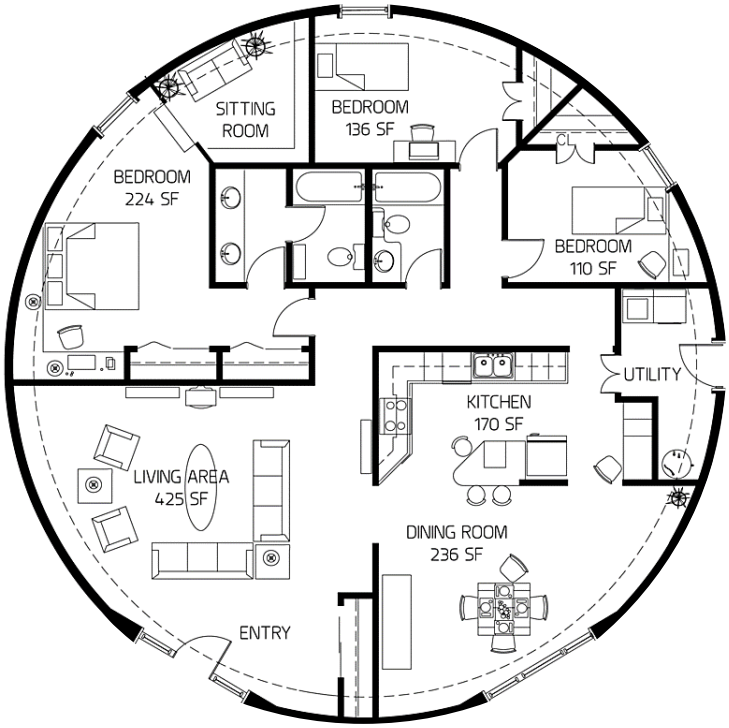
Deep Learning zur Objekterkennung in Grundriss-Bildern



[1]

Abschlussbericht der Kooperationsphase 2019/20

Durchgeführt am Deutschen Forschungszentrum für Künstliche Intelligenz
Betreuer: Dr. Stefan Agne, Christoph Balada
Betreuer am Hector-Seminar: Dr. Joachim Götz, Ingmar Oehme

Annika Nassal, PF14
nassalan@hector-seminar.de

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

In unserem Alltag sind Computer allgegenwärtig. Sie können Sprache verarbeiten, Antworten geben und uns bei der Navigation ohne große Schwierigkeiten den schnellsten Weg zu unserem Ziel unter Einbezug der Verkehrsdaten zeigen. Die Möglichkeiten von Computern wachsen ständig und so übernehmen sie immer öfter Aufgaben, die sonst Menschen mit deutlich größerem Zeit- und Arbeitsaufwand bearbeiten würden. Doch es gibt Aufgaben, in deren Bewältigung Computer noch weit hinter uns Menschen liegen – die Interpretation visueller Daten beispielsweise. Während Menschen ohne Probleme Informationen aus einem Bild extrahieren können und wissen, was dargestellt ist, sind Bilder für den Computer zunächst einfach nur Daten in Form von Zahlen. Dies wird zu einem Problem, wenn Computer Aufgaben für uns durchführen sollen, die auf Bilddokumenten beruhen. Ein bekanntes Beispiel hierfür ist die Klassifikation von Bildern anhand des dargestellten Tiers – Katze oder Hund. Dieses Problem wurde mit einem immer häufiger erfolgreich eingesetzten informatischen Konzept gelöst – dem neuronalen Netz. Neuronale Netze gehören zum Bereich der künstlichen Intelligenz und simulieren mithilfe von Mathematik ein Netz aus Neuronen ähnlich dem menschlichen Gehirn. Im hier vorgestellten Projekt wird mithilfe eines solchen neuronalen Netzes konkret das Problem der Analyse von Grundrissplanbildern mit eingezeichneter Möblierung bearbeitet. Dabei sollen die Pläne eingelesen und darauf alle Objektsymbole erkannt und bestimmt werden. Schwierigkeiten, die hierbei auftreten können, sind beispielsweise eine sehr variierende Symbolwahl in Plänen unterschiedlichen Ursprungs und die Feststellung der Position der Objekte auf dem Plan. Für die Lösung der Aufgabe soll ein passendes neuronales Netz trainiert werden, bis es die höchstmögliche Effizienz erreicht hat. Für das Training wird ein Datensatz aus einem Wissenschaftswettbewerb zum Einsatz kommen.

Fragestellung:

Wie kann ein neuronales Netz angewandt werden, um Möbelsymbole in Grundrissplänen zu erkennen und deren Positionen und Objekttypen zu bestimmen?

2 Durchführung

2.1 Ansatz

Um das Ziel zu erreichen, auf einem Grundrissbild alle Möbelstücke und Einrichtungsobjekte zu erkennen und deren Position zu bestimmen, wurde ein mehrschrittiger Ansatz gewählt. Dieser beinhaltet für die Objektklassifizierung ein Faltungsnetz[1], das zwischen den verschiedenen Möbelkategorien unterscheiden kann. Dafür soll ein neuronales Netz unter Variation der Hyperparameter[2] in vielen Durchläufen trainiert werden, um das beste Ergebnis zu erzielen. Für das Training des Netzes muss der Datensatz zunächst aufbereitet und dann die Trainingsdaten daraus extrahiert werden. Das Netz selbst soll dann auf Ausschnitten der Grundrisspläne erkennen, welches Symbol zu sehen ist oder ob keine Objektklasse klar erkannt werden kann und daher vermutlich auch kein Symbol auf dem Bildausschnitt ist. Dafür gibt einen Vektor mit den Wahrscheinlichkeiten für die verschiedenen Klassen aus. Wenn dann alle Ausschnitte des Planes klassifiziert wurden, ist es im nächsten Schritt möglich, mithilfe von Schwellwerten aus den Wahrscheinlichkeiten die tatsächlichen Positionen und Objekttypen zu extrahieren.

[1] Form des neuronalen Netzes, welche Faltungsschichten beinhaltet, die das Bild mithilfe der mathematischen Faltung oder bildlich dargestellt einem Filter verarbeiten

[2] Durch den Menschen festgelegte Parameter eines neuronalen Netzes

2.2 Datenvorverarbeitung und -vervielfältigung

Der verwendete Grundrissplan-Datensatz des Wettbewerbes „ICDAR2019-ORF“ der „International Conference on Document Analysis and Recognition 2019“ [3] [1] beinhaltet knapp 250 Pläne, auf denen mehr als 7000 Objekte aus 12 Kategorien zu finden sind. Diese Kategorien sind Toilette, Dusche, Badewanne, Waschbecken, Bidet, Tisch, Stuhl, Sofa, Sessel, Nachttisch, Bett und Herd, wobei diese Kategorien auf unterschiedlichen Plänen unterschiedlich dargestellt werden. Die Bilder stehen als PNG-Dateien zur Verfügung und die Metadaten sind in einer JSON-Datei festgehalten.

2.2.1 Aufbereiten des Datensatzes

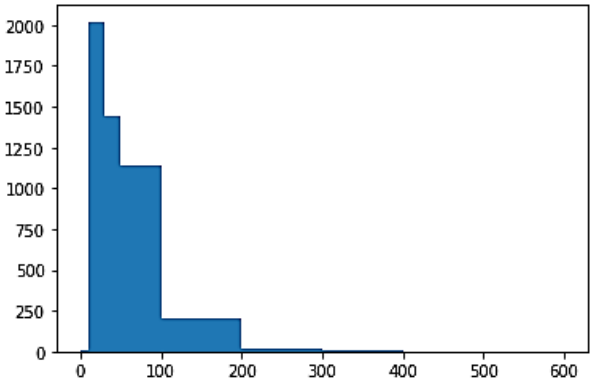


Abb. 1: Mengenverteilung (y) der Symbolbilder nach Größe (x)

Schon früh während der Arbeit mit dem Datensatz konnten Fehler in und Probleme mit den Daten festgestellt werden, weshalb dieser aufbereitet werden musste. Zunächst fiel auf, dass die Metadaten Informationen zu Grundrissbildern enthielten, die im Download nicht mitgeliefert wurden. Um diesen Fehler zu beheben, wurde ein Programm geschrieben, das die Informationen, die keinem Bild zugeordnet werden konnten, aus den Metadaten entfernt und in Anhang 9.1 zu sehen ist. Das selbe Programm entfernte auch Fehlinformationen, die in den Metadaten enthalten waren. So waren beispielsweise Angaben zu Positionen von Symbolen außerhalb der Größe des Grundrissplans vorhanden, was nicht möglich ist. Des Weiteren hatten die Symbole auf den Grundrissplänen sehr variierende Größen, also unterschiedliche Seitenlängen in Pixeln, was ein Problem darstellte, da die Eingabegröße des Neuronalen Netzes, das zur Klassifizierung verwendet werden sollte, fest ist. Die Größe des größten Symbols war dafür nicht geeignet, da auf anderen Grundrissbildern die Symbole sehr klein waren. Um einen Überblick zu erhalten, wie viele Symbole mit welcher Größe vorhanden sind, wurden die Metadaten mit einem Programm ausgewertet, das diese Information als Histogramm darstellt. Das Histogramm für die Höhe der Bilder ist in Abb. 1 zu sehen. Darauf ist zu erkennen, dass viele Symbole zwischen 100*100 und 10*10 Pixel groß sind. Die Verteilung für die Breite der Bilder wich nur geringfügig von der für die Höhe ab. Also wurde das Projekt eingegrenzt auf die Erkennung von Symbolen die in ein Feld von 100*100 Pixeln passen. Diejenigen Symbole, die zu groß oder zu klein sind, wurden aus den Metadaten gelöscht, sodass das Netz beim Training im Abgleich mit den korrekten Lösungen keine verfälschten Werte ausgibt. Am Ende waren für das Training 1600 verschiedene Symbole übrig, die später vervielfältigt werden.

[3] Eigene Übersetzung: Internationale Konferenz zur Dokumentenanalyse und -erkennung 2019; zweijährlich stattfindende wissenschaftliche Konferenz

2.2.2 Zuschneiden und Vervielfältigen der Trainingssymbole

Um nun für das Training des Netzes zur Klassifizierung der Objektsymbole diese 100*100 Pixel großen Symbolbildchen zu erhalten, mussten diese aus den Grundrissplänen ausgeschnitten werden. Der zugehörige Quellcode ist in Anhang 9.2 zu finden.

Darin werden zunächst die aufbereiteten Metadaten eingelesen und die Grundrisspläne als Numpy[4] Arrays importiert und in einer Liste gespeichert. Daraufhin müssen die Pläne den richtigen Metadaten zugeordnet werden, da die Liste keine Dateinamen „speichert, sondern jeder Plan nur einen Listenindex hat. So kann einfacher auf die Bild-Arrays zugegriffen werden.

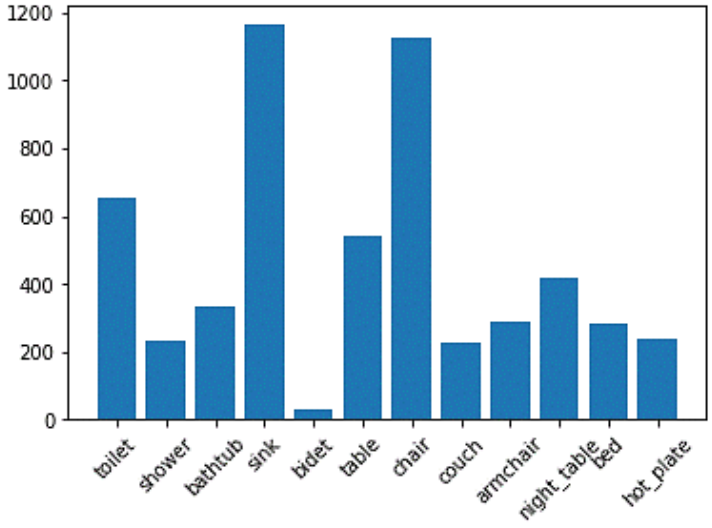


Abb. 2:Mengenverteilung (y) der Klassen (x) bei einem festen Vervielfältigungsfaktor von 20

Nachdem alle Daten importiert und sortiert sind, wird nun eine Schleife genutzt, um alle Annotationen, wie die Informationen für jedes Symbolbild in den Metadaten bezeichnet sind, nacheinander durchzugehen. Innerhalb der Schleife werden dann die sogenannte Bounding Box, die Positionsangabe des aktuellen Symbols auf dem Grundrissplan, und die Bild-ID verwendet, um das Symbol auf dem entsprechenden Plan zu finden. Ein Quadrat der Kantenlänge 100*100 mit dem Symbol im Zentrum wird aus den Bilddaten herauskopiert und in einer Liste gespeichert. Daraufhin wird in einer zweiten Liste an der entsprechenden Stelle die Nummer der Objektkategorie, die auf dem Quadrat dargestellt ist, abgespeichert. Bei der Auswahl des Quadrats gibt es einen Ausnahmefall, in dem das Symbol nicht im Zentrum positioniert werden kann. Findet sich der Mittelpunkt des Symbols näher als 50 Pixel am Rand des Grundrissbildes, so muss das Quadrat in dieser Richtung am Rand ausgerichtet werden. Anderenfalls würde das Quadrat leere Pixel kopieren, was zu einem Fehler bei der Verarbeitung durch das Netz führen würde. Für die Lösung dieses Problems wurde eine Funktion geschrieben, die die Koordinaten der linken oberen Ecke des Quadrats von 100*100 Pixeln für beide Dimensionen innerhalb des Intervalls [0; Ausdehnung des Bildes -100] hält. Nachdem das Standardsymbolbild dann abgespeichert wurde, wird es nun vervielfältigt. Das gleicht die geringe Datenmenge aus, die im Datensatz zur Verfügung steht. Die Vervielfältigung beinhaltet immer eine Abwandlung des Standardbildes, da nur so die Verallgemeinerung in der Klassifizierung durch das Netz gewährleistet werden kann. So ist beispielsweise ein Verdrehen des Bildes eine Möglichkeit, dem Netz die Fähigkeit zu geben, unbekannte Symbole beliebiger Orientierung zu bestimmen. Für die Rotation wird die Bibliothek Imutils[5] verwendet. Neben der Rotation werden zudem eine Skalierung mithilfe der Bibliothek OpenCV[6] und eine handgeschriebene Verschiebung des Symbols innerhalb des Quadrates von 100*100 Pixeln genutzt. Die Werte der Rotation, Verschiebung und Skalierung werden für jeden Aufruf zufällig neu bestimmt und auch die Verteilung der verschiedenen Vervielfältigungen pro Symbol ist zufällig. So soll gewährleistet werden,

[4] Bibliothek für wissenschaftliche Berechnungen in Python

[5] Bibliothek für Bildverarbeitung

[6] Bibliothek für Bildverarbeitung und -erkennung

dass möglichst viele verschiedene Versionen jedes Symbols entstehen. Die allgemeine Anzahl der Vervielfältigungen pro Symbolbildchen wurde zunächst auf 20 festgelegt. Es stellte sich jedoch heraus, dass manche Objektklassen deutlich häufiger vertreten waren als andere, was in dem Histogramm in Abb. 2 gezeigt wird. Dies ist für die Auswertung des Netzes unvorteilhaft. Daher wurde der konstante Wert durch eine umgekehrte Skalierung mit Toleranzbereich ersetzt. Der Toleranzbereich betrug dabei $\pm$ 60 Symbole um den Mittelwert der Symbolanzahlen und der zu skalierende Wert war 20 als Basiswert für die Vervielfältigungen. Die mathematische Formulierung der Skalierung ist in **Fehler! Verweisquelle konnte nicht gefunden werden.** dargestellt und wird an einem Beispiel gezeigt. Die Ergebnisse des Beispiels sind in Abb. 3 visualisiert. Es ist zu erkennen, dass v_n für sehr kleine Bildanzahlen sehr groß wird und umgekehrt und es einen konstanten Bereich gibt. Die Klasse „bidet" war, wie in Abb. 2 zu erkennen, allerdings so wenig in den Daten vorhanden, dass die hohe Vervielfältigung auch keine Lösung war, weil die vielen Vervielfältigungen sehr weniger Ursprungsbilder sich sehr ähnlich waren. Die Klasse wurde darum und aufgrund häufiger Verwechslung später mit der Klasse „toilet" verbunden.

Allgemeine Theorie	**Beispiel**
Ursprüngliche Anzahl von Bildern pro Klasse: $\boldsymbol{a} = \{a_1; a_2; \dots a_{12}\}$ Durchschnitt der Anzahlen: $\bar{a}$ Vervielfältigungsfaktor für jedes Bild pro Klasse: $\boldsymbol{v} = \{v_1; v_2; \dots v_{12}\}$ Für jedes a_n: Falls $(\bar{a} - 60) < a_n < (\bar{a} + 60)$: $v_n = 20$ Falls $(\bar{a} + 60) < a_n$: $v_n = 20 * \frac{\bar{a}+60}{a_n}$ Falls $a_n < (\bar{a} - 60)$: $v_n = 20 * \frac{\bar{a}-60}{a_n}$	Gegeben: Anzahl von Bildern pro Klasse $\boldsymbol{a} =$ {185;67;70;318;9;161;381;46;77;145;63;78} Durchschnitt der Anzahlen $\boldsymbol{\bar{a} = 133,\overline{3}}$ Gesucht: Vervielfältigungsfaktor für jedes Bild pro Klasse $\boldsymbol{v} = \{\boldsymbol{v1}; \boldsymbol{v2}; \dots \boldsymbol{v12}\}$ Ergebnis: $\boldsymbol{v}$ ={20,00; 21,79; 20,86; 12,14; 162,22; 20,00; 10,13; 31,74; 20,00; 20,00; 23,17; 20,00}

Tab. 1: Umgekehrte Skalierung mit einem Toleranzbereich allgemein und an einem Beispiel

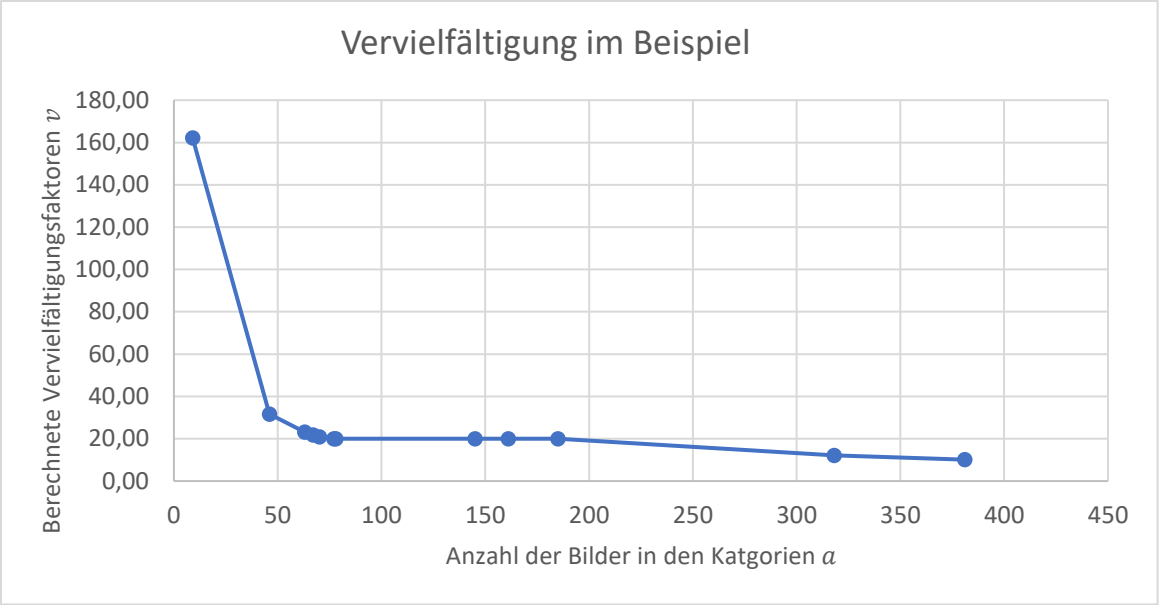


Abb. 3: Graphische Darstellung der Ergebnisse des Beispiels

2.3 Aufbau und Training des Netzes

Für die Symbolerkennung auf den Quadraten von 100*100 Pixeln wurde ein neuronales Netz, genauer ein Faltungsnetz verwendet. Netze dieses Typs sind in der Bildklassifikation weit verbreitet, da sie als hoch performant auf solchen Aufgaben gelten. Dies bestätigen auch die Autoren des Buches „Deep Learning“ mit ihrer Feststellung „Convolutional Networks have been tremendously successful in practical applications“[7] [2] (S. 321). Die anfänglichen Gewichte im Netz wurden von Tensorflow[8] Keras[9] bezogen und sind auf dem ImageNet-Datensatz[10] vortrainiert. Das bietet den Vorteil, dass ein spezifisches Training für die Aufgabe schneller und erfolgreicher abläuft, da die Gewichte bereits an die Erkennung von Strukturen in Bildern angepasst sind. Es wurden die Netze InceptionResNetV2 und ResNet50 von Keras ausgewählt. Diese wurden dann mit einem Teil der Bilder mit den Objektsymbolen trainiert. Um eine möglichst hohe Genauigkeit des Netzes zu erreichen, wurden in mehreren automatisierten Trainingsdurchläufen die Hyperparameter Epochen[11], Batch Size[12], Lernrate[13] und Optimizer[14] variiert und die Ergebnisse festgehalten und verglichen. Das Programm, das für das Training und die Evaluation der Durchläufe inklusive der nachfolgend beschriebenen Optimierung zuständig ist, ist in Anhang 9.3 zu sehen.

---

[7] Eigene Übersetzung: Faltungsnetze waren in praktischen Anwendungsfällen enorm erfolgreich.

[8] Bekanntes Framework für maschinelles Lernen

[9] Bibliothek für die Implementierung von neuronalen Netzen

[10] „a very large collection of human annotated photographs designed by academics for developing computer vision algorithms“ (Eigene Übersetzung: Ein große Sammlung von durch Menschen bezeichneten Fotos, die von Wissenschaftlern für die Entwicklung von Bilderkennungsalgorithmen entwickelt wurde.) [10]

[11] Trainingsdurchläufe über den ganzen Trainingsdatensatz.

[12] Anzahl der Trainingselemente, die vor jedem Optimierungsschritt durch das Netz bearbeitet werden

[13] Bestimmt die Stärke der Anpassung in jedem Optimierungsschritt

[14] Bestimmt die konkrete Umsetzung der Anpassung in jedem Optimierungsschritt

2.4 Optimieren des Netzes mithilfe von Metriken

Die Genauigkeit des Netzes auf den Daten, mit denen es trainiert, ist aufgrund des sogenannten „Overfitting“ [15] und anderer Probleme möglicherweise nicht repräsentativ. Um das zu überprüfen und dem entgegenzuwirken, wird vor dem Training der Datensatz nach einem Prinzip aufgeteilt, das der bekannte KI-Forscher Andrew Ng in seinem Buch „Machine Learning Yearning“ beschreibt. Er empfiehlt die Aufteilung „Training set — Which you run your learning algorithm on; Dev (development) set — Which you use to tune parameters, select features, and

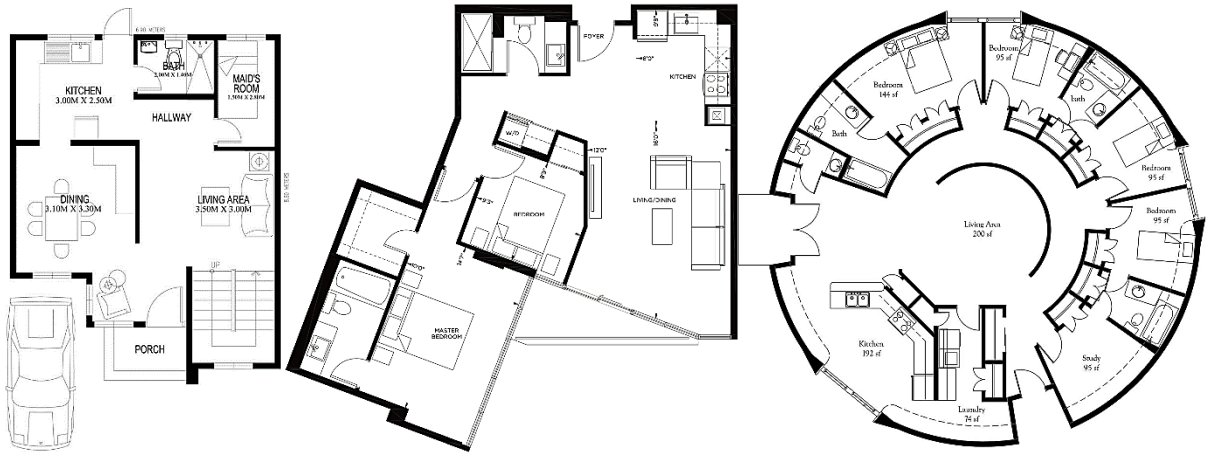


Abb. 4: Beispiele aus Trainings-, Validierungs- und Testdatensatz [1]

make other decisions regarding the learning algorithm […]; Test set — which you use to evaluate the performance of the algorithm, but not to make any decisions regarding what learning algorithm or parameters to use“[16] [3]. Dabei wird darauf geachtet, dass die anderen Teile auch Eigenschaften enthalten, die im Trainingsanteil nicht vorhanden sind, um die Fähigkeit des Netzes zur Verallgemeinerung zu überprüfen. Für die Aufteilung der Grundrisspläne in diesem Projekt wurde als Kriterium festgelegt, dass nur Pläne, die ausschließlich rechtwinklig aufeinander stehende Wände zeigen, für das Training verwendet werden. Der aussortierte Teil wurde wieder in zwei Kategorien unterteilt. Der sogenannte Validierungsdatensatz enthält nur Pläne mit geraden, aber auch diagonal und nicht rechtwinklig verlaufenden Wänden und der Testdatensatz enthält schließlich die Pläne, auf denen auch gekrümmte Wände und runde Gebäude gezeigt werden. Je ein Beispiel für jeden Teil des Datensatzes ist in Abb. 4 zu sehen.

Nach jeder Epoche des Trainings wird nun ein sogenanntes Callback aufgerufen. Darin wird das Netz auf die Validierungsdaten angewandt. Die Ergebnisse des Netzes werden dann mit den Musterlösungen verglichen und das Netz wird mithilfe von daraus berechneten Metriken bewertet. Die genutzten Metriken sind die Konfusionsmatrix, die Precision-Recall-Kurve, der Average-Precision-Score und der F1-Score.

[15] Überanpassung des Netzes an die Trainingsdaten und damit eine fehlende Verallgemeinerung; Netz hat die Trainingsdaten „auswendiggelernt“ und erreicht infolge nur eine geringe Genauigkeit auf allen anderen Daten

[16] Eigene Übersetzung: Trainingsdatensatz — darauf trainiert der Lernalgorithmus; Entwicklungs-/Validierungsdatensatz — wird zur Optimierung von Parametern, zur Bestimmung der besten Netzeigenschaften und zur weiteren Entscheidungsfindung über die Gestaltung des Algorithmus genutzt; Testdatensatz — darauf wird die Leistung des Algorithmus evaluiert, aber es werden damit keine Entscheidungen über die Wahl des Lernalgorithmus oder der Parameter getroffen

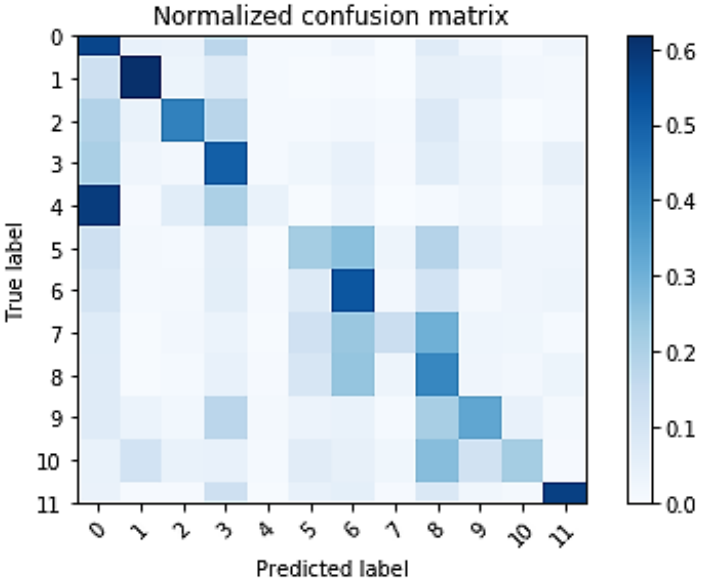


Abb. 5: Beispielhafte Konfusionsmatrix aus Epoche 48 des Trainings eines ResNet 50 mit Batch Size 32 und Lernrate 0,001

Die Konfusionsmatrix ist die einzige Metrik, die für alle Klassen gemeinsam berechnet wird. Sie wird als tabellenartiges Diagramm dargestellt. Die x-Achse gibt dabei die vom Netz vorhergesagte Klasse an und die y-Achse die tatsächliche Klasse. Das entstehende Raster ist über eine Farbskala gefüllt, die die Häufigkeitsverteilung der vom Netz vorhergesagten Klasse für jede Klasse angibt. Optimal ist ein Netz dann, wenn bei der Vorhersage immer die vorhergesagte Klasse mit der tatsächlichen Klasse übereinstimmt. Die Konfusionsmatrix hat dann einen diagonalen, intensiv gefärbten Streifen und ist sonst vollkommen weiß. In Abb. 5 ist ein Beispiel aus dem Training eines ResNet 50 dargestellt. Darauf ist zu erkennen, dass die Klassen 1 und 11 bereits sehr gut trainiert sind, da sowohl in der Zeile, als auch in der Spalte der Klassen nur ein dunkles Feld zu sehen ist. Klasse 0 beispielsweise wird zwar oft korrekt erkannt, was daran zu erkennen ist, dass in der Zeile nur ein sehr dunkles Feld ist. Dafür wird die Klasse 4 aber oft als Klasse 0 erkannt, was im Diagramm durch das dunkle Feld auf Höhe der 4 in der Spalte 0 dargestellt ist. An der hellen Spalte von Klasse 4 ist zu erkennen, dass das Netz fast nie ein Bild als Objekt der Kategorie 4 klassifiziert. [4]

Die Precision-Recall-Kurve wird für jede Klasse einzeln berechnet und alle Kurven können am Ende in einem Diagramm gezeichnet werden. Für die Kurve werden je Klasse zunächst Precision und Recall berechnet. Die Precision wird dabei definiert als der Quotient aus den korrekt als diese Klasse vorhergesagten Bildern und allen als diese Klasse vorhergesagten Bildern. Der Recall ist festgelegt als das Verhältnis von Bildern dieser Klasse, die korrekt als Bilder dieser Klasse vorhergesagt wurden und allen Bildern dieser Klasse. Am Beispiel der Klasse „toilet“ wäre die Precision also der Quotient aus der Anzahl aller korrekt erkannten Toiletten und der Anzahl aller als Toilette erkannten Objekten und der Recall wäre der Quotient aus allen korrekt erkannten Toiletten und allen Toiletten. Beide Berechnungen sind in Tab. 2 dargestellt. Um nun mehrere Precision-Recall-Paare für eine Kurve zu erhalten, wird ein sogenannter Threshold, also ein Schwellwert benutzt.

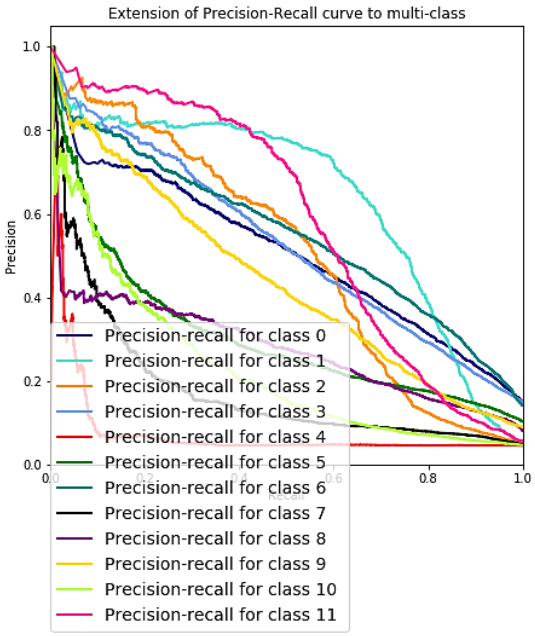


Abb. 6: Beispielhafte Precision-Recall-Kurve für alle zwölf Klassen

Das Netz gibt für jedes Bild für jede Klasse eine Wahrscheinlichkeit aus, mit der das Objekt auf dem Bild dieser Klasse zuzuordnen ist. Der Threshold gibt an, ab welcher Wahrscheinlichkeit das Objekt als erkannt gilt. Ein geringerer Threshold bedeutet also, dass das Netz sich

weniger sicher sein muss, um das Bild der aktuellen Klasse zuzuordnen. Daraus resultiert dann ein höherer Recall-Wert, da mehr Bilder als die aktuelle Klasse erkannt werden, aber ein geringerer Precision-Wert, da das Netz sich weniger sicher war und sich so mehr Fehler einschleichen. Optimal ist ein Netz dann, wenn bei einem hohen Threshold für alle Klassen ein hoher Recall-Wert und ein hoher Precision-Wert festgestellt werden können. Je nach Anwendung ist dabei oft entweder die Precision oder der Recall von größerer Bedeutung. Beispielsweise soll ein Klassifikator, der Krebstumore auf Bildern erkennt, lieber zu oft Alarm geschlagen haben und dann falschliegen, als dass er zwar immer richtig liegt, wenn er einen Tumor erkennt, er aber viele übersieht. Der Recall hat hier also eine höhere Priorität als die Precision. In einem Spamerkennungsprogramm wiederum wird angestrebt, wirklich nur Spam und auf keinen Fall wichtige Mails als Spam zu klassifizieren, während eine nicht als Spam erkannte, aber unerwünschte Mail im Posteingang das kleinere Übel darstellt. Im Beispiel in Abb. 6 wären die Klassen 1 und 11 am besten, da deren Kurven in ihrem Verlauf dem Punkt (1|1) am nächsten sind. Besonders schlecht ist die Klasse 4, da ihr Graph am nächsten an (0|0) verläuft. [2], [5]

TP (True Positive) = korrekt als aktuelle Klasse erkannt

FP (False Positive) = fälschlicherweise als aktuelle Klasse erkannt

TN (True Negative) = korrekt als nicht die aktuelle Klasse erkannt

FN (False Negative) = fälschlicherweise als nicht die aktuelle Klasse erkannt

$$Precision\ P = \frac{TP}{TP + FP};\ Recall\ R = \frac{TP}{TP + FN}$$

Tab. 2: Berechnung von Precision und Recall [6]

Der Average Precision Score ist definiert als die Fläche unter dem Graphen der Precision-Recall-Kurve. Er beschreibt die Aufsummierung aller Precision-Werte, die jeweils mit der Differenz des aktuellen Recall-Wertes zum Recall-Wert bei dem vorangegangenen Threshold multipliziert werden, wie in Tab. 3. [7]

$$Average\ Precision\ Score\ AP = \sum_{n} (R_n - R_{n-1}) * P_n$$

Tab. 3: Berechnung des Average Precision Score [7]

Der F1-Score ist ein anderes Bewertungsmaß für neuronale Netze, für das Precision und Recall kombiniert werden. Um diese Metrik anzugeben, wird das harmonische Mittel der beiden Werte angegeben, welches wie in Tab. 4 gezeigt berechnet wird. Da sich beide Werte zwischen Null und Eins bewegen, liegt auch der F1-Score in diesem Bereich und erreicht sein Optimum bei Eins. Die Verwendung des harmonischen Mittels führt dazu, dass ein sehr kleiner Wert mehr ins Gewicht fällt und nicht von einem sehr großen Wert ausgeglichen wird. [8]

$$F1 = \frac{2}{P^{-1} + R^{-1}} = \frac{2 * P * R}{P + R}$$

Tab. 4: Berechnung des F1-Scores [8]

Schließlich wurde eine Funktion implementiert, die das Training unter Variation der Hyperparameter automatisiert wiederholt. Die Ergebnisse der F1-Scores als Leitmetrik wurden abgespeichert und mithilfe einer PowerQuery-Abfrage in Excel geladen. Dort wurden das arithmetische und das harmonische Mittel der jeweils zwölf Werte gebildet und mithilfe von Farbskalen

die besten Hyperparameter abgelesen. Diese wurden gesondert erneut trainiert und verglichen, um so die optimale Hyperparameter-Konfiguration für das Netz zu finden.

2.5 Vor- und Nachverarbeitung von Anwendungsdaten

Ist das beste Netz gefunden und trainiert worden, könnten nun auch unbekannte Daten verarbeitet werden. Für die Klassifizierung von Objekten auf den Grundrissplänen muss allerdings zunächst die Iteration des 100*100 Pixel großen Eingabequadrates über den zu bearbeitenden Plan implementiert werden.

Wenn die Ergebnisse des Netzes für das Eingabequadrat vorhanden sind, werden sie zugeordnet zu dem Pixel auf dem Grundrissplan, der dem Pixel Y: 50; X: 50 im Eingabequadrat entspricht, abgespeichert. Nachdem dann alle Eingabequadrate verarbeitet worden sind, werden die Informationen aus den Netzausgaben extrahiert. Hierfür wird für jeden Pixel festgestellt, welcher der Wahrscheinlichkeitswerte am höchsten ist und der Pixel dieser Klasse zugeordnet. Dann wird überprüft, ob die Wahrscheinlichkeit, die der Pixel für seine Klasse beinhaltet, den Schwellwert überschreitet, ab dem davon ausgegangen werden kann, dass tatsächlich ein Objekt rund um den Pixel vorhanden ist. Der Schwellwert wird experimentell und mithilfe der Precision-Recall-Kurve bestimmt. Schließlich müssen die Ergebnisse noch nach der Sinnhaftigkeit überprüft werden. Einzelne Pixel, die keine Nachbarn derselben Klasse haben, können hier beispielsweise als Fehler gewertet und entsprechend gelöscht werden. Zum Schluss wird die Information aus den Ergebnissen in Bildform gebracht und kann so an den Anwender übermittelt werden. Wie man diese Möglichkeit der Vor- und Nachverarbeitung umsetzen könnte, ist in Anhang 9.4 dargestellt.

3 Ergebnisse

Da die Ergebnisse in einzelnen Test besser waren, wurde aus den beiden möglichen Optimizern ADAM[17] und SGD[18] letzterer ausgewählt. Durch weitere einzelne Tests wurden die besten beiden Netze, ResNet50 und InceptionResNetV2 bestimmt. Daraufhin wurden die Batch Size, die Lernrate und die Epochenanzahl variiert und die F1-Scores[19] abgespeichert.

Wie im Anhang 9.5 in Tabelle 1 und Tabelle 2 zu sehen, hat sich die Performance des ResNet50 durch das Entfernen der unterrepräsentierten Klasse „bidet“ verbessert. Die Werte stiegen zudem erneut, als die Vervielfältigung der Validierungsdaten, die nicht nötig ist, entfernt wurde, wie in Tabelle 3 zu erkennen. Des Weiteren zeigt sich im Vergleich von Tabelle 2 und Tabelle 4, dass das Netz InceptionResNetV2 bei denselben Einstellungen besser abschneidet als das ResNet50. Schließlich ist in Tabelle 5 zu sehen, dass wie beim ResNet50 auch die Entfernung der Vervielfältigung der Validierungsdaten und zudem die Einführung einer zufälligen Rotation der Trainingsdaten bei der Vervielfältigung die Werte deutlich anheben. Die besten Ergebnisse lieferte das InceptionResNet50 nach diesen Korrekturen in der Epoche 20 mit einer Batch Size von 128 und einer Lernrate von 0.01. Die Genauigkeit („Accuracy“) auf den Trainingsdaten betrug hier gerundet 0,999. Der durchschnittliche F1-Score auf den Validierungsdaten betrug hier 0,999103231 (in Tabelle 5 auf zwei Nachkommastellen gerundet). Die genauen Ergebnisse des Validierungsdatensatzes für allen Klassen sind in Tab. 5 zu sehen. Es ist zu erkennen, dass einzig Klasse 1 keinen perfekten F1-Score von 1,0 hat und auch der Average Precision Score nur bei Klasse 1 und Klasse 3 unter 1,0 liegt.

Klasse	0	1	2	3	4	5	6	7	8	9	10
F1	1	0,99	1	1	1	1	1	1	1	1	1
AP	1,00	0,99	1,0	0,99	1,0	1,0	1,0	1,0	1,0	1,0	1,0

Tab. 5: Ergebnisse des Validierungsdatensatzes bei dem besten Netz, einem InceptionResNetV2 (BS 128; LR 0,01; 20 Epochen)

[17] „adaptive moment estimation“, „an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments“ [12]; eigene Übersetzung: Adaptive Moment-Abschätzung, ein Algorithmus für Gradienten-basierte Optimierung erster Ordnung von stochastischen Zielfunktionen basierend auf adaptiven Schätzungen von Momenten geringer Ordnung

[18] „stochastic gradient descent“, eigene Übersetzung: stochastischer Gradientenabstieg; Basisalgorithmus zur Optimierung von neuronalen Netzen mithilfe des Gradientenabstiegs, der für die Kostenberechnung einen Teil des Datensatzes zufällig auswählt

[19] Aus Gründen der Übersichtlichkeit wurden alle Werte auf höchstens zwei Nachkommastellen gerundet. Für die Bestimmung des besten Netzes wurden mehr Nachkommastellen berücksichtigt.

In Abb. 7 ist zu erkennen, dass auch die Konfusionsmatrix der Validierungsdaten die erwünschte Diagonale aufweist. Es wird also jede Klasse korrekt vorhergesagt. Des Weiteren zeigt die Precision-Recall-Kurve in Abb. 8, dass selbst für einen Schwellwert von beinahe 1,0 noch sowohl Precision als auch Recall aller Klassen bei 1,0 liegen.

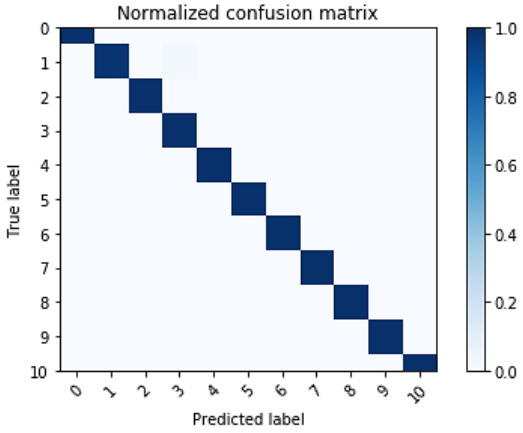


Abb. 7: Konfusionsmatrix aus dem Training mit dem besten Netz bei den optimalen Hyperparametern

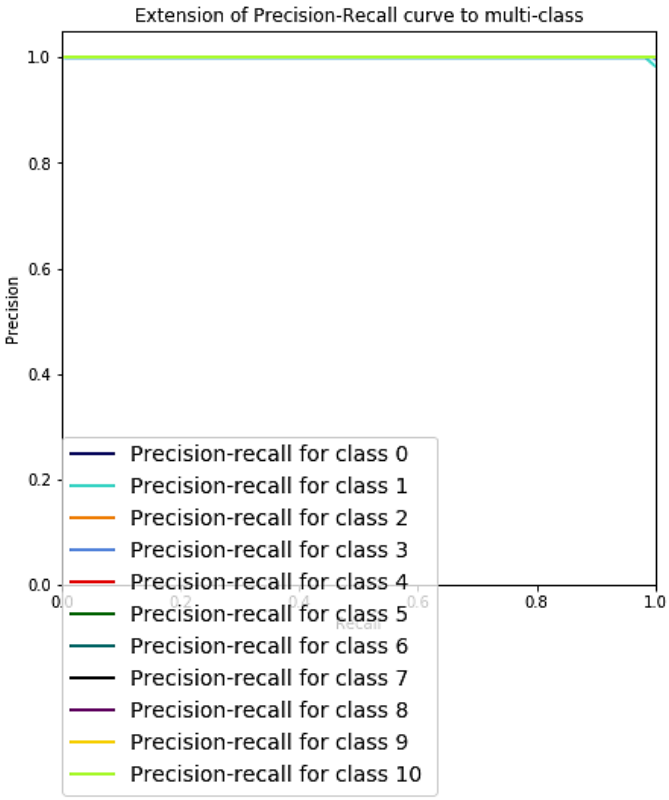


Abb. 8: Precision-Recall-Kurve aus dem Training mit dem besten Netz bei den optimalen Hyperparametern

Das Netz wurde auch auf die Testdaten angewandt. Hier fielen die Ergebnisse erwartungsgemäß etwas niedriger aus, da die gekrümmten Wände dem Netz unbekannte Strukturen sind. Trotz dieser Schwierigkeit erreichte das Netz aber für manche Klassen F1-Scores von zwischen 0,8 und 0,9, wie in **Fehler! Verweisquelle konnte nicht gefunden werden.** zu sehen ist. Der niedrigste Wert ist 0,33. Der Average-Precision-Score lag, wie in **Fehler! Verweisquelle konnte nicht gefunden werden.** zu erkennen, bei mehreren Werten über 0,9 und das Minimum betrug 0,39.

Klasse	0	1	2	3	4	5	6	7	8	9	10
F1	0.77	0.68	0.89	0.78	0.46	0.84	0.68	0.7	0.33	0.4	0.81
AP	0.87	0.81	0.98	0.90	0.61	0.92	0.75	0.81	0.39	0.57	0.91

Tab. 6: Ergebnisse des besten Netzes auf den Testdaten

In Abb. 9 ist die Konfusionsmatrix der Ergebnisse auf den Testdaten zu sehen. Es ist zu erkennen, dass unter anderem die Klassen 2, 3 und 10 auch in den Testdaten sehr gut erkannt wurden, während vor allem bei den Klassen 4, 8 und 9 der Transfer des Gelernten weniger gut funktioniert hat. In Abb. 10 ist zudem die zugehörige Precision-Recall-Kurve dargestellt. Darauf ist zu sehen, dass die Klassen 2, 3 und 10 gut abschneiden, was sich mit den Ergebnissen der Konfusionsmatrix deckt. Zudem ist zu sehen, dass die Klassen 4, 8 und 9 genau wie in der Matrix auch in der Kurve am schlechtesten sind.

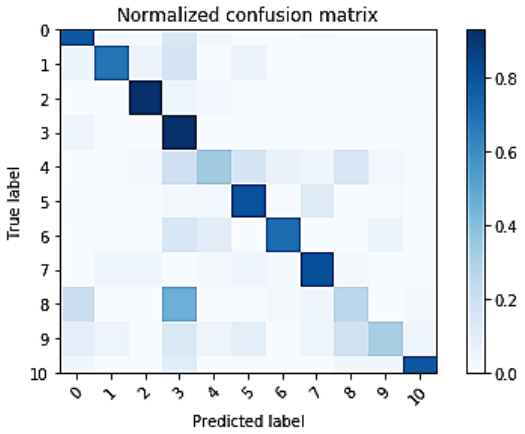


Abb. 9: Konfusionsmatrix der Ergebnisse auf den Testdaten

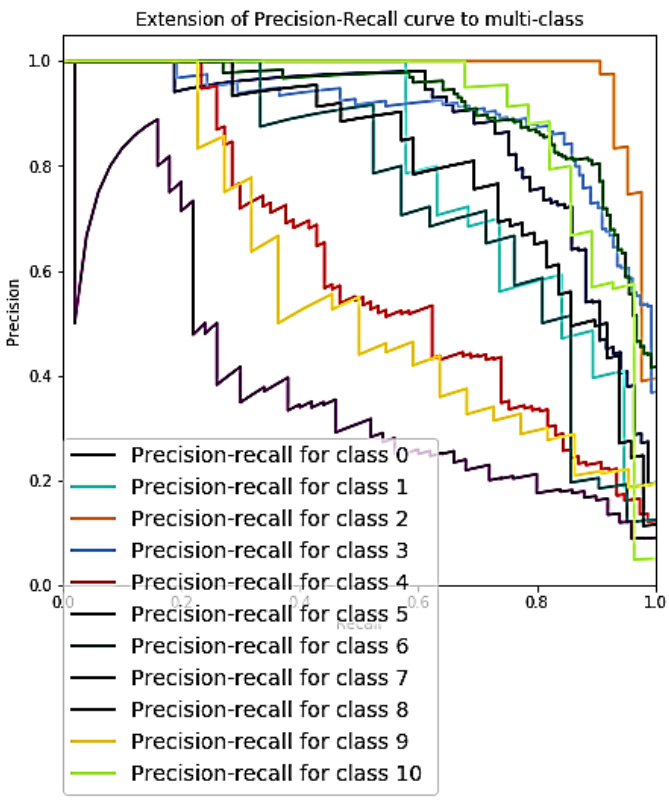


Abb. 10: Precision-Recall-Kurve der Ergebnisse des besten Netzes auf den Testdaten

4 Fehleranalyse und Deutung der Ergebnisse

Zunächst wurde festgestellt, dass darauf geachtet werden muss, bei jedem Training des Netzes die Gewichte neu zu initialisieren, da sonst keine sinnvollen Ergebnisse erzielt werden können. Des Weiteren stellte es sich als hilfreich für das Netz heraus, die Validierungsdaten nicht zu vervielfältigen, da dabei sonst bereits schwierigere und unbekannte Daten noch komplexer werden. Eine erfolgreiche Vereinfachung stellte jedoch dar, die Trainingsdaten zufällig zu rotieren statt nur in 45°-Schritten. Dadurch wurden die Unterschiede zwischen Trainings- und Validierungsdatensatz geringer und das Netz musste weniger unbekannte Eigenschaften in den Validierungsdaten verarbeiten.

Dass das InceptionNetV2 durchschnittlich ein wenig besser abschneidet als das ResNet50, könnte auf die tiefere Netzarchitektur mit mehr trainierbaren Parametern zurückzuführen sein, da so mehr Strukturen trainiert werden können.

Bezüglich der Hyperparameter beim Training des InceptionNetV2 ohne Vervielfältigung der Validierungsdaten und mit zufälliger Rotation der Trainingsdaten lässt sich aus den Ergebnissen in Anhang 9.5 Tabelle 5 ableiten, dass kleinere Batch Sizes wie 32 und 64 weder mit sehr großen noch sehr kleinen Lernraten besonders gut funktionieren und hier eher die Mitte mit

0,01 und 0,001 gewählt werden sollte. Für die größte mögliche Batch Size von 128 stellten sich größere Lernraten von 0,01 und 0,1 als erfolgreicher heraus und diese Batch Size funktionierte mit der Lernrate 0,01 auch am besten von allen. Größere Lernraten waren aufgrund von fehlendem Arbeitsspeicher in der Grafikkarte nicht mehr möglich. Dass die größte mögliche Batch Size am besten funktioniert, deckt sich mit der Aussage des Buches „Hands-On Machine Learning“, „many researchers and practitioners recommend using the largest batch size that can fit in GPU RAM“ as „hardware accelerators like GPU can process them efficiently“[20] [9].

Die hohe Genauigkeit des besten InceptionNetV2 auf den Trainingsdaten ist erfreulich, für sich aber wenig aussagekräftig. Die ebenfalls hohen Werte auf den Validierungsdaten hingegen zeigen, dass das Netz sehr verallgemeinert trainiert hat und wirklich gute Vorhersagen über die Objektkategorien treffen kann. Auch die Ergebnisse auf dem Testdatensatz weisen auf ein erfolgreiches, verallgemeinertes Training hin. Insbesondere, wenn die im Vergleich zum Validierungsdatensatz noch weiter erhöhte Schwierigkeit bedacht wird, ist es ein sehr gutes Ergebnis, dass mehrere Klassen F1-Scores von über 0,7 bis 0,8 erreicht haben. An den Werten der Testdaten wird zudem ersichtlich, dass der F1-Score eine eher kritische Metrik ist, da der Average Precision Score immer etwas höher ausfiel. Auch deshalb sind die Werte auf Validierungs- und Testdatensatz sehr zufriedenstellend.

Ein Grund, weshalb manche Klassen bessere Werte lieferten als andere, könnte die variierende Datenmenge sein. Objekttypen, für die eine geringere Anzahl und damit auch weniger verschiedene Symbole im Training vorhanden waren, können weniger verallgemeinert und erfolgreich trainiert werden und sind daher gerade im schwierigeren Testdatensatz oft etwas weniger gut vom Netz zu erkennen als Objekttypen mit vielen und vielfältigen Daten. Des Weiteren könnten die Symbole bei den etwas weniger guten Klassen größere Varianz in der Darstellung und auch in der absoluten Größe aufweisen, wodurch es für das Netz schwerer ist, es auf Gemeinsamkeiten zu reduzieren.

[20] Eigene Übersetzung: Viele Wissenschaftler und Anwender empfehlen die Nutzung der größten Batch Size, die in den GPU RAM passt, da Hardware-Beschleuniger wie die GPU diese dann effizient verarbeiten können.

5 Zusammenfassung und Ausblick

5.1 Erweiterte Fragestellungen

Die Erkennung von Objekten in Grundrissplänen ist nur ein Teil der Dokumentenanalyse. Das übergeordnete Ziel ist, alle Informationen, die der Mensch auf einem Grundrissplan erkennen kann, für den Computer und Programme nutzbar und so digital verarbeitbar zu erfassen. Dazu gehören bei den Grundrissplänen neben den Objekten auch die Bestimmung des Raumtyps anhand des Objektvorkommens und dafür auch die Einteilung in Räume durch die Detektion von Wänden und Türen. Dieser Teil der Grundrissplananalyse wird beispielsweise in dem Paper „Deep Floor Plan Recognition Using a Multi-Task Network with Room-Boundary-Guided Attention“ von der Chinese University of Hong Kong untersucht und die Ergebnisse könnten mit den Ergebnissen dieses Projekts kombiniert werden[21] [10]. Des Weiteren können Maße und Größenverhältnisse im Plan erfasst werden. Auch auf dem Bild befindliche Texte sollten zudem eingelesen und in einem weitaus anspruchsvolleren Schritt nicht nur als Text, sondern als Information verarbeitet werden.

5.2 Anwendung und Nutzen des Projekts

Die Bestimmung von Objekten und deren Positionen in Grundrissplänen ist also mit einem neuronalen Netz vom Typ InceptionResNetV2 in einem zweiteiligen Ansatz möglich und kann vor allem im Bereich der Einrichtung und Vermietung von Wohnungen verwendet werden. So kann beispielsweise die Einrichtung einer Wohnung vereinfacht werden, indem die gewonnenen Informationen in eine Empfehlung für eine Zusammenstellung von Möbelstücken und Einrichtungsgegenständen des gewünschten Anbieters oder Herstellers entsprechend eines bestimmten Stils oder einer Preisvorgabe umgesetzt werden. Auf der Basis dieser Empfehlung können dann Anpassungen gemacht werden, bis der Kunde vollständig zufrieden ist. Dadurch würde die Dauer bis zur fertigen Einrichtung verkürzt, die Kundenzufriedenheit erhöht und die Arbeit eines Einrichtungsanbieters erleichtert werden.

Auch bei der Vermietung kann eine höhere Zufriedenheit des Mieters und gleichzeitig eine schnellere Verarbeitung der Wünsche erreicht werden. Wenn Vermieter Grundrisspläne ihrer möblierten Immobilien mithilfe des neuronalen Netzes analysieren und die Informationen bei einem Vermietungsportal einreichen, können Kunden schneller die perfekte Wohnung finden, da auch nach der Einrichtung gefiltert werden kann.

Wird die Objekterkennung als Teil einer detaillierteren Grundrissplananalyse eingesetzt, eröffnen sich noch weitere Anwendungsfelder. So könnte beispielsweise die Analyse von Grundrissen von Wohnhäusern, Bürogebäuden, Schulen und anderen öffentlichen Einrichtungen als Grundlage für eine computergestützte Bestimmung des besten Fluchtweges oder Zugangsweges für Rettungskräfte genutzt werden. Die Informationen des Grundrissplanes helfen hier, Engstellen durch bestimmte Möbelanordnungen und Türdurchgänge zu erkennen und den kürzesten Weg von jedem Standpunkt zu entwerfen.

Auch die Brandschutzmaßnahmen könnten mithilfe von Computern leichter optimiert werden, da mithilfe von Informationen über die Positionen von kritischen Objekten wie dem Herd und über die Raumtypen Simulationen der Ernstfallszenarien vereinfacht werden können. So können Probleme im alten Brandschutzsystem erkannt und Verbesserungsvorschläge gemacht oder ein von Grund auf neues System vom Computer entworfen werden.

[21] Der Fokus dieses Projekts lag auf der Erkennung von Wänden, Fenstern, Türen und Räumen und deren Typen.

Schließlich kann der Computer mithilfe der Grundrissplananalyse auch bei der effizienten Verlegung von Strom- und Wasserleitungen unterstützen, da die Wände und die Positionen von Waschbecken, Duschen und elektrischen Geräten, welche hierfür relevant sind, erkannt werden und die Informationen digital verarbeitbar zur Verfügung stehen. Dadurch kann Material gespart werden und auch die Arbeitseffizienz nimmt zu.

5.3 Fazit

Für die Erkennung von Möbelsymbolen in Grundrissplänen war das Faltungsnetz InceptionNetV2 sehr gut geeignet, da es nach angepasstem Training die Fähigkeit zur Verallgemeinerung hat, die aufgrund der vielen verschiedenen Varianten eines Symbols nötig ist. Bei der richtigen Vorverarbeitung der Daten kann das Netz dann sehr zuverlässig Objektsymbole auf den Plänen erkennen, solange diese in ihren Eigenschaften nicht zu sehr von den Trainingsdaten abweichen. Ein solches oder ähnliches Netz kann dann eigenständig oder als Teil eines größeren Systems vielseitig eingesetzt werden. Es kann bei der Einrichtung und Vermietung von Wohnungen und ebenso bei der Sicherheitstechnik und der Verlegung von Leitungen die Zuständigen bei ihrer Arbeit unterstützen und den Komfort und die Effizienz für Dienstleister und Kunden nachhaltig erhöhen.

6 Danksagung

Ich möchten mich bei allen bedanken, die meine Arbeit am Projekt unterstützt haben. In erster Linie gilt mein Dank Christoph Balada und Dr. Stefan Agne vom Deutschen Forschungszentrum für künstliche Intelligenz in Kaiserslautern, die mir dieses Projekt und damit einzigartige Erfahrungen ermöglicht haben und mich inhaltlich und organisatorisch bei allen Fragen unterstützt haben. Zudem möchte ich mich bei Herrn Prof. Dr. Prof. h.c. Andreas Dengel dafür bedanken, dass er meine Bewerbung weitergegeben und so die Durchführung meiner Kooperationsphase an seinem Institut möglich gemacht hat.

Dr. Hans-Werner und Josephine Hector danke ich für die langjährige Förderung im Hector-Seminar und die Möglichkeit, wissenschaftliche Arbeit hautnah zu erleben. Schlussendlich möchte ich mich bei meinen Kursleitern Herr Dr. Götz und Herr Oehme bedanken, die mir jederzeit bei Fragen und insbesondere beim Überarbeiten der Dokumentation beratend zur Seite standen.

7 Selbständigkeitserklärung

Ich versichere, dass ich diese schriftliche Projektarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass ich alle Stellen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, durch Angabe der Quellen als Entlehnung kenntlich gemacht habe.

Königsbach-Stein, den 17.07.2020 Annika Nassal

Unterschrift

8 Quellenverzeichnis

[1] ICDAR, *Grundrissplan-Datensatz,* 2019.

[2] I. Goodfellow, Y. Bengio und A. Courville, Deep Learning, Cambridge, MA: Massachusetts Institute of Technology, 2016.

[3] A. Ng, „Machine Learning Yearning," 2018. [Online]. Available: https://www.deeplearning.ai/machine-learning-yearning/. [Zugriff am 10 07 2020].

[4] „Confusion matrix," 2019. [Online]. Available: https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py. [Zugriff am 12 Juli 2020].

[5] „sklearn.metrics.precision_recall_curve," 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_curve.html?highlight=precision%20recall%20curve#sklearn.metrics.precision_recall_curve. [Zugriff am 12 Juli 2020].

[6] W. Kohersen, „towards data science," 03 03 2018. [Online]. Available: https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c. [Zugriff am 03 Juli 2020].

[7] „sklearn.metrics.average_precision_score," 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html. [Zugriff am 11 Juli 2020].

[8] „sklearn.metrics.f1_score," 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html. [Zugriff am 11 Juli 2020].

[9] A. Géron, Hands-On Machine Learning with ScikitLearn, Keras & Tensorflow, 2 Hrsg., Canada: O'Reilly Media, 2019.

[10] Z. Zeng , X. Li, Y. K. Yu und C.-W. Fu, The Chinese University of Hong Kong, 2019. [Online]. Available: https://openaccess.thecvf.com/content_ICCV_2019/papers/Zeng_Deep_Floor_Plan_Recognition_Using_a_Multi-Task_Network_With_Room-Boundary-Guided_ICCV_2019_paper.pdf. [Zugriff am 12 Juli 2020].

[11] J. Brownlee, „A Gentle Introduction to the ImageNet Challenge (ILSVRC)," 01 Mai 2019. [Online]. Available: https://machinelearningmastery.com/introduction-to-the-imagenet-large-scale-visual-recognition-challenge-ilsvrc/. [Zugriff am 16 07 2020].

[12] D. P. Kingma und J. L. Ba, „Adam: A Method for Stochastic Optimization," 2015. [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf?source=post_page----------. [Zugriff am 16 07 2020].

9 Anhang

9.1 Quellcode Metadatenbereinigung

```
#Imports left out
###START LOADING DATA
script_dir = os.path.dirname(__file__) #<-- absolute dir the script is in
rel_path = "../Flurplandaten/flo2plan_icdar_instances.json"
new_path = os.path.join(script_dir,
"../Flurplandaten/floorplan_metadata_cleaned_nobidet.json")
metadata_path = os.path.join(script_dir, rel_path)

#Load json file
metadata = json.loads(open(metadata_path, 'r').read())

#List for images
imgs = []

#Dict for pairing of names and idices of images
imgs_nametoind = {}

#Path for accessing images
rel_path = "../Flurplandaten/images"
path = os.path.join(script_dir, rel_path)

#Counter to get index of image
count = 0

#Loop through elements (-->e) in directory
for e in listdir(path):
    #Get the element's link
    link = join(path, e)
    #Load image and transform it to B&W numpy array
    curr_img = np.array(PIL.Image.open(link).convert('L'))
    #Add array-shaped image to list
    imgs.append(curr_img)

    imgs_nametoind[e] = count

    count+= 1

#Dict for pairing indices (meaning the place of the image in the list with
the images) and ids (meaning the number adressing this image in the metadata)
of images
imgs_idtoind = {}

#Loop through all image entries (-->ie) in metatdata
for ie in metadata['images']:
    #Check if image exists in imported image list
    if ie['file_name'] in imgs_nametoind:
        #Get the index to the current file name
        index = imgs_nametoind[ie['file_name']]
        #Get the id of this file
        img_id = ie['id']
        #Link id to the index
        imgs_idtoind[img_id] = index
###END LOADING DATA



###START ACTUAL CLEANING
n = 0

#Check all annotations
while n < len(metadata['annotations']):

    image_id = metadata['annotations'][n]['image_id']

    #Check if image exists
```

```
    if image_id in imgs_idtoind:

        #Get image data
        bbox = metadata['annotations'][n]['bbox']
        img_width = metadata['images'][imgs_idtoind[image_id]]['width']
        img_height = metadata['images'][imgs_idtoind[image_id]]['height']

        #Delete image if it is larger than 100*100 or smaller than 10*10 or
        #if it is too close to the edges of the floorplan
        if (bbox[3] < 10) or (bbox[2] < 10) or (bbox[3] > 100) or (bbox[2]
> 100) or (img_height < (bbox[1] + bbox[3])) or (img_height < bbox[3])or
(img_width < (bbox[0]+bbox[2])) or (img_width < bbox[2]):
            del metadata['annotations'][n]

        #Overwrite all bidets as toilets and let other classes follow one
down
        if(metadata['annotations'][n]['category_id'] == 5):
            metadata['annotations'][n]['category_id'] = 1
        elif(metadata['annotations'][n]['category_id']>5):
            metadata['annotations'][n]['category_id']                    =
metadata['annotations'][n]['category_id'] - 1
    n += 1

x = 0

#Delete bidet class, let other classes follow one down
while x < len(metadata['categories']):
    if(metadata['categories'][x]['id'] == 5):
        del metadata['categories'][x]
    if(metadata['categories'][x]['id'] > 5):
        metadata['categories'][x]['id'] = metadata['categories'][x]['id'] -
1

    x += 1

print(metadata['categories'])

###END CLEANING

###WRITE INTO FILE AND SAVE
with open(new_path, 'w') as f:
        json.dump(metadata, f)
```

9.2 Quellcode Datenvorverarbeitung

```
#Imports left out
random.seed()


###START FUNCTION SECTION
def getNormalizedNumbersOfAugmentation(old_numbers):
    average = np.sum(old_numbers)/len(old_numbers)
    augmentation_numbers = np.zeros(len(old_numbers))
    for i in range(len(old_numbers)):
        if(average-60 < old_numbers[i] < average+60):
            augmentation_numbers[i] = 20
        elif(average-60 > old_numbers[i]):
            augmentation_numbers[i] = 20*(average-60)/old_numbers[i]
        elif(average+60 < old_numbers[i]):
            augmentation_numbers[i] = 20*(average+60)/old_numbers[i]
    return augmentation_numbers
```

```
def trim(value, min, max):
    #Keeps value between min and max
    if(value<min):
        return min
    elif(value>max):
        return max
    else:
        return value
def invert(img):
    #inverts an image over 255
    inverted_image = 255 - img
    return inverted_image
def StandardAnnotation(img, bound):
    #Cuts out the annotation as the centre of a 100x100 image
    #Get x-coordinate of upper left corner rounded down
    annot_x1 = trim(int((bound[0]) - (100 - bound[2])/2), 0, img.shape[1]-
100)
    #Get y-coordinate of upper left corner rounded down
    annot_y1 = trim(int((bound[1]) - (100 - bound[3])/2), 0, img.shape[0]-
100)
    #Get width and height
    annot_width = 100
    annot_height = 100
    #Calculate x-coordinate of lower right corner
    annot_x2 = annot_x1 + annot_width
    #Calculate y-coordinate of lower right corner
    annot_y2 = annot_y1 + annot_height
    return img[annot_y1:annot_y2, annot_x1:annot_x2]


def Offset2dAnnotation(img, bound):
    # Calculate actual coordiantes of upper left corner of 100x100 area
    corner_100x100_no_offset_x = int(((bound[0])) - (100 - bound[2])/2)
    corner_100x100_no_offset_y = int(((bound[1])) - (100 - bound[3])/2)
    #Get offset (min half annotation should be still in 100x100)
    offset_x = random.randint(-50, 50)
    offset_y = random.randint(-50, 50)
    #Get coordinates of upper left corner (100x100) with offset
    corner_100x100_offset_x1 = trim(corner_100x100_no_offset_x + offset_x,
0, img.shape[1]-100)
    corner_100x100_offset_y1 = trim(corner_100x100_no_offset_y + offset_y,
0, img.shape[0]-100)
    #Set width and height
    width = 100
    height = 100
    #Get coordinates of lower right corner (100x100)
    corner_100x100_offset_x2 = corner_100x100_offset_x1 + width
    corner_100x100_offset_y2 = corner_100x100_offset_y1 + height
```

```
    result     =     img[corner_100x100_offset_y1:corner_100x100_offset_y2,
corner_100x100_offset_x1:corner_100x100_offset_x2]
    if(result.shape[0] != 100 or result.shape[1] != 100):
        print("Size issue")
    return result
def RotateAnnotation(img, bound):
    #Rotation of offset or standard annotation
    case = random.randint(0, 1)
    if case == 0:
        rot_img = StandardAnnotation(img, bound)
    else:
        rot_img = Offset2dAnnotation(img, bound)
    #Get the angle (*45°) to rotate the image with
    angle = random.randint(1, 360)
    #Save the original scale of the image (in case it was not 100x100)
    orig_height = rot_img.shape[0]
    orig_width = rot_img.shape[1]
    #Rotate the image
    rotated = invert(imutils.rotate_bound(invert(rot_img), angle))
    #Crop the image back to 100*100
    y1 = int(abs((rotated.shape[0]-orig_height)/2))
    y2 = y1 + 100
    x1 = int(abs((rotated.shape[1]-orig_width)/2))
    x2 = x1 + 100
    result = rotated[y1:y2, x1:x2]
    if(result.shape[0] != 100 or result.shape[1] != 100):
        print("Size issue")
    return result
#Skalierung mit opencv.resize, interpolation cubic
def RescaleAnnotation(img, bound):
    #Cut the annotation with or without offset
    case = random.randint(0, 1)
    if case == 0:
        rot_img = StandardAnnotation(img, bound)
    else:
        rot_img = Offset2dAnnotation(img, bound)
    #Save the original size of the image (in case it is not 100)
    orig_height = rot_img.shape[0]
    orig_width = rot_img.shape[1]
    #Get the scale factor
    scale = random.randint(11, 14)/10

    #Scale the image with the factor
    scale_img      =      cv2.resize(rot_img,      (int(orig_height*scale),
int(orig_width*scale)), interpolation = cv2.INTER_CUBIC)
```

```
    #Cut the scaled image back to 100*100
    y1 = int(abs((scale_img.shape[0]-orig_height)/2))
    y2 = y1 + 100
    x1 = int(abs((scale_img.shape[1]-orig_width)/2))
    x2 = x1 + 100
    result = scale_img[y1:y2, x1:x2]
    if(result.shape[0] != 100 or result.shape[1] != 100):
        print("Size issue")
    return result
def HotKeyEncode(int_list, num_categories):
    HotKeyList = []
    for element in int_list:
        hotkey = np.zeros(num_categories)
        hotkey[element-1] = 1
        HotKeyList.append(hotkey)
    return HotKeyList
###END FUNCTION SECTION
###START LOADING DATA
training_annotations_per_category = [185, 67, 70, 318, 161, 381, 46, 77,
145, 63, 78]
augmentation_numbers_for_training =
getNormalizedNumbersOfAugmentation(training_annotations_per_category)
validation_annotations_per_category = [182, 60, 61, 313, 123, 296, 48, 97,
107, 39, 66]
augmentation_numbers_for_validation =
getNormalizedNumbersOfAugmentation(validation_annotations_per_category)
script_dir = os.path.dirname(__file__) #<-- absolute dir the script is in
rel_path = "../Flurplandaten/floorplan_metadata_cleaned_nobidet.json"
metadata_path = os.path.join(script_dir, rel_path)
#Load json file
metadata = json.loads(open(metadata_path, 'r').read())
#List for images
imgs = []
#Dict for pairing of names and idices of images
imgs_nametoind = {}
#Path for accessing images
rel_path = "../Flurplandaten/images_roundings"
path = os.path.join(script_dir, rel_path)
#Counter to get index of image
count = 0
#Loop through elements (-->e) in directory
for e in listdir(path):
    #Get the element's link
    link = join(path, e)
    #Load image and transform it to B&W numpy array
    curr_img = np.array(PIL.Image.open(link).convert('L').convert('RGB'))
```

```
    #Add array-shaped image to list
    imgs.append(curr_img)
    imgs_nametoind[e] = count
    count+= 1
#Dict for pairing indices (meaning the place of the image in the list with the images)
#and ids (meaning the number adressing this image in the metadata) of images
imgs_idtoind = {}
#Loop through all image entries (-->ie) in metatdata
for ie in metadata['images']:
    #Check if image exists in imported image list
    if ie['file_name'] in imgs_nametoind:
        #Get the index to the current file name
        index = imgs_nametoind[ie['file_name']]
        #Get the id of this file
        img_id = ie['id']
        #Link id to the index
        imgs_idtoind[img_id] = index
###END LOADING DATA
###START PREPROCESSING
#List of images
annotations = []
#List of objects in annotations
object_categories = []
#Loop through all annotations
for annot in metadata['annotations']:
    #Get bounding box
    bbox = annot['bbox']
    #Get image id
    img_id = annot['image_id']
    #Get object on annotation
    object_category = annot['category_id']
    #Check if the image to the id exists
    if img_id in imgs_idtoind:
        #Get the image
        annot_img = imgs[imgs_idtoind[img_id]]
        #Add current annotation to list
        annotations.append(StandardAnnotation(annot_img, bbox))
        object_categories.append(object_category)

        #Create 20 varied Verions of this Annotation (only for training annotations, otherwise comment)
        for i in range(20):
            #Type of augmentation
            augmentation = random.randint(0, 2)
```

```
            #Offset only
            if augmentation == 0:
                aug_annot = Offset2dAnnotation(annot_img, bbox)
                annotations.append(aug_annot)
                object_categories.append(object_category)
            #Rotate
            elif augmentation == 1:
                aug_annot = RotateAnnotation(annot_img, bbox)
                annotations.append(aug_annot)
                object_categories.append(object_category)
            #Rescale
            elif augmentation == 2:
                aug_annot = RescaleAnnotation(annot_img, bbox)
                annotations.append(aug_annot)
                object_categories.append(object_category)
script_dir = os.path.dirname(__file__)
annotation_path = os.path.join(script_dir,
"../Flurplandaten/preprocessed__test_annotations_nobidet.p")
pickle.dump(annotations, open(annotation_path, "wb"))
object_categories_encoded = HotKeyEncode(object_categories, 11)
object_path = os.path.join(script_dir,
"../Flurplandaten/object_list_for_test_annotations_nobidet.p")
pickle.dump(object_categories_encoded, open(object_path, "wb"))
###END PREPROCESSING
###NOTES:
#BBox: Wert in der Breite, Wert in der Höhe, Breite, Höhe
```

9.3 Quellcode Training des Netzes

```
#Imports left out
gpu_options = tf.GPUOptions(allow_growth=True)
session =
tf.InteractiveSession(config=tf.ConfigProto(gpu_options=gpu_options))
###GLOBAL VARIABLES
new_path = os.path.join(os.path.dirname(__file__),
"../Formal/f1_scores_automated_training_9_nobidet_IncResV2_randomrotation.j
son")
error_path =
os.path.join(os.path.dirname(__file__),"../Flurplandaten/FalsePredictions")
all_categories = np.array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                           [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                           [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
                           [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
                           [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                           [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
                           [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
                           [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
                           [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
```

```
                            [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
                            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]])
trainResults = {}
batchSizes = [32, 64, 128, 256]
learnRates = [0.1, 0.01, 0.001, 0.0001]
###FUNCTION SECTION
#Create array of zeros and one one from array of probabilities
def standartize(input_array):
    output_array = []
    for array in input_array:
        output_part = np.zeros(11)
        highest = np.argmax(array, axis = 0)
        output_part[highest] = 1
        output_array.append(output_part)
    return np.array(output_array)
#Plot confusion matrix
def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(11)
    plt.xticks(tick_marks, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
rotation=45)
    plt.yticks(tick_marks, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()


#Plot precision recall curve
def plot_precision_recall_curve(recall, precision):
    # setup plot details
    colors = cycle(['navy', 'turquoise', 'darkorange', 'cornflowerblue',
'red', 'green', 'teal', 'black', 'purple', 'gold', 'greenyellow',
'deeppink'])
    plt.figure(figsize=(7, 8))
    lines = []
    labels = []
    for i, color in zip(range(11), colors):
        l, = plt.plot(recall[i], precision[i], color=color, lw=2)
        lines.append(l)
        labels.append('Precision-recall for class {0} '.format(i)
    fig = plt.gcf()
    fig.subplots_adjust(bottom=0.25)
    plt.xlim([0.0, 1.0])
```

```
    plt.ylim([0.0, 1.05])
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Extension of Precision-Recall curve to multi-class')
    plt.legend(lines, labels, loc=(0, -.38), prop=dict(size=14))
    plt.show()
#Instatiate model
def getModel():
    #Get model
    #InceptionResnetV2
    base_model = keras.applications.InceptionResNetV2(weights = 'imagenet',
include_top = False, classes = 11, input_shape = (100,100,3))
    x = keras.layers.Flatten()(base_model.output)
    predictions = keras.layers.Dense(11, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)
    return model
#Callback that is called on beginning and end of epoch and on end of training
for evaluation with validation and test dataset
class evaluationCallback(keras.callbacks.Callback):
    #Initialise batch size and learn rate variables
    def __init__(self, currentBatchSize, currentLearnRate):
        self.currentBatchSize = currentBatchSize
        self.currentLearnRate = currentLearnRate
    #Give epoch overview
    def on_epoch_begin(self, epoch, log = None):
        print("Starting epoch {}".format(epoch+1))
    #Evaluate with validation dataset
    def on_epoch_end(self, epoch, log = None):
        global validation_categories
        print("Learn Rate: {}; Batch Size: {}".format(self.currentLearnRate,
self.currentBatchSize)
        #Result of net - exact probabilitites
        result = model.predict(validation_annot_array)
        #Result of net - numbers of the predicted classes
        predicted_result = np.argmax(result, axis=1)
        #Result of net - binary with eleven zeros and one one
        standartized_predicted_result = standartize(result)
        #True result - numbers of the true classes
        true_result = np.argmax(validation_categories, axis=1)
        #Get all annotations that were not predicted correctly to find
pattern
        mask = predicted_result==true_result
        current_error_path = os.path.join(error_path,
"{},{},{}".format(self.currentBatchSize, self.currentLearnRate, epoch))
        if(not os.path.exists(current_error_path)):
            os.makedirs(current_error_path)
```

```
        for number, annotation in enumerate(validation_annot_array[~mask]):
            plt.imsave(os.path.join(current_error_path,
"wrong_annotation{}.png".format(number)), annotation)
            print(current_error_path)
        #Confusion matrix
        cm = metric.confusion_matrix(true_result, predicted_result)
        np.set_printoptions(precision=2)
        cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        plt.figure()
        plot_confusion_matrix(cm_normalized, title='Normalized confusion
matrix')
        #F1-Score
        f1 = metric.f1_score(true_result, predicted_result, average = None)
        print("F1-Score is {}".format(f1))
        #Save results and errors only for automated training
        if(train_mode == "Y"):
            #Error image path
            current_error_path = os.path.join(error_path,
"{},{},{}".format(self.currentBatchSize, self.currentLearnRate, epoch))
            if(not os.path.exists(current_error_path)):
                os.makedirs(current_error_path)
            #Save error images
            for number, annotation in
enumerate(validation_annot_array[~mask]):
                plt.imsave(os.path.join(current_error_path,
"wrong_annotation{}.png".format(number)), annotation)
                print(current_error_path)
            #Save training results for evaluation
            with open(new_path, 'w') as path:
                trainResults["SGD: {},{}, Epoch:
{}".format(self.currentBatchSize, self.currentLearnRate, epoch)] =
f1.tolist()
                json.dump(trainResults, path)
        #Save info for precision-recall-curve
        precision = dict()
        recall = dict()
        thresholds = dict()
        #Category-specific metrics
        for current_category in all_categories:
            category_number = np.argmax(current_category, axis = 0)
            #Array of 0s & one 1 showing which true results are of the
current category (1)
            y_true_aktuell = np.array([np.array_equal(current_category, t)
for t in validation_categories], dtype=np.uint8)
            #Result of net - exact probabilities for one class
            y_pred_aktuell = result[:, category_number]
            #Expecting only probabilities for current class as 1-dim vector
```

```
            precision[category_number], recall[category_number],
thresholds[category_number] = metric.precision_recall_curve(y_true_aktuell,
y_pred_aktuell)
            average_prec = metric.average_precision_score(y_true_aktuell,
y_pred_aktuell)
            print('Average Precision of class {} is
{}'.format(category_number, average_prec))
        plot_precision_recall_curve(recall, precision)
    #Evaluate net with test dataset
    def on_train_end(self, log = None):
        print("We are now checking the performance on the test dataset!")
        test_result = model.predict(test_annot_array)
        test_predicted_result = np.argmax(test_result, axis=1)
        #True result - numbers of the true classes
        test_true_result = np.argmax(test_categories, axis=1)
        #Confusion matrix
        cm = metric.confusion_matrix(test_true_result,
test_predicted_result)
        np.set_printoptions(precision=2)
        cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        plt.figure()
        plot_confusion_matrix(cm_normalized, title='Normalized confusion
matrix')
        #F1-Score
        f1 = metric.f1_score(test_true_result, test_predicted_result,
average = None)
        print("F1-Score is {}".format(f1))
        #Save info for precision-recall-curve
        precision = dict()
        recall = dict()
        thresholds = dict()
        #Category-specific metrics
        for current_category in all_categories:
            category_number = np.argmax(current_category, axis = 0)
            #Array of 0s & one 1 showing which true results are of the
current category (1)
            y_true_aktuell = np.array([np.array_equal(current_category, t)
for t in test_categories], dtype=np.uint8)
            #Result of net - exact probabilities for one class
            y_pred_aktuell = test_result[:, category_number]
            #Expecting only probabilities for current class as 1-dim vector
            precision[category_number], recall[category_number],
thresholds[category_number] = metric.precision_recall_curve(y_true_aktuell,
y_pred_aktuell)
            average_prec = metric.average_precision_score(y_true_aktuell,
y_pred_aktuell)
            print('Average Precision of class {} is
{}'.format(category_number, average_prec))
        plot_precision_recall_curve(recall, precision)
```

```
#Automated training of the net with varying batchSize and learn rate
def trainNet(train_annot_array, train_categories):
    global model
    for batchSize in batchSizes:
        for learnRate in learnRates:
            #Reinstantiate model
            model = getModel()
            #Choose optimizer
            opti = keras.optimizers.SGD(lr = learnRate)
            #Compile model
            model.compile(optimizer = opti, loss =
"categorical_crossentropy", metrics=["accuracy"])
            print(model.summary())
            print("We have batch size {} and learn rate {}".format(batchSize,
learnRate))
            #Actual training with callback for evaluation
            model.fit(x = train_annot_array, y =
np.array(train_categories), batch_size = batchSize, epochs = 50, callbacks
= [evaluationCallback(batchSize, learnRate)])
###IMPORT DATASET
#Get path of the script to work with relative paths later
script_dir = os.path.dirname(__file__)
#Get training data
rel_path_train_annot =
"../Flurplandaten/preprocessed__training_annotations_nobidet_randomangle.p"
train_annot_path = os.path.join(script_dir, rel_path_train_annot)
rel_path_train_categories =
"../Flurplandaten/object_list_for_training_annotations_nobidet_randomangle.
p"
train_categories_path = os.path.join(script_dir, rel_path_train_categories)
train_annot = np.array(pickle.load(open(train_annot_path, 'rb')))
train_annot_array = np.reshape(train_annot, (train_annot.shape[0], 100,
100, 3))
train_categories = pickle.load(open(train_categories_path, 'rb'))
#Get validation data
rel_path_validation_annot =
"../Flurplandaten/preprocessed__validation_annotations_nobidet_noaugmentati
on.p"
validation_annot_path = os.path.join(script_dir, rel_path_validation_annot)
rel_path_validation_categories =
"../Flurplandaten/object_list_for_validation_annotations_nobidet_noaugmenta
tion.p"
validation_categories_path = os.path.join(script_dir,
rel_path_validation_categories)
validation_annot = np.array(pickle.load(open(validation_annot_path, 'rb')))
validation_annot_array = np.reshape(validation_annot,
(validation_annot.shape[0], 100, 100, 3))
validation_categories = pickle.load(open(validation_categories_path, 'rb'))
#Get test data
```

```
rel_path_test_annot =
"../Flurplandaten/preprocessed__test_annotations_nobidet.p"
test_annot_path = os.path.join(script_dir, rel_path_test_annot)
rel_path_test_categories =
"../Flurplandaten/object_list_for_test_annotations_nobidet.p"
test_categories_path = os.path.join(script_dir, rel_path_test_categories)
test_annot = np.array(pickle.load(open(test_annot_path, 'rb')))
test_annot_array = np.reshape(test_annot, (test_annot.shape[0], 100, 100,
3))
test_categories = pickle.load(open(test_categories_path, 'rb'))
###TRAINING
#Ask if training should be automated
train_mode = input("Do you want to train automatedly? Y/N: ")
if(train_mode == "Y"):
    print("Automated training is started.")
    #Train model automatedly while varying hyperparameters
    trainNet(train_annot_array, train_categories)
else:
    #Train model non-automatedly
    batchSize = int(input("You want to train non-automatedly. Enter batch
size:"))
    learnRate = float(input("Now enter the learn rate:"))
    epochs = int(input("How many epochs do you want to train?"))
    opti = keras.optimizers.SGD(lr = learnRate)
    model = getModel()
    model.compile(optimizer = opti, loss = "categorical_crossentropy",
metrics=["accuracy"])
    print(model.summary())
    model.fit(x = train_annot_array, y = np.array(train_categories),
batch_size = batchSize, epochs = epochs, callbacks =
[evaluationCallback(batchSize, learnRate)])
#Save net
net_path = os.path.join(script_dir,
"../Netze/try10_IncResV2_randomrotation.h5")
model.save(net_path)
```

9.4 Vor- und Nachverarbeitung von Anwendungsdaten – Pseudocode und grafische Darstellung

```
script_dir = directory of script
floorplan_name = input(„Which image do you want to open?“)
floorplan_path = script_dir + ../images/ + floorplan_name

floorplan= array(open(floorplan_path).convert(BlackAndWhite).convert(RGB))

net_path = script_dir + /net.h5
net = load(net_path)

predictions = empty matrix of 100 * 100

for pixel in floorplan:

    image = floorplan part (size 100*100, top left corner x-Value|y-Value)
    prediction = net.predict(image)
```

```
    predicted_class = prediction with highest probability
    predicted_probability = probability of predicted_class

    if predicted_probability > threshold:
        predictions[x-Value][y-Value] = predicted_class

for prediction in predictions:
    if no neighbour with same class:
        delete prediction

floorplan = floorplan.add(boxes to cover the areas with predictions, labels)
plot floorplan
save floorplan
```

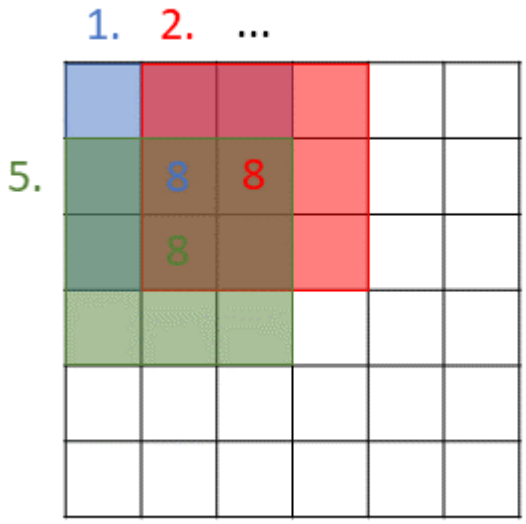


*Abbildung 1: Grafische Darstellung der Verarbeitung von Grundrissplänen in der praktischen Anwendung; vereinfachte Darstellung mit einem Eingabequadrat von 3*3 Pixeln und einem Plan von 6*6 Pixeln*

9.5 Ergebnisse der Trainingsdurchläufe

Epochen	SGD: 128, 0.01	SGD: 128, 0.1	SGD: 32, 0.0001	SGD: 32, 0.001	SGD: 32, 0.01	SGD: 32, 0.1	SGD: 64, 0.0001	SGD: 64, 0.001	SGD: 64, 0.01	SGD: 64, 0.1
1	0,27	0,02	0,10	0,21	0,38	0,06	0,10	0,15	0,35	0,06
2	0,32	0,28	0,14	0,27	0,33	0,07	0,12	0,21	0,39	0,09
3	0,36	0,16	0,17	0,31	0,35	0,06	0,13	0,25	0,39	0,01
4	0,37	0,37	0,19	0,34	0,37	0,08	0,13	0,27	0,40	0,08
5	0,38	0,38	0,21	0,35	0,41	0,08	0,13	0,29	0,38	0,01
6	0,38	0,38	0,22	0,35	0,41	0,08	0,13	0,31	0,38	0,04
7	0,38	0,41	0,22	0,36	0,40	0,07	0,15	0,33	0,39	0,12
8	0,36	0,40	0,23	0,37	0,39	0,06	0,15	0,34	0,39	0,11
9	0,36	0,41	0,24	0,37	0,38	0,10	0,16	0,34	0,38	0,14
10	0,35	0,39	0,24	0,37	0,40	0,10	0,17	0,35	0,40	0,05
11	0,37	0,36	0,25	0,38	0,36	0,09	0,18	0,35	0,40	0,16
12	0,38	0,33	0,25	0,38	0,35	0,09	0,18	0,36	0,39	0,20
13	0,38	0,41	0,26	0,37	0,37	0,12	0,19	0,36	0,38	0,20

14	0,38	0,38	0,27	0,37	0,36	0,09	0,20	0,37	0,37	0,22
15	0,37	0,39	0,27	0,37	0,38	0,10	0,20	0,36	0,39	0,23
16	0,39	0,41	0,28	0,37	0,37	0,08	0,21	0,36	0,39	0,22
17	0,38	0,39	0,28	0,37	0,39	0,10	0,21	0,36	0,39	0,25
18	0,39	0,39	0,29	0,37	0,40	0,14	0,21	0,37	0,39	0,25
19	0,38	0,42	0,29	0,37	0,40	0,14	0,22	0,36	0,41	0,22
20	0,38	0,40	0,30	0,37	0,42	0,14	0,22	0,36	0,40	0,21
21	0,36	0,41	0,30	0,38	0,41	0,15	0,22	0,36	0,41	0,20
22	0,38	0,38	0,30	0,37	0,42	0,14	0,23	0,36	0,40	0,25
23	0,37	0,40	0,31	0,37	0,42	0,17	0,23	0,36	0,40	0,24
24	0,38	0,40	0,31	0,37	0,41	0,16	0,24	0,36	0,40	0,25
25	0,39	0,39	0,32	0,36	0,42	0,14	0,24	0,36	0,39	0,26
26	0,38	0,41	0,31	0,37	0,39	0,16	0,25	0,36	0,39	0,25
27	0,39	0,40	0,32	0,37	0,40	0,15	0,25	0,36	0,40	0,22
28	0,39	0,41	0,32	0,37	0,41	0,16	0,26	0,35	0,40	0,23
29	0,39	0,41	0,32	0,37	0,39	0,18	0,25	0,36	0,40	0,27
30	0,39	0,41	0,32	0,37	0,41	0,15	0,26	0,36	0,41	0,25
31	0,38	0,41	0,32	0,37	0,42	0,19	0,26	0,36	0,40	0,26
32	0,37	0,40	0,33	0,37	0,42	0,16	0,27	0,36	0,41	0,26
33	0,38	0,42	0,33	0,37	0,42	0,18	0,26	0,37	0,40	0,24
34	0,39	0,39	0,33	0,37	0,41	0,15	0,27	0,36	0,40	0,26
35	0,37	0,42	0,33	0,37	0,41	0,16	0,27	0,36	0,40	0,27
36	0,36	0,44	0,33	0,37	0,41	0,19	0,27	0,36	0,40	0,27
37	0,38	0,40	0,33	0,36	0,41	0,18	0,28	0,36	0,40	0,24
38	0,38	0,42	0,33	0,36	0,42	0,19	0,28	0,36	0,40	0,25
39	0,38	0,42	0,34	0,37	0,42	0,20	0,29	0,36	0,39	0,26
40	0,38	0,41	0,33	0,37	0,41	0,20	0,29	0,37	0,39	0,25
41	0,38	0,41	0,34	0,37	0,42	0,13	0,29	0,37	0,40	0,25
42	0,37	0,40	0,34	0,38	0,41	0,18	0,29	0,36	0,39	0,25
43	0,38	0,41	0,34	0,37	0,41	0,19	0,30	0,36	0,40	0,27
44	0,38	0,40	0,34	0,37	0,41	0,21	0,29	0,36	0,40	0,24
45	0,39	0,42	0,33	0,38	0,41	0,19	0,30	0,36	0,40	0,26
46	0,39	0,40	0,34	0,37	0,41	0,20	0,30	0,36	0,39	0,25
47	0,39	0,41	0,34	0,38	0,41	0,15	0,30	0,36	0,40	0,27
48	0,39	0,39	0,34	0,37	0,40	0,20	0,30	0,36	0,41	0,25
49	0,39	0,40	0,34	0,37	0,42	0,16	0,30	0,36	0,40	0,27
50	0,39	0,40	0,35	0,37	0,34	0,21	0,30	0,36	0,40	0,27

Tabelle 1: Ergebnisse des Trainings mit einem ResNet50

Epo-chen	SGD: 32, 0.0001	SGD: 32, 0.001	SGD: 32, 0.01	SGD: 32, 0.1	SGD: 64, 0.0001	SGD: 64, 0.001	SGD: 64, 0.01	SGD: 64, 0.1
1	0,12	0,24	0,38	0,09	0,11	0,16	0,37	0,06
2	0,14	0,31	0,41	0,08	0,12	0,22	0,40	0,08
3	0,16	0,34	0,41	0,08	0,14	0,26	0,42	0,08
4	0,19	0,36	0,42	0,08	0,15	0,30	0,43	0,10
5	0,20	0,38	0,43	0,09	0,16	0,31	0,42	0,12
6	0,22	0,38	0,41	0,09	0,17	0,34	0,42	0,14

7	0,23	0,40	0,42	0,11	0,18	0,35	0,43	0,19
8	0,24	0,40	0,42	0,10	0,19	0,36	0,42	0,19
9	0,26	0,41	0,43	0,09	0,19	0,37	0,43	0,20
10	0,26	0,41	0,43	0,09	0,20	0,37	0,43	0,22
11	0,27	0,40	0,43	0,10	0,21	0,37	0,44	0,23
12	0,28	0,41	0,43	0,10	0,21	0,38	0,42	0,22
13	0,28	0,41	0,43	0,11	0,22	0,38	0,42	0,24
14	0,29	0,40	0,42	0,12	0,23	0,39	0,43	0,23
15	0,29	0,40	0,42	0,10	0,23	0,39	0,44	0,24
16	0,30	0,40	0,43	0,11	0,24	0,39	0,43	0,24
17	0,30	0,40	0,44	0,13	0,24	0,39	0,42	0,25
18	0,30	0,40	0,44	0,17	0,25	0,39	0,43	0,25
19	0,31	0,40	0,42	0,17	0,25	0,39	0,43	0,25
20	0,31	0,40	0,44	0,20	0,26	0,40	0,43	0,25
21	0,32	0,40	0,44	0,21	0,26	0,39	0,43	0,23
22	0,32	0,40	0,44	0,18	0,26	0,39	0,43	0,26
23	0,33	0,41	0,44	0,21	0,27	0,39	0,44	0,24
24	0,33	0,41	0,44	0,23	0,27	0,40	0,44	0,25
25	0,33	0,40	0,43	0,21	0,27	0,39	0,44	0,25
26	0,33	0,41	0,41	0,20	0,28	0,39	0,43	0,27
27	0,34	0,41	0,43	0,22	0,28	0,39	0,43	0,25
28	0,34	0,41	0,43	0,21	0,28	0,39	0,43	0,25
29	0,34	0,40	0,44	0,23	0,29	0,39	0,44	0,25
30	0,35	0,40	0,44	0,23	0,29	0,39	0,43	0,26
31	0,35	0,41	0,44	0,20	0,29	0,39	0,44	0,26
32	0,35	0,40	0,43	0,22	0,30	0,39	0,43	0,25
33	0,35	0,41	0,44	0,23	0,30	0,40	0,43	0,26
34	0,35	0,40	0,45	0,22	0,30	0,39	0,44	0,26
35	0,36	0,40	0,42	0,24	0,30	0,39	0,44	0,26
36	0,36	0,41	0,44	0,23	0,30	0,39	0,44	0,26
37	0,36	0,41	0,44	0,23	0,31	0,39	0,44	0,26
38	0,37	0,41	0,43	0,24	0,31	0,39	0,43	0,26
39	0,37	0,41	0,44	0,24	0,31	0,39	0,43	0,26
40	0,36	0,41	0,43	0,23	0,32	0,39	0,43	0,26
41	0,37	0,41	0,43	0,24	0,32	0,39	0,42	0,25
42	0,37	0,41	0,44	0,23	0,32	0,39	0,43	0,27
43	0,37	0,41	0,44	0,24	0,32	0,39	0,43	0,26
44	0,37	0,41	0,44	0,21	0,33	0,39	0,43	0,27
45	0,37	0,41	0,45	0,24	0,32	0,39	0,44	0,27
46	0,37	0,41	0,45	0,24	0,33	0,39	0,43	0,26
47	0,38	0,42	0,44	0,24	0,33	0,39	0,43	0,26
48	0,38	0,41	0,43	0,24	0,33	0,39	0,44	0,25
49	0,38	0,41	0,44	0,24	0,33	0,39	0,43	0,25
50	0,38	0,41	0,43	0,23	0,34	0,39	0,44	0,26

Tabelle 2: Ergebnisse des Trainings mit einem ResNet50 ohne die Klasse "bidet"

Epochen	SGD: 128, 0.0001	SGD: 128, 0.001	SGD: 128, 0.01	SGD: 128, 0.1	SGD: 32, 0.0001	SGD: 32, 0.001	SGD: 32, 0.01	SGD: 32, 0.1	SGD: 64, 0.0001	SGD: 64, 0.001	SGD: 64, 0.01	SGD: 64, 0.1
1	0,10	0,12	0,38	0,07	0,12	0,31	0,46	0,05	0,12	0,20	0,37	0,11
2	0,10	0,14	0,45	0,42	0,13	0,39	0,52	0,08	0,15	0,28	0,50	0,05
3	0,11	0,21	0,47	0,36	0,18	0,41	0,50	0,08	0,15	0,31	0,51	0,10
4	0,14	0,27	0,48	0,47	0,21	0,42	0,53	0,11	0,16	0,34	0,47	0,10
5	0,11	0,30	0,53	0,44	0,24	0,44	0,55	0,09	0,18	0,36	0,48	0,08
6	0,11	0,32	0,49	0,49	0,27	0,46	0,49	0,11	0,20	0,38	0,51	0,10
7	0,13	0,35	0,47	0,50	0,29	0,48	0,50	0,12	0,22	0,40	0,52	0,08
8	0,12	0,38	0,52	0,41	0,31	0,47	0,51	0,12	0,23	0,40	0,51	0,10
9	0,12	0,37	0,49	0,47	0,32	0,47	0,48	0,12	0,25	0,42	0,54	0,10
10	0,12	0,40	0,51	0,43	0,32	0,47	0,51	0,18	0,25	0,44	0,48	0,09
11	0,13	0,40	0,52	0,41	0,35	0,48	0,52	0,17	0,26	0,44	0,51	0,09
12	0,12	0,41	0,50	0,46	0,34	0,46	0,54	0,22	0,27	0,45	0,52	0,08
13	0,13	0,40	0,51	0,50	0,36	0,47	0,53	0,27	0,28	0,45	0,49	0,04
14	0,14	0,41	0,49	0,52	0,37	0,48	0,51	0,29	0,29	0,47	0,52	0,12
15	0,14	0,41	0,51	0,47	0,37	0,48	0,56	0,31	0,30	0,45	0,51	0,14
16	0,15	0,42	0,52	0,51	0,38	0,47	0,54	0,28	0,31	0,46	0,50	0,17
17	0,15	0,43	0,51	0,52	0,38	0,47	0,56	0,35	0,32	0,47	0,52	0,20
18	0,16	0,43	0,52	0,55	0,38	0,46	0,51	0,32	0,31	0,46	0,51	0,18
19	0,17	0,43	0,52	0,53	0,40	0,46	0,52	0,32	0,31	0,46	0,51	0,23
20	0,19	0,43	0,52	0,59	0,40	0,46	0,54	0,31	0,32	0,46	0,51	0,25
21	0,19	0,42	0,52	0,51	0,41	0,45	0,54	0,34	0,33	0,46	0,51	0,29
22	0,19	0,44	0,49	0,54	0,40	0,47	0,53	0,33	0,34	0,47	0,54	0,30
23	0,19	0,45	0,51	0,52	0,41	0,47	0,56	0,31	0,34	0,45	0,51	0,33
24	0,20	0,45	0,51	0,52	0,41	0,45	0,52	0,32	0,34	0,46	0,52	0,20
25	0,23	0,46	0,51	0,50	0,41	0,46	0,55	0,31	0,35	0,46	0,50	0,33
26	0,23	0,45	0,49	0,52	0,41	0,45	0,53	0,30	0,34	0,46	0,51	0,31
27	0,23	0,47	0,52	0,49	0,42	0,47	0,54	0,37	0,35	0,46	0,52	0,34
28	0,23	0,44	0,52	0,50	0,43	0,46	0,57	0,36	0,35	0,46	0,52	0,34
29	0,24	0,44	0,52	0,53	0,44	0,47	0,54	0,39	0,35	0,45	0,53	0,33
30	0,25	0,46	0,50	0,52	0,43	0,46	0,54	0,35	0,35	0,44	0,54	0,32
31	0,25	0,44	0,52	0,55	0,44	0,47	0,57	0,38	0,36	0,46	0,50	0,32
32	0,25	0,44	0,51	0,55	0,45	0,45	0,54	0,36	0,36	0,46	0,53	0,34
33	0,26	0,45	0,51	0,50	0,45	0,47	0,55	0,38	0,36	0,45	0,53	0,33
34	0,26	0,45	0,52	0,54	0,45	0,49	0,56	0,36	0,36	0,45	0,53	0,36
35	0,26	0,45	0,54	0,54	0,43	0,47	0,57	0,31	0,37	0,45	0,51	0,35
36	0,26	0,44	0,53	0,52	0,44	0,48	0,58	0,37	0,37	0,45	0,53	0,34
37	0,27	0,44	0,53	0,57	0,44	0,48	0,56	0,38	0,37	0,47	0,53	0,33
38	0,27	0,44	0,52	0,55	0,46	0,48	0,55	0,39	0,37	0,45	0,51	0,36
39	0,26	0,44	0,53	0,54	0,46	0,48	0,57	0,37	0,37	0,45	0,48	0,34
40	0,27	0,44	0,52	0,50	0,46	0,47	0,58	0,35	0,39	0,46	0,51	0,34
41	0,28	0,44	0,52	0,52	0,47	0,47	0,57	0,37	0,38	0,44	0,51	0,31
42	0,27	0,44	0,51	0,54	0,45	0,48	0,57	0,36	0,39	0,45	0,51	0,30
43	0,27	0,42	0,52	0,49	0,46	0,48	0,57	0,37	0,39	0,46	0,51	0,34

44	0,28	0,43	0,52	0,51	0,46	0,47	0,54	0,39	0,39	0,46	0,53	0,32
45	0,28	0,42	0,53	0,53	0,46	0,47	0,56	0,38	0,39	0,45	0,52	0,36
46	0,29	0,42	0,53	0,55	0,47	0,46	0,58	0,36	0,39	0,47	0,50	0,33
47	0,29	0,43	0,51	0,53	0,47	0,47	0,55	0,35	0,39	0,45	0,51	0,34
48	0,29	0,42	0,53	0,57	0,47	0,47	0,56	0,35	0,40	0,45	0,52	0,34
49	0,29	0,42	0,53	0,58	0,47	0,46	0,56	0,38	0,39	0,45	0,54	0,32
50	0,30	0,43	0,52	0,58	0,48	0,47	0,59	0,37	0,40	0,45	0,53	0,38

Tabelle 3: Ergebnisse des Trainings eines ResNet50 ohne Klasse "bidet" und ohne Vervielfältigung im Validierungsdatensatz

Epo-chen	SGD: 32,0.0001	SGD: 32,0.001	SGD: 32,0.01	SGD: 32,0.1	SGD: 64,0.0001	SGD: 64,0.001	SGD: 64,0.01	SGD: 64,0.1
1	0,12	0,40	0,49	0,21	0,11	0,27	0,51	0,52
2	0,13	0,47	0,51	0,34	0,12	0,35	0,52	0,58
3	0,15	0,49	0,52	0,41	0,14	0,39	0,54	0,47
4	0,19	0,48	0,53	0,45	0,17	0,41	0,52	0,57
5	0,24	0,49	0,55	0,12	0,19	0,42	0,54	0,53
6	0,27	0,51	0,54	0,24	0,20	0,42	0,51	0,28
7	0,28	0,52	0,54	0,40	0,21	0,43	0,53	0,50
8	0,30	0,50	0,54	0,23	0,22	0,45	0,53	0,58
9	0,31	0,51	0,52	0,12	0,23	0,47	0,52	0,58
10	0,32	0,50	0,53	0,18	0,25	0,46	0,51	0,56
11	0,33	0,50	0,52	0,28	0,26	0,48	0,51	0,56
12	0,34	0,50	0,54	0,29	0,27	0,46	0,52	0,58
13	0,35	0,50	0,53	0,20	0,29	0,49	0,51	0,58
14	0,35	0,49	0,52	0,32	0,30	0,48	0,53	0,59
15	0,38	0,50	0,53	0,34	0,31	0,48	0,51	0,60
16	0,40	0,48	0,53	0,34	0,32	0,46	0,51	0,51
17	0,40	0,49	0,50	0,36	0,33	0,47	0,52	0,55
18	0,41	0,49	0,51	0,35	0,33	0,47	0,52	0,59
19	0,43	0,50	0,52	0,33	0,34	0,47	0,52	0,55
20	0,43	0,50	0,52	0,36	0,34	0,48	0,52	0,62
21	0,43	0,52	0,52	0,36	0,35	0,46	0,51	0,61
22	0,43	0,51	0,54	0,37	0,35	0,48	0,52	0,60
23	0,44	0,51	0,52	0,35	0,36	0,48	0,53	0,59
24	0,44	0,49	0,51	0,37	0,38	0,47	0,53	0,60
25	0,44	0,51	0,53	0,40	0,38	0,47	0,52	0,59
26	0,45	0,50	0,53	0,41	0,38	0,47	0,52	0,59
27	0,46	0,49	0,53	0,39	0,37	0,47	0,52	0,59
28	0,47	0,49	0,54	0,38	0,39	0,47	0,51	0,60
29	0,46	0,49	0,53	0,29	0,40	0,47	0,51	0,63
30	0,47	0,49	0,52	0,32	0,40	0,47	0,51	0,61
31	0,47	0,50	0,53	0,38	0,42	0,45	0,51	0,62
32	0,47	0,50	0,53	0,40	0,42	0,45	0,51	0,63
33	0,47	0,51	0,53	0,33	0,41	0,46	0,51	0,60
34	0,48	0,51	0,53	0,40	0,41	0,45	0,50	0,61
35	0,47	0,51	0,53	0,27	0,43	0,46	0,50	0,62

36	0,48	0,51	0,53	0,40	0,43	0,47	0,51	0,62
37	0,48	0,50	0,53	0,14	0,44	0,47	0,52	0,61
38	0,48	0,50	0,52	0,27	0,44	0,46	0,51	0,63
39	0,48	0,51	0,52	0,36	0,44	0,47	0,51	0,60
40	0,49	0,52	0,52	0,37	0,44	0,46	0,51	0,63
41	0,50	0,51	0,53	0,28	0,43	0,47	0,51	0,59
42	0,50	0,51	0,52	0,38	0,45	0,47	0,51	0,61
43	0,50	0,51	0,52	0,39	0,45	0,46	0,52	0,62
44	0,50	0,52	0,52	0,27	0,45	0,47	0,52	0,60
45	0,49	0,50	0,53	0,38	0,45	0,47	0,52	0,60
46	0,50	0,52	0,53	0,38	0,46	0,48	0,50	0,60
47	0,51	0,52	0,53	0,38	0,46	0,47	0,51	0,59
48	0,50	0,50	0,52	0,34	0,46	0,46	0,51	0,59
49	0,50	0,51	0,52	0,38	0,47	0,47	0,51	0,61
50	0,50	0,52	0,52	0,41	0,46	0,46	0,51	0,59

Tabelle 4: Training eines InceptionNetV2 ohne Klasse "bidet"

Epo-chen	SGD: 128, 0.0001	SGD: 128, 0.001	SGD: 128, 0.01	SGD: 128, 0.1	SGD: 32, 0.0001	SGD: 32, 0.001	SGD: 32, 0.01	SGD: 32, 0.1	SGD: 64, 0.0001	SGD: 64, 0.001	SGD: 64, 0.01	SGD: 64, 0.1
1	0,06	0,21	0,70	0,79	0,21	0,60	0,83	0,08	0,16	0,22	0,46	0,03
2	0,10	0,28	0,83	0,87	0,05	0,81	0,76	0,05	0,20	0,38	0,84	0,03
3	0,11	0,37	0,93	0,93	0,44	0,88	0,95	0,09	0,29	0,54	0,86	0,04
4	0,13	0,44	0,96	0,97	0,53	0,83	0,91	0,05	0,35	0,60	0,97	0,07
5	0,14	0,52	0,97	0,98	0,59	0,95	0,92	0,07	0,40	0,68	0,96	0,01
6	0,15	0,56	0,98	0,96	0,29	0,92	0,98	0,09	0,48	0,76	0,93	0,10
7	0,16	0,61	0,99	0,99	0,67	0,97	0,97	0,11	0,58	0,81	0,89	0,06
8	0,16	0,65	0,99	0,98	0,70	0,97	0,99	0,09	0,50	0,85	0,93	0,03
9	0,18	0,70	0,99	0,99	0,76	0,96	0,98	0,14	0,52	0,87	0,96	0,09
10	0,19	0,74	1,00	0,94	0,62	0,96	0,91	0,03	0,71	0,88	0,97	0,05
11	0,21	0,78	1,00	0,97	0,65	0,98	0,97	0,49	0,72	0,89	0,99	0,02
12	0,22	0,80	1,00	0,99	0,81	0,40	0,97	0,26	0,77	0,87	0,99	0,03
13	0,22	0,82	1,00	1,00	0,86	0,21	0,96	0,58	0,49	0,84	0,12	0,04
14	0,24	0,84	1,00	1,00	0,02	0,99	1,00	0,14	0,78	0,95	0,44	0,11
15	0,25	0,84	1,00	1,00	0,56	0,99	0,99	0,05	0,83	0,95	0,97	0,05
16	0,26	0,86	1,00	1,00	0,64	0,99	0,95	0,61	0,81	0,95	0,97	0,10
17	0,27	0,87	1,00	1,00	0,68	0,99	1,00	0,24	0,86	0,97	0,98	0,09
18	0,27	0,88	1,00	0,97	0,72	1,00	0,90	0,27	0,78	0,71	0,04	0,09
19	0,29	0,90	1,00	1,00	0,74	1,00	0,63	0,42	0,83	0,97	0,64	0,01
20	0,31	0,91	1,00	0,99	0,77	1,00	0,96	0,32	0,04	0,98	0,10	0,04
21	0,32	0,92	1,00	1,00	0,79	0,25	0,86	0,89	0,58	0,99	0,99	0,06
22	0,33	0,92	1,00	1,00	0,81	0,99	0,97	0,85	0,70	0,98	0,96	0,01
23	0,35	0,93	1,00	1,00	0,82	1,00	1,00	0,81	0,76	0,99	0,99	0,04
24	0,35	0,93	1,00	1,00	0,85	0,99	1,00	0,29	0,79	0,98	0,98	0,10
25	0,36	0,94	1,00	1,00	0,84	1,00	0,98	0,09	0,60	0,99	0,98	0,12
26	0,37	0,94	1,00	1,00	0,87	1,00	0,98	0,19	0,38	0,99	0,98	0,09
27	0,38	0,86	1,00	1,00	0,88	0,99	1,00	0,48	0,80	0,99	0,99	0,09

28	0,40	0,96	0,99	1,00	0,90	0,99	1,00	0,78	0,73	0,99	0,99	0,07
29	0,40	0,96	1,00	1,00	0,91	0,96	0,97	0,56	0,83	0,99	0,99	0,03
30	0,41	0,96	1,00	1,00	0,92	1,00	0,93	0,85	0,60	0,99	0,99	0,13
31	0,42	0,96	1,00	1,00	0,92	0,99	1,00	0,90	0,82	0,99	0,99	0,11
32	0,43	0,96	1,00	1,00	0,92	0,98	0,68	0,02	0,85	0,52	0,99	0,15
33	0,43	0,97	1,00	1,00	0,94	0,99	0,73	0,22	0,87	0,99	0,99	0,03
34	0,44	0,97	1,00	1,00	0,94	0,98	0,97	0,30	0,84	0,99	0,98	0,03
35	0,46	0,98	1,00	1,00	0,94	0,96	0,94	0,06	0,89	0,99	0,99	0,06
36	0,46	0,98	1,00	1,00	0,95	1,00	0,97	0,66	0,88	0,99	0,99	0,08
37	0,45	0,98	0,99	1,00	0,95	0,56	1,00	0,51	0,90	1,00	0,99	0,06
38	0,47	0,98	1,00	1,00	0,96	1,00	0,98	0,56	0,91	1,00	0,99	0,02
39	0,47	0,98	1,00	1,00	0,73	1,00	0,97	0,95	0,90	1,00	0,76	0,06
40	0,49	0,98	1,00	0,99	0,96	1,00	1,00	0,70	0,92	1,00	0,98	0,01
41	0,50	0,98	1,00	0,99	0,97	1,00	1,00	0,85	0,77	0,99	0,97	0,07
42	0,50	0,98	1,00	1,00	0,97	1,00	0,99	0,83	0,94	0,98	0,74	0,07
43	0,51	0,99	1,00	1,00	0,97	1,00	0,99	0,97	0,91	0,99	0,89	0,03
44	0,52	0,99	1,00	1,00	0,97	1,00	0,96	0,07	0,94	1,00	0,99	0,10
45	0,52	0,99	1,00	1,00	0,98	0,97	1,00	0,95	0,95	1,00	0,98	0,04
46	0,53	0,99	1,00	1,00	0,97	1,00	1,00	0,34	0,95	1,00	0,99	0,19
47	0,53	0,99	1,00	1,00	0,98	1,00	1,00	0,51	0,96	1,00	0,99	0,10
48	0,55	0,99	1,00	1,00	0,99	1,00	1,00	0,16	0,95	0,75	0,98	0,22
49	0,55	0,99	1,00	1,00	0,99	1,00	0,90	0,92	0,95	0,92	0,98	0,29
50	0,56	0,99	1,00	1,00	0,98	0,98	0,97	0,82	0,93	0,09	1,00	0,19

Tabelle 5:Training eines InceptionNetV2 ohne Klasse "bidet", ohne Vervielfältigung der Validierungsdaten und mit zufälliger Rotation der Trainingsdaten