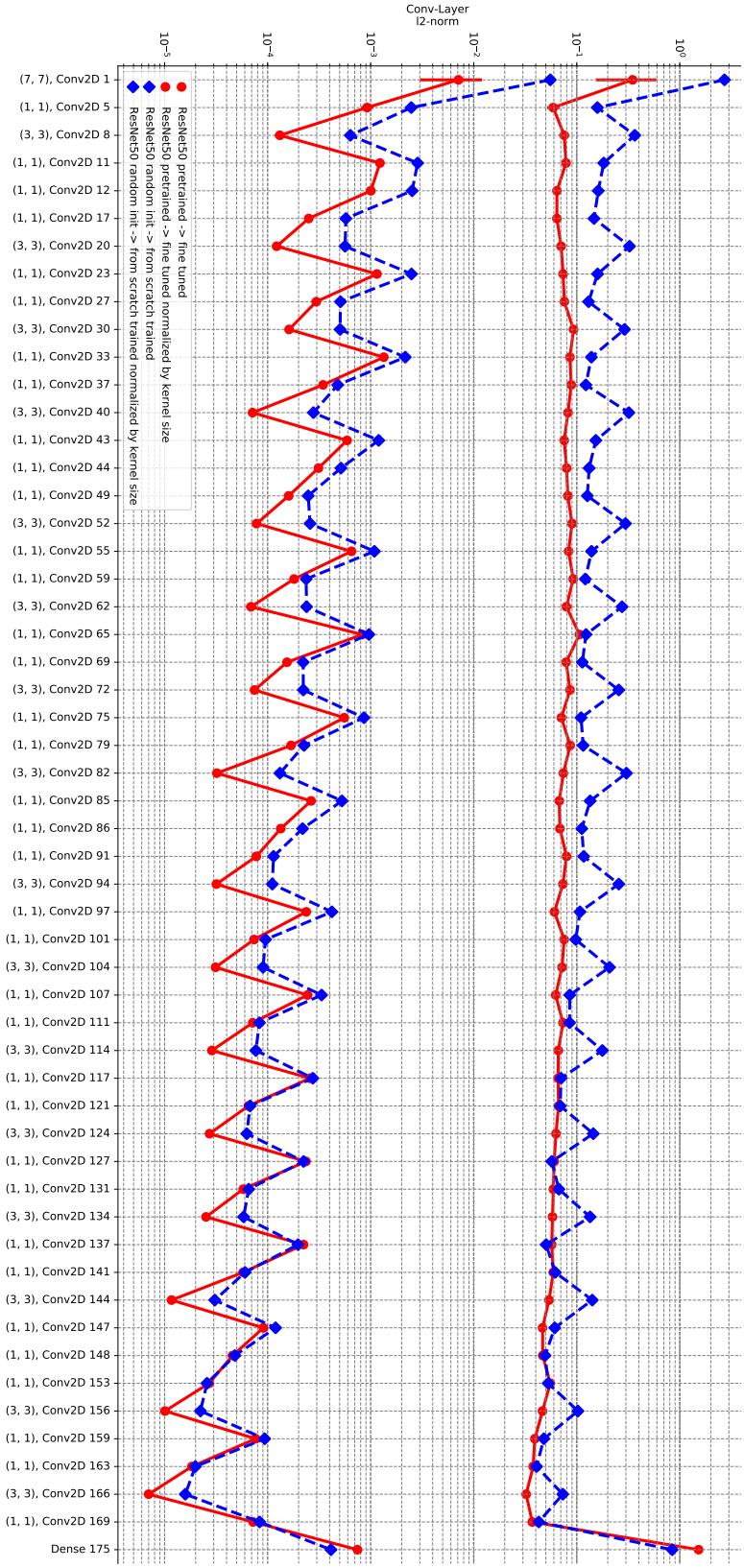


Abbildung A.22: Gewichtsänderung: ResNet50

Zu sehen ist die schichtweise Änderung der Gewichte, logarithmisch abgetragen. Die Berechnungsvorschrift findet sich in Abschnitt 5.3.2.

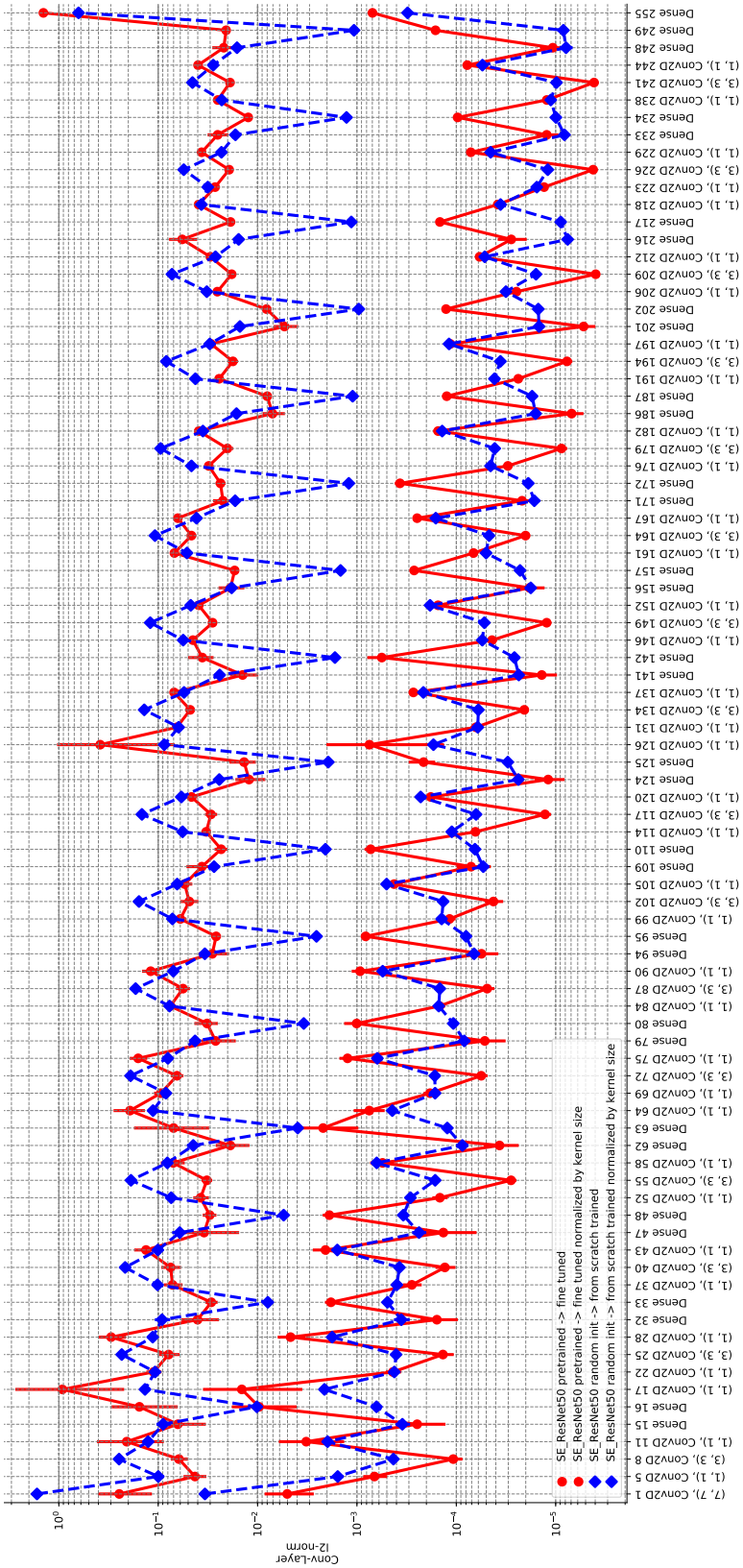


Abbildung A.23: Gewichtsänderung: SE-ResNet50

Zu sehen ist die schichtweise Änderung der Gewichte, logarithmisch abgetragen. Die Berechnungsvorschrift findet sich in Abschnitt 5.3.2.

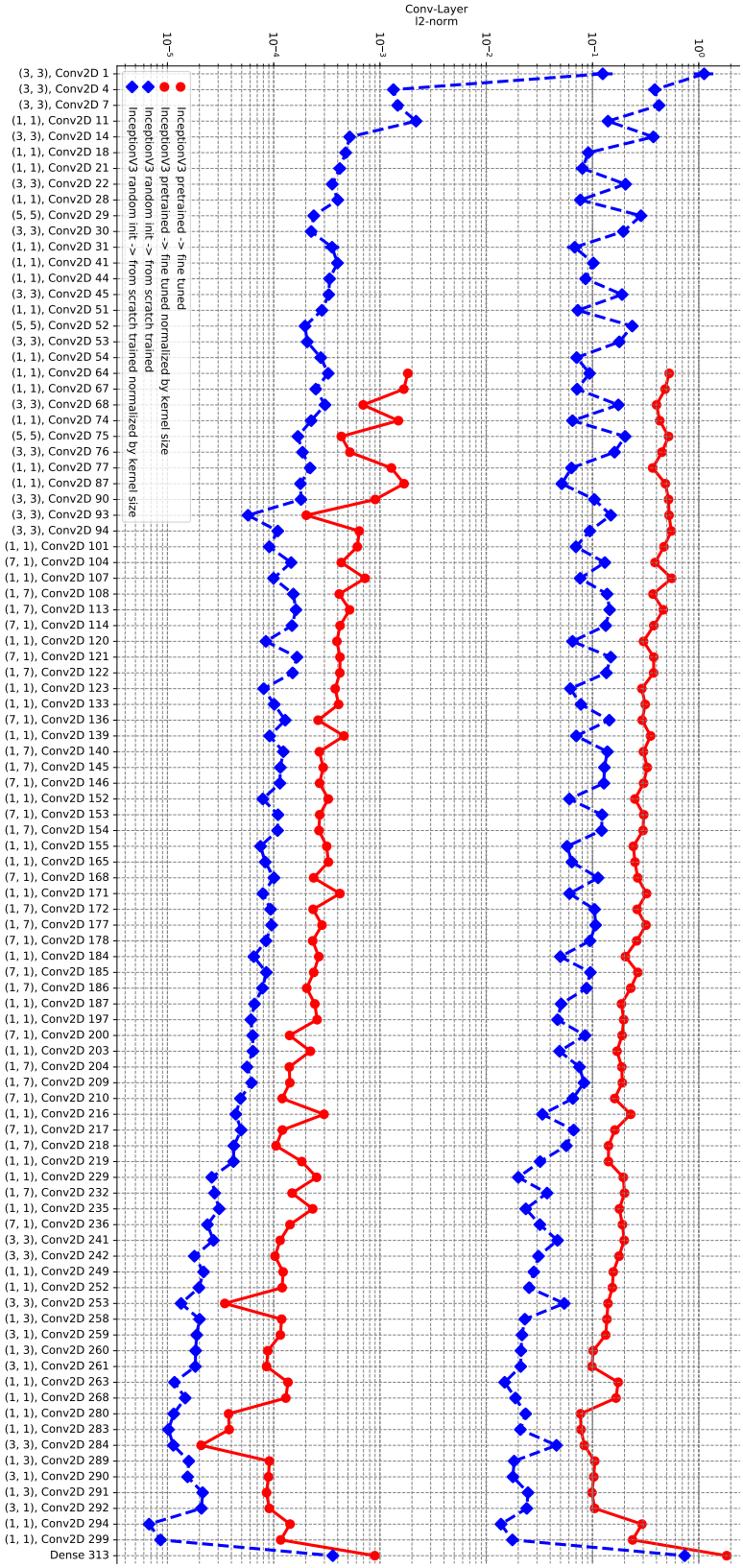


Abbildung A.24: Gewichtsänderung: InceptionNetV3

Zu sehen ist die schichtweise Änderung der Gewichte, logarithmisch abgetragen. Die Berechnungsvorschrift findet sich in Abschnitt 5.3.2.

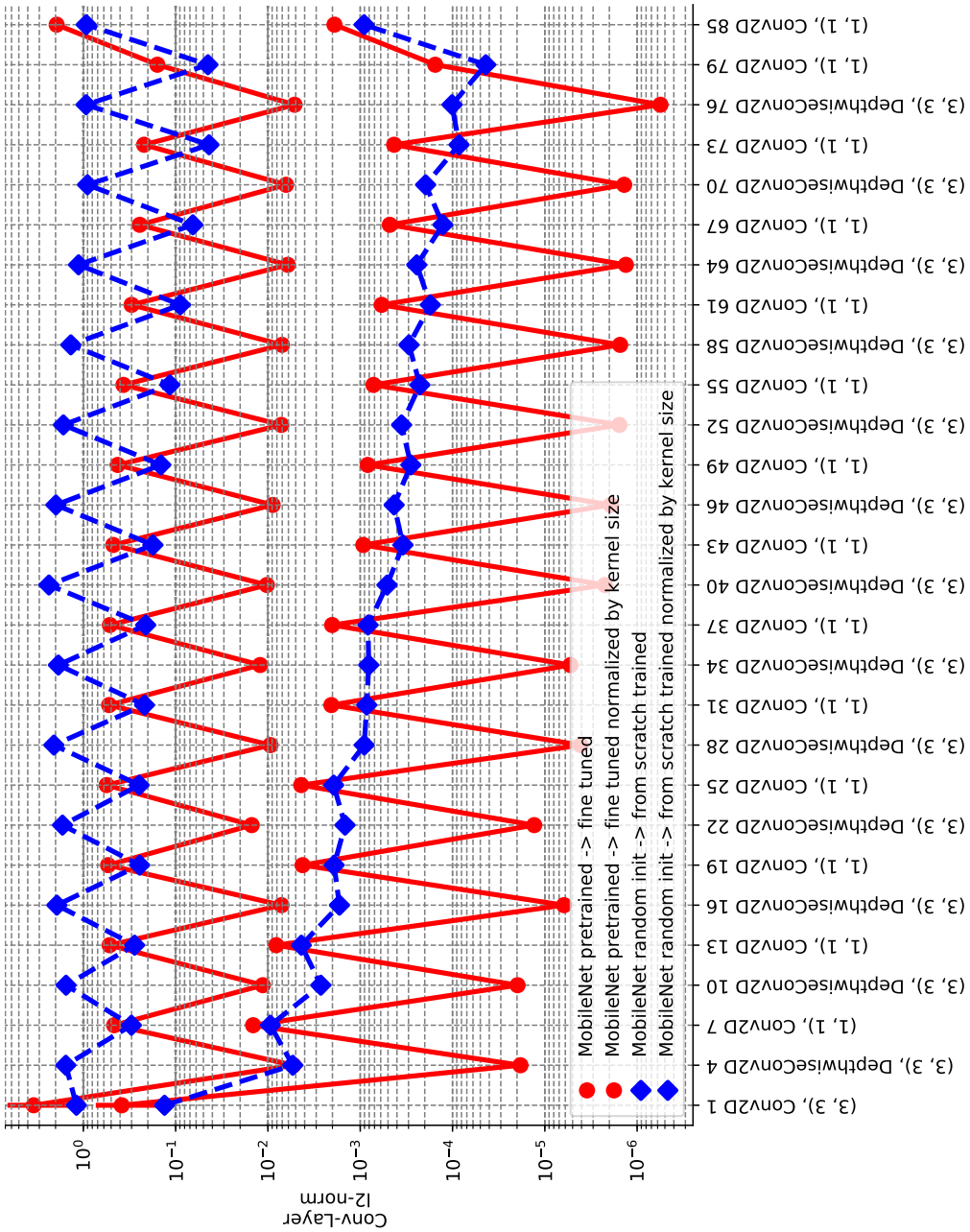


Abbildung A.25: Gewichtsänderung: MobileNet

Zu sehen ist die schichtweise Änderung der Gewichte, logarithmisch abgetragen. Die Berechnungsvorschrift findet sich in Abschnitt 5.3.2.

A.8 Architekturüberblick

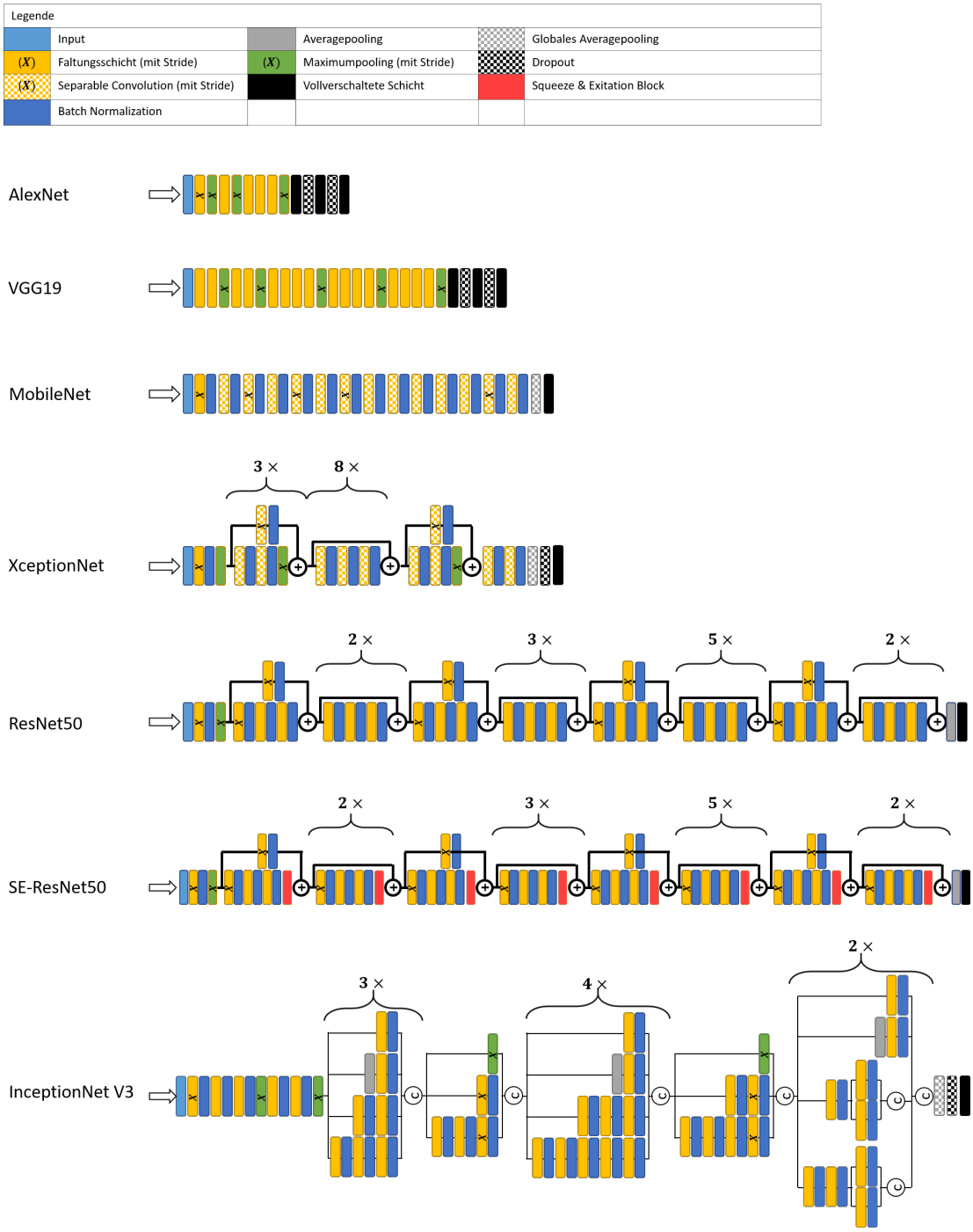
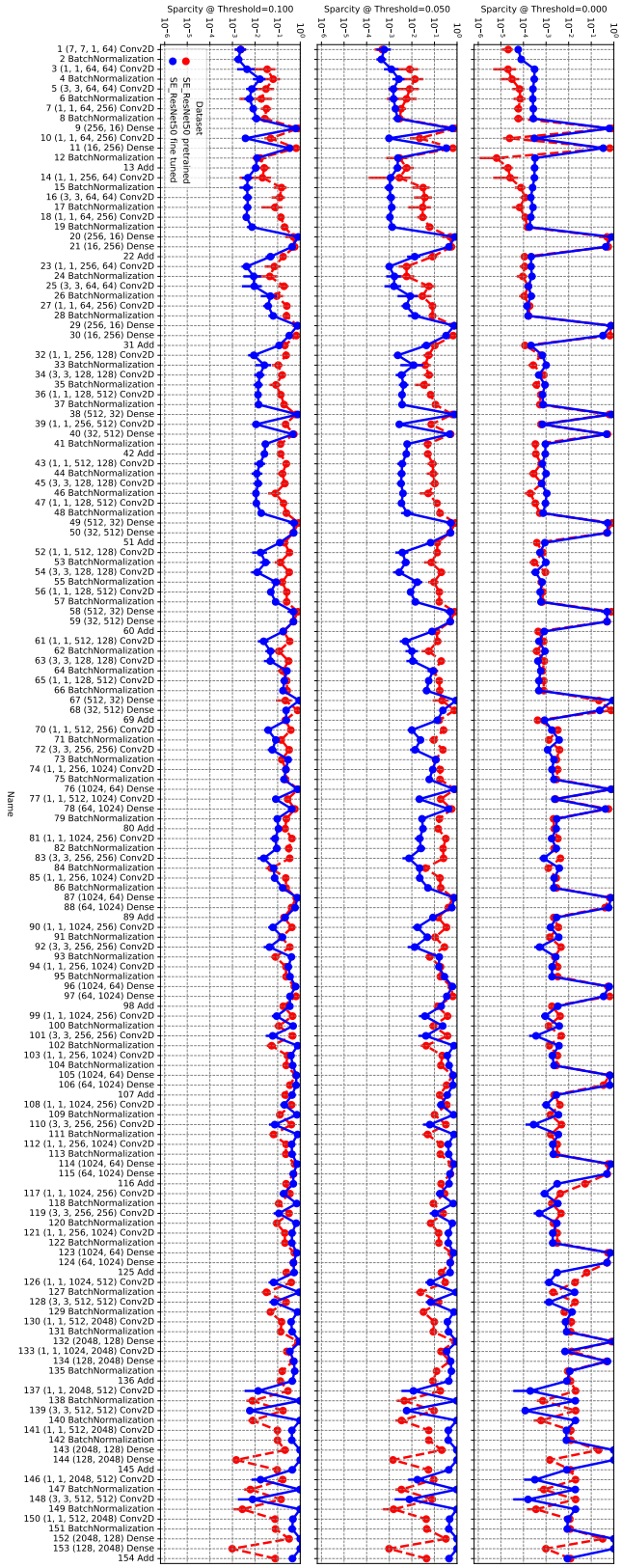


Abbildung A.26: Schematische Gegenüberstellung aller Architekturen

A.9 Sparsity-Diagramm



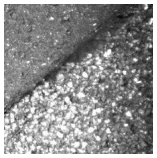


Abbildung A.27: Sparsity-Verlauf des SE-ResNet50

Zu sehen ist der Sparsity-Verlauf SE-ResNet50. Für die Berechnung wurde das abgebildete Bildbeispiel verwendet. Das SE-ResNet50 besitzt über weite Teile des Netzes eine sehr geringe Sparsity. Wie gut zu erkennen ist, steigt die Sparsity durch einen höheren Threshold erheblich.

A.10 Feature Attention

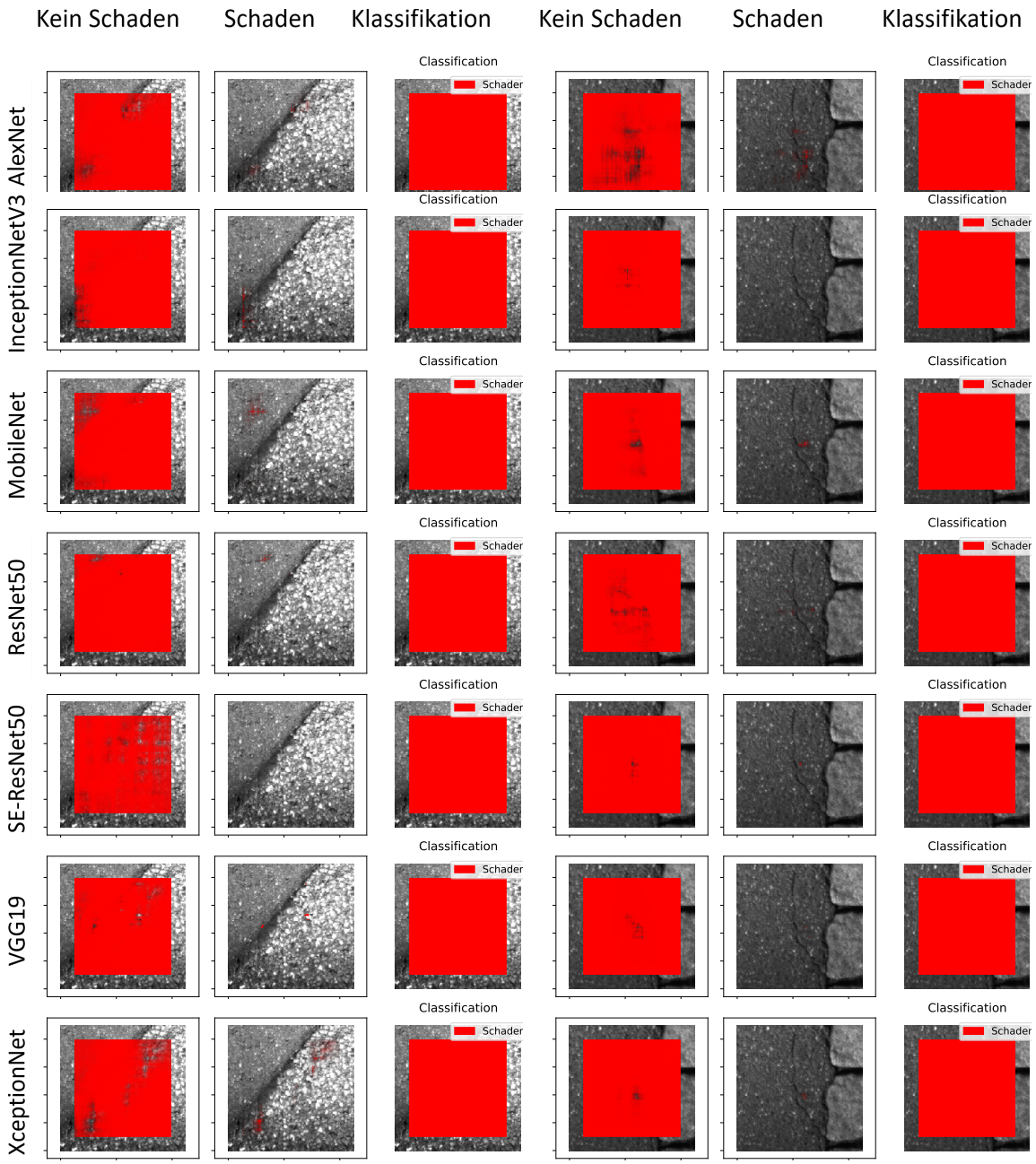


Abbildung A.28: Visualisierung der Feature-Attention zweier Positivbeispiele

Zu sehen ist die Feature-Attention der beiden Ausgabeneuronen (links und mittig) und die resultierende Klassifikation anhand des stärker aktivierten Neurons (rechts).

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Ahmed et al., 2015] Ahmed, E., Jones, M., and Marks, T. K. (2015). An improved deep learning architecture for person re-identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3908–3916.

[An et al., 2018] An, Y.-K., Jang, K.-Y., Kim, B., and Cho, S. (2018). Deep learning-based concrete crack detection using hybrid images. In *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems 2018*, volume 10598, page 1059812. International Society for Optics and Photonics.

[Cheplygina et al., 2018] Cheplygina, V., de Bruijne, M., and Pluim, J. P. (2018). Not-so-supervised: a survey of semi-supervised, multi-instance, and transfer learning in medical image analysis. *arXiv preprint arXiv:1804.06353*.

[Chollet, 2016] Chollet, F. (2016). Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. `https://github.com/fchollet/keras`.

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE.

[Dhall et al., 2016] Dhall, A., Goecke, R., Joshi, J., Hoey, J., and Gedeon, T. (2016). Emotiw 2016: Video and group-level emotion recognition challenges. In *Proceedings of the 18th ACM International Conference on Multimodal Interaction*, pages 427–432. ACM.

[Donahue et al., 2014] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2014). Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655.

[Eisenbach et al., 2017] Eisenbach, M., Stricker, R., Seichter, D., Amende, K., Debes, K., Sesselmann, M., Ebersbach, D., Stoeckert, U., and Gross, H.-M. (2017). How to get pavement distress detection ready for deep learning? a systematic approach. In *Int. Joint Conf. on Neural Networks (IJCNN)*, pages 2039–2047. IEEE.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Gopalakrishnan et al., 2018] Gopalakrishnan, K., Gholami, H., Vidyadharan, A., Choudhary, A., and Agrawal, A. (2018). Crack damage detection in unmanned

aerial vehicle images of civil infrastructure using pre-trained deep learning model. *International Journal for Traffic and Transport Engineering*, 8:1.

[Gopalakrishnan et al., 2017] Gopalakrishnan, K., Khaitan, S. K., Choudhary, A., and Agrawal, A. (2017). Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection. *Construction and Building Materials*, 157:322–330.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Heuritech, 2017] Heuritech (2017). convnets-keras. `https://github.com/heuritech/convnets-keras`.

[Hinton and Roweis, 2003] Hinton, G. E. and Roweis, S. T. (2003). Stochastic neighbor embedding. In *Advances in neural information processing systems*, pages 857–864.

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

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

[Hu et al., 2017] Hu, J., Shen, L., and Sun, G. (2017). Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[Kilgarriff, 2000] Kilgarriff, A. (2000). Wordnet: An electronic lexical database.

[Kim et al., 2018] Kim, B., Lee, Y., and Cho, S. (2018). Deep learning-based rapid inspection of concrete structures. In *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems 2018*, volume 10598, page 1059813. International Society for Optics and Photonics.

[Kim et al., 2017] Kim, S., Kim, W., Noh, Y.-K., and Park, F. C. (2017). Transfer learning for automated optical inspection. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 2517–2524. IEEE.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Längkvist et al., 2014] Längkvist, M., Karlsson, L., and Loutfi, A. (2014). A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24.

[Langner et al., 2010] Langner, O., Dotsch, R., Bijlstra, G., Wigboldus, D. H., Hawk, S. T., and Van Knippenberg, A. (2010). Presentation and validation of the radboud faces database. *Cognition and emotion*, 24(8):1377–1388.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

[Li et al., 2017a] Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., and Liu, H. (2017a). Feature selection: A data perspective. *ACM Computing Surveys (CSUR)*, 50(6):94.

[Li et al., 2017b] Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., and Liu, H. (2017b). scikit-feature. `https://github.com/jundongl/scikit-feature`.

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

[Mou et al., 2016] Mou, L., Meng, Z., Yan, R., Li, G., Xu, Y., Zhang, L., and Jin, Z. (2016). How transferable are neural networks in nlp applications? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 479–489.

[Ng, 2016] Ng, A. (2016). Nuts and bolts of building ai applications using deep learning. NIPS.

[Ng et al., 2015] Ng, H.-W., Nguyen, V. D., Vonikakis, V., and Winkler, S. (2015). Deep learning for emotion recognition on small datasets using transfer learning. In *Proceedings of the 2015 ACM on international conference on multimodal interaction*, pages 443–449. ACM.

[Pearson, 1901] Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Razavian et al., 2014] Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

[Shi et al., 2017] Shi, Z., Siva, P., and Xiang, T. (2017). Transfer learning by ranking for weakly supervised object annotation. *arXiv preprint arXiv:1705.00873*.

[Shin et al., 2016] Shin, H.-C., Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., and Summers, R. M. (2016). Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285–1298.

[Shwartz-Ziv and Tishby, 2017] Shwartz-Ziv, R. and Tishby, N. (2017). Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., et al. (2015). Going deeper with convolutions. Cvpr.

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826.

[Wattenberg et al., 2016] Wattenberg, M., Viegas, F., and Johnson, I. (2016). How to use t-sne effectively. *Distill*.

[Wu et al., 2016] Wu, Y. et al. (2016). Tensorpack. `https://github.com/tensorpack/`.

[Yosinski et al., 2014] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.

[Zeiler and Fergus, 2014] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.

[Zhang et al., 2018] Zhang, K., Cheng, H. D., and Zhang, B. (2018). Unified approach to pavement crack and sealed crack detection using preclassification based on transfer learning. *Journal of Computing in Civil Engineering*, 32(2):04018001.