Institut für Visualisierung & Interaktive Systeme, Abt. Visualisierung

Masterarbeit

Einfluss von Bildausschnittgröße und Hyperparametern auf Computer Vision Modelle

Dominik Matthias Lekar

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Daniel Weiskopf
Betreuer/in:	Dr. Sandeep Vidyapu Ruben Bauer M.Sc.
Beginn am:	1. November 2022
Beendet am:	11. Mai 2023

Kurzfassung

Lange Zeit galten faltende neuronale Netze (CNN) als Stand der Technik in vielen Aufgabengebieten der Computer Vision, wie beispielsweise der Bildklassifizierung. Dank technischem Fortschritt konnten sich jedoch in den letzten Jahren auch andere Computer Vision Modelle etablieren. Insbesondere Vision Transformer(ViT) und mehrschichtige Perzeptron(MLP) Modelle verarbeiten die Bilder in Bildausschnitten. Häufig wird in Arbeiten jedoch nur entweder eine Größe mit unterschiedlichen Hyperparametern, oder aber mehrere Ausschnittsgrößen mit denselben Parametern evaluiert. Auf diese Weise lassen sich keine Rückschlüsse ziehen, welche Wechselwirkungen zwischen unterschiedlichen Ausschnittsgrößen und anderen Hyperparametern bestehen.
Diese Arbeit beschäftigt sich daher mit der Untersuchung ebendieser Wechselwirkungen. Hierbei wird ein Framework implementiert, mit dem sich eine automatisierte Evaluierung durchführen lässt. In jedem Trainingsdurchlauf wird der Wert eines Hyperparameters gleichmäßig variiert und mit mehreren Ausschnittsgrößen evaluiert. Untersucht werden neben der Genauigkeit des Modells auch Laufzeit und Speicherbedarf des Trainingsprozesses. Die Ergebnisse werden anhand eines Beispieldatensatzes vorgestellt und anschließend für jeden der evaluierten Hyperparameter zu einem Wert zusammengefasst, der sich anschließend mit den Werten der restlichen Ausschnittsgrößen vergleichen lässt. Dabei kann gezeigt werden, dass sich Hyperparameter in manchen Situationen durchaus unterschiedlich auf andere Ausschnittsgrößen auswirken, es jedoch häufig schwierig ist allgemeine Schlüsse zu ziehen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Danksagung

Ich möchte mich an dieser Stelle bei Dr. Sandeep Vidyapu, Ruben Bauer M.Sc und Prof. Dr. Daniel Weiskopf vom Visualisierungsinstitut der Universität Stuttgart für die Betreuung und Prüfung dieser Arbeit bedanken.

1. Einleitung

Neuronale Netze gewannen in den letzten Jahren immer mehr an Bedeutung. Dank des technischen Fortschritts, konnte die Rechenleistung von Computern kontinuierlich gesteigert werden. Dieses ermöglicht das Trainieren von immer komplexer werdenden Modellen auf immer größeren Datensätzen. Seit der Einführung von AlexNet zählen faltende neuronale Netze (CNNs) lange Zeit als state-of-art Modelle, welche die besten Leistungen in vielen verschiedenen Aufgabengebieten liefern.
Transformer Modelle haben sich besonders in der maschinellen Sprachverarbeitung als wettbewerbsfähige Konkurrenz erwiesen. Seit der Einführung des Vision Transformers (ViT) konnten sich Diese auch im Bereich der Computer Vision fest etablieren. Neben dem Vision Transformer haben sich, dank kontinuierlich wachsender Datensätze, auch mehrschichtige Perzeptron(MLP) Modelle als eine zusätzliche konkurrenzfähige Alternative bewährt.

1.1. Motivation

Vision Transformer, MLP, aber auch einige CNNs arbeiten auf Bildausschnitten bei der Bildverarbeitung. Daher wird das Eingabebild zunächst in einzelne Ausschnitte unterteilt, bevor diese weiter verarbeitet werden. Mit der Einführung von ViT hat ist sich eine Standardgröße von 16x16 (alternativ 14x14) großen Pixeln für einzelnen Bildausschnitte etabliert, welche auch häufig von aufbauenden Arbeiten übernommen wird. Einige Autoren untersuchen bei ihren Modellen zudem meist ein bis zwei weitere Ausschnittsgrößen wie beispielsweise 32x32 Pixel, oder aber anderen Parametern, wie der Bildauflösung oder Modellgröße. Nur selten werden sowohl Parameter als auch Ausschnittsgröße variiert, sodass deren Wechselwirkung unklar bleibt.

1.2. Zielsetzung

Ziel dieser Arbeit ist es zu untersuchen, wie die Wechselwirkungen zwischen unterschiedlichen Ausschnittsgrößen und einzelnen Hyperparametern aussehen. Hierbei sollen fünf Modelle aus der Computer Vision Domäne ausgewählt und auf mehreren Datensätzen evaluiert werden. Der Fokus der Evaluierung liegt dabei Vergleiche zwischen unterschiedlichen Ausschnittsgrößen ziehen können. Hierbei soll der Wert eines einzelnen Hyperparameter in jedem Durchlauf geändert und anschließend untersucht werden, ob es Abweichung bei unterschiedlichen Ausschnittsgrößen gibt. Durch die Auswertung der Evaluierung soll veranschaulicht werden, ob bzw. bei welchen Hyperparametern dies der Fall ist. Untersucht werden sollen die Metriken Genauigkeit, Laufzeit und Speicheranforderungen des Trainingsdurchlaufs.
Für die Umsetzung der Zielsetzung soll ein Framework entwickelt werden, welches die Durchführung der Evaluierung ermöglicht. Da in jedem Durchlauf nur der Wert eines Hyperparameters geändert

wird, gibt es entsprechend viele Trainingsdurchläufe. Das Framework soll deshalb die Durchführung möglichst automatisiert realisieren können.
Zudem soll eine grafische Benutzeroberfläche (GUI) implementiert werden. Diese soll den Benutzer bei der Erstellung von Trainingsdurchläufen unterstützten und die Ergebnisse veranschaulichen.

1.3. Aufbau der Arbeit

In den Grundlagen werden zunächst alle relevanten Punkte erläutert. Dies beinhaltet neben der grundlegenden Funktionsweise der unterschiedlichen Modelltypen, alle weiteren Aspekte, die für das Verständnis dieser Arbeit von Relevanz sind.
Nach den Grundlagen werden verwandte Arbeiten vorgestellt, die für diese Arbeit relevant sind.
Daraufhin wird der untersuchte Ansatz beschrieben. Es werden die Anforderung beschrieben, welche für die Evaluierung gestellt wurden und die ausgewählten Modelle und Datensätze vorgestellt.
Daraufhin wird die Implementierung des entwickelten Frameworks und der dazugehörenden GUI vorgestellt.
Anschließend wird die Durchführung vorgestellt. Hierbei beinhaltet sind neben den Ergebnissen, auch der Prozess der Durchführung und dabei aufgetretene Probleme. Neben einer detaillierten Betrachtung anhand eines Datensatzes, werden die Verläufe je Parameter und Ausschnittsgröße zu einem Wert zusammengefasst, der anschließend verglichen wird
Zum Schluss werden die Erkenntnisse in einem Fazit zusammengefasst und einen Ausblick auf künftige Arbeiten geben.

2. Grundlagen

2.1. Computer Vision Domänen

Computer Vision ist ein Teilgebiet der künstlichen Intelligenz, bei dem Computermodelle selbstständig aus Daten wie Bildern oder Videos lernen, indem sie visuelle Merkmale erkennen und verarbeiten können. Zu den Anwendungsgebieten von Computer Vision zählen zahlreiche Domänen, die sich mit der Bild- und Videoverarbeitung auseinandersetzen. Unter anderem zählen dazu Bildklassifizierung, Objekterkennung und Bildsegmentierung welche in Abbildung 2.1 zu sehen sind. Bei der Bildklassifizierung muss das Modell das Eingabebild verarbeiten und die korrekte Klasse des Bildes bestimmen können. Bei der Objekterkennung muss neben der Klassifizierung, auch die Position des Objekts im Bild bestimmt werden, hierbei wird eine Box bestimmt, die Größe und Position des Objektes angibt. Im Gegensatz dazu wird bei der Segmentierung das Objekt im Bild pixelgenau segmentiert.

Abbildung 2.1.: Visualisierung von unterschiedlichen Anwendungsgebieten.
Angelehnt an Quelle: [Sta]

2.1.1. Lernverfahren

Neben der Anwendungsdomäne, muss auch beachtet werden, welche Gegebenheiten vorhanden sind. Abhängig von der Situation existieren unterschiedliche Ansätze von Lernverfahren.

Überwachtes Lernen Überwachtes Lernen ist ein Teilgebiet des maschinellen Lernens. Hierbei werden annotierte Datensätze verwendet, um Modelle darauf zu trainieren. Durch die vorhandenen Annotationen lässt sich der Trainingsprozess des Modells überwachen. Während des Trainings wird die Eingabe an das Modell übergeben und dieses passt seine Gewichte während des Trainings an, basierend darauf, wie die Ausgabe des Modells mit der Annotation der Eingabe (englisch auch ground truth genannt) übereinstimmt. Die Bildklassifizierung zählt hierbei auch zum überwachten Lernen. Ein Beispiel für diesen Vorgang ist die Eingabe eines Bildes, dessen Ausgabe anschließend

verglichen wird. Wenn das Modell das Bild korrekt mit der entsprechenden Annotation klassifiziert, wird die Ausgabe als korrekt betrachtet. Andernfalls, als inkorrekt.
Neben der Klassifizierung existieren noch andere Anwendungen, bei denen überwachtes Lernen zum Einsatz kommt. Ein weiteres Beispiel ist die Regression, bei der die Beziehung zwischen abhängigen und unabhängigen Variablen untersucht wird, um etwa Prognosen über die Verkaufszahlen eines Unternehmens machen zu können [IBM23].

Unbewachtes Lernen Neben dem überwachtem Lernen gibt es das unbewachte Lernen. Hierbei werden Verfahren des maschinellen Lernens angewendet, um Informationen aus einem Datensatz zugewinnen, der keine Annotationen, Zielwerte oder ähnliches enthält. Anstatt dem Modell zu sagen, was falsch oder richtig ist, soll es Muster von alleine erkennen können. Eine mögliche Aufgabe ist die Clusteranalyse
. Beim überwachten Lernen werden Modelle auf sehr großen Datensätzen trainiert, um ihre Leistung zu optimieren. Die Erstellung solcher Daten erfordert jedoch einen großen Arbeitsaufwand. Dieser ist beim unbewachten Lernen nicht nötig. Zudem hat es den Vorteil, dass beim unbewachten Lernen, die Modelle meist besser auf zukünftige, ungesehene Eingaben reagieren. [Pat19]

Transfer-Lernen Herkömmliche Modelle müssen von Anfang an trainiert werden, was ein ressourcenintensiver Prozess ist. Um wettbewerbsfähige Leistung zu erzielen, bedarf es einer großen Menge an Daten. Im Gegensatz dazu ist das Transfer-Lernen eine effizientere Methode, welche auch bei kleinen Datensätzen zu guten Ergebnissen führen kann. Beim Transfer-Lernen werden bereits vortrainierte Modelle verwendet und an eine neue Problemstellung angepasst. Je nach Bereich, Aufgabe und verfügbaren Daten, können unterschiedliche Methoden und Wege verwendet werden [Bah23]. Bevor man eine Strategie auswählt, sollte man sich fragen,

- Welcher Teil des Wissens kann übertragen werden, um die Ergebnisse der Zielaufgabe zu verbessern?
- Wann sollte man Wissen übertragen und wann nicht? Eine falsche Wahl könnte die Ergebnisse der Zielaufgabe sogar verschlechtern.
- Wie überträgt man das Wissen aus dem vortrainierten Model, auf Grundlage der aktuellen Problemstellung? [Bah23]

Insbesondere im Kontext dieser Arbeit muss sich auch gefragt werden, ob Transfer-Lernen auch mit unterschiedlich großen Bildausschnitten angewendet werden kann.

2.2. Neuronale Netze

Ursprung Bereits 1943 haben die Forscher Warren McCullock und Walter Pitts das McCulloch-Pitts (MCP) Neuron vorgestellt, welches ihr erstes Konzept einer vereinfachten Gehirnzelle darstellt. Sie schlugen vor, dass neuronale Netze als Modell für das menschliche Gehirn dienen können und eine Vernetzung mehrerer solcher Zellen es ermöglicht, komplexere Aufgaben zu lösen. [MP43] Aufbauend auf der Arbeit von McCullock und Pitts hat Frank Rosenblatt 1958 das Perzeptron vorgestellt, welches als erstes neuronales Netz betrachtet wird [Ros58].

2.2.1. Perzeptron

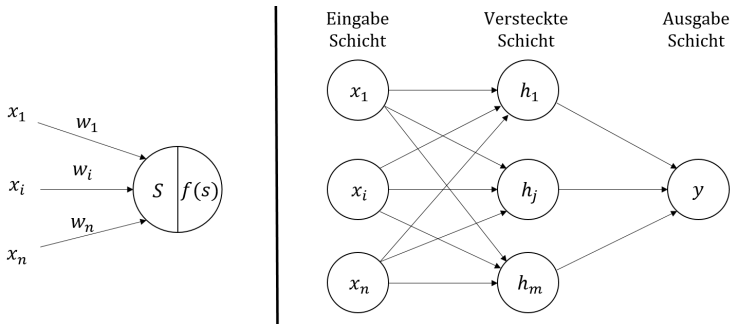


Abbildung 2.2.: Darstellung eines einfach Perzeptron (links) und einem Aufbau eines mehrschichtigen Perzeptron Modells (rechts).

Abbildung 2.2 zeigt den Aufbau eines einfachen Perzeptron. Es besteht aus einer Eingabeschicht mit den Werten x_1 bis x_n und hat einen Ausgabeknoten. Jeder Eingangswert x_i wird mit dem entsprechenden Gewicht w_i multipliziert und deren Produkt mit allen anderen aufsummiert. Erreicht die Summe einen gewissen Schwellenwert, welcher durch eine Aktivierungsfunktion f modelliert werden kann, so wird das Perzeptron aktiviert. Die Ausgabe ergibt sich entsprechend:

$$Ausgabe = \begin{cases} 1 & \text{falls } \sum_i w_i x_i > f \\ 0 & \text{sonst} \end{cases}$$

Mehrschichtiges Perzeptron (MLP) Ein mehrschichtiges Perzeptron (MLP) hat entsprechend mehrere Knoten in einer Schicht. Die einzelnen Knoten werden häufig auch als Neuronen bezeichnet. Ein MLP Modell besteht aus mindestens drei Schichten: der Eingabeschicht, der versteckten Schicht und der Ausgabeschicht. Sowohl die Zahl der Neuronen in der verdeckten Schicht, also auch die Anzahl der verdeckten Schichten, kann variieren. Hierbei sind alle Neuronen mit der vorherigen Schicht vernetzt, oder auch vollständig verbunden. Der Graph des MLP Models ist somit azyklisch und voll vernetzt in Richtung Ausgabeschicht.

2.2.2. Aktivierungsfunktionen

Ohne Aktivierungsfunktionen ist das Modell aus Abbildung 2.2 eine lineare Kombination der Eingabewerte und Gewichte jedes Knoten. Dies schränkt das Lernvermögen des Modells ein, da sich komplexe Probleme schlecht durch lineare Gleichungen modellieren lassen [**Brodtman2021**]. Aufgrund dessen wurden nicht linear Aktivierungsfunktionen eingeführt, um dieser Eigenschaft entgegenzuwirken. Bei der Bilderverbreitung werden unter anderem verwendet:

ReLU Kurz Recitified Linear Unit, ist eine lineare Funktion, die erst ab einem bestimmten Schwellenwert angewendet wird und somit als Ganzes nichtlinear. Ist x negativ wird 0 zurückgegeben, ansonsten der Wert von x.

$$f(x) = \max(0, x$$

GELU Die Gaussian Error Linear Unit (GELU) ist eine weitere nichtlineare Aktivierungsfunktion, welche die Eingabewerte nach einer gaußschen Verteilung gewichtet und wird definiert als:

$$GELU(x) = \frac{1}{2}x\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$$

Wobei erf die (Gaußsche) Fehlerfunktion ist [HG16].

Softmax Die Softmax Funktion nimmt einen Vektor mit k realen Werten und normalisiert ihn in einen Vektor mit k realen Werten an, bei denen die Summe der einzelnen Werte des Vektors 1 ergibt. $\overrightarrow{z}$ entspricht dem Eingabevektor und k der Anzahl der Werte [Woo].

$$\sigma(\overrightarrow{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

2.2.3. Schichten von neuronalen Netzen

Faltende-Schicht Faltende Schichten werden eingesetzt, um mithilfe eines Filters Merkmale aus Bildern zu extrahieren. Beispielsweise kann ein Filter verwendet werden, um Kanten oder Farbverläufe im Bild zu erkennen. Hierbei wird ein Filter mit einer festen Größe auf die Eingabe angewendet und wie ein Fenster über die Eingabe geschoben. In Abbildung 2.4 wird ein 3x3 großer Filter auf der Eingabe angewendet. Jeder Wert in der Eingabe wird mit dem Wert des Kernel an entsprechender Position multipliziert und alle Produkte anschließend aufaddiert. Anschließend verschiebt sich der Filter um ein Feld nach rechts. Alternativ kann die Schrittbreite (englisch stride) auch angegeben werden. Bei einem Wert von 2 würde der Filter hierbei ein Feld überspringen und direkt 2 Felder nach rechts wandern.

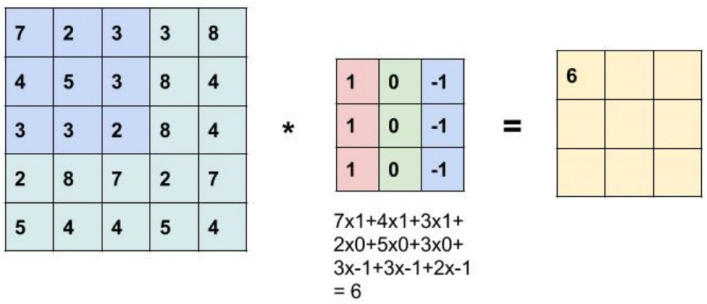


Abbildung 2.3.: Beispiel eines convolutionellen Filter mit der Größe 3x3, Quelle: [Voh19]

Pooling-Schichten Pooling Schichten werden häufig nach Convolutionellen Schicht eingesetzt. Mithilfe von pooling lassen sich extrahierten Merkmale aus der vorherigen Schicht reduzieren, indem diese zusammengefasst oder aggregiert werden. Auf diese Weise lassen sich die Anzahl der Parameter im Netzwerk minimieren [Qay22].
Die Pooling Schicht geht ähnlich wie die faltende Schicht mit einem Fenster über die Eingabe. Hierbei wird jedoch kein Filter angewandt, sondern nur die Eingabe betrachtet.
Beim Max Pooling wird der maximale Wert in dem Fenster bestimmt. In dem obigen Beispiel aus Abbildung 2.4 würde dies der Wert 7 sein.
Mithilfe des Durchschnitts-Pooling wird der Mittelwert bestimmt, indem alle Werte innerhalb des Fensters addiert und anschließend durch die Anzahl der Werte geteilt wird.

Normalisierungs-Schicht Diese Schicht normalisiert die Eingangswerte. Ba et al. haben gezeigt, dass Normalisierung der eingehenden Werte zu besser Laufzeit und Generalisierung führen kann [BKH16].

Dense-Schicht Eine Dense-Schicht, oft auch voll-vernetzte Schicht genannt, ist ähnlich wie in Abbildung 2.2 eine Schicht, bei der die Knoten mit allen Knoten aus der vorherigen Schicht vernetzt sind.

2.2.4. Vision Transformer

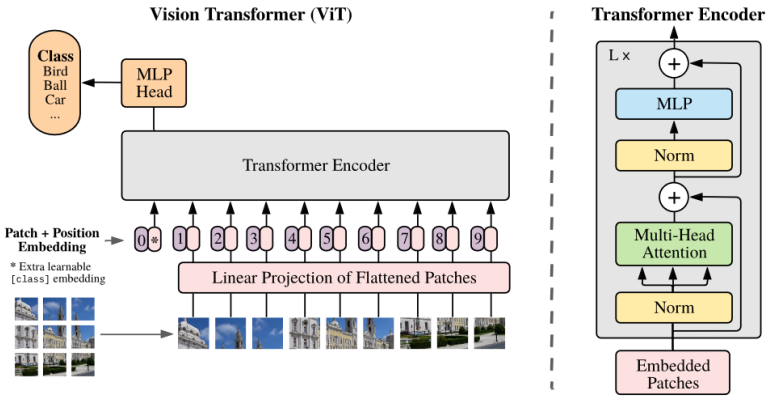


Abbildung 2.4.: Modellübersicht des Vision Transformers (ViT), Quelle: [DBK+20]

Dosovitskiy et al. haben mit ihrem Vision Transformer (ViT) die Grundlage gelegt, bei denen ein Transformer-Modell zum ersten Mal Anwendung in der Computer Vision Domäne fand. Das Modell basiert auf der aufmerksamkeitsbasierten (Attention) Transformer Architektur aus der natürlichen Sprachverarbeitung. Hierbei werden die Bildausschnitte zunächst mithilfe von linearer Projektion auf einen eindimensionalen Vektor abgebildet, die zudem eine positionelle Codierung bekommen, um sie anschließend dem Transformer Netzwerk übergeben zu können [DBK+20]. Der Selbst-Aufmerksamkeitsmechanismus ermöglicht es weitreichende Abhängigkeiten und kontextuelle Informationen in den Eingabedaten zu erfassen, indem es eine gewichtete Summe der Eingabedaten bestimmt, bei denen die Gewichte basierend auf der Ähnlichkeit der Eingaben

berechnet werden [Boe23]. Häufig ist es jedoch der Fall, dass mehrere Aspekte der Eingabedaten betrachtet werden müssen, um akkuratere Schlussfolgerungen ziehen zu können. Hierbei werden mehrere Köpfe (Multi-Head) eingesetzt, welche simultan Selbst-Aufmerksamkeit betreiben und am Ende zu einer Ausgabe vereinigt werden [Lip22].

2.2.5. Hyperparameter

Hyperparameter sind Parameter, die vor dem Trainingsprozess bestimmt werden und die Rahmenbedingungen des Trainingsprozesses definieren und Einfluss auf diesen haben. Neben Lernrate und Gewichtszerfall gibt es ein je nach Implementierung eine Vielzahl weiterer Hyperparameter. Im folgenden Abschnitt werden die für diese Arbeit relevanten vorgestellt.

Bildausschnittsgröße Der Wert bestimmt die Größe des Bildausschnitts, in die das Eingabebild aufgeteilt wird, und damit auch die Anzahl der Bildausschnitte. Diese sind immer quadratisch, ein Wert von 16 entspricht daher einem Ausschnitt von 16x16 Pixeln. Hierbei ist zu beachten, dass dieser Wert nur die Größe des Ausschnitts bestimmt, nicht die Anzahl. Es existieren auch Modelle, die mit überlappenden Ausschnitten arbeiten.

Stapelgröße Große Datensätze mit Millionen von Bildern können aufgrund von Ressourcenlimits nicht auf einmal trainiert werden. Stattdessen wird der Trainingsdatensatz in gleich große Gruppen unterteilt, auf denen das Modell trainiert und anschließend seine Gewichte aktualisiert. Der Wert der Stapelgröße bestimmt wie groß die einzelnen Gruppen sind.

Epochen Die Anzahl der Epochen legt fest, wie häufig über den Datensatz iteriert wird. Eine Epoche entspricht einem Durchgang über den gesamten Datensatz, während die Stapelgröße definiert, wie häufig die Gewichte pro Epoche aktualisiert werden.

Bildauflösung Der Wert der Bildauflösung definiert, auf welche Größe das Bild während der Vorverarbeitung skaliert wird. Ein Wert von 224 bedeutet eine Skalierung auf 224x224 Pixel. Unterschiedliche Werte der Bildauflösung bedeuten entsprechend, dass ein unterschiedlicher Grad an Detail erhalten bleibt und dem Modell übergeben wird, aber auch die Anzahl der Pixel, die verarbeitet werden müssen.

Modelltiefe Die Modelltiefe legt fest, wie tief ein Model ist. Dabei kann es sich um die Anzahl einzelner versteckter Schichten, oder um die Anzahl von komplexen Blöcken, die mehrere Schichten beinhalten, handeln. Je nach Model kann der Wert eine positive Zahl, oder auch eine Liste von Zahlen sein, bei dem die Länge der Liste die Anzahl der Stufen der Architektur und die Werte die Anzahl an Elementen in der entsprechenden Stufe definiert.

Versteckte Dimension Der Wert der versteckten Dimension legt die Anzahl der Neuronen fest, die sich in der entsprechenden versteckten Schicht befinden.

DropOut und DropPath Dieser Wert legt fest, mit welcher Wahrscheinlichkeit einzelne Neuronen, beziehungsweise Pfade deaktiviert werden.

Lernrate Die Lernrate ist ein wichtiger Parameter, der festlegt, wie stark die Gewichte auf Basis des Verlustgradienten pro Iteration angepasst werden.

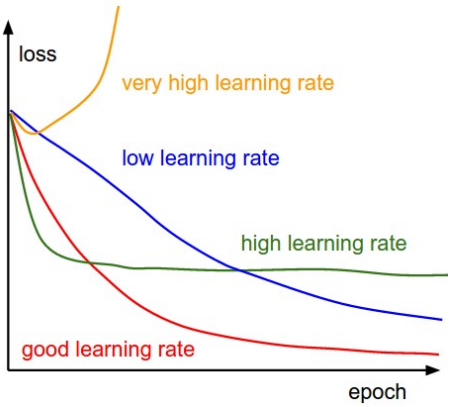


Abbildung 2.5.: Darstellung von Verlustverläufen mit unterschiedlichen Lernraten. Quelle: [Zul18]

Abbildung 2.5 veranschaulicht die Verläufe von Verlustwerten mit unterschiedlichen Lernraten. Ein geringerer Wert führt entsprechend zu einer langsameren Anpassung der Gewichte. Entsprechend langsam konvergiert der Verlust gegen ein Minimum. Eine zu hohe Lernrate kann jedoch dazu führen, dass das gesuchte Minimum nicht erreicht wird [Zul18].

2.2.6. Optimierung des Netzwerks

Damit ein neuronales Netz seine Lernfähigkeit steigern kann, muss es auf den Trainingsdaten lernen und seine Gewichte ständig anpassen.

Verlustfunktion Zunächst muss eine Verlustfunktion definiert werden, mit der sich die Differenz zwischen den Wahrheitswerten und von dem Modell ausgegebenen Werten berechnen lässt. Eine Funktion, die häufig bei der Bildklassifizierung verwendet wird, ist die Kreuzentropie. Diese wird definiert als:

$$L_{CE} = -\sum_{i=1}^{k} W(x_i) \log A(x_i)$$

Hierbei entspricht k der Anzahl der Klassen, x_i der Werte der i-ten Klasse. $W(x_i)$ ist der Wahrheitswert und $A(x_i)$ die Ausgabe des Modells für die entsprechende Klasse ist.

Lernraten-Planer Die gewählte Lernrate muss nicht zwingend einen festen Wert behalten. So kann diese auch während des Trainingsprozess verändert werden. Hierfür existieren verschiedene Planer, die unterschiedliche Ansätze ermöglichen. So existieren unter anderem stufenförmige, exponentielle oder zyklische Planungsmethoden. Alternativ können die Werte der Lernrate je Epoche auch vom Benutzer selber definiert werden.

Rückpropagierung Nachdem der Stapel von Bildern durch das neuronale Netz propagiert ist, lässt sich mithilfe der Verlustfunktion der aktuelle Verlust des Netzes bestimmen. Um den Verlust reduzieren zu können, muss der Gradient des Verlustes berechnet werden. Dies geschickt mittels Rückpropagierung. Hierbei wird rückwärts, daher von der Ausgabeschicht in Richtung Eingabeschicht rückpropagiert. Der Gradient wird bestimmt, indem für jede Verbindung in jeder Schicht der Beitrag zum Fehler gemessen wird [Ger19].

Optimierung Gradient Descent ist ein Optimierungsalgorithmus, dessen Ziel es ist, das Minimum der Verlustfunktion zu finden, indem es mithilfe des berechneten Gradienten die Gewichte des neuronalen Netzes stufenweise anpasst, um sich in Richtung Minimum zu bewegen. Ein Problem dieses Algorithmus ist jedoch, dass es sehr kostspielig ist den Gradient auf der gesamten Eingabe zu berechnen. Im Gegensatz dazu verwendet der stochastische Gradient Descent, kurz SGD nur eine zufällige Auswahl aus den Eingabedaten, um den Gradienten zu berechnen [Ger19].
Adam (kurz für *adaptive moment estimation*) ist ein weiterer Optimierungsalgorithmus, der den exponentiell zerfallenden Durchschnitt aus vorherigen Gradienten berücksichtigt [Ger19]. AdamW stellt eine Modifikation von Adam darf, welche den Gewichtszerfall von der Aktualisierung des Gradienten entkoppelt [LH17].

Hyperparameter Optimierung

Neben der Optimierung der Gewichte durch Optimierungsalgorithmen ist auch Optimierung von Hyperparametern ein wichtiger Bestandteil von neuronalen Netzen, um gute eine Lernfähigkeit zu ermöglichen. Hierbei existieren eine Vielzahl von Ansätzen. Neben Optimierungsalgorithmen, die durch ein eigenständiges Lernverfahren die Suche nach Parametern optimieren, wie etwa der bayes'schen Optimierung gibt es simple Ansätze, die abhängig vom zeitlichen Budget zur Anwendung kommen. Darunter zählen unter anderem:

Gittersuche Bei der Gittersuche wird eine Liste an Werten für jeden Hyperparameter erstellt, der bei der Optimierung berücksichtigt werden soll. Anschließend werden systematisch alle möglichen Kombinationen von Parameterwerten trainiert und die beste Kombination im Rahmen des zeitlichen Budgets übernommen.

Zufallssuche Ähnlich wie bei der Gittersuche werden für jeden Hyperparameter der Wertebereich vordefiniert. In diesem Fall werden nur die Grenzen der Werte angegeben, der Algorithmus wählt daraufhin zufällige Werte aus, die innerhalb des definierten Suchraums liegen.

Testplaner Gittersuche hat den Nachteil, dass der Suchraum schnell steigt mit der Anzahl der Hyperparameter und den jeweiligen Werten. Bei großem Suchraum, können viele Wertekombination zu suboptimalen Ergebnissen führen, die dennoch einen gesamten Trainingsdurchlauf erfordern. Testplaner versuchen diesem Problem entgegenzuwirken, indem sie testweise Wertekombinationen durchprobieren. Hierbei werden alle Wertekombinationen nur für eine gewisse Anzahl an Epochen trainiert. Nach jeder Testphase werden die Durchläufe verworfen, die am schlechtesten abschneiden. Die Anzahl der Epochen und verworfenen Durchläufe in jeder Testphase lassen sich vorher festlegen. Ziel ist es, den gleichen Suchraum in weniger Zeit abdecken zu können.

Datenvorverarbeitung

Die Vorverarbeitung ist ein wichtiger Schritt bei neuronalen Netzen. Mithilfe von Datenaugmentierung lassen sich Genauigkeit, als auch die Fähigkeit zu generalisieren, steigern [CZSL20]. Steiner et al. haben gezeigt, dass sich mit gut gewählter Augmentierung die Genauigkeit des Modells auf ein Niveau steigern lässt, wie das Trainieren auf einem 10 Mal so großen Datensatz [SKZ+22]
. Hierbei kommt eine Kombination aus Mixup [ZCDL17] und RandAugment [CZSL20] zum Einsatz.
Neben einfachen Augmentierungstechniken wie das Rotieren oder Spiegeln von Bildern wurden auch stärkere Augmentierungen eingeführt.

RandAugment RandAugment führt eine Reihe von zufällig ausgewählten Operationen durch. Hierfür nimmt es zwei Parameter als Eingabe, mit denen sich die Anzahl der Operationen und die Intensität der jeweiligen Operationen festlegen lässt.

Mixup Mixup nimmt zwei Bilder und überlagert diese in ein neues Bild. Mithilfe des Alpha Parameters lässt sich festlegen ob, beziehungsweise welches der beiden Bilder eher im Vordergrund erscheint.

Generalisierung

Generalisierung ist eine wichtige Eigenschaft, die ein Modell haben soll. Ein Modell, welches gut generalisiert, kann auch gute Ausgaben bei Daten produzieren, die es bisher nicht gesehen hat. Um die Generalisierungsfähigkeit einzuschätzen, werden Datensätze in zwei Teile aufgeteilt. Mit dem Trainingssatz wird das Modell, wie in den vorherigen Abschnitten erläutert, trainiert. Anschließend wird das Modell in jeder Epoche mit dem Validierungssatz evaluiert. Hierbei wird auch ein Validierungsverlust berechnet. Es werden jedoch keine Gewichte aktualisiert, um zu vermeiden, dass sich das Modell Merkmale von den Validierungsdaten antrainiert. Entsprechend wird auch die Genauigkeit auf dem Validierungssatz als Evaluierungsgrundlage verwendet.
Dabei kann es zu unerwünschten Verhalten des Modells vorkommen. Unter Überanpassung versteht man eine zu starke Anpassung an den Trainingsdatensatz, indem es genaue Vorhersagen auf dem Trainigssatz erzeugt, jedoch nicht auf dem Validierungssatz, oder sonstigen ungesehenen Daten. Dieser Effekt kann unter anderem begünstigt werden durch zu kleinen Trainingsätzen, oder zu hoher Modellkomplexität [Ser].

Regularizierung

Es existieren verschiedene Ansätze, um dem Effekt der Überanpassung eines Modells entgegenzuwirken. Neben den populären Ansätzen wie Gewichtszerfall und L_2 Regularisierung, wird auch Dropout verwendet. Hierbei werden einzelne Neuronen zu einer gewissen Wahrscheinlichkeit deaktiviert [SHK+14]. Der Wahrscheinlichkeitswert zählt auch zu den Hyperparametern, die zu Beginn vordefiniert werden. Eine Abwandlung davon ist DropPath, bei der nicht einzelne Neuronen, sondern Pfade deaktiviert werden [LMS17].

2.2.7. Klassifizierungs-Metriken

Um die Leistung eines Models bei der Bildklassifizierung bewerten zu können, benötigt es einer Evaluierungsmetrik. Hierbei stehen mehrere Optionen zur Wahl, die abhängig von dem Datensatz und den Rahmenbedingungen der Analyse unterschiedlich nützlich sind.

Genauigkeit Die Metrik zur Bewertung der Modelle umfasst die Erfassung des Anteils der korrekt klassifizierten Bilder im Verhältnis zur Gesamtzahl der validierten Bilder. Diese Methode gilt als einfaches und intuitives Mittel zur Bewertung der Leistung von Modellen und wird häufig bei der Bildklassifizierung verwendet. Die Metrik wird definiert als:

$$\text{Genauigkeit} = \frac{\text{Anzahl korrekt klassifizierter Bilder}}{\text{Gesamtanzahl an Bilder}}$$

Top-K Genauigkeit Neben der Genauigkeit gibt es noch die Top-k Genauigkeit. Ähnlich wie bei der Genauigkeit wird das Verhältnis zwischen korrekt klassifizierten Bildern zu der Gesamtzahl gemessen, mit dem Unterschied das die korrekte Ausgabe des Models keine einzelne Klasse, sondern die besten k Einschätzungen ausgibt. Die einfache Genauigkeit kann daher auch als Top-1 Genauigkeit interpretiert werden.

Präzision und Recall Präzision misst das Verhältnis der wahren Positiven (korrekt identifizierte positive Fälle), zu allen positiven Vorhersagen des Modells. Sie wird definiert als:

$$\text{Präzision} = \frac{\text{Anzahl der wahren Positiven}}{\text{Anazhl der wahren Positiven + Anzahl der falsch Positiven}}$$

Recall misst das Verhältnis der wahren Positiven zu allen tatsächlich positiven Vorhersagen. Beide Metriken finden Anwendung, wenn Kosten von falschen Vorhersagen besonders hoch sind, wie beispielsweise in der medizinischen Diagnostik [HST+22]. Recall wrid definiert als:

$$\text{Recall} = \frac{\text{Anzahl der wahren Positiven}}{\text{Anzahl der wahren Positiven + Anzahl der falsch Negativen}}$$

F1 Wert Der F1 Wert ist eine Metrik, welche Präzision und Recall in eine Metrik vereint. Sie wird definiert als:

$$\text{F1 score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

2.3. Technische Grundlagen

Bibliotheken für maschinelles Lernen Zwei beliebte Bibliotheken, welche für diese Arbeit in Betracht gezogen wurden, sind Tensorflow und PyTorch. TensorFlow ist eine quelloffene Bibliothek für maschinelles Lernen, die von Google entwickelt wurde und mithilfe der High-Level API Keras das Erstellen und Trainieren von Modellen ermöglicht. Neben Python werden auch andere Programmiersprachen unterstützt wie C++, JavaScript oder MATLAB [**Tensorflowapi**]. Zudem bietet TensorFlow eine direkte Integration in das Visualisierungswerkzeug TensorBoard, mit dem sich unter anderem Trainingsergebnisse, Modellgraphen oder Histogrammen von Gewichten, visuell darstellen lassen [**Tensorboard**].
PyTorch hingegen ist eine quelloffene Bibliothek, die mit Python auf Torch aufbaut und von Meta AI entwickelt wurde.

Slurm Slurm ist ein quelloffener Auftrags-Manager, der für die Zuweisung von Rechenressourcen, die Verwaltung von Aufträgen und die Überwachung der Ressourcennutzung auf Linux-Clustern zuständig ist. Slurm ermöglicht eine effiziente und effektive Verwaltung von Arbeitsabläufen in großem Maßstab und ermöglicht eine effiziente Ressourcennutzung für einen optimalen Auftragsdurchsatz [**Slurm**].

3. Verwandte Arbeiten

Steiner et al. untersuchen die Leistung von Vision-Transformern (ViT) im Vergleich zu CNNs. Dabei werden die Wechselwirkung zwischen Trainingsdatengröße, Modelgröße, Berechnungsbudget, Regularisierung, und Datenaugmentierung analysiert. Wenn ViT-Modelle auf kleineren Datensätzen trainiert werden, erfordern sie eine starke Regularisierung des Modells oder Datanaugmentierung. Hierbei schlagen sie mehrere Kombinationen von Parametern für Datenaugmentierung mit RandAugment und Mixup vor. Dabei konnte gezeigt werden, dass sorgfältig ausgewählte Regularisierung und Datenaugmentierung die Genauigkeit des Models auf ein Niveau gesteigert werden konnte, wie dem Trainieren auf einem Datensatz mit 10-facher Größe. Diese Aussage gilt jedoch nicht für beliebig große Datensätze. So berichten die Autoren, dass es auf dem Oxford Pets Datensatz, unabhängig von der investieren Trainingszeit, nicht möglich schien, für ViT eine Genauigkeit zu erreichen, die auch nur annähernd an die von vortrainierten Modellen mittels Transfer-Lernen heranreicht. Nebenbei wurden auch zwei unterschiedliche Ausschnittsgrößen untersucht mit Variationen der Modellgröße. Als Fazit wird empfohlen ein Vergrößern der Ausschnittsgröße, dem Verkleinern des Models vorzuziehen. [SKZ+21]
Lv et al. verfolgen einen ganz anderen Ansatz und verwenden keinerlei Techniken für eine starke Datenaugmentierung wie RandAugment oder Mixup und verzichten ganz auf Regularisierung. Dennoch konnte ihr Model gute Ergebnisse auf kleinen Datansätzen wie CIFAR10 oder Oxford Pets liefern und im direkten Vergleich zu anderen MLP, CNN und Transformer Modellen diese bei der Genauigkeit überbieten. [LBW22]
Beyer et al. untersuchen die Flexibilität von ViT. Dabei konnten die Autoren aufzeigen, dass herkömmliche ViTs nicht flexibel sind in Bezug auf die Ausschnittsgröße. Für den Trainingsprozess wird eine feste Ausschnittsgröße benötigt, die zu Beginn definiert werden muss. Weiterhin wird untersucht, ob vortrainierte ViTs, mit anderen Ausschnittsgrößen evaluiert werden können. Hierbei werden die Einbettungsgewichte der Ausschnitte und die Positionseinbettung mithilfe von bilinearer Interpolation auf unterschiedliche Ausschnittsgrößen skaliert. Untersucht wurde ViT-B mit den Ausschnittsgrößen 16 und 30. In beiden Fällen konnten die Autoren beobachten, dass ein Ändern der Ausschnittsgröße des vortrainierten ViTs zu schlechteren Genauigkeiten führt. Für beide Ausschnittsgrößen sinkt die Genauigkeit, je weiter sich die evaluierenden Ausschnittsgröße von der vortrainierten Größe entfernt. Aufgrund dieser Befunde führen die Autoren ihr Model FlexiViT vor, eine flexible Version von ViT, die es ermöglicht, die Ausschnittsgröße zu ändern, ohne das Modell neu zu trainieren. Das Model wird mit variierenden Ausschnittsgrößen trainiert, die während des Trainingsprozesses zufällig bestimmt werden. Es kann gezeigt werden, dass FlexiViT eine gute Leistung über eine weite Reihe von Ausschnittsgrößen erbringt. Zudem stellen die Autoren fest, dass eine flexible Ausschnittsgröße den Trainingsprozess beschleunigen kann. [BIK+22]
Touvron et al. untersuchen mit ihrem PatchConvNet Model unterschiedliche Parameter auf dem ImageNet Datensatz. Unter anderem adaptieren sie einen anderen Ansatz zur Erstellung der Ausschnitte. Anstelle eines faltenden Filters, bei dem Filtergröße und Schrittweite auf den Wert der Ausschnittsgröße gesetzt werden, wir ein kleines CNN Netzwerk mit $3x3$ Faltungen und Schrittweite 2 verwendet. Neben einer Evaluierung ihres Models mit unterschiedlichen Modellgrößen und

Bildauflösung wurde auch die stochastische Tiefe untersucht mithilfe des Hyperparameters DropPath. Die Autoren kommen zu dem Ergebnis, dass die stochastische Tiefe, wie bei ViT ein wichtiger Faktor ist und die Optimierung dieses Hyperparameters sehr wichtig ist. [TCE+21]
Chhabra et al. haben eine weitere Regularisierungstechnik names PatchSwap für ViTs entwickelt. Für die Evaluierung wurde hierfür PatchSwap mit anderen Techniken wie Mixup verglichen. Evaluiert wurde unter anderem auf den Datensätzen CIFAR10, CIFAR100 und FashionMNIST, mit unterschiedlichen Ausschnittsgrößen. Aus den Ergebnissen lässt sich ablesen, dass der Austausch der Datenaugmentierungstechnik nur minimal durch die Ausschnittsgröße beeinflusst wird. So scheint der Anstieg an Genauigkeit von geringsten auf höchsten Wert verhältnismäßig bei allen Größen auf ähnlichem Niveau(Zum Vergleich auf CIFAR100 stieg die Genauigkeit bei Ausschnittsgröße 4 um 4,3%, 8 um 3.9% und bei 16 um 4.3% an) [CVL22].
Godbole et al. liefern einen guten Einsteig, der sich mit der Bestimmung von Hyperparametern befasst. Hierfür stellen die Autoren eine Einleitung bereit, bei denen sie ihre Erfahrungen teilen. Neben einer stufenweisen Anleitung, wie sich an ein neues Projekt herangehen lässt, stellen die Autoren auch zahlreiche Informationen bereit, die sich mit der Optimierung des Trainingsprozesses befassen [GDG+21].

Neben dem Trainieren der neuronalen Netze ist es auch wichtig, die erhobenen Ergebnisse anschaulich darzustellen, um bestehende Relationen einfacher interpretieren zu können. Heyen et al. haben sich mit der Analyse von Klassifikationsmodellen beschäftigt. In ihrem Artikel stellen die Autoren ClaVis vor, einem visuellen Analysesystem, welches mithilfe von unterschiedlichen Visualisierungstechniken dem Benutzer ermöglichen eine Vielzahl von Klassifikatoren, die mit unterschiedlichen Konfigurationen von Hyperparametern trainiert wurden, darzustellen [HMN+20].

4. Ansatz

Es soll eine Evaluierung durchgeführt werden, bei denen jeweils die Werte eines einzelnen Parameters variiert werden und gleichzeitig dieselbe Variation auf unterschiedlichen Ausschnittsgrößen stattfindet. Der Schwerpunkt der Evaluierung soll dabei auf dem Einfluss der Ausschnittsgrößen liegen und wie sich unterschiedliche Größen in Kombination mit anderen Hyperparametern einerseits auf die Genauigkeit des Modells und andererseits auf die Laufzeit und Speicheranforderung des Trainingsprozesses auswirken.
Für die Umsetzung soll ein Framework implementiert werden, welches die automatisierte Evaluierung von Computer Vision Modelle mit einer großen Menge von Konfigurationen ermöglicht.

4.1. Herangehensweise

4.1.1. Auswahl der Modelle

Um eine Auswahl an Modellen erstellen zu können, wurde zunächst Literaturrecherche betrieben, bei denen relevante Publikationen und deren Implementierung in Betracht gezogen werden können. Hierbei gab es mehrere Anforderungen an das dazugehörige Modell, die im Folgenden aufgelistet werden.

Aktualität Um eine Aktualität der Modelle und somit den aktuellen technischen Stand zu repräsentieren, sollen Modelle evaluiert werden, die entweder State-of-the-Art sind, oder sich mit solchen vergleichen, gleichzeitig jedoch einen Kompromiss zwischen Leistung und Modellkomplexität eingehen.

Quellcode Verfügbarkeit Der Quellcode des Modells ist öffentlich verfügbar. Publikationen, welche ihren Quellcode nicht in ihrer Veröffentlichung zur Verfügung stellen, sind häufig Publikationen, die sich mit einer sehr speziellen Problemstellung befassen, wie beispielsweise dem Erkennen von Krebszellen auf Bildern von Zellen, die mit einem Mikroskop aufgenommen wurden. Diese Datensätze stammen meist aus medizinischen Instituten und sind nicht frei zugänglich. Die Verfügbarkeit gewährleistet, dass sich das Modell in den Trainingsprozess einbinden und mit unterschiedlichen Hyperparametern trainieren lässt.

Verwendetes Framework Neben der allgemeinen Quellcodeverfügbarkeit, ist es zudem wichtig, dass alle Modelle in demselben Framework implementiert sind, um Unterschiede bei der Evaluation zu vermeiden. Die beiden prominentesten Frameworks sind PyTorch [PGM+19a] und Tensorflow [ABC+16]. Paszke et al. konnten aufzeigen, dass es je nach Model zu messbaren Unterschieden kommen kann. Während ResNet-50 in PyTorch einen 6% höheren Durchsatz hat, sind es bei VGG-19 bis zu 80%. Weiterhin hat PyTorch einen höheren Speicherverbrauch. Insbesondere zu Beginn während dem Laden des Datensatzes ist dieser bis zu doppelt so hoch [PGM+19b].
Zu Beginn der Recherche wurden Modelle für beide Frameworks in Betracht gezogen, entschieden wurde sich letztendlich für PyTorch aufgrund einer vielseitigeren Auswahl. Die Webseite Papers With Code bietet hierbei eine gute Anlaufstelle zur Recherche.

Auswahl von vordefinierten Modellen Viele der SOTA Modelle sind meist sehr komplex. Einige davon haben trainierbare Parameter in Milliardenhöhe, die auf sehr großen, teilweise auch privaten Datensätzen trainiert werden. Für eine ausführliche Evaluierung sollen möglichst viele Datenpunkte erstellt werden. Um dies in dem gegebenen zeitlichen Rahmen bewerkstelligen zu können, sollen daher Modelle bevorzugt werden, die entweder bereits moderate Größe haben, oder aber mehrere Modell-Versionen vordefinieren. So werden häufig Modelle, ähnlich wie dem ViT Model, in Kategorien wie Vit-L (large) ViT-B (base), ViT-S (small) und ViT-T (tiny) unterteilt. Kleinere Modelle sind weniger komplex und haben häufig deutlich weniger trainierbare Parameter und damit eine bessere Laufzeit, jedoch mit dem Kompromiss etwas an Genauigkeit einzubüßen.

Verwendung von Bildausschnitten Damit das Modell für die Evaluierung verwendet werden kann, muss es mit Bildausschnitten arbeiten. Insbesondere ist es wichtig, dass sich die Größe des Bildausschnitts vorher fest deklarieren lässt und als Variable dem Modell übergeben werden kann.

Anwendungsdomäne Computer Vision Modelle kommen in vielen Anwendungsdomänen zu Einsatz. Darunter zählen unter anderem Bildklassifizierung, Objekterkennung und semantische Segmentierung. Die Modelle werden jedoch auf großen Datensätzen trainiert, um beste Ergebnisse zu erreichen, was ein zeitintensiver Prozess ist. Lv et al. haben gezeigt, dass die Genauigkeit von Modellen stark von dem gewählten Datensatz abhängen kann [LBW22]. Auch Steiner et al. berichten, dass die Genauigkeit stark von dem verwendeten Datensatz abhängen kann [SKZ+21].Liu et al. berichten, dass standard Vision Transformers wettbewerbsfähige Leistungen bei der Bildklassifierung erreichen, jedoch Schwierigkeiten bei anderen Anwendungen, wie der Objekterkennung oder Semgentierung [LMW+22]. Den gewonnenen Erkenntnissen nach wurde sich für Bildklassifizierung als Anwendungsdomäne entschieden.

Ausgewählte Modelle

CaiT CaiT (Class-Attention in Image Transformers) ist eine Weiterentwicklung des Transformers ViT. Das Modell führt einen klassenorientierten Aufmerksamkeitsmechanismus ein, der sich auf die Darstellung der Klassen-Token konzentriert. Zudem werden die Klassen-Token erst später in die Architektur eingefügt. Die entstandene Architektur ähnelt weiterhin der Encoder/Decoder-Architektur mit dem Unterschied, dass Transformer-Schichten welche den Selbst-Aufmerksamkeitsmechanismus zwischen Bildausschnitten beinhalten, von den Schichten für klassenorientierter Aufmerksamkeit

getrennt werden.
Im Vergleich zum ursprünglichen ViT, hat das entstandene Modell eine verbesserte Leistung. Dank einer Schichtenskalierungsstrategie eine bessere Tiefenskalierung und durch den klassenorientierten Aufmerksamkeitsmechanismus eine verbesserte Laufzeit [TCS+21].

SplitMixer Borji & Lin stellen ihr Modell SplitMixer in einem ähnlichen Szenario wie Lv et al. mit ihrem MDMLP Modell [LBW22] vor. SplitMixer ist ein einfaches und leichtgewichtiges Modell, welches Ähnlichkeit zu MLP Modellen hat, jedoch auch faltende Schichten verwendet. Das Modell ist angelehnt an ConvMixer [TK22], verwendet jedoch eindimensionale Faltungsfilter. Den Ergebnissen der Autoren nach schafft es auf gleichen Niveau wie ConvMixer zu bleiben und dabei an Modellgröße zu verlieren [BL22].

WaveMLP WaveMLP ist ein weiteres MLP-basiertes Modell. Als besonderes Merkmal zählt hierbei die Aggregierung der einzelnen Bildausschnitte. Diese werden als Wellenfunktion mit zwei Teilen repräsentiert. Die Amplitude repräsentiert das originale Merkmal und die Phase einen komplexen Wert, der sich abhängig vom semantischen Inhalt der Bilder ändert [THG+22].

ConvNeXt ConvNeXt ist ein Modell, dass auf Faltungsschichten aufgebaut. Liu et al. haben festgestellt, dass standardmäßige Version von Vision Transformern CNNs bei der Bildklassifierung überholen konnten, jedoch ihre Schwächen bei anderen Aufgaben wie der Objekterkennung und Segmentierung haben. Es wurden die vorhanden Unterschiede untersucht und aus dem gewonnen Wissen ein standard ResNet CNN Modell ”modernisiert” [LMW+22].

MLP-Mixer MLP-Mixer ist ein reines MLP Modell, welches zwei Arten von Schichten enthält, eine Schicht wendet MLPs auf den Bildausschnitt an, während eine zweite Schicht unabhängig davon die Merkmale über mehrere Ausschnitte hinweg verarbeitet [THK+21].

4.1.2. Auswahl der Datensätze

Zusätzlich zur Auswahl der fünf Modelle muss auch eine geeignete Auswahl an Datensätzen getroffen werden, auf denen die Modelle evaluiert werden sollen. Dabei wurden folgende Anforderungen berücksichtigt:

Verfügbarkeit Der Datensatz muss öffentlich zugänglich sein, um die Ergebnisse jederzeit reproduzieren zu können

Popularität Es sollen Datensätze bevorzugt werden, die auch in anderen Studien und veröffentlichen verwendet werden, sodass sich Ergebnisse mit anderen Arbeiten besser vergleichen lassen

Vielfallt Die Auswahl der Datensätze soll möglichst Vielfältigkeit sein, um mögliche Unterschiede in den Ergebnisse erkenntlich zu machen. Dabei berücksichtigt wurden: Inhalt der Bilder, Größe des Datensatzes, Anzahl der enthaltenen Klassen und die Auflösung der Bilder

Datensatz	Anzahl Klassen	Trainigsgröße	Testgröße	Besonderheit
CIFAR10	10	50,000	10,000	Auflösung 32x32 Pixel
Oxford-IIIT Pets	32	3680	3699	
Flowers102	102	2040	6149	
Stanford Cars	196	8,144	8,041	

Tabelle 4.1.: Auswahl von Datensätzen, welche für die Evaluierung verwendet werden

Größe Um möglichst viele Datenpunkte sammeln zu können, sollen möglichst kleine Datensätze evaluiert werden, die eine schnellere Trainingszeit ermöglichen

Als Orientierungspunkte wurden hierfür die standardmäßig bereitgestellten Datensätze durch die Torchvision Bibliothek, als auch die Internetseite PapersWithCode, verwendet. Die Wahl fiel auf die in Tabelle **??** aufgelisteten Datensätze. CIFAR10 hat als Besonderheit eine Auflösung von 32x32 Pixeln, ist dafür aber von der Größenordnung um ein zehnfaches größer. Die Oxford Pets und Flowers102 Datensätze beinhalten deren Namen entsprechend Bilder mit dem Fokus auf Haustiere beziehungsweise Blumen. Sie sind in etwa gleich Groß, unterscheiden sich jedoch in der Anzahl der vorhandenen Klassen. Diese ist bei dem Flowers102 Datensatz mehr als doppelt so groß. Der Stanford Cars Datensatz beinhaltet Bilder von Fahrzeugen und hat eine etwa drei- bis vierfach größere Trainingsgröße als Oxford Pets und Flowers102.

4.2. Implementierung

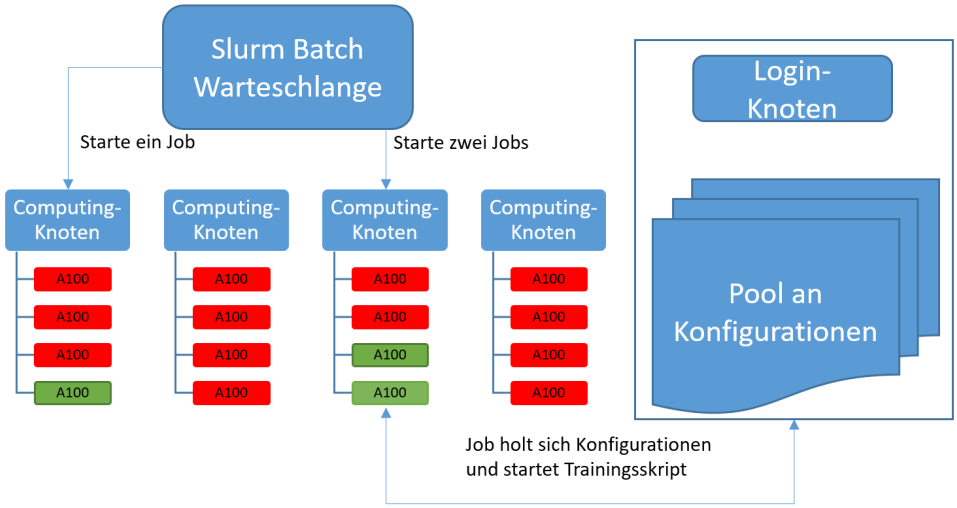
Implementiert wurde das Framework in Python 3.9 und PyTorch 1.13.1 mit der CUDA Version 11.7. Die Implementierung zur Evaluierung besteht aus 7 Elementen, welche modular aufgebaut sind. Es wurde Wert darauf gelegt, dass einzelne Bestandteile bei Bedarf erweitert oder ersetzt werden können.

4.2.1. Importierte Module

Für jedes Modell wurde der Quellcode in zwei Python Dateien aufgeteilt. In einem wird die Architektur des Modells definiert und in einer anderen Datei die Basisparameter. Vier der fünf Modelle nutzen hierfür ein *Argsparser* Objekt, um ihre Parameter zu definieren. Nur Splitmixer verwendet ein Wörterbuch als Deklarierungen. Beide Klassen entsprechen einer paarweisen Schlüssel-Wert Struktur, werden jedoch unterschiedlich aufgerufen. Um eine einheitlichere Verwaltung zu ermöglichen, wurde daher die Splitmixer Implementierung in ein entsprechendes *Argsparser* Objekt übersetzt.
Als Training-Skripte sollen die originalen Skripte in Betracht gezogen werden, welche auch von den Autoren verwendet wurden. Das Evaluierung-Framework dient daher primär als Wrapper, mit dem sich eine automatisierte Werteanpassung ermöglichen lässt. Vier der fünf Modelle nutzen hierbei dasselbe ImageNet Trainingsskript als Grundlage, welches häufig von anderen Autoren adaptiert wird. Die Autoren von Splitmixer verwenden eine eigene, schlankere Implementierung.

4.2.2. Automatisierte Trainingsdurchläufe

Für die Evaluierung wurde der Rechnerverbund des Instituts verwendet. Auf diesem läuft das Slurm System. Es existieren insgesamt vier Rechen-Knoten, welche jeweils vier NVIDIA A100 Grafikkarten mit je 40GB Grafikspeicher besitzen.
Aus technischen Limitierungen sind Aufträge an ein zeitliches Limit von 48 Stunden gebunden. Um eine möglichst hohe GPU Zuweisung zu erreichen, wurde daher ein eigenes Verwaltungssystem entwickelt, das die zugewiesenen Grafikkarten möglichst dauerhaft während des 48 stündigen Zeitfensters auslastet. Abbildung 4.1 stellt eine vereinfachte Form des Slurm Systems dar.



[H]

Abbildung 4.1.: Vereinfachte Darstellung der Architektur des Servers und wie Aufträge automatisiert ein Training initialisieren können

Per SSH Zugriff gelangt man auf den Login-Knoten, von dem man mit dem Befehl *sbatch script.sh* einen Auftrag in die Warteschlange einreihen kann. Falls gewünscht, hat der Benutzer die Möglichkeit, einen bestimmten Rechen-Knoten für die Ausführung seines Jobs anzugeben. Dies kann notwendig sein, wenn nur bestimmte Knoten für die spezifischen Anforderungen des Benutzers konfiguriert wurden. Alternativ kann der Benutzer den Knoten für die Ressourcenzuweisung undefiniert lassen. In diesem Fall wird der SLURM-Manager die frühest verfügbaren Ressourcen zuweisen, die den Anforderungen des Auftrags entsprechen.
Für die Evaluierung werden Aufträge erstellt, die jeweils eine der 16 A100 GPUs anfordern. Die Zuweisung und Abarbeitung der Aufträge erfolgt automatisiert durch das Slurm System. Um alle 16 GPUs zu beanspruchen, müssten also 16 Aufträge zeitgleich eingereicht werden, sofern alle GPUs unbenutzt sind. In dem Beispiel in Abbildung 4.1 werden 3 Aufträge direkt an die nicht reservierten (grün) GPUs übergeben werden, während alle weiteren Aufträge in die Slurm Warteschlange eingereicht werden. Aus Fairnessgründen erfolgt die Erstellung der Aufträge nicht automatisiert, sondern muss von dem Benutzer manuell gehandhabt werden.

Bei jedem Auftrag wird das *eval_worker* Modul gestartet. Die Funktionsweise ist in Algorithmus 4.1 dargestellt. Das Skript ruft eine offene Konfiguration aus dem Pool von Konfigurationen ab, der sich auf dem Login-Knoten befindet, und initiiert einen Trainingsprozess. Um gleichzeitige

Algorithmus 4.1 Eval_worker Pseudocode

```
1: config_path ← getConfigPath()
2: while true do
3:     nextConfig ← getNextConfig()
4:     if nextConfig is empty then
5:         break
6:         print("config queue is empty, finishing")
7:     else
8:         args ← args_manager.getArgs(nextConfig)
9:         try
10:            eval(args)
11:        catch Exception as e
12:            print(e)
13:    end if
14: end while
```

Zugriffe auf Dateien zu vermeiden, werden in der Unterfunktion $getNextConfig()$ entsprechende Maßnahmen implementiert.
Sofern eine offene Konfiguration existiert, wird diese durch das Python-Modul $os.rename$ umbenannt, indem ein $\_locked\_$ Präfix vor die entsprechende Datei angehängt wird. Die unterliegende C Bibliothek, die dabei aufgerufen wird, stellt eine atomare Operation dar. Wurde die Datei erfolgreich umbenannt und damit reserviert, wird der Trainingsprozess gestartet. Dateien, die dieses Präfix bereits enthalten, werden ignoriert, um doppelte Ausführungen zu vermeiden. Ist der Trainingsprozess erfolgreich abgeschlossen, wird die Datei aus dem Pool an offenen Konfigurationen entfernt.
Um mögliche Fehler während des Trainings zu überbrücken, ist der Aufruf des Trainingsprozesses innerhalb eines try-$catch$-Blocks eingebettet. Mögliche Fehlerquellen können fehlerhafte Eingaben in der Konfigurationen oder die Verwendung von Parametern sein, die mehr Grafikspeicher benötigen, als verfügbar ist.

Argumentparser Das Modul $args\_manager$ verwaltet die Erstellung eines uniformen $argsparse.namespace$ Objektes, welches an das Trainings-Skript übergeben werden kann. Es liest sowohl die Basisparameter der Modelle, als auch die Evaluierungs-Parameter ein, und überschreibt gegebenenfalls vorhandene Werte, wenn diese von dem Benutzer mit übergeben wurden. Dadurch lässt sich bewerkstelligen, dass der Eingabeaufwand durch den Benutzer minimal gehalten wird.

4.2.3. Trainingsprozess

Algorithmus 4.2 stellt ein vereinfachtes Beispiel dar. Als Parameter wird ein Argsparse Namespace Objekt übergeben, welche zuvor von dem $args\_manager$ erstellt wurde. Der Datensatz-Manager verwaltet die Erstellung der $DataLoader$ Objekte, die für das Trainieren und Validieren des Modells benötigt werden. Insbesondere wird hier der Datensatz durch Augmentierung vorverarbeitet. Das Modell-Bauer Modul ist entsprechend für die Instanziierung des Modells zuständig.
Möchte man Model oder Datensatz zu einem späteren Zeitpunkt ändern, kann man dies leicht durch Ersetzen der entsprechenden Zeilen mit seiner eigenen Methode, bewerkstelligen. Alternativ können auch die entsprechenden Module erweitert werden.
Falls das gesamte Skript ausgetauscht werden soll, muss nur das Protokolierungs-Modul $custom\_logger$ mit übernommen werden. Dieses initialisiert zu Beginn jeder Epoche unter anderem die Messung der Laufzeit mittels eines CUDA Events Objektes und misst die Laufzeit jeder Epoche mit. Nebenbei werden auch Speicherbelegung, Trainings- und Evaluierungsverlust

Algorithmus 4.2 Pseudocode für den Trainingsablauf

```
1: function TRAIN(args)
2:     #setup training
3:     model = model_builder(args)
4:     train_loader, val_loader = dataset_manager.get_loaders(args)
5:     logger = custom_logger(len(val_loader), args)
6:
7:     #start training
8:     for epoch in args.epochs do
9:         logger.init_epoch()
10:
11:        #train and validate one epoch
12:        train_loss, val_loss, val_acc = train_one_epoch(model, train_loader, val_loader, args)
13:
14:        logger.update_variables(epoch, train_loss, val_loss, val_acc)
15:        logger.end_epoch()
16:        logger.print_epoch_info()
17:        logger.write_epoch_results()
18:    end for
19:
20:    logger.end_training()
21: end function
```

und Evaluierungsgenauigkeit mitverfolgt. Für jeden Durchlauf wird ein separates Protokoll erstellt, bei dem diese Werte für jede Epoche erfasst werden. Dazu wird für jede Instanz des $eval\_worker$ Moduls eine Datei erstellt, indem die Zusammenfassung des Durchlaufs in einer Zeile je Trainingsprozess festgehalten.
Weiterhin zu beachten ist, dass der Datenaugmentierung-Prozess Mixup erst während des Trainings stattfindet. Möchte man diesen Prozess in einem eigenen Skript anwenden, so muss auch diese Funktionen entsprechend übernommen werden.

4.2.4. weitere Hilfsfunktionen

Hierunter befinden sich weitere Hilfsfunktionen, welche die Erstellung und Verwaltung von Konfigurationen erleichtern. Diese sind für die Verwaltung der verschiedenen YAML-Konfigurationen zuständig.

Konfigurationsstapel So kann unter anderem aus einer gestapelten Ansammlung an Konfigurationen, wie in Abbildung 4.2 dargestellt ist, eine Liste an YAML-Konfigurationsdateien erstellt werden. Hierbei wird beispielsweise das WaveMLP Model für den Datensatz Flowers102 konfiguriert. Der auszuwertende Parameter wird mit der Variable $param$ deklariert, und der entsprechende Parameterwert mit der Varible $pval$ definiert. Hierbei entspricht jede Zeile einem Trainingsdurchlauf. Es wird Modell, Datensatz und Ausschnittsgröße ausgelesen und in diesem Beispiel jeweils Stapelgröße oder Bildauflösung mit dem definierten Wert trainiert, anstelle des Basiswertes des entsprechenden Parameters.
Die Basiskonfiguration für den Flowers102-Datensatz wurde mit einer Stapelgröße (hier als batch_size bezeichnet) von 128 und einer Bildauflösung von 224x224 erstellt. Um Ressourcen zu schonen und die Auswertung einer größeren Anzahl von Werten zu ermöglichen, wird der Trainingsprozess nur einmal mit den Basiswerten durchgeführt. Der Durchlauf mit einer Bildauflösung von 224 hat entsprechend die gleiche Konfiguration und ist bereits abgedeckt. Diese Zeile wird folglich auskommentiert, was die Erstellung der YAML-Datei mit dieser spezifischen Konfiguration

ausschließt.
Stattdessen werden die Ergebnisse der Basis mithilfe eines weiteren Skripts vervollständigt, indem die vorhandenen Ergebnisse der Basiskonfiguration auf die ausgelassenen Durchläufe kopiert werden.

```
#Model: wavemlp, dataset: flowers
#patch: 16

#batch size
{model: wavemlp, ds: flowers, param: batch_size, pval: 128, patch_size: 16}
{model: wavemlp, ds: flowers, param: batch_size, pval: 32, patch_size: 16}
{model: wavemlp, ds: flowers, param: batch_size, pval: 64, patch_size: 16}
{model: wavemlp, ds: flowers, param: batch_size, pval: 256, patch_size: 16}

#img_res
{model: wavemlp, ds: flowers, param: img_res, pval: 128, patch_size: 16}
#{model: wavemlp, ds: flowers, param: img_res, pval: 224, patch_size: 16}
{model: wavemlp, ds: flowers, param: img_res, pval: 288, patch_size: 16}
{model: wavemlp, ds: flowers, param: img_res, pval: 384, patch_size: 16}
```

Abbildung 4.2.: Beispielhafte Stapekonfigurationen für das WaveMLP-Modell auf dem Flowers-Datensatz. Die getesteten Parameterwerte umfassen die Stapelgröße und die Bildauflösung bei einer festen Ausschnittsgröße von 16

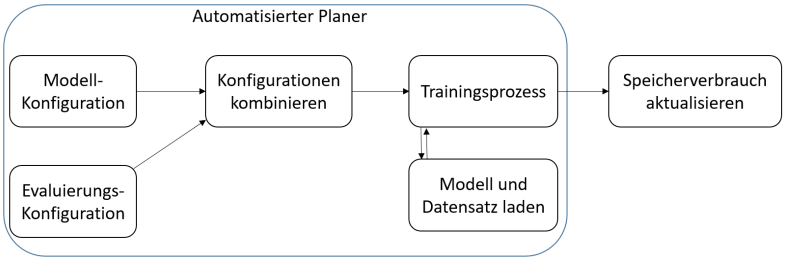


Abbildung 4.3.: Vereinfachte Darstellung der Architektur des Servers und wie Aufträge automatisiert ein Training initialisieren können

Anpassung der gemessenen Speicherbelegung Abbildung 4.3 zeigt den bisherigen Prozessablauf, der bisher erläutert wurde. Bei der Auswertung der Ergebnisse ist jedoch eine Problematik aufgefallen, die genauer in Kapitel 5 erläutert wird. Es hat sich herausgestellt, dass es starke Schwankungen bei der Messung des Speicherbedarfs bestehen. Um einen erneuten Durchlauf zu vermeiden, wurden die Ergebnisse nachbearbeitet. Hierbei werden die vorhandenen Protokolle eingelesen und der Speicherbedarf wird dabei neu bestimmt. Anstelle der Speicherspitze, wird für jeden Durchlauf der durchschnittliche Speicherbedarf ab der zweiten Epoche erfasst.

4.2.5. Grafische Benutzeroberfläche

Neben der Implementierung zur Evaluierung soll auch eine Grafische Benutzeroberfläche (GUI) implementiert werden. Als Programmiersprache wurde ebenfalls Python gewählt. Für die GUI Programmierung in Python bestehen zahlreiche Bibliotheken. In Betracht gezogen wurden die beiden Bibliotheken PyQT5 und Plotly Dash.

Batch Generator Das ursprüngliche Konzept zielte darauf ab, eine effiziente Verwaltung aller Konfigurationen zu ermöglichen und deren Ergebnisse darzustellen. Während der Experimentierphase wurde jedoch schnell erkannt, dass die Verwaltung von Hunderten von Konfigurationen eine mühsame Aufgabe darstellt. Aus diesem Grund wurde beschlossen, auf eine manuelle Verwaltung zu verzichten und stattdessen einen Generator zu erstellen, der in Abbildung 4.4 zu sehen ist.
Der Benutzer kann neben Model und Datensatz, die Ausschnittsgrößen angeben, die evaluiert werden sollen und die Größen Dimension der zu untersuchenden Parameter. Mehrere Werte bei diesen drei Feldern können durch Komma Trennung angegeben werden. Der Wert n steht für die Anzahl der Parameter und m für die Anzahl der Werte pro Parameter, die evaluiert werden sollen. Anschließend kann man durch Drücken auf den Knopf "Generate Grid" ein n x m großes Gitter erstellen, bei dem man die Namen der Parameter, die Basiswerte und alle sonstigen Werte definieren kann. Abschließend wird durch Drücken auf den Knopf "Write to file" ein Stapel an Konfigurationen erstellt, wie auch in Abbildung 4.2 abgebildet ist. Auch hier wird die Basis Konfiguration nur einmal erstellt.

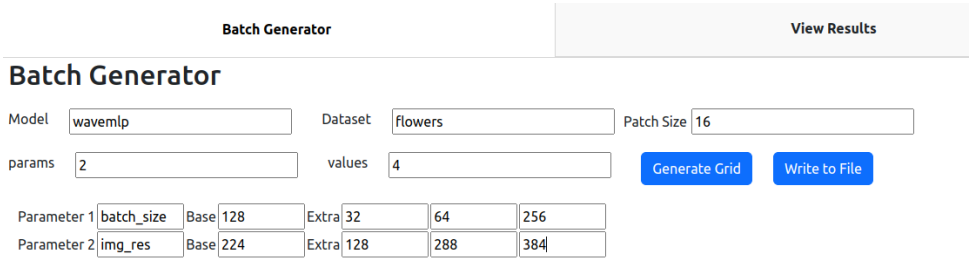


Abbildung 4.4.: GUI-Batch Generator Tab

Ergebnistabelle In dem zweiten Tab werden alle existierenden Ergebnisse in eine Tabelle geladen und diese dargestellt. Alle Zeilen in der Tabelle können einzeln gefiltert und sortiert werden, sodass die Verwaltung der Ergebnisse erleichtert wird. Wurden die Konfiguration per Hand erstellt, oder Trainingsdurchläufe abgebrochen, beispielsweise aufgrund von zu hoher Speicheranforderungen, kann es vorkommen, dass einzelne Einträge fehlen. Mithilfe der Filter- und Sortierfunktion Lassen sich schnell fehlende Einträge identifizieren und können bei Bedarf von dem Benutzer genauer untersucht werden.

Die zweite Aufgabe der GUI besteht in der Darstellung der Ergebnisse. Es wurde entschieden, die GUI in Form eines Dashboards unter Verwendung des Dash-Frameworks zu implementieren. Der Vorteil dabei ist, dass Plotly-Graphen direkt und nativ unterstützt werden, was eine interaktive Visualisierung der Ergebnisse ermöglicht. Zusätzlich reduziert die Verwendung des Dash-Frameworks den Implementierungsaufwand im Falle von Anpassungen oder Erweiterungen der GUI.

Visualisierung In dem dritten Tab werden die Ergebnisse visuell dargestellt. für die Wahl der Darstellung wurde sich für eine Scatterplot-Matrix entschieden. hierbei werden alle Parameter paarweise in einer Matrix an Diagrammen dargestellt. Zu untersucht werden sollen die Genauigkeit, Zeitaufwand und Speicherverbrauch. Bei den drei Werten ergeben sich entsprechend eine 3 x 3 große Matrix, bei der jeder der drei Parameter sich einmal auf der y-Achse und x-Achse der Matrix befindet.
Je Datensatz werden die fünf Modelle mit jeweils sieben verschiedenen Parametern evaluiert. Dies hat zur Folge, dass man pro Datensatz 35 verschiedene Abbildungen erhält. Die Scatterplot-Matrix lässt sich in drei Teile unterteilen. In dem unteren Teil der Matrix befinden sich drei Diagramme die bereits eine paarweise Darstellung der drei Parameter enthalten. die Diagonale der Matrix enthält den gleichen Parameter für x und y-Achse und wird je nach gewählter Bibliothek unter anderem gar nicht dargestellt. In dem oberen Teil der Matrix befinden sich dieselben Diagramme wie in dem unteren Teil, nur dass die Achsen vertauscht und entsprechend die Datenpunkte gespiegelt sind.
Um die Darstellung etwas kompakter zu halten, wurde eine abgewandelte Form der Scatterplot-Matrix erstellt. Hierbei wird nur der untere Teil der Matrix dargestellt. Auf diese Weise lässt sich verwirklichen, dass auf gleiche Fläche mehr Ergebnisse abgebildet werden können. Insbesondere werden die einzelnen Teilmatrizen in einem $\lceil \frac{N}{2} \rceil$ x 2 großen Gitter abgebildet, wobei N der Anzahl der Teilmatrizen entspricht. Daraus ergibt sich eine Plotly Figur mit $2\lceil \frac{N}{2} \rceil$ x 4 Graphen. Der obere rechte Teil jeder Teilmatrix wird leer gelassen und kann für Annotationen oder zur Positionierung der Legende verwendet werden. Anders als bei einer Scatterplot-Matrix teilen sich jedoch die einzelnen Diagramme keine Achsenwerte. In einem Vergleich mit geteilten Achsen hat sich gezeigt, dass viele Verläufe dadurch nur schwer ersichtlich werden, da sich die Wertegrenzen, besonders mit mehreren Ausschnittsgrößen in einem Schaubild, deutlich unterscheiden. Die daraus entstehende Abbildung lässt sich dank der Funktionalität der Plotly Bibliothek interaktiv analysieren. Dank der Zoom-Funktion können Werte, wie in dem eben genannten Beispiel besser untersucht werden. Mithilfe der Hover-Funktion können zudem schnell weitere Informationen über den entsprechenden Daten Punkt abgerufen werden.
Dank der Exportierfunktion lassen sich die Schaubilder als PNG Datei speichern, sodass dieses auch für die Darstellung der Ergebnisse in dieser Arbeit verwendet werden können.
Ein entsprechendes Beispiel der Visualisierung ist in Abbildung 5.5 zu sehen. Neben einer visuellen Abtrennung der einzelnen Teilmatrizen, wird zudem der größte Parameterwert in dem dazugehörigen Diagramm als dunkler Stern kodiert, sodass man die Reihenfolge der Datenpunkte besser einordnen kann.

5. Durchführung

In dem folgenden Kapitel wird die Durchführung der Evaluierung erläutert. Diese ist in drei Teile gegliedert. Zunächst wird die Herangehensweise und eine Vorabevaluierung erläutert, bei der die finalen Hyperparameterwerte bestimmt werden. Insbesondere wird auf aufgetretene Probleme hingewiesen und alle weiteren Aspekte vorgestellt, die untersucht wurden. Anschließend werden die Ergebnisse zuerst an einem Beispiel im Detail vorgestellt und abschließend zusammengefasst.

5.1. Wahl der zu evaluierenden Hyperparamter

Für eine uniforme Evaluierung und somit bessere Vergleichbarkeit musste zunächst eine Liste an Hyperparametern erstellt werden, die bei allen Modellen anpassbar sind. Ein Beispiel für einen Parameter, der damit ausgeschlossen ist, ist der Architektur-spezifische Hyperparameter Schichtenskalierung bei dem Transformer-Modell CaiT. Im folgenden Abschnitt wird die Wahl der Parameter vorgestellt und anschließend alle verwendeten Werte aufgelistet. Zur Auswahl standen unter anderem folgenden Parameter als Option:

- Ausschnittsgröße
- Stapelgröße
- Bildauflösung
- Modellgröße
- Versteckte Dimensionsgröße´
- Datenaugmentierung
- Lernrate
- Regularisierung in Form von DropOut bzw. DropPath

Aus zeitlichen Limitierungen konnten nicht beliebig viele Hyperparameter untersucht wurden. Es wurde sich daher auf eben genannte Auswahl beschränkt. Hierbei sollten nach Möglichkeit Parameter gewählt werden, die sich nicht nur auf die Genauigkeit auswirken, sondern auch auf Laufzeit und Speicherbedarf, um einen Einfluss bei jedem der drei Evaluierungsmetriken aufzeigen zu können. Neben der Ausschnittsgröße, die elementarer Teil der Evaluierung ist, bieten sich dafür Stapelgröße, Bildauflösung, Modelltiefe und Größe der versteckten Dimension an. Weiterhin wurde sich für die Datenaugmentierung entschieden, da während der Vorabevaluierung auch Einfluss auf die Laufzeit beobachtet werden konnte. Lernrate und Regularisierung in Form von DropOut beziehungsweise DropPath wurden hinzugefügt, da diese als besonders wichtig erachtet werden[TCE+21].

Als Ausschnittsgröße wurden die gängigen Größen von 16 und 32 gewählt. Da sich das Eingabebild bei manchen Modellen exakt durch die Ausschnittsgröße teilen lassen muss, um stets gleich große Ausschnitte zu bekommen, kamen als dritte Ausschnittsgröße nur die Werte 8 und 64 infrage. Frühe Tests mit der Ausschnittsgröße 64 haben zu schlechteren Ergebnissen geführt, weshalb die Wahl als dritte Größe auf den Wert 8 fiel. Bei dem CIFAR10 Datensatz wurden entsprechend kleinere Größen von 2, 4 und 8 gewählt, da hier die Auslösung deutlich geringer ist.

Als Ansatz wurde nach der Anleitung von Godbole et al. vorgegangen[GDG+21]. Die Modellarchitekturen der Modelle wurden bereits von den Autoren des jeweiligen Modells erstellt und sollten daher für die Basiskonfiguration nicht verändert werden. Bei Splitmixer wurde das Standardmodell ausgewählt, da dieses bereits sehr leichtgewichtig ist. Bei allen anderen Modellen wurde die kleinste Version(meist mit Tiny gekennzeichnet) gewählt, da diese im Vergleich zu Splitmixer deutlich größer sind.
Als nächster Schritt wurde die Stapelgröße bestimmt. Borji & Lin verwenden eine Stapelgröße von 512 für CIFAR10 und 128 für den Flowers102 Datensatz bei ihrem Splitmixer Modell [BL22]. Lv et al. hingegen einen Wert von 128 für CIFAR10 und 16 für Flowers102 [LBW22]. Den Ergebnissen der Autoren zufolge, konnte Splitmixer bei beiden Datensätzen bessere Genauigkeiten erreichen. Hierbei sollte jedoch angemerkt werden, dass mit unterschiedlichen Ausschnittsgrößen trainiert wurde. In ersten Experimenten hat sich jedoch gezeigt, dass die verwendeten Stapelgrößen von Splitmixer eine gute Grundlage darstellen. Zudem begünstigt eine größere Stapelgröße die Laufzeit und somit die Gesamtanzahl an Durchläufen innerhalb des zeitlichen Rahmens. Weiterhin wurde untersucht, wieviel Speicher dabei belegt wird. Idealerweise sollte dieser nicht höher als 50% des vorhandenen Grafikspeichers einer einzelnen A100 GPU sein und somit bei ca. 20GB liegen. Auf diese Weise ist sichergestellt, dass genug Speicher als Reserve vorhanden ist für Parameterwerte, die durch eine Wertänderung den Speicherbedarf erhöhen. Hierbei muss jedoch festgehalten werden, dass CaiT einen exponentiellen Speicherbedarf aufweist, abhängig von der Ausschnittsgröße. Für eine vollständige Evaluierung auf den Ausschnittsgrößen 8, bzw. 2 für CIFAR10 müsste die Stapelgröße auf den Wert 16, teilweise 8 heruntergesetzt werden, welches die Evaluierung aus zeitlichen Gründen einschränken würde. Es wird daher in Kauf genommen, dass es bei diesem Modell für die jeweils kleinste Ausschnittsgröße häufig zu fehlenden Werten kommen kann, da mehr Speicher benötigt wird als vorhanden ist.

5.1.1. Vorabevaluierung

In einer ersten Experimentierphase musste zunächst entschieden werden, mit welchem Trainingsskript die Evaluierung durchgeführt werden soll. Vier der fünf Modelle verwenden das ImageNet Trainingsskript, während Splitmixer hingegen eine eigene, deutlich schlankere Implementierung verwendet. Probeweise wurde das ImageNet Skript mit dem von Splitmixer vertauscht, um zu sehen, ob hier unterschiedliche Ergebnisse zu beobachten sind. Getestet wurden die beiden Modelle CaiT und WaveMLP. Während bei CaiT nur eine geringe Verbesserung verzeichnet wurde (< 1%), konnte bei WaveMLP eine deutliche Verbesserung verzeichnet werden. (>10%). Weiterhin untersützt das ImageNet Skript die Verwendung eines exponentiell gleitenden Mittelwertes (EMA). Dieser hat jedoch zu schlechteren Ergebnisse geführt. Touvron et al. haben zudem festgestellt, dass die Verwendung von EMA nur einen sehr geringen (0,1%) Einfluss auf die Genauigkeit von ihrem

Model hat [TCD+21]. Aus diesen beiden Gründen wurde sich für das Splitmixer Skript mit einem normal gleitenden Mittelwert entschieden.

Um aussagekräftigere Ergebnisse zu bekommen, sollen zwei weitere Vorabevaluierungen durchgeführt werden. Mithilfe einer einfachen Gittersuche sollen mehrere Parameter untersucht werden, um die Basis der Parameter zu verbessern. Hierbei soll angemerkt werden, dass es nicht Ziel der finalen Evaluierung ist die bestmöglichen Genauigkeiten zu erreichen, vielmehr sollte versucht werden sicherzustellen, dass zu geringe Genauigkeiten keinen negativen Einfluss auf die Interpretierbarkeit der Ergebnisse haben.
In einem ersten Durchlauf soll der Einfluss der Parameter für die Datenaugmentierung untersucht werden. Hierfür eignet sich das Splitmixer Modell auf dem CIFAR10 und Flowers102 Datensatz. Diese Kombination wurde bereits von den Autoren verwendet und erzielte gute Ergebnisse. Dabei wurde unter anderem RandAugment zur Augmentierung verwendet. Die Autoren stellen eine Kombination mit Mixup in Aussicht, welche die Ergebnisse nochmals verbessern könnte [BL22]. Entsprechend wurde sich für eine Kombinations aus beiden Techniken entschieden. Die Werte für die Datenaugmentierung wurden angelehnt an dem Ansatz von Steiner et al., welche ebenfalls eine Kombination von RandAugment und Mixup verwenden. Gewählt wurde jeweils eine schwache, eine mittlere und eine starke Augmentierung, nach Vorgaben der Autoren [SKZ+21].

Ausschnittsgröße	Augmentierung		
	[2, 10, 0.2]	[2, 15, 0.5]	[2, 20, 0.8]
	leicht	moderat	stark
2	93.44	**94.35**	93.42
4	90.53	**91.41**	90.67

Tabelle 5.1.: Genauigkeit des Modells Splitmixer auf dem CIFAR10 Datensatz mit unterschiedlich starker Datenaugmentierung. Das jeweils beste Ergebnis ist hervorgehoben.

In Tabelle 5.1 können die Ergebnisse für den CIFAR10 Datensatz entnommen werden. Für beide Ausschnittsgrößen erzielte eine moderate Augmentierung die besten Ergebnisse. Der Eintrag [2, 15, 0.5] steht hierbei für 2 Operationen während der RandAugment Transformierung mit einer Intensität von 15. Der Wert 0.5 entspricht dem Alpha Wert für Mixup.
Abbildung 5.1 zeigt zudem den Trainings- und Validierungsverlust mit leichter und moderater Datenaugmentierung. Vergleicht man beide Ergebnisse, fällt auf, dass der Validierungsverlust in beiden Schaubilder sehr ähnlich ist. Der Trainingsverlust ist bei leichter Augmentierung etwas geringer, was ein Anzeichen von Überanpassung sein kann, da sich das Modell auf dem Trainingsdaten verbessert, ohne dabei besser generalisieren zu können.

Die beiden Datensätze Pets und Flowers102 sind im Vergleich zu CIFAR10 deutlich kleiner. Daher soll auch hier untersucht werden, ob eine moderate Augmentierung als Basis gewählt werden soll, oder ob es aufgrund der geringeren Anzahl an Eingaben eine Abweichung besteht. Getestet wurde diesmal auf dem Flowers102 Datensatz nur mit der Ausschnittsgröße 8 und mit moderater beziehungsweise starker Augmentierung. Hierbei hat sich gezeigt, dass eine starke

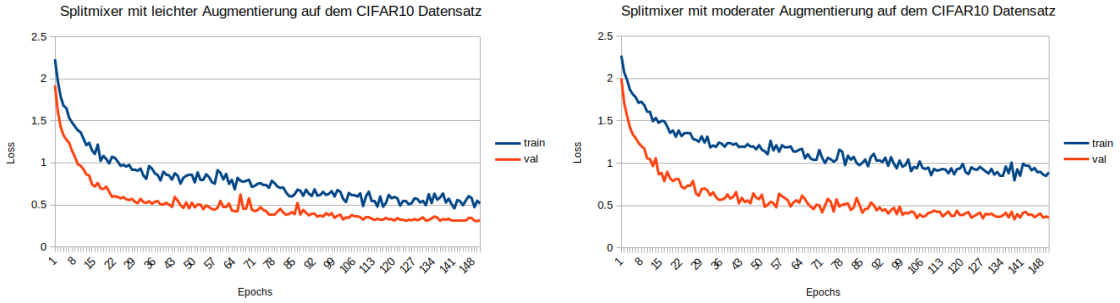


Abbildung 5.1.: Trainsverlauf von Splitmixer mit leichter(links) und moderater(rechts) Augmentierung auf dem CIFAR10 Datensatz

Augmentierung zu besseren Ergebnissen führt. Die Genauigkeit lag mit 62, 08% über der moderaten Augmentierung(61, 33%).

Betracht man in Abbildung 5.2 die Schaubilder der Trainingsverläufe wird deutlich, dass aufgrund der kleineren Datensatzgröße eine stärkere Augmentierung von Vorteil sein kann und zu gleichmäßigeren Verläufen führt. Die Intensität wird daher bei der Basis für den Flowers102 und Pets Datensatz von 15 auf 20 und der Alphawert von 0,5 auf 0,8 erhöht.

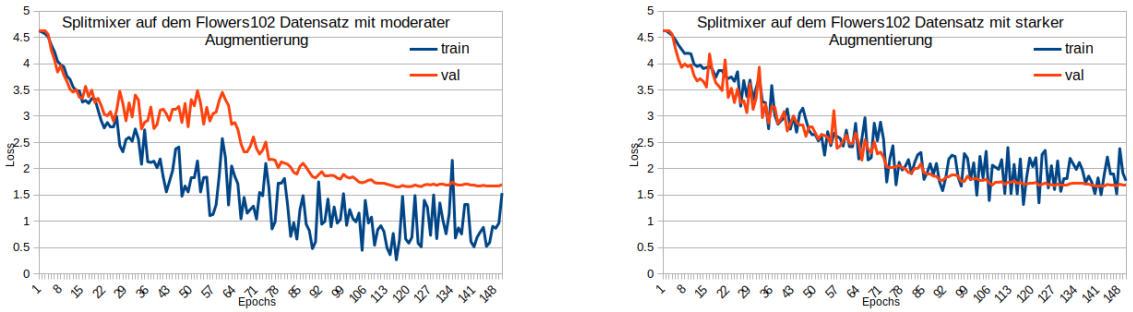


Abbildung 5.2.: Trainsverlauf von Splitmixer mit moderater(links) und starker(rechts) Augmentierung auf dem Flowers102 Datensatz

Touvron et al. argumentieren, dass neben der Lernrate, die stochastische Tiefe einer der wichtigsten Hyperparameter ist, der optimiert werden sollte[TCE+21]. Deswegen soll eine zweite Vorabevaluierung durchgeführt werden, bei denen die beiden Parameter Lernrate und DropPath evaluiert werden sollen. Hierbei wurden 2 Modelle gewählt und auf einem der drei Datensätze mit unterschiedlichen Werten für die Parameter Lernrate und DropPath trainiert.

Die Ergebnisse können aus Tabelle 5.2 entnommen werden. Beide Modelle erreichten die höchste Genauigkeit mit einer Lernrate von 0,001 und einen DropPath Wert von 0.2. Diese Werte wurden entsprechend für die Basiskonfiguration verwendet.

Model	Lernrate	DropPath		
		0.05	0.1	0.2
WaveMLP	0.01	27.82	26.57	27.36
WaveMLP	0.001	40.3	40.05	40.49
CaiT	0.01	31.77	32.41	32.62
CaiT	0.001	33.51	33.12	33.77

Tabelle 5.2.: Genauigkeit der Modelle WaveMLP und CaiT auf dem Flowers102 Datensatz mit unterschieden Werten für Lernrate und DropPath. Das jeweils beste Ergebnis ist hervorgehoben

5.1.2. weitere Tests und Erkenntnisse

Eine weitere Erkenntnis ist während der Evaluierung auf dem Oxford Pets Datensatz aufgekommen. Während die Durchläufe für CIFAR10 und Flowers102 abgeschlossen waren, ist aufgefallen, dass die Ergebnisse von zuvor durchgeführten Tests abweichen auf dem Oxford Pets Datensatz. Als ein möglicher Grund erschien die Augmentierung zu sein, da in früheren Tests teilweise andere Werte verwendet wurden. Um diesem Problem nachzugehen wurden sicherheitshalber erneut alle Modelle mit allen drei Parameteroptionen auf den drei Datensätzen evaluiert.

Die Ergebnisse sind in Tabelle 5.3 dargestellt. Der jeweils beste Wert ist unterstrichen. Bei CIFAR10 entspricht dies der mittleren Augmentierung für alle fünf Modelle.
Auf dem FLowers102 Datensatz schwankt es je nach Model. Die Unterschiede sind jedoch minimal, sodass auf einen erneuten Durchlauf verzichtet wurde, da die Evaluierung für diesen Datensatz bereits vollständig abgeschlossen war.
Bei dem Oxford Pets Datensatz konnte festgestellt werden, dass zwei der fünf Modelle unter starker Augmentierung merkbar an Genauigkeit verlieren. In diesem Fall schien es sinnvoll die Evaluierung nicht weiterzuführen, sondern neu zu starten. Eine leichte Augmentierung scheint besonders Splitmixer negativ zu beeinflussen. Als Basis wurde daher wie bei dem CIFAR10 Datensatz eine mittlere Datenaugmentierung gewählt.

	Augmentierung		Model				
Datensatz	Werte	Bezeichnung	Splitmixer	WaveMLP	CaiT	MLPMixer	ConvNeXt
CIFAR10	2, 10, 0.2	leicht	90.5	88.68	89.80	85.57	93.25
	2, 15, 0.5	mittel	91.41	89.88	90.14	87.3	94.30
	2, 20, 0.8	schwer	90.67	89.02	88.59	86.89	94.29
Flowers102	2, 10, 0.2	leicht	47.5	41.14	35.66	40.26	37.98
	2, 15, 0.5	mittel	51.01	42.49	34.83	39.92	39.84
	2, 20, 0.8	schwer	48.38	40.49	33.77	39.22	39.2
Oxford Pets	2, 10, 0.2	leicht	**38.29**	31.64	24.23	26.92	29.24
	2, 15, 0.5	mittel	45.73	31.97	19.1	30.25	25.15
	2, 20, 0.8	schwer	47.09	31.83	20.03	**27.06**	**20.74**

Tabelle 5.3.: Erneute Evaluierung aller Modelle auf den Datensätzen CIFAR10, Flowers102 und Oxford Pets mit unterschiedlich starken Augmentierung. Als Ausschnittsgröße wurde 4 für CIFAR10 und 16 für die beiden anderen Datensätze verwendet. Das beste Ergebnis ist unterstrichen, auffallende Abweichung sind vervorgehoben.

Die finalen Hyperparameterwerte können aus Tabelle 5.4 und Tabelle 5.5 entnommen werden. Die Werte für Modelltiefe und versteckte Dimensionsgröße entsprechen den Werten, die von den jeweiligen Autoren vordefiniert wurden. Die restlichen Parameter wurden eben erläutert. Der Basiswert ist jeweils unterstrichen. Dieser wird für alle Durchläufe verwendet, sofern es nicht der aktive Parameter ist, bei dem die Werte variiert werden. Splitmixer verwendet ursprünglich 100 Epochen [BL22] für den Trainingsprozess. Lv et al. hingegen 200 Epochen [LBW22]. Beyer et al. untersuchen den Einfluss von unterschiedlichen Änderungen mit unterschiedlicher Epochenanzahl. Hierfür wurde 90, 150 und 300 Epochen werden [BZK22]. Für die Evaluierung wurde sich entsprechend auch für 150 Epochen als Mittelwert entschieden.

CIFAR10				Pets/Flowers102				
				Pets	Flowers102			
Datenaugmentierung	Stapelgröße	Bildauflösung	Ausschnittsgröße	Datenaugmentierung	Datenaugmentierung	Stapelgröße	Bildauflösung	Ausschnittsgröße
2, 10, 0.2	128	24	2	2, 10, 0.2	2, 10, 0.2	32	128	8
2, 15, 0.5	256	32	4	2, 15, 0.5	2, 15, 0.5	64	244	16
2, 20, 0.8	512	40	8	2, 20, 0.8	2, 20, 0.8	128	288	32
-	1024	48	-	-	-	256	384	-

Tabelle 5.4.: Finale Werte für die Parameter Datenaugmentierung, Stapelgröße, Bildauflösung und Ausschnittsgröße, abhängig vom jeweiligen Datensatz

Model	Parameter			
	Versteckte Dimension	Modelltiefe	Lernrate	Regularisierung
splitmixer	64	4	0.0001	-
	128	8	0.001	
	256	12	0.01	
	512	16	-	
WaveMLP	[24, 48, 120, 192]	[2, 2, 2, 2]	0.0001	0.05
	[32, 64, 160, 256]	[2, 2, 4, 2]	0.001	0.1
	[48, 96, 240, 384]	[2, 3, 6, 3]	0.01	0.2
	[64, 128, 320, 512]	[2, 3, 10, 3]	-	0.3
CaiT	96	6	0.0001	0.05
	128	12	0.001	0.1
	192	18	0.01	0.2
	288	24	-	0.3
MLP-Mixer	128	4	0.0001	0.05
	256	6	0.001	0.1
	512	8	0.01	0.2
	1024	10	-	0.3
ConvNext	96	6	0.0001	0.05
	192	12	0.001	0.1
	384	18	0.01	0.2
	768	24	-	0.3

Tabelle 5.5.: Finale Werte aller Modelle für die Parameter versteckte Dimension, Modelltiefe, Lernrate und Regularisierung

Während der Auswertung der Ergebnisse konnten inkonsistente Beobachtungen beim Speicherbedarf gemacht werden. Als mögliche Ursache hat sich dabei die cuDNN Bibliothek herausgestellt. Standardmäßig ist der Wert $cudnn.benchmark$ auf Wahr gesetzt. Hierbei wird ein Optimierungsarlgorithmus verwendet, der zu Beginn jedes Trainings den am besten geeigneten Kernel sucht. Dabei scheint es jedoch zu deutlichen Messspitzen zu kommen.

In Abbildung 5.3 kann das Verhältnis zwischen dem gemessenen Höchstwert und dem Trainingsdurchschnitt gesehen werden. In einigen Durchläufen konnte ein deutlich erhöhter Höchstwert festgestellt werden. Aufgrund des Optimierungsalgorithmus durch die cuDNN Bibliothek, wird hierbei zu Beginn mehr Speicher belegt, als es vom Training verlangt wird. Dies hatte zur Folge, dass der Messwert nach der ersten Epoche teilweise deutlich höher ist. Die scheint vor allem WaveMLP regelmäßig zu betreffen. Die stärksten Abweichungen konnten jedoch bei MLP-Mixer notiert werden.

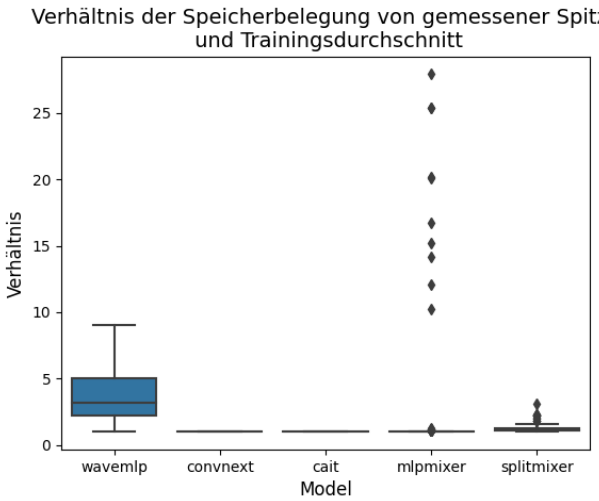


Abbildung 5.3.: Boxplot-Diagramm für das Verhältnis zwischen gemessener Spitze und Durchschnitt für die Speicherbelegung

Hier lag der höchste Wert des Verhältnisses bei ca 27. Das bedeutet, dass für die Optimierung in dieser Konfiguration 27-mal mehr Speicherplatz belegt wurde, als für das eigentliche Training des Modells erforderlich gewesen wäre.

5.2. Ergebnisse

In diesem Abschnitt werden die finalen Ergebnisse präsentiert und analysiert. Für jeden Datensatz wurden neben der Ausschnittsgröße, sieben Parameter untersucht. Für ein besseres Verständnis soll zunächst ein Datensatz im Detail betrachtet werden. Hierfür wurde sich für den CIFAR10 Datensatz entschieden.

5.2.1. Detaillierte Auswertung anhand des CIFAR10 Datensatzes

Daten Augmentierung

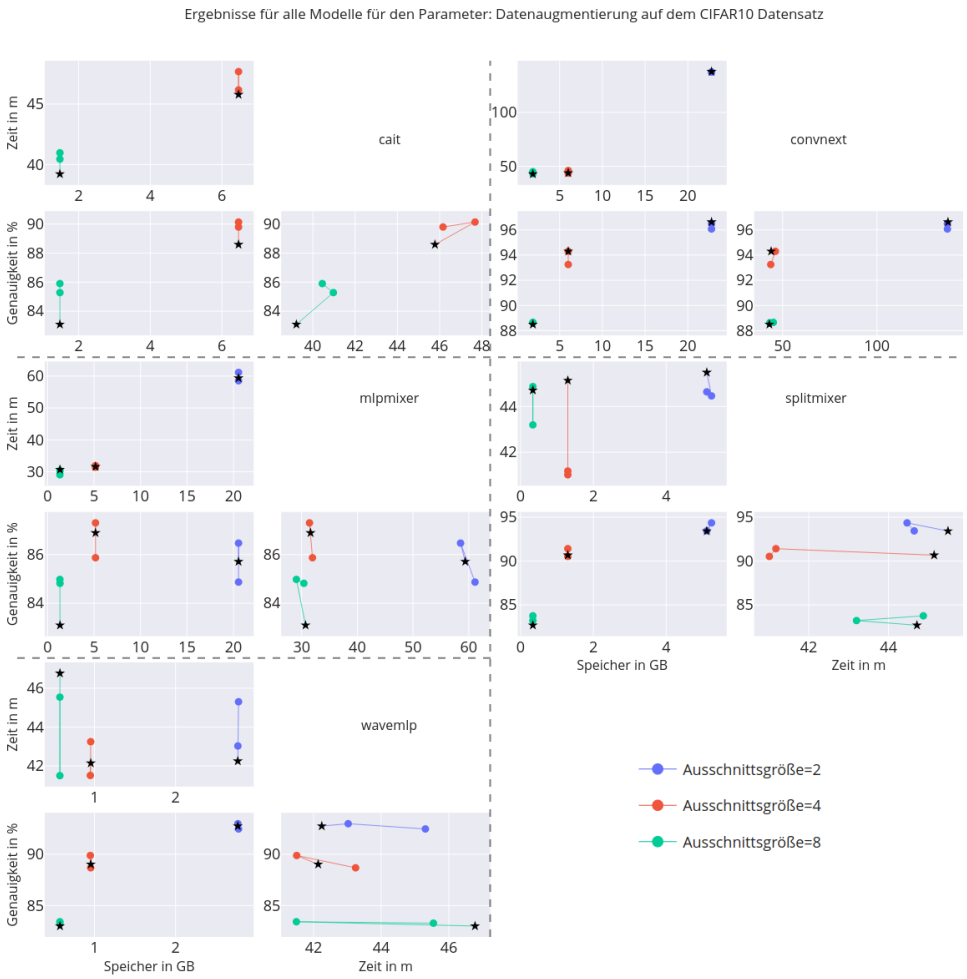


Abbildung 5.4.: Einfluss von unterschiedlicher Augmentierung auf alle Modelle für den CIFAR10 Datensatz.

Abbildung 5.4 zeigt den Einfluss unterschiedlich starker Augmentierungen für alle Modelle auf dem CIFAR10 Datensatz. Als erneuter Hinweis: der schwarze Stern markiert den größen Wert in der entsprechenden Reihe. Im Falle der Augmentierung entspricht dies der starken Augmentierung,

da hier von leicht nach stark sortiert wurde. Zudem sollte beachtet werden, dass jedes Modell eine eigene Achsenskalierung besitzt, um die Verläufe der Datenpunkte besser betrachten zu können. Auffällig ist hierbei, dass es unterschiedlich großen Einfluss bei den Laufzeiten kommt.

Stapelgröße

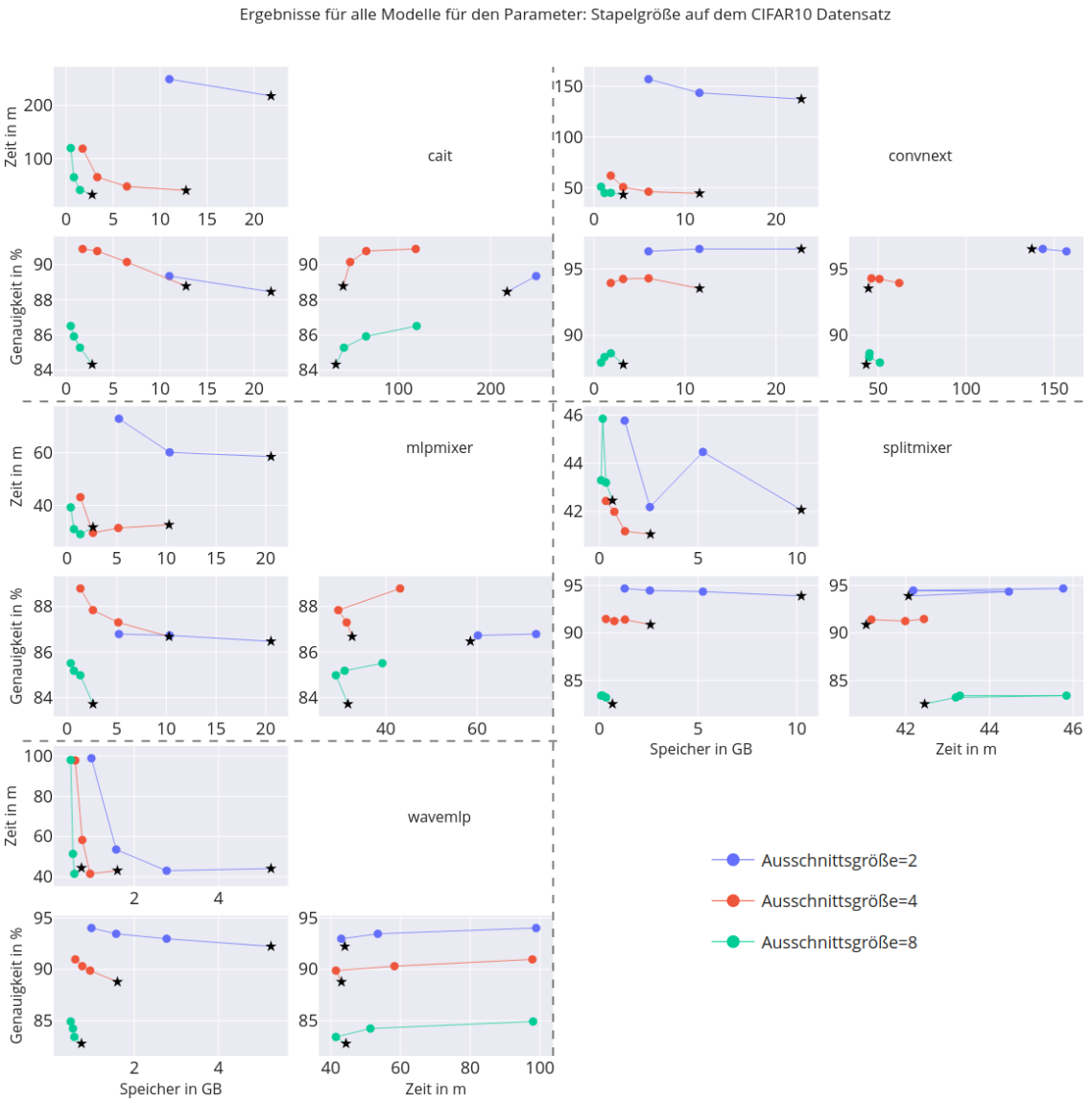


Abbildung 5.5.: Einfluss von unterschiedlicher Batch-Größe auf alle Modelle für den CIFAR10 Datensatz.

In Abbildung 5.5 lässt sich erkennen, dass mit steigender Stapelgröße, der Speicherbedarf steigt und die Laufzeiten sinken. Aufällig ist hier bei dem Modell Splitmixer ein Sprung nach oben bei Ausschnittsgröße zwei. Zur Überprüfung wurde ein separater Testlauf mit 10 Epochen gestartet und anschließend die Laufzeit auf 150 Epochen skaliert. Dabei konnte bestätigt werden, dass

Stapelgröße 512 länger braucht als mit den Werten 256 und 1024. Unklar ist wodurch dieser Effekt ausgelöst wird.
Weiterhin kann beobachtet werden, dass die Verläufe mit unterschiedlichen Ausschnittsgrößen ähnlich sind, es jedoch Unterschiede im direkten Vergleich mit anderen Modellen gibt.

Bildauflösung

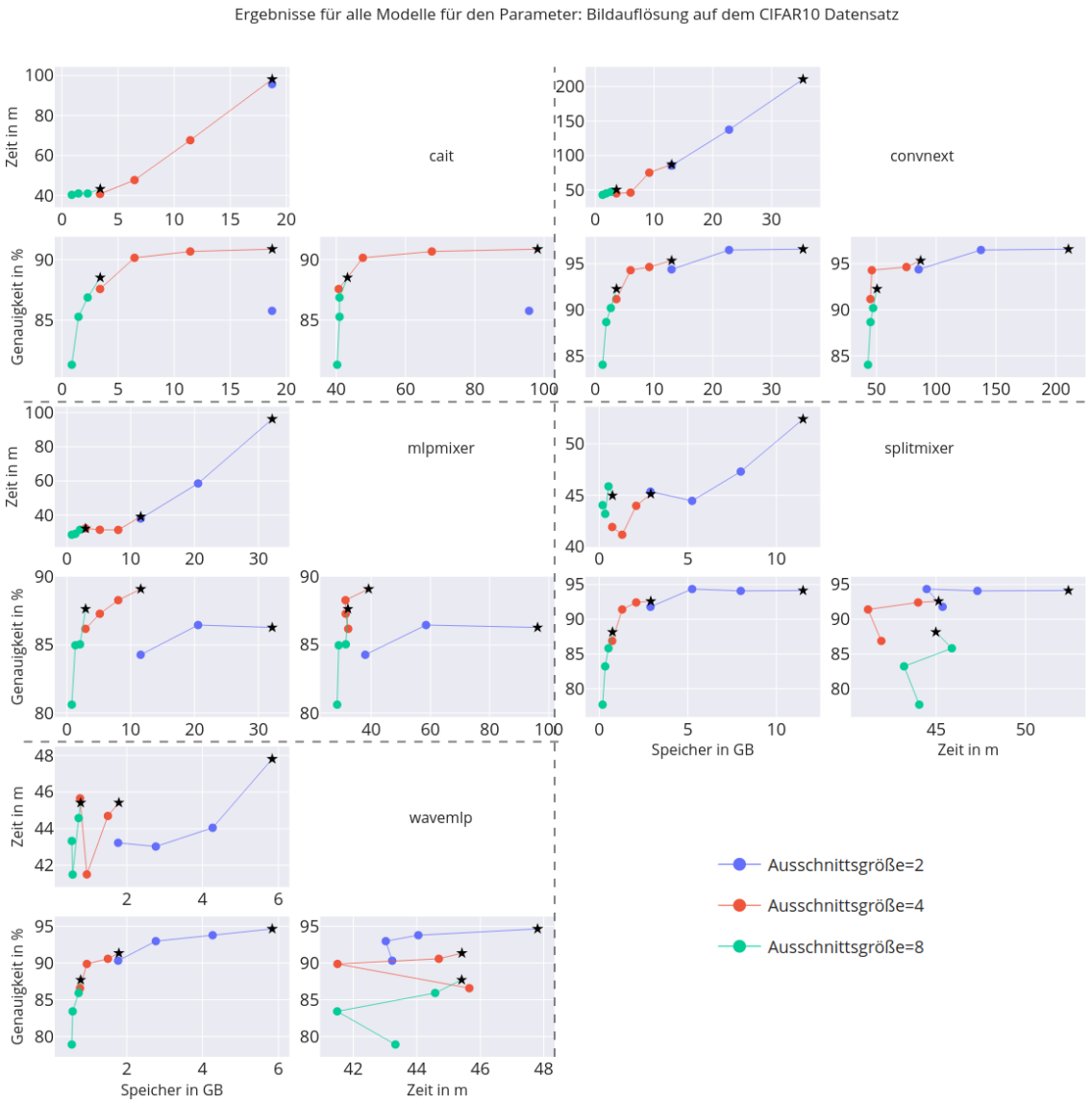


Abbildung 5.6.: Einfluss von unterschiedlicher Bildauflösung auf alle Modelle für den CIFAR10 Datensatz.

Beim Betrachten der Bildauflösung in Abbildung 5.6 ist zu erkennen, dass sich in allen Fällen eine verbesserte Genauigkeit mit steigender Bildauflösung zeigt. Mit einer höheren Auflösung steigt die auch Anzahl der Bildausschnitte an, folglich erhöhen sich auch Laufzeit und Speicherverbrauch.

Auffällig sind hier die Laufzeiten von WaveMLP. Auch hier wurde mit einem Testdurchlauf überprüft, ob sich die Ergebnisse bestätigen lassen. Geprüft wurde mit der Ausschnittsgröße 4, da hier der Effekt am stärksten zu sein scheint. Hierbei lag die Laufzeit mit geringster Auflösung (24) zwischen den Werten 40 und 48, welches den höchsten Wert darstellt. Eine gewisse Ungenauigkeit lässt sich durch die geringere Anzahl an Epochen erklären, jedoch zeigte sich auch hier dasselbe Muster.

Modeltiefe

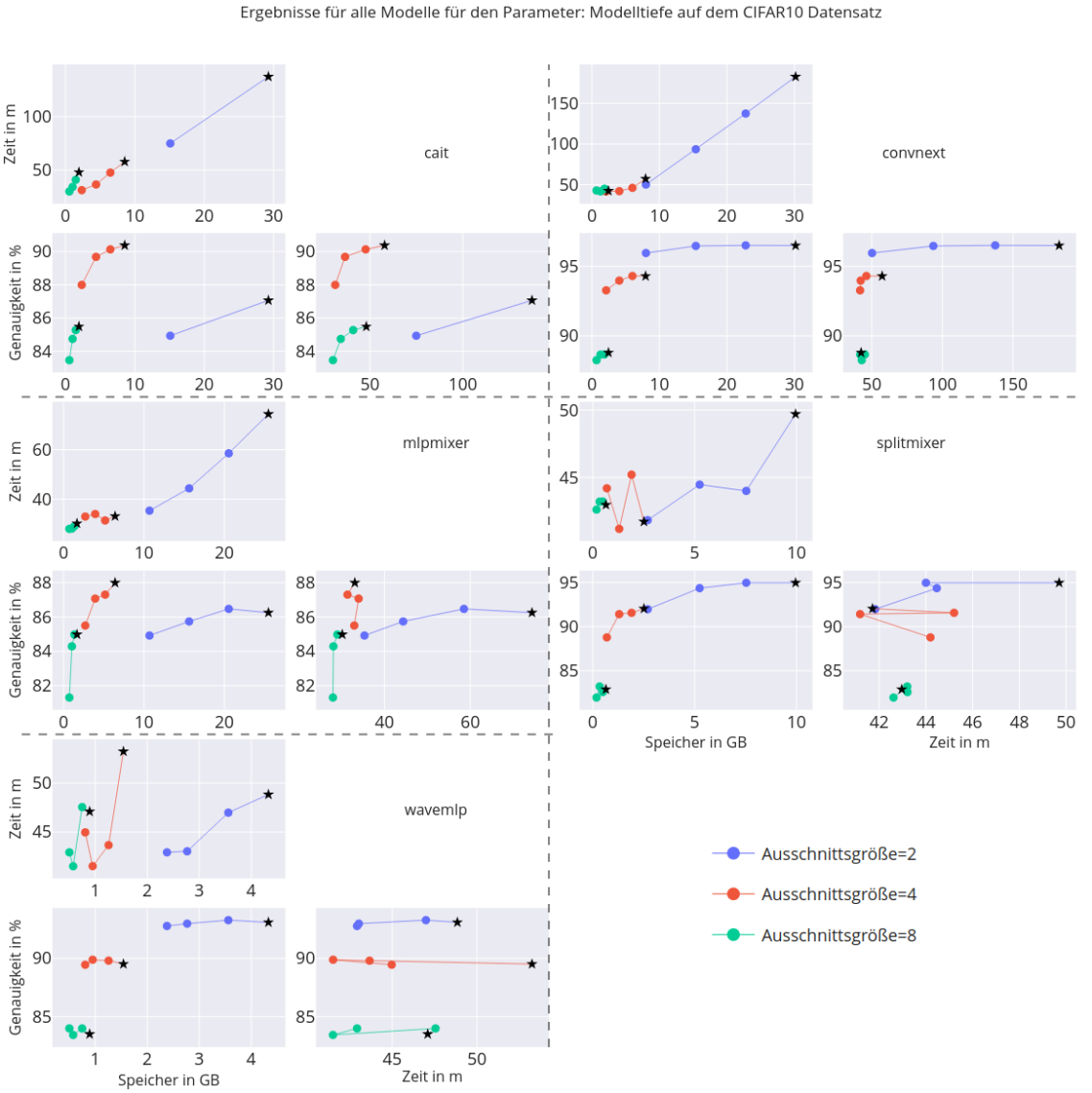


Abbildung 5.7.: Einfluss von unterschiedlicher Modeltiefe auf alle Modelle für den CIFAR10 Datensatz.

Die Analyse von Abbildung 5.7 zeigt, dass sich mit Ausschnittsgröße 2 die Genauigkeit am wenigsten verbessern lässt. Zudem wirkt sich die Ausschnittsgröße unterschiedlich auf die einzelnen Modelle aus. So steigt bei MLP-Mixer die Laufzeit vor allem mit kleinster Ausschnittsgröße, während es bei WaveMLP der mittlere Wert mit größtem Einfluss ist. Auffällig ist auch hier erneut die Laufzeit für Splitmixer.

Versteckte Dimension

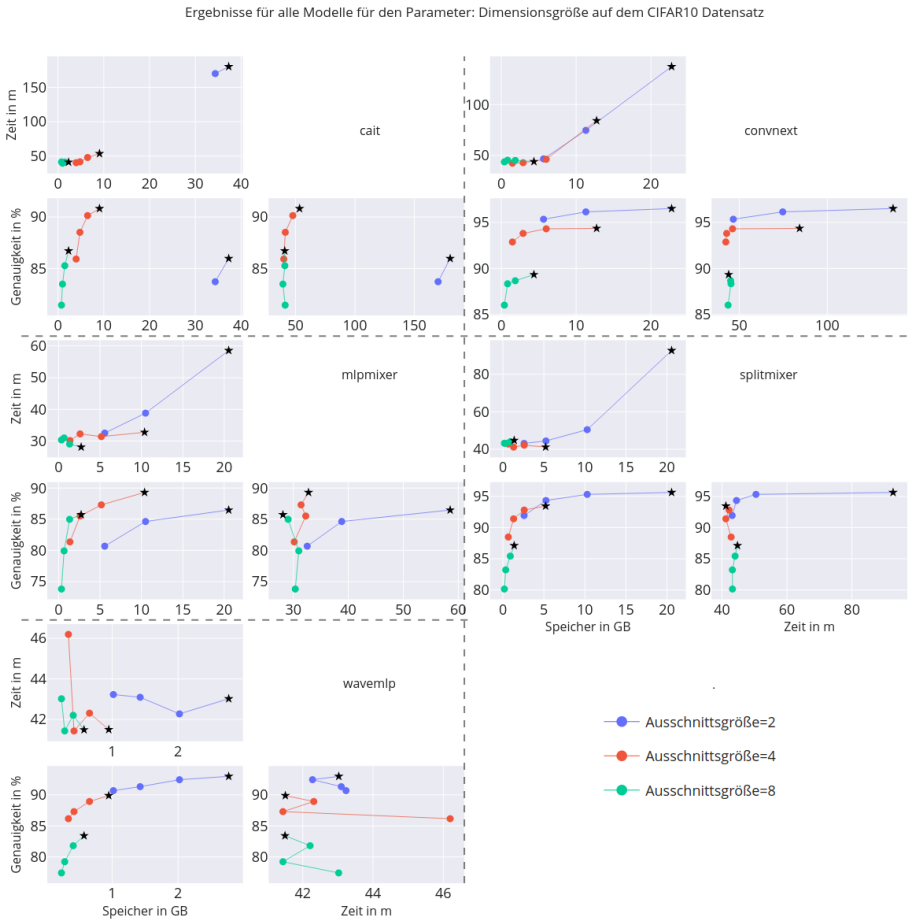


Abbildung 5.8.: Einfluss von unterschiedlichen Werten für die versteckte Dimension auf allen Modellen für den CIFAR10 Datensatz.

Die Versteckte Dimension ist neben der Modelltiefe, der zweite Hyperparameter, der Einfluss auf die Größe des Modells und der trainierbaren Parameter hat. Entsprechend sehen die Verläufe in Abbildung 5.8 ähnlich wie in Abbildung 5.7 aus. Festgehalten werden kann, dass dieser Hyperparameter einen größeren Einfluss auf die Genauigkeit zu scheinen hat als die Modelltiefe. Auffällig sind hier die Werte von WaveMLP. Hierbei erscheint besonders mit dem kleinsten Wert die Laufzeit bei allen drei Ausschnittsgrößen am höchsten. Auch hier wurde Ausschnittsgröße 4 stichprobenartig mit größtem und kleinsten Wert überprüft. In dem Test mit 10 Epochen weichen die Ergebnisse dieses mal leicht ab. So lag die Laufzeit für den geringsten Wert der versteckten Dimension unter dem der Evaluierung und die Laufzeit des größten Wertes darüber. Insgesamt

lagen die Werte etwa gleich auf (5% Unterschied). Unklar ist wieso die Ergebnisse abweichen. Ein Faktor könnten technische Probleme, wie die Überlastung des Servers entweder während der Evaluierungszeit oder während des 10-Epochen Tests gewesen sein.

Lernrate

Abbildung 5.9.: Einfluss von unterschiedlichen Werten für die Lernrate auf allen Modellen für den CIFAR10 Datensatz.

Abbildung 5.9 veranschaulicht den Einfluss verschiedener Lernraten auf die Ergebnisse der Modelle. Diese scheint sich nicht auf den Speicherbedarf auszuwirken. Auch bei der Laufzeit scheint sich nur einen Einfluss auf den beiden Modellen WaveMLP und Splitmixer bemerkbar zu machen.

Regularisierung

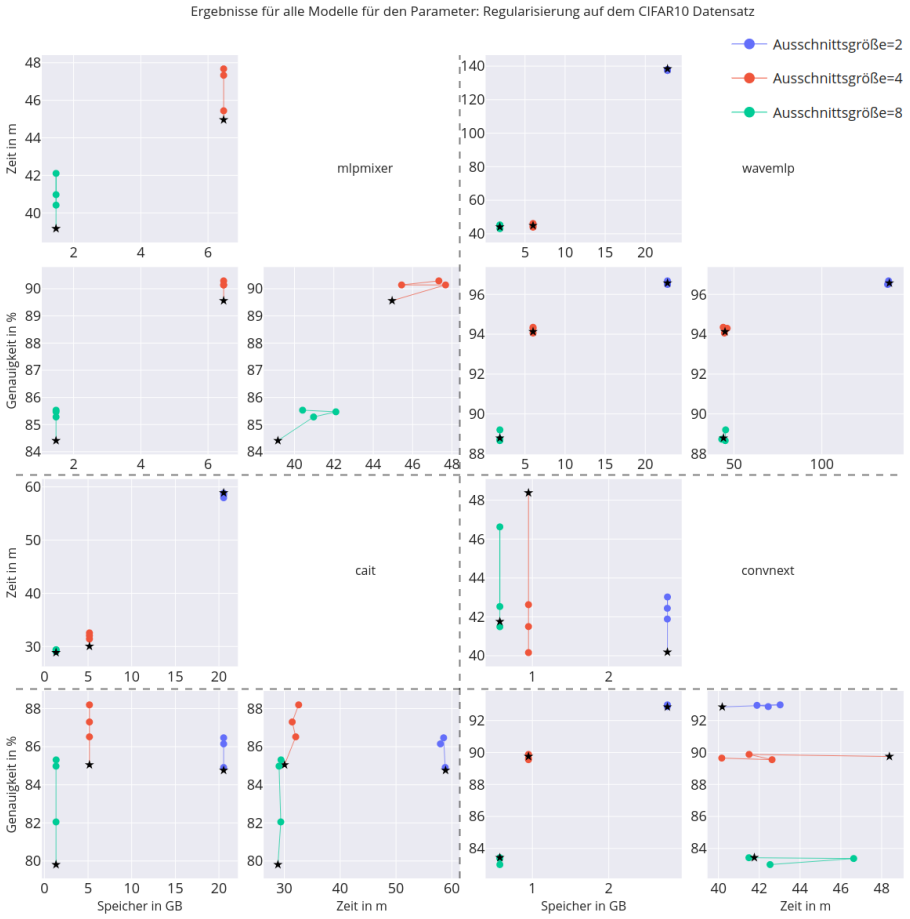


Abbildung 5.10.: Einfluss von unterschiedlichen Werten für die Lernrate auf allen Modellen für den CIFAR10 Datensatz.

Für die Regularisierung nutzt MLP-Mixer das DropOut-Verfahren, alle anderen in Abbildung 5.10 zu sehenden Modelle benutzen DropPath. Splitmixer verwendet standardmäßig keines davon, daher wurde es für diesen Teil ausgeschlossenen. Während DropPath bei WaveMLP kaum bemerkbare Auswirkungen hat, scheint der Effekt bei CaiT am stärksten bei der Genauigkeit zu sein. Im Gegensatz dazu ändert sich vorallem bei ConvNeXt die Laufzeit.

Ergebnisse zu FLowers102, Oxford Pets und Stanford Cars

Die Schaubilder zu diesen beiden Datensätzen sind an sich recht ähnlich, unterschieden sich jedoch teilweise zu den vorgestellten Ergebnissen für den CIFAR10 Datensatz. Schaubilder für den Pets Datensatz werden dem Anhang beigefügt. Interessant ist es zu beobachten, dass manche der im vorherigen Abschnitt vorgestellten Abweichungen, wie beispielsweise WaveMLP bei der Bildauflösung in Abbildung 5.6, nicht zu beobachten sind. Dafür jedoch Schwankungen an anderen Stellen zu beobachten sind.

Weiterhin fällt die Genauigkeit für ConvNeXt bei großer Modelltiefe oder versteckter Dimension auf unter 10%. Die Ursache dafür ist unklar, jedoch konnte im Protokoll beobachtet werden, dass ab einem bestimmten Punkt nur noch der Wert NaN für den Verlust zurückgegeben wird und das Modell mit diesem Wert entsprechend Probleme hat seine Genauigkeit zu erhöhen.
Aufgrund von technischen Problemen zum Ende der Bearbeitungszeit konnte der Stanford Cars Datensatz nicht vollständig abgearbeitet werden, daher wurde dieser nicht ausgewertet.

5.2.2. Verallgemeinerung der Ergebnisse

Algorithmus 5.1 Berechnung des Durchschnitts für alle Kombinationen an Modell, Datensatz, Parameter und Ausschnittsgröße (hier als patch size benannt)

function COMPUTE_MEAN($datasets$, $models$, $params$, $patch\_sizes$)
  for each $dataset$ in $datasets$ **do**
    for each $model$ in $models$ **do**
      for each $parameter$ in $parameters$ **do**
        for each $patchsize$ in $patchsizes$ **do**
          $S \leftarrow \sum_{i=1}^{N-1} \frac{x_{i+1}}{x_i}$
          $\overline{S} \leftarrow \frac{S}{N-1}$
        end for
      end for
    end for
  end for
end function

Um die Wechselwirkung zwischen Parameter und Ausschnittsgröße besser einstufen zu können, werden die Verläufe der eben gezeigten Ergebnisse zu einem Wert je Ausschnittsgröße zusammengefasst. Der Ansatz ist in Algorithmus 5.1 zu sehen. Die Variable $patch\_size$ entspricht dabei der Ausschnittsgröße. Hierbei wird jeder Eintrag durch den Vorherigen geteilt, um den Anteil der Veränderung festzuhalten. Bei 4 Parameterwerten ergeben sich daraus 3 anteilige Veränderungen. Diese werden anschließend zu einem Durchschnitt zusammengefasst. Geht man beispielsweise davon aus, dass eine Verdoppelung des Wertes für die Stapelgröße in jedem Schritt einen etwa doppelt so hohen Speicherbedarf hat, so sollte auch der berechnete Wert etwa dem Faktor 2 entsprechen.

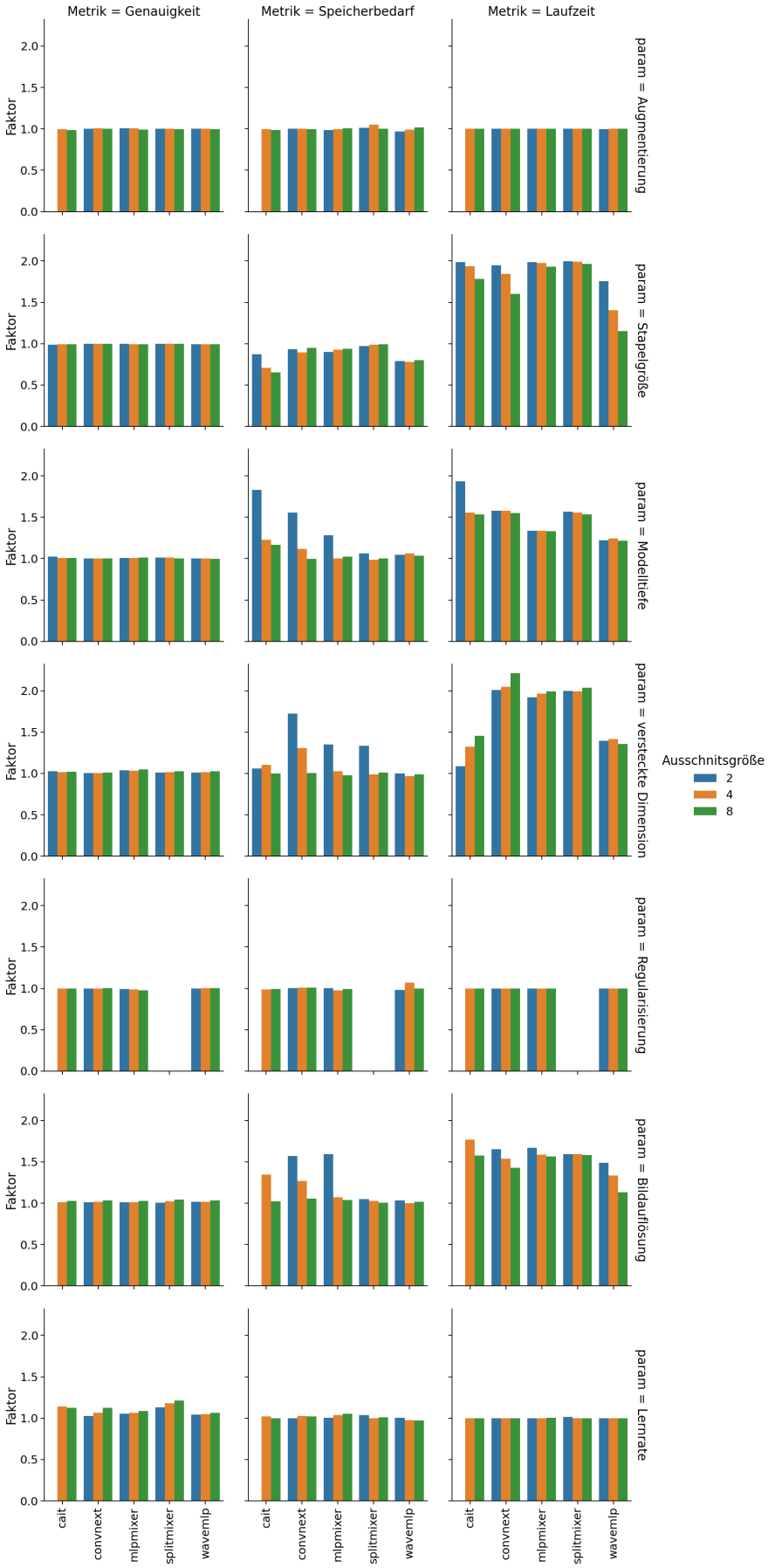


Abbildung 5.11.: Berechnung eines Durchschnitts für alle Parameter mit allen Modellen und Ausschnittsgrößen auf dem CIFAR10 Datensatz.

In Abbildung 5.11 sind die zusammengefassten Ergebnisse zu sehen. Vergleicht man jeweils die drei farblichen Balken für jedes Modell lässt sich feststellen, dass besonders die Lernrate die Genauigkeit mit unterschiedlichen Ausschnittsgrößen beeinflusst. So ist bis auf CaiT stets das Muster zu erkennen, dass größere Ausschnitte mehr von einer Änderung der Lernrate beeinflusst wurden. Auch bei der Bildaufösung, scheint besonders Ausschnittsgröße 8 eine etwas stärkere Abweichung zu verzeichnen. Deutlicher fallen die Unterschiede bei der Laufzeit und Speicherbedarf aus. So kann beobachtet werden, dass der Einfluss der Stapelgröße auf den Speicherbedarf mit steigender Ausschnittsgröße abnimmt. Besonders die Modelltiefe und Größe der versteckten Dimension scheinen abhängig von der Ausschnittsgröße einen unterschiedlichen starken Effekt zu haben. Im Gegensatz dazu hat die versteckte Dimension beim Speicherbedarf den gegenteiligen Effekt. Hier scheint der meiste Einfluss bei Ausschnittsgröße 8 zu liegen. Auch bei der Stapelgröße und Bildauflösung lässt sich erkennen, dass der Einfluss bei kleineren Größen stärker ausfällt.
Die Schaubilder zu dem Oxford Pets und Flowers Datensatz können im Anhang gefunden werden. Wie auch bei der detaillierten Betrachtung fallen diese ähnlich aus, haben jedoch einige Unterschiede zu dem CIFAR10 Datensatz. Besonders der Einfluss von Modelltiefe und versteckter Dimension fällt hier deutlich geringer aus. Im Gegensatz dazu ist der Effekt für die Genauigkeit bei den Parametern Lernrate und Stapelgröße höher. In vielen Fällen scheinen die Einflüsse je nach Modell zu schwanken und vor allem auch unterschiedlich auszufallen, sodass sich nur schwierig Aussagen treffen lassen können. So lässt sich beispielsweise bei der versteckten Dimension in beiden Schaubildern erkennen, dass bei Splitmixer und WaveMLP der Einfluss mit fallen Ausschnittsgröße zunimmt. Bei MLP-Mixer liegt wiederum die größe Abweichung bei mittlerer Größe und bei ConvNeXt und CaiT bei größter Ausschnittsgröße.

Unabhängig der Ergebnisse lässt sich weiterhin festhalten, dass die Ergebnisse für Splitmixer mit denen der Autoren übereinstimmen. Bei CIFAR10 konnte eine leichte Steigerung beobachtet werden (+1%), bei Flowers102 eine leichter Fall(-2%), welches sich jedoch aufgrund von unterschiedlichen Ausschnittsgrößen erklären lassen könnte (7 statt 8).Weiterhin fallen die Ergebnisse auch ähnlich mit denen von MDMLP aus. In dieser Durchführung waren die Genauigkeiten etwas höher bei CIFAR10, dafür jedoch fallen besonders die ViT und MLP Modelle schlechter aus als bei der Untersuchung von MDMLP. Unterschiede die zu diesen Abweichungen beigetragen haben könnten, sind unter anderem unterschiedliche Ansätze bei der Augmentierung und Regularisierung, oder aufgrund eines anderen Optimierers mit anderer Lernratenplanung.

6. Zusammenfassung und Ausblick

6.1. Zusammenfassung

Es wurde ein Framework entwickelt, welches die Evaluierung von einzelnen Parametern mit unterschiedlichen Ausschnittsgrößen ermöglicht. Für einen möglichst hohen Trainingsdurchsatz wurde ein automatisiertes Verwaltungssystem entwickelt, welches mithilfe des Slurm Systems eine hohe GPU Ausschöpfung ermöglicht.
Zusätzlich wurde eine GUI entwickelt die zur Unterstützung bei der Erstellung von Trainingskonfigurationen, Verwaltung von Ergebnissen und deren visuellen Darstellung dient. Es wurden insgesamt fünf Modelle auf mehreren Datenbanken trainiert. Dabei untersucht wurden abhängig von der Ausgangsauflösung des Datensatzes drei verschiedene Bildausschnitte, mit insgesamt sieben verschiedenen Hyperparametern, welche jeweils drei bis vier Werte zugewiesen wurden. Aufgrund von vorhandener Literatur wurde sich gegen Transfer-Lernen entschieden. Es wurde in Kauf genommen, dass die zu erwartenden Ergebnisse entsprechend schlechter ausfallen, dafür aber eine fairere Evaluierung auf unterschiedlichen Ausschnittsgrößen erlaubt.
Aus den Ergebnissen lässt sich mitnehmen, dass eine simple Übernahme von Parameterwerten keine solide Grundlage für eine Auswertung dient. Basierend aus den initialen Erkenntnissen wurden mehrere Vorabevaluierungen durchgeführt, um im Rahmen des Möglichen eine solidere Grundlage zu schaffen. Aufgrund der Breite der Analyse war es jedoch zeitlich nicht möglich die Hyperparameter der Modelle für alle Konstellationen feiner abzustimmen. So empfehlen Godbole et al. bereits beim Ändern der Stapelgröße eine komplette, neue Optimierung der restlichen Hyperparameter [GDG+21].
Zudem konnte gelernt werden, dass die Speicherbelegungsspitzen durch des standardmäßig aktiven *cudnn.benchmark* Moduls, fälschlicherweise überhöhte Werte erfasst werden. Hierbei wird während der Suche nach dem besten Optimierungsalgorithmus mehr Speicher belegt, als von dem entsprechenden Model benötigt wird. Infolgedessen wurde die Evaluierung auf den durchschnittlichen Speicherbedarf nach der Initialisierung angepasst.

Die Ergebnisse wurden an einem Datensatz detaillierter vorgestellt und dabei besondere Abweichungen hervorgehoben, die berücksichtigt werden sollten. Diese wurden teilweise untersucht und konnten in den meisten Fällen bestätigt werden. Die gewählten Datensätze ermöglichen ein schnelles Training. Die Laufzeit pro Epoche lag hierbei teilweise bei unter 15 Sekunden. Da alle Durchläufe nur einmal getätigt wurden, ist es aber nicht auszuschließen, dass es vor allem bei der Laufzeit zu Schwankungen kommen kann.

Anschließend wurden die gemessenen Werte für jeden Parameter zu einem Wert zusammengefasst. Ziel ist es zu untersuchen, ob dieser Faktor von unterschiedlichen Ausschnittsgrößen unterschiedlich stark beeinflusst wird. So konnte beispielsweise beobachtet werden, dass sich Lernrate bei dem

CIFAR10 Datensatz stärker auf größere Ausschnittsgrößen auswirkt. Bei Oxford Pets und Flowers102 scheint es wiederum das genaue Gegenteil zu sein, hier fällt der Einfluss im Schnitt auf kleinere Ausschnitsgrößen größer aus. Aber auch innerhalb eines Datensatzes scheinen Abweichungen vorhanden zu sein. So ist es nur schwierig, allgemein geltende Aussagen zu treffen. Vielmehr konnte aufgezeigt werden, dass abhängig von der Ausschnittsgröße durchaus unterschiedliche Auswirkungen existieren, bei denen jedoch im Einzelfall unterschieden werden muss.

Limitierungen Aufgrund von zeitlichen Limitierungen konnte keine ausführliche Hyperparameteroptimierung für jedes Modell und jeden Datensatz durchgeführt werden. Die erfassten Ergebnisse wurden mit dem CIFAR10 Datensatz vorgestellt, da hier im Vergleich mit vorhandener Literatur gleiche, oder sogar bessere Ergebnisse bei der Genauigkeit erfasst werden konnten. Dennoch scheinen einzelne Werte teilweise unregelmäßige Ergebnisse zu erzeugen, welche jedoch unter Umständen Einfluss auf die Evaluierung haben.

Ausblick

Beyer et al. haben gezeigt, dass ViT nicht flexibel ist, um mit variierenden Ausschnittsgrößen umzugehen, die von der trainierten Ausschnittsgröße abweichen [BIK+22]. Aus diesem Grund wurde entschieden, kein Transfer-Lernen mit vortrainierten Modellen durchzuführen. Um eine SOTA Leistung zu erzielen, werden die meisten hier verwendeten Modelle auf großen Datensätzen wie Imagenet, welches 14 Millionen Bilder, oder JFT-300M, welches 300 Millionen Bilder enthält, trainiert. Durch Anwendung von Transfer-Lernen auf diese vortrainierten Modelle kann die Genauigkeit der Modelle bei den hier verwendeten Datensätzen mittels Feinabstimmung deutlich verbessert werden. Das Ziel dieser Arbeit ist es daher, den relativen Einfluss einzelner Parameter in Kombination mit unterschiedlichen Ausschnittsgrößen zu untersuchen. In der Praxis ist es jedoch empfehlenswert, vortrainierte Modelle zu verwenden und sie auf den entsprechenden Datensatz feinzustimmen. So erreicht beispielsweise CaiT einen Genauigkeit von 99.2% auf CIFAR10, oder 98.8% auf dem Flowers102 Datensatz mittels Transfer-Lernen [TCS+21].
Ein möglicher Ansatz wäre eine weiterführende Untersuchung, bei der man sich auf ein bis zwei Modelle beschränkt und diese auf dem ImageNet oder ähnlich großen Datenätzen mit unterschiedlichen Ausschnittsgrößen trainiert. Diese würde eine Grundlage für das Transfer-Lernen schaffen. Interessant wäre zu sehen, ob sich beim Transfer-Lernen besser Schlussfolgerungen ziehen lassen, besonders in Bezug auf die Allgemeingültigkeit, oder es hier ebenso auf den Einzelfall ankommt.
Eine weitere Möglichkeit wäre hier sich auf ein oder zwei Modelle zu fokussieren und die Hyperparameter ausführlicher zu optimieren. Mit einer besser optimierten Basis fallen die Ergebnisse unter Umständen anders aus. Interessant wäre zu sehen, ob die Einflüsse in diesem Fall schwächer, oder vielleicht sogar stärker ausfallen, da man sich mit dem Variieren eines Parameters von dem gefunden Optimum womöglich weiter wegbewegt, als es in dieser Evaluierung der Fall ist.

Literaturverzeichnis

[ABC+16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng. *TensorFlow: A system for large-scale machine learning*. 2016. arXiv: 1605.08695 [cs.DC] (zitiert auf S. 28).

[Bah23] P. Baheti. *transfer learning guide*. 2023. URL: https://www.v7labs.com/blog/transfer-learning-guide/ (zitiert auf S. 14).

[BIK+22] L. Beyer, P. Izmailov, A. Kolesnikov, M. Caron, S. Kornblith, X. Zhai, M. Minderer, M. Tschannen, I. Alabdulmohsin, F. Pavetic. *FlexiViT: One Model for All Patch Sizes*. 2022. arXiv: 2212.08013 [cs.CV] (zitiert auf S. 25, 56).

[BKH16] J. L. Ba, J. R. Kiros, G. E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML] (zitiert auf S. 17).

[BL22] A. Borji, S. Lin. *SplitMixer: Fat Trimmed From MLP-like Models*. 2022. arXiv: 2207.10255 [cs.CV] (zitiert auf S. 29, 38, 39, 42).

[Boe23] G. Boesch. *Vision Transformers (ViT) in Image Recognition – 2023 Guide*. https://viso.ai/deep-learning/vision-transformer-vit/. Accessed May 5, 2023. 2023 (zitiert auf S. 18).

[BZK22] L. Beyer, X. Zhai, A. Kolesnikov. *Better plain ViT baselines for ImageNet-1k*. 2022. arXiv: 2205.01580 [cs.CV] (zitiert auf S. 42).

[CVL22] S. Chhabra, H. Venkateswara, baoxin Li. „PatchSwap: A Regularization Technique for Vision Transformers“. In: *33rd British Machine Vision Conference 2022, BMVC 2022, London, UK, November 21-24, 2022*. BMVA Press, 2022. URL: https://bmvc2022.mpi-inf.mpg.de/0996.pdf (zitiert auf S. 26).

[CZSL20] E. D. Cubuk, B. Zoph, J. Shlens, Q. V. Le. „Randaugment: Practical automated data augmentation with a reduced search space“. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (Juni 2020). DOI: 10.1109/cvprw50498.2020.00359. URL: http://dx.doi.org/10.1109/CVPRW50498.2020.00359 (zitiert auf S. 21).

[DBK+20] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. arXiv: 2010.11929 [cs.CV] (zitiert auf S. 17).

[GDG+21] V. Godbole, G. E. Dahl, J. Gilmer, C. J. Shallue, Z. Nado. *Deep Learning Tuning Playbook*. https://github.com/google-research/tuning_playbook. 2021 (zitiert auf S. 26, 38, 55).

[Ger19] A. Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd. O'Reilly Media, Inc., 2019. ISBN: 1492032646 (zitiert auf S. 20).

[HG16] D. Hendrycks, K. Gimpel. *Gaussian Error Linear Units (GELUs)*. 2016. arXiv: 1606.08415 [cs.LG] (zitiert auf S. 16).

[HMN+20] F. Heyen, T. Munz, M. Neumann, D. Ortega, N. T. Vu, D. Weiskopf, M. Sedlmair. „ClaVis: An Interactive Visual Comparison System for Classifiers". In: *Proceedings of the International Conference on Advanced Visual Interfaces (AVI)*. AVI '20. ACM, 2020, 9:1–9:9. ISBN: 9781450375351. DOI: 10.1145/3399715.3399814. URL: https://doi.org/10.1145/3399715.3399814 (zitiert auf S. 26).

[HST+22] S. A. Hicks, I. Strümke, V. Thambawita, M. Hammou, M. A. Riegler, P. Halvorsen, S. Parasa. „On evaluation metrics for medical applications of artificial intelligence". In: *Scientific Reports* 12.1 (Apr. 2022), S. 5979. ISSN: 2045-2322. DOI: 10.1038/s41598-022-09954-8. URL: https://doi.org/10.1038/s41598-022-09954-8 (zitiert auf S. 22).

[IBM23] IBM. *Supervised Learning - IBM*. 2023. URL: https://www.ibm.com/topics/supervised-learning (zitiert auf S. 14).

[LBW22] T. Lv, C. Bai, C. Wang. *MDMLP: Image Classification from Scratch on Small Datasets with MLP*. 2022. arXiv: 2205.14477 [cs.CV] (zitiert auf S. 25, 28, 29, 38, 42).

[LH17] I. Loshchilov, F. Hutter. *Decoupled Weight Decay Regularization*. 2017. arXiv: 1711.05101 [cs.LG] (zitiert auf S. 20).

[Lip22] P. Lippe. *Transformers and Multi-Head Attention*. https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial6/Transformers_and_MHAttention.html. 2022 (zitiert auf S. 18).

[LMS17] G. Larsson, M. Maire, G. Shakhnarovich. *FractalNet: Ultra-Deep Neural Networks without Residuals*. 2017. arXiv: 1605.07648 [cs.CV] (zitiert auf S. 22).

[LMW+22] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, S. Xie. „A ConvNet for the 2020s". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (Juni 2022). DOI: 10.1109/cvpr52688.2022.01167. URL: http://dx.doi.org/10.1109/CVPR52688.2022.01167 (zitiert auf S. 28, 29).

[MP43] W. S. McCulloch, W. Pitts. „A logical calculus of the ideas immanent in nervous activity". In: *Bulletin of Mathematical Biophysics* 5.4 (1943), S. 115–133. DOI: 10.1007/BF02478259 (zitiert auf S. 14).

[Pat19] A. A. Patel. *Hands-On Unsupervised Learning Using Python: How to Build Applied Machine Learning Solutions from Unlabeled Data*. O'Reilly Media, 2019. ISBN: 1492035645 (zitiert auf S. 14).

[PGM+19a] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala. „PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural*

Information Processing Systems 32. Curran Associates, Inc., 2019, S. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (zitiert auf S. 28).

[PGM+19b] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG] (zitiert auf S. 28).

[Qay22] R. Qayyum. *Introduction To Pooling Layers In CNN*. 2022. URL: https://towardsai.net/p/l/introduction-to-pooling-layers-in-cnn (besucht am 05. 05. 2023) (zitiert auf S. 17).

[Ros58] F. Rosenblatt. „The perceptron: a probabilistic model for information storage and organization in the brain.“ In: *Psychological review* 65 6 (1958), S. 386–408 (zitiert auf S. 14).

[Ser] A. W. Services. *What is Overfitting?* URL: https://aws.amazon.com/de/what-is/overfitting/ (zitiert auf S. 21).

[SHK+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. „Dropout: A Simple Way to Prevent Neural Networks from Overfitting“. In: *Journal of Machine Learning Research* 15.56 (2014), S. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html (zitiert auf S. 22).

[SKZ+21] A. Steiner, A. Kolesnikov, X. Zhai, R. Wightman, J. Uszkoreit, L. Beyer. *How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers*. 2021. arXiv: 2106.10270 [cs.CV] (zitiert auf S. 25, 28, 39).

[SKZ+22] A. P. Steiner, A. Kolesnikov, X. Zhai, R. Wightman, J. Uszkoreit, L. Beyer. „How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers“. In: *Transactions on Machine Learning Research* (2022). ISSN: 2835-8856. URL: https://openreview.net/forum?id=4nPswr1KcP (zitiert auf S. 21).

[Sta] A. Startups. *Object Segmentation vs. Object Detection - Which one should you use?* https://www.augmentedstartups.com/blog/Object-detection-vs-Object-segmentation (zitiert auf S. 13).

[TCD+21] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, H. Jégou. *Training data-efficient image transformers & distillation through attention*. 2021. arXiv: 2012.12877 [cs.CV] (zitiert auf S. 39).

[TCE+21] H. Touvron, M. Cord, A. El-Nouby, P. Bojanowski, A. Joulin, G. Synnaeve, H. Jégou. *Augmenting Convolutional networks with attention-based aggregation*. 2021. arXiv: 2112.13692 [cs.CV] (zitiert auf S. 26, 37, 40).

[TCS+21] H. Touvron, M. Cord, A. Sablayrolles, G. Synnaeve, H. Jegou. „Going deeper with Image Transformers“. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (Okt. 2021). DOI: 10.1109/iccv48922.2021.00010. URL: http://dx.doi.org/10.1109/ICCV48922.2021.00010 (zitiert auf S. 29, 56).

[THG+22] Y. Tang, K. Han, J. Guo, C. Xu, Y. Li, C. Xu, Y. Wang. „An Image Patch is a Wave: Phase-Aware Vision MLP“. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (Juni 2022). DOI: 10.1109/cvpr52688.2022.01066. URL: http://dx.doi.org/10.1109/CVPR52688.2022.01066 (zitiert auf S. 29).

[THK+21] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, M. Lucic, A. Dosovitskiy. *MLP-Mixer: An all-MLP Architecture for Vision*. 2021. arXiv: 2105.01601 [cs.CV] (zitiert auf S. 29).

[TK22] A. Trockman, J. Z. Kolter. *Patches Are All You Need?* 2022. arXiv: 2201.09792 [cs.CV] (zitiert auf S. 29).

[Voh19] R. Vohra. *Convolutional Neural Networks*. 2019. URL: https://medium.datadriveninvestor.com/convolutional-neural-networks-3b241a5da51e (besucht am 23. 04. 2023) (zitiert auf S. 16).

[Woo] T. Wood. *Machine Learning Glossary and Terms: Softmax Layer*. https://deepai.org/machine-learning-glossary-and-terms/softmax-layer (zitiert auf S. 16).

[ZCDL17] H. Zhang, M. Cisse, Y. N. Dauphin, D. Lopez-Paz. *mixup: Beyond Empirical Risk Minimization*. 2017. arXiv: 1710.09412 [cs.LG] (zitiert auf S. 21).

[Zul18] H. Zulkifli. *Understanding Learning Rates and How It Improves Performance in Deep Learning*. https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10. 2018 (zitiert auf S. 19).

Alle URLs wurden zuletzt am 10. 05. 2023 geprüft.

A. Anhang

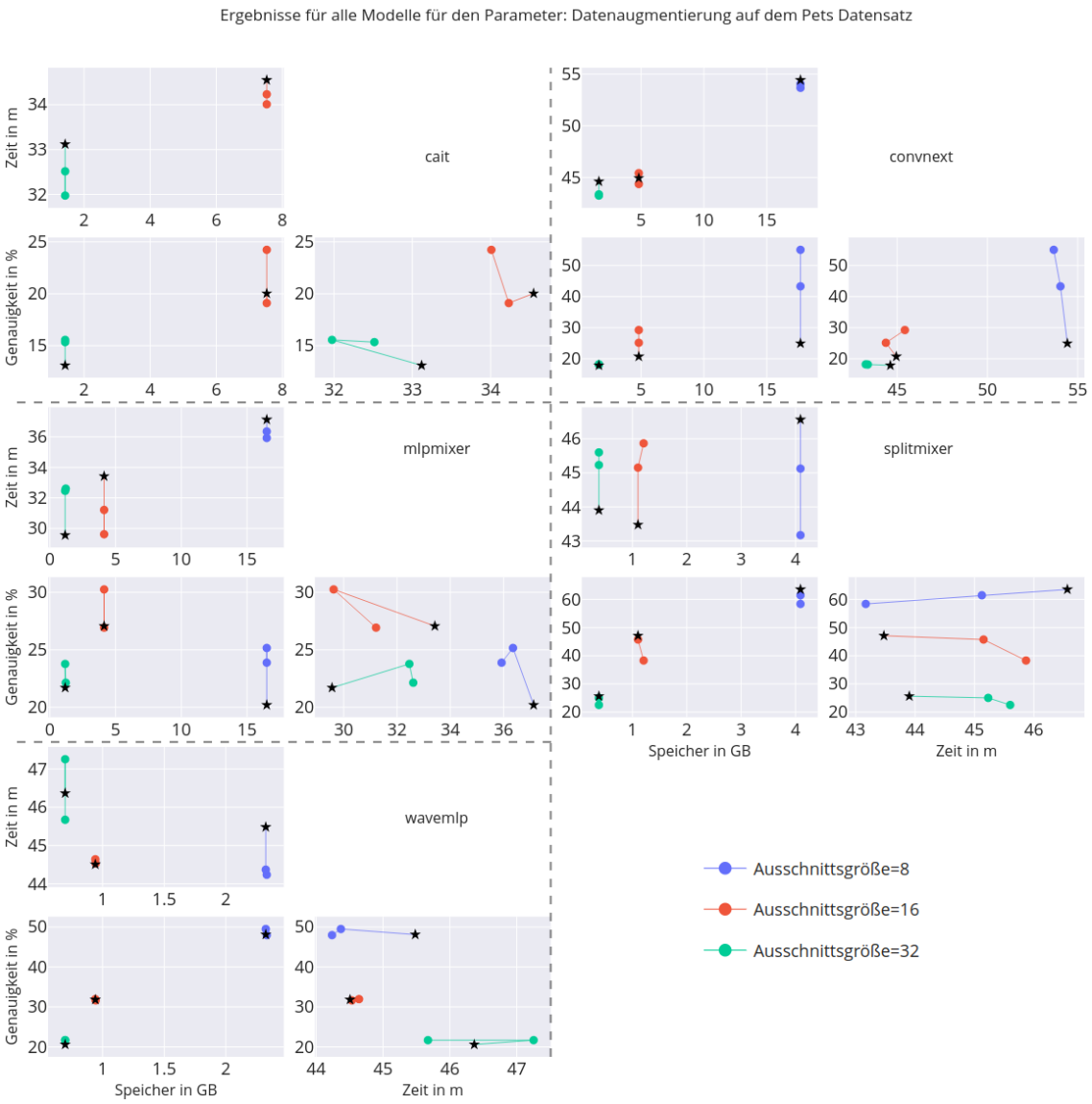


Abbildung A.1.: Detaillierte Ansicht aller Modelle auf dem Pets Datensatz mit dem Parameter Augmentierung

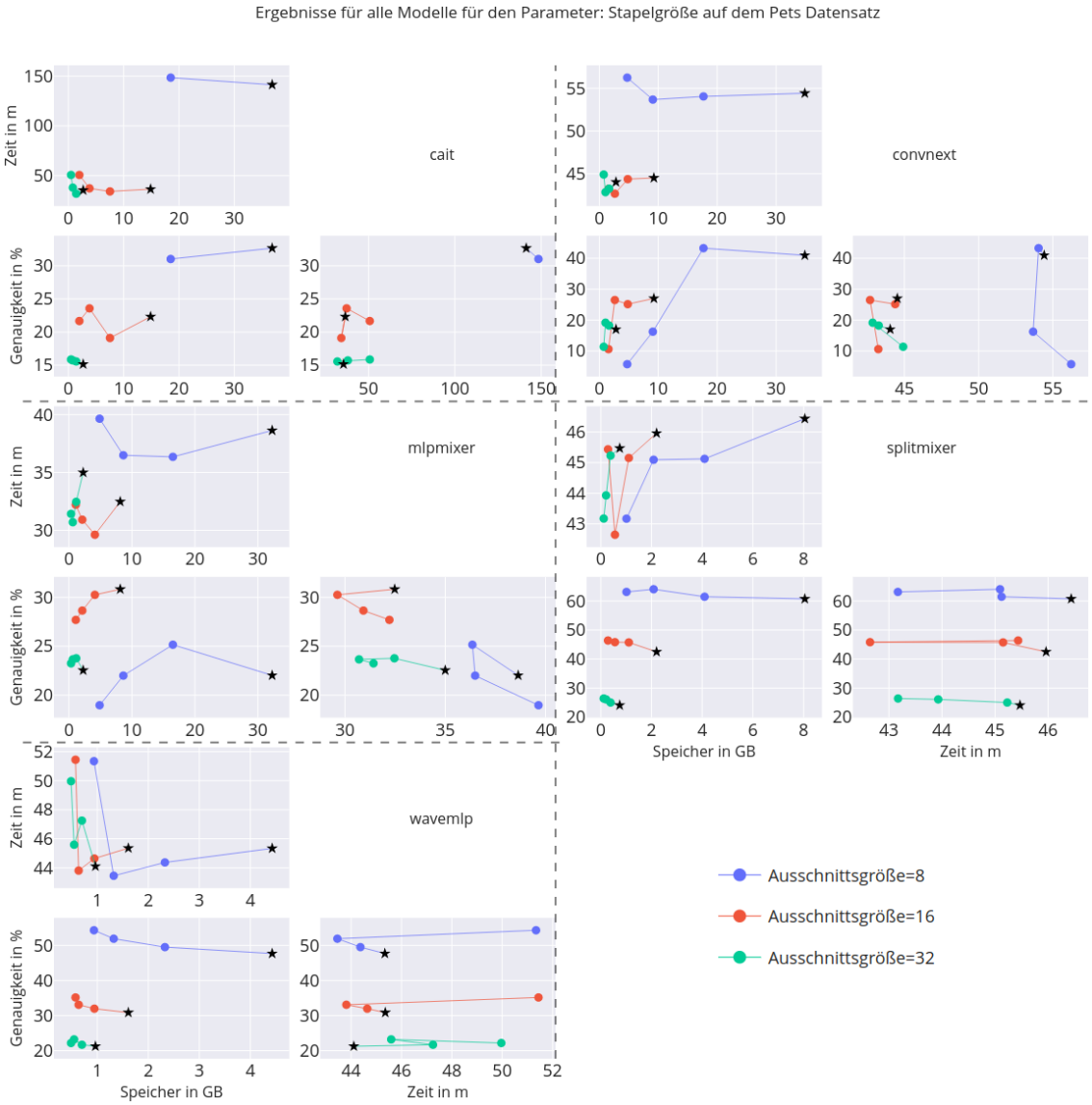


Abbildung A.2.: Detaillierte Ansicht aller Modelle auf dem Pets Datensatz mit dem Parameter Stapelgröße

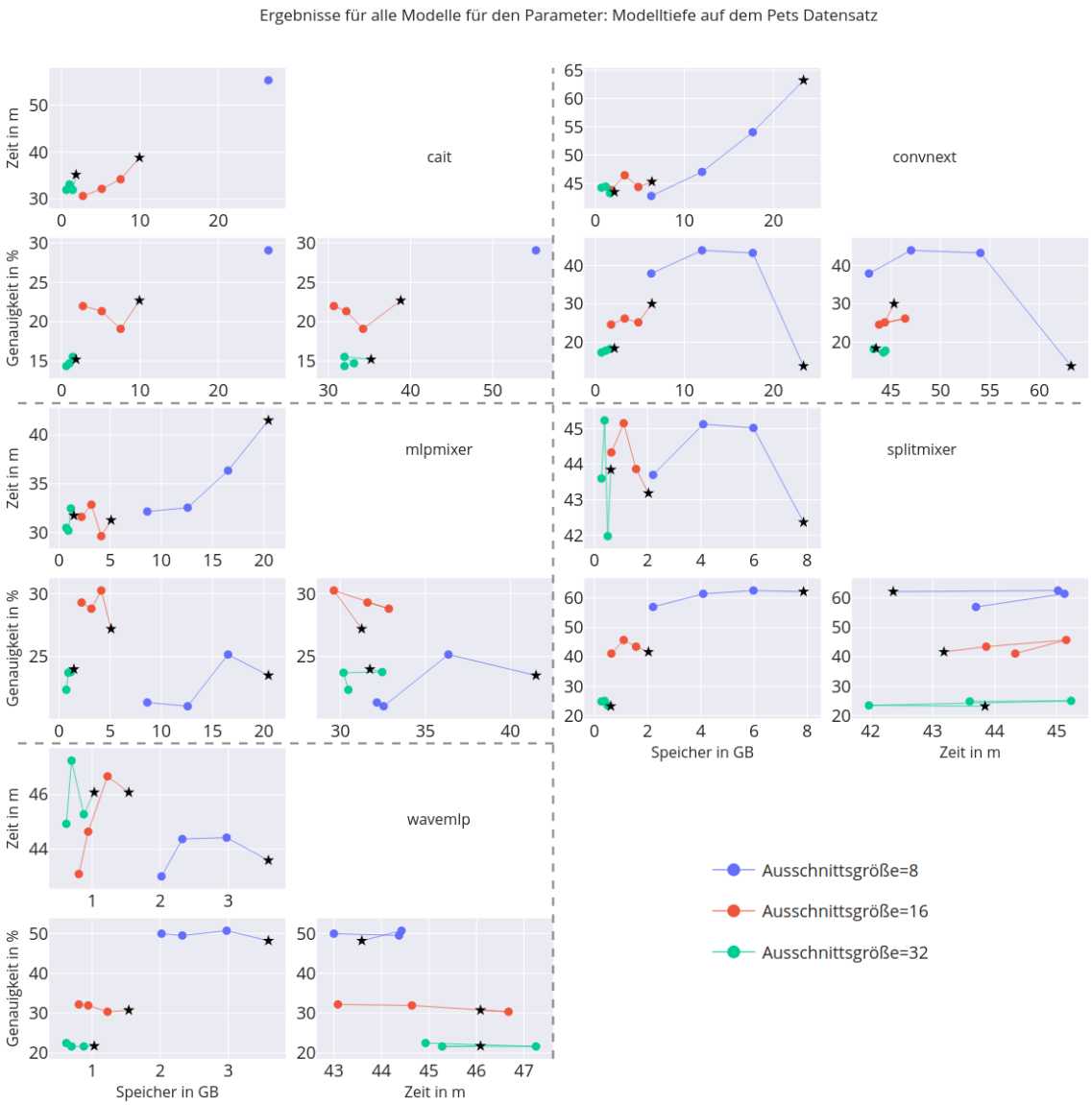


Abbildung A.3.: Detaillierte Ansicht aller Modelle auf dem Pets Datensatz mit dem Parameter Modelltiefe

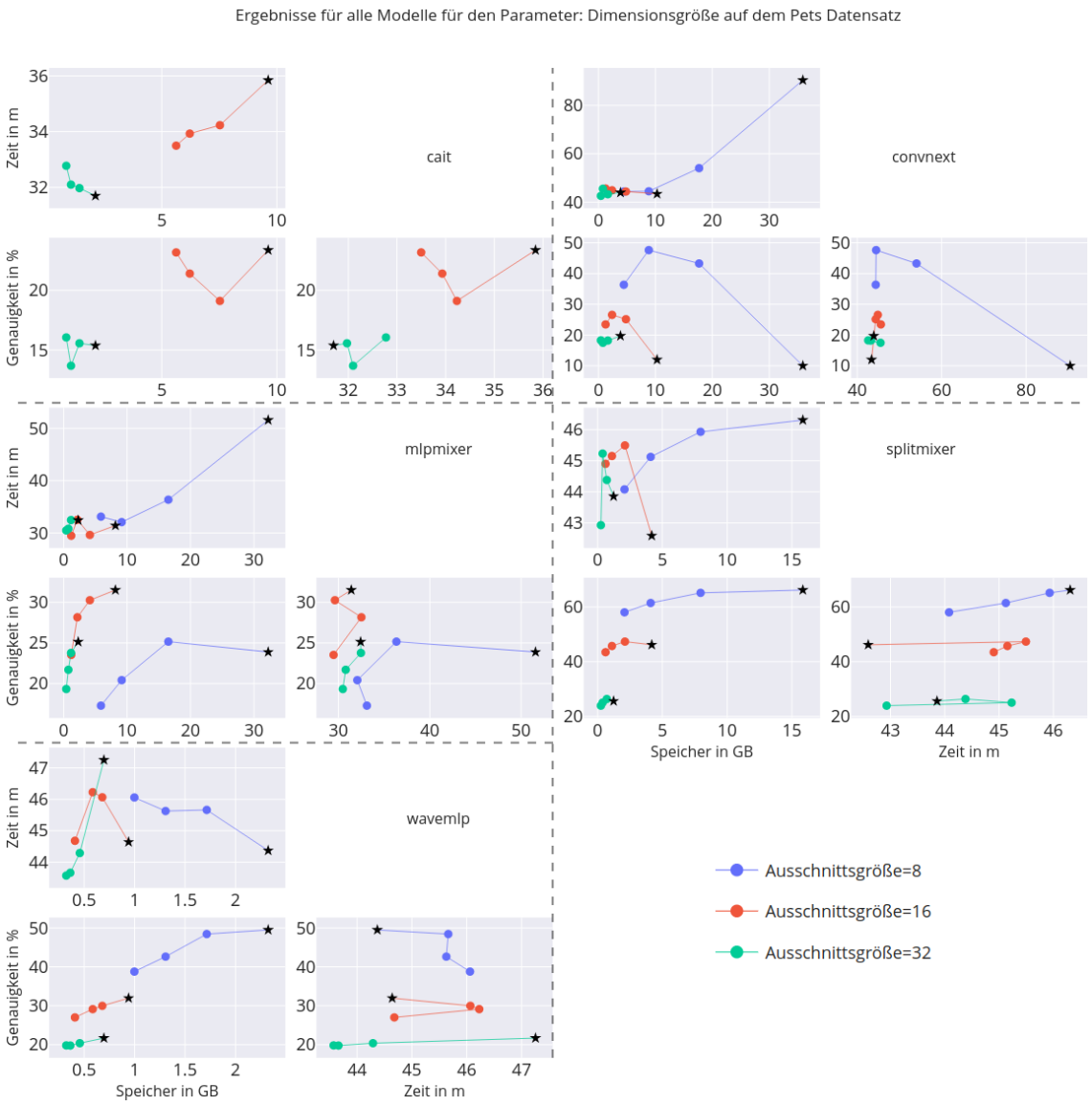


Abbildung A.4.: Detaillierte Ansicht aller Modelle auf dem Pets Datensatz mit dem Parameter verdeckte Dimension

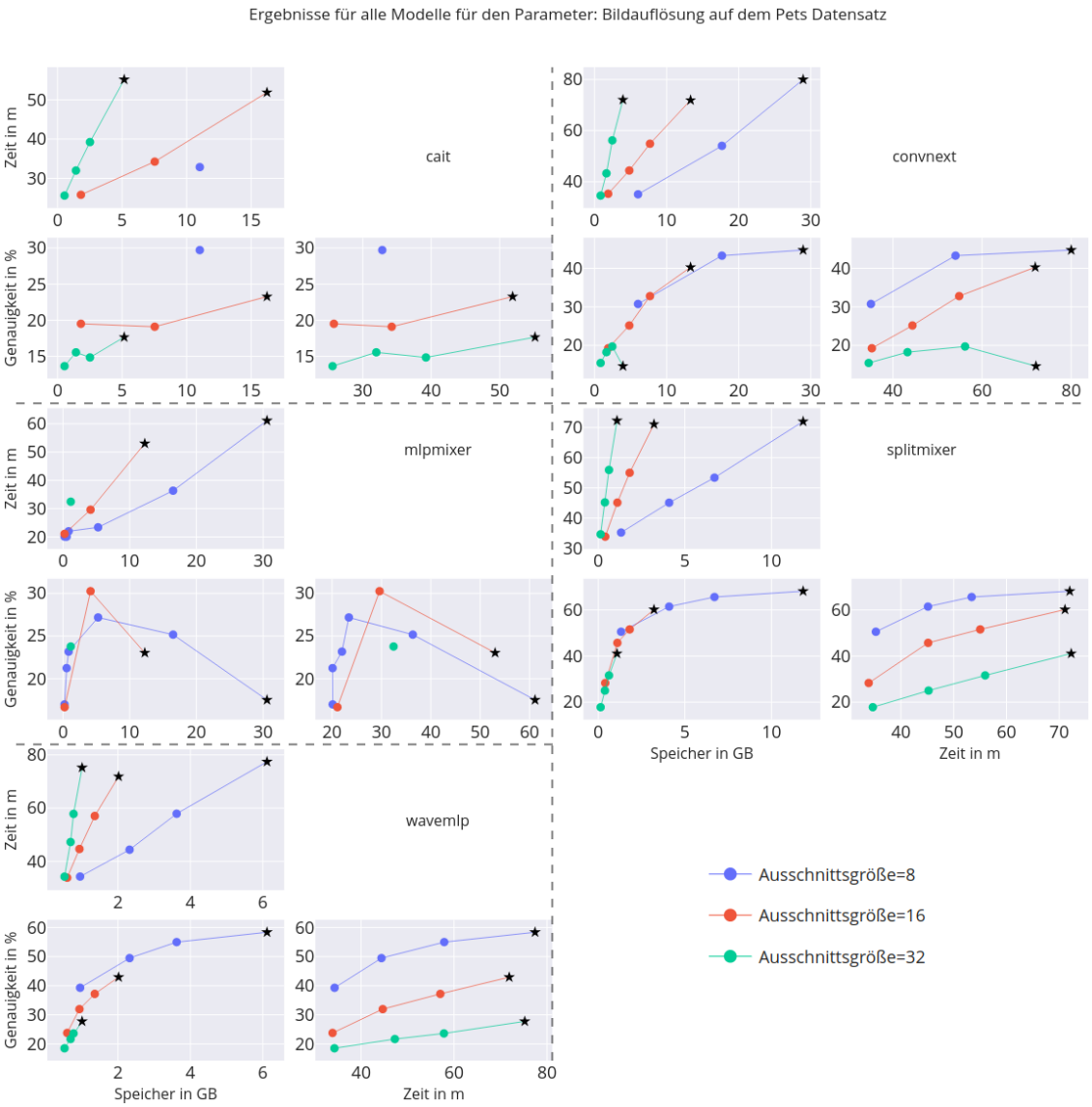


Abbildung A.5.: Detaillierte Ansicht aller Modelle auf dem Pets Datensatz mit dem Parameter Bildauflösung

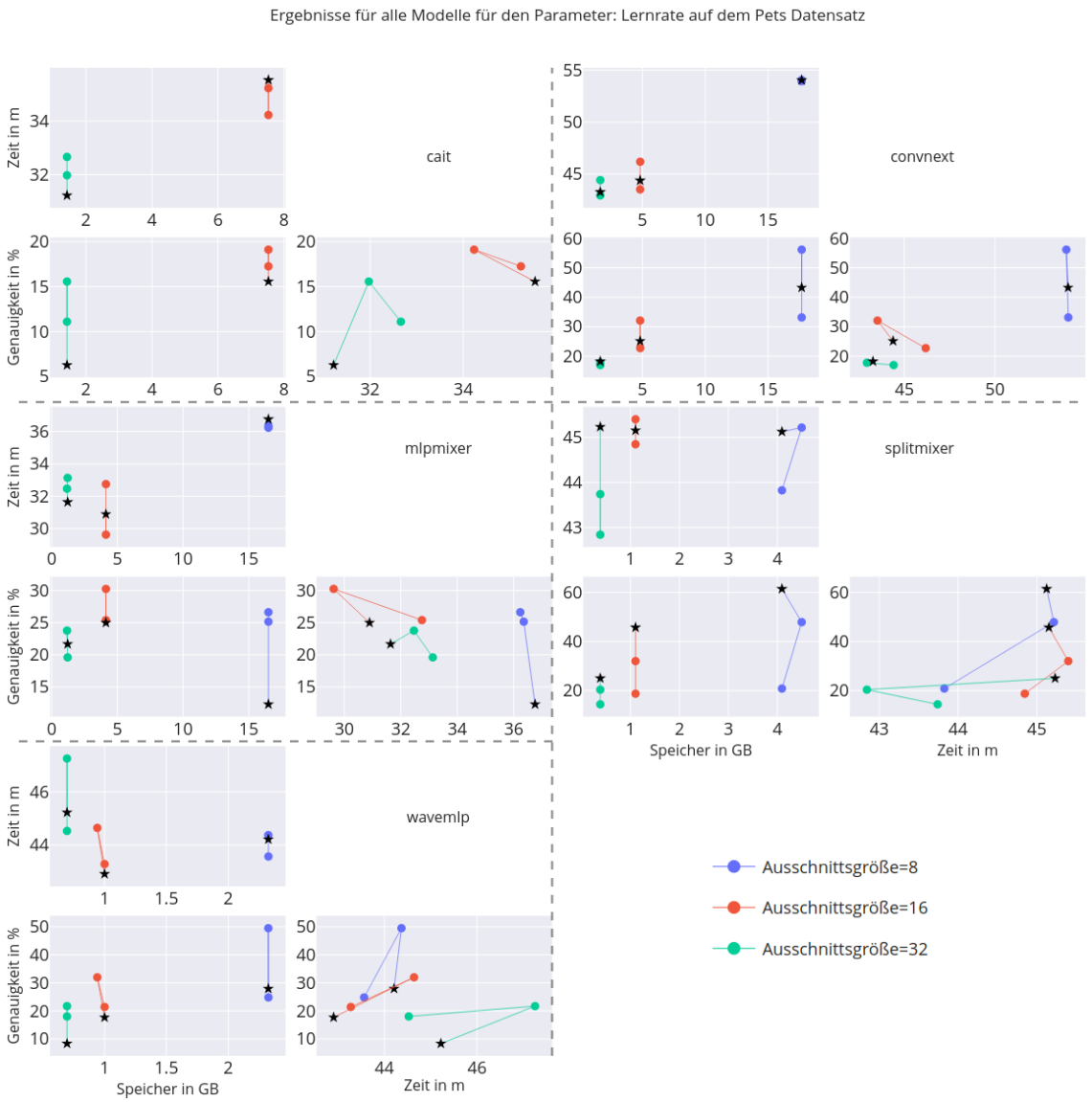


Abbildung A.6.: Detaillierte Ansicht aller Modelle auf dem Pets Datensatz mit dem Parameter Lernrate

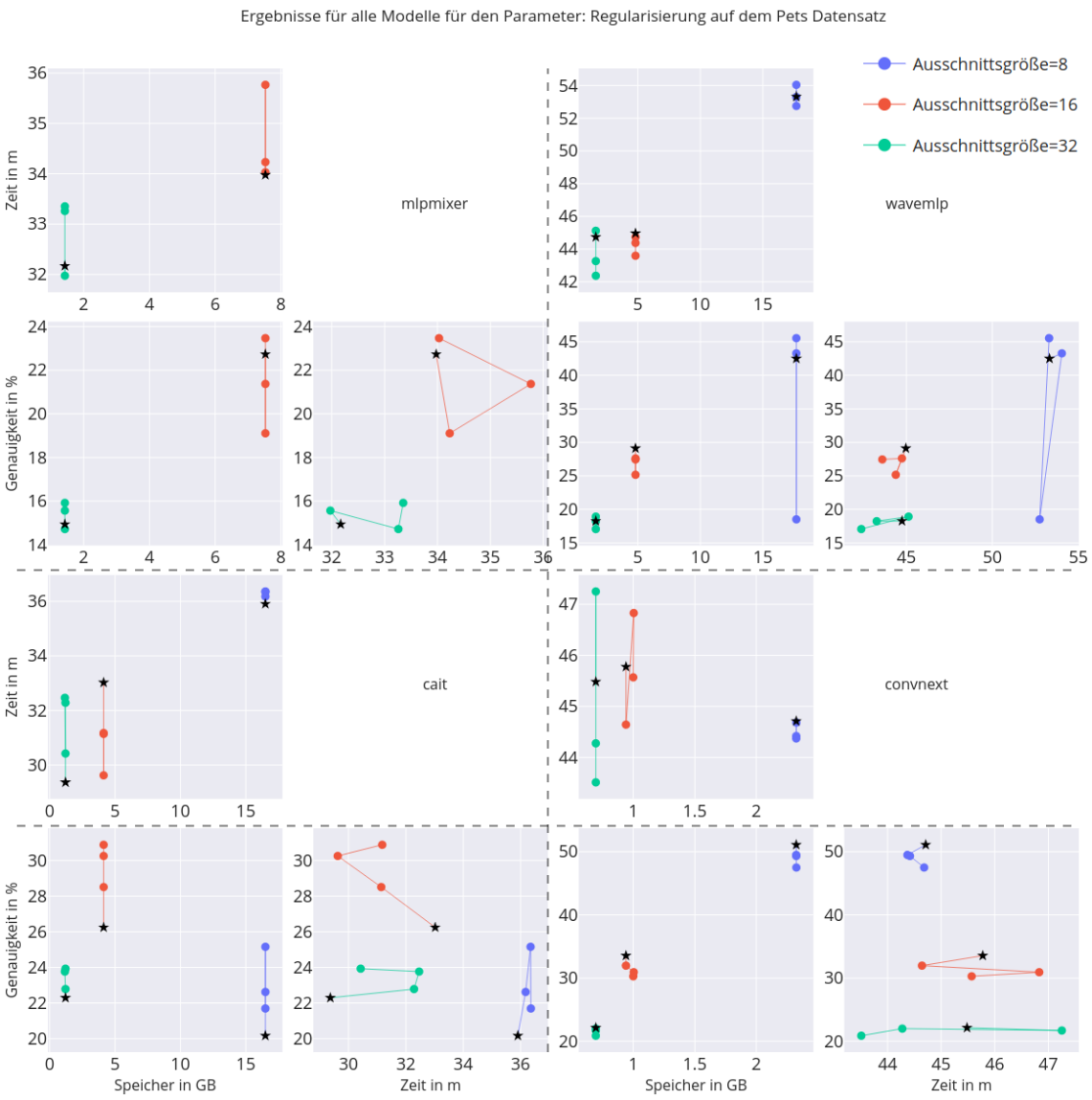


Abbildung A.7.: Detaillierte Ansicht aller Modelle auf dem Pets Datensatz mit dem Parameter Regularisierung

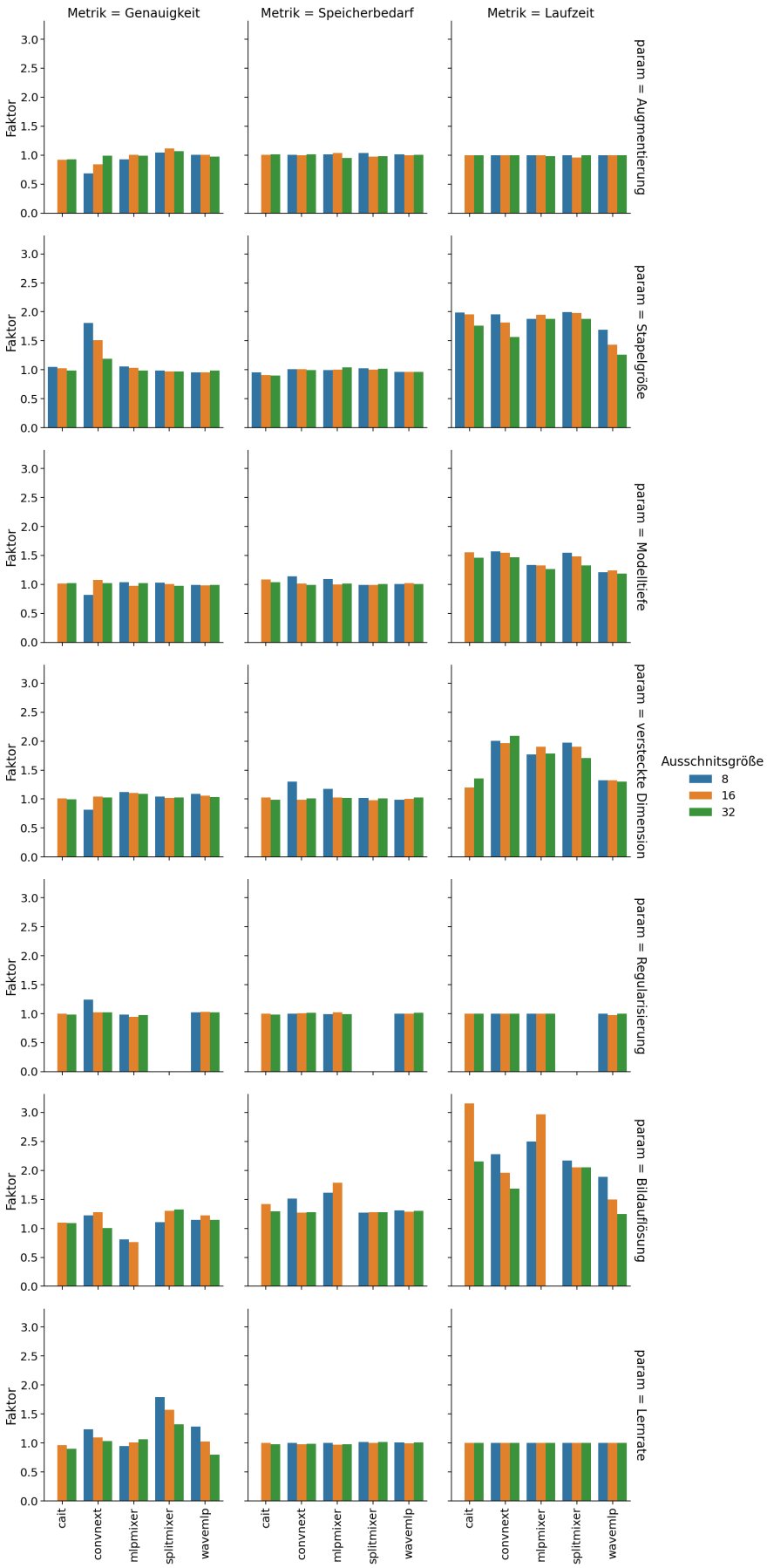


Abbildung A.8.: Zusammenfassung aller Werte auf dem Pets Datensatz

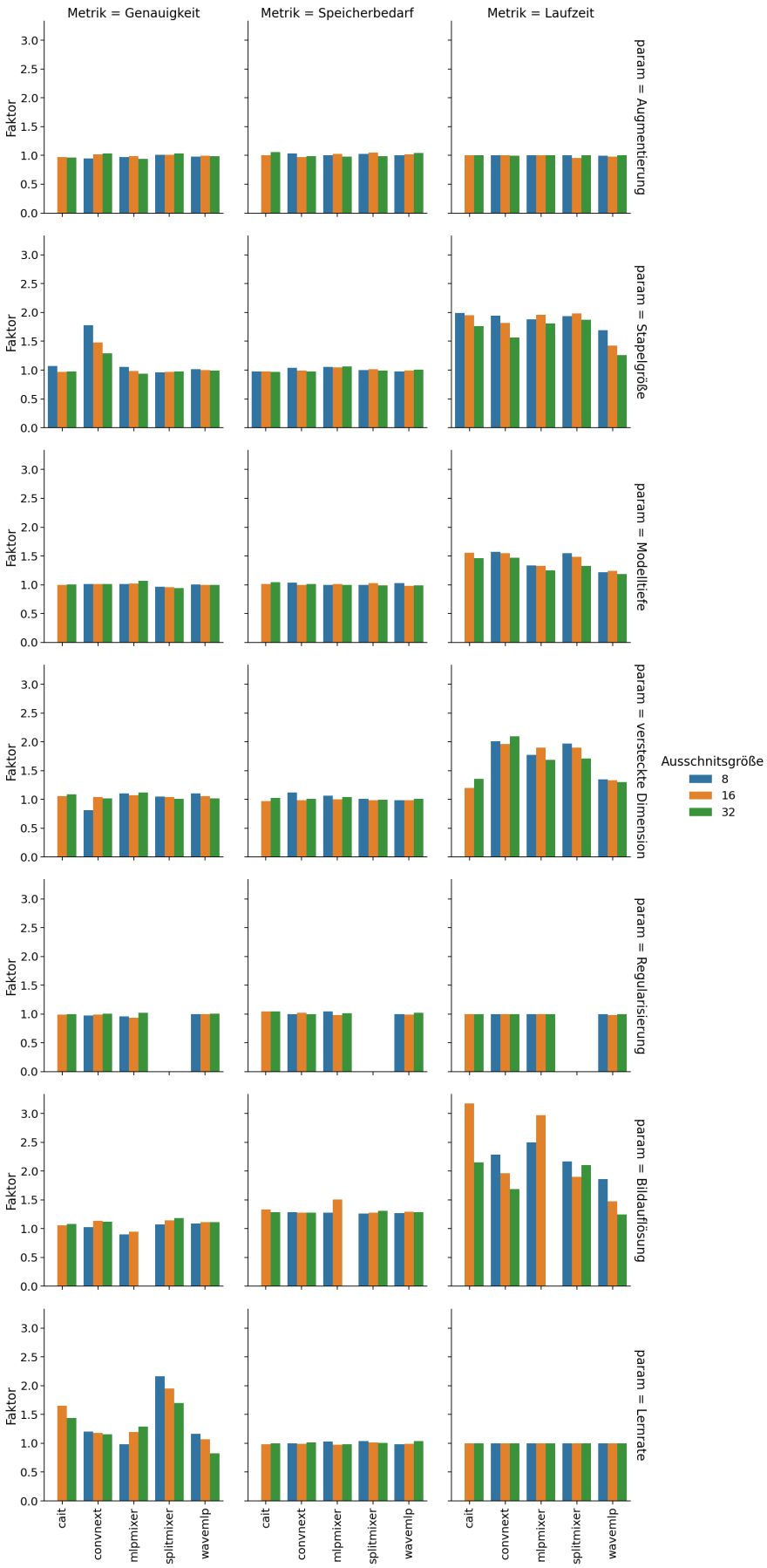


Abbildung A.9.: Zusammenfassung aller Werte auf dem Flowers102 Datensatz

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift