Hochschule Konstanz
Fakultät Elektrotechnik
und Informationstechnik

Produkterkennung mit Convolutional neural networks

Bachelorarbeit

Patrik Seitz

Matrikelnummer: 296298

Studiengang: Automobilinformationstechnik

Erstbetreuer: Prof. Dr. Michael Froehlich **Zweitbetreuer**: Dipl.-Ing. Sebastian Langhammer

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Bachelorarbeit selbstständig und nur unter Benutzung der angegebenen Literatur und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und ist auch noch nicht veröffentlicht. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

22.12.2020 Patrik Seitz

Datum Unterschrift

Literaturverzeichnis

1. Andrew Ng. Objekterkennung / YOLO Algorithmus / Deep Learning. Video [Folienskript]. Zugriff am 12.12.2020 unter https://www.youtube.com/watch?v=ArPaAX_PhIs&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF

2. Benjamin Aunkofer. 13.1.2019. Training eines Neurons mit dem Gradientenverfahren. Onlineartikel. Zugriff am 12.12.2020 unter https://data-science-blog.com/blog/2019/01/13/training-eines-neurons-mit-dem-gradientenverfahren/

3. Fei-Fei Li. März. (2015). How we're teaching computers to understand pictures. Ted Talk, Video und Transkript. Zugriff am 12.12.2020 unter https://www.ted.com/talks/fei_fei_li_how_we_re_teaching_computers_to_understand_pictures?referrer=playlist-what_are_we_really_teaching_ai

4. Jason Brownlee. 22.4.2019. Pooling Layers. Onlineartikel. Zugriff am 12.12.2020 unter https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

5. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. 9.5.2016. You Only Look Once: Unified, Real-Time Object Detection.

6. Julia Fischer und Kevin Pochwyt. Neuronale Netze. Onlineartikel. Zugriff am 12.12.2020 unter https://user.phil.hhu.de/~petersen/SoSe17_Teamprojekt/AR/neuronalenetze.html

7. Kunal Kushwaha. Logistic Regression. Video [Folienskript]. Zugriff am 12.12.2020 unter https://www.youtube.com/watch?v=UzaIxfRIcEY&t=5856s

8. Mingxing Tan. 29.5.2020. EfficientNet. Onlineartikel. Zugriff am 12.12.2020 unter https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

9. Rikiya Yamashita, Mizuho Nisho, Richard Kinh Gian Do, Koari Togashi. 22.6.2018. Convolutional neural networks: an overview an application in radiology.

10. Navaneeth Bodla, Bharat Singh, Rama Chellappa, Larry S. Davis. 8.8.2017. Improving Object Detection With One Line of Code (Non-maximum supression).

11. Rohan Arora. 29.1.2020. Onlineartikel. Convolutional implementation of the sliding window algorithm. Zugriff am 12.12.2020 unter https://medium.com/ai-quest/convolutional-implementation-of-the-sliding-window-algorithm-db93a49f99a0

12. Romain Beaumont. 20.7.2020. Image Embeddings. Onlineartikel. Zugriff am 12.12.2020 unter https://rom1504.medium.com/image-embeddings-ed1b194d113e

13. Univ.-Prof. Dr. Axel C. Schwickert. Arbeitspapier. Künstliche Neuronale Netze - Aufbau und Funktionsweise. Universität Mainz.

14. Wilhelm Burger, Mark James Burge. 2005. Digitale Bildverarbeitung Eine Einführung mit Java und ImageJ.

Abkürzungsverzeichnis

CNN	Convolutional Neural Network
vgl	Vergleich
NN	Neural networks
LG	Logistische Regression
SVM	Support Vector Machine

Abbildungsverzeichnis

Inhalt

1 Einleitung

„*Stück für Stück können wir Maschinen Augenlicht geben. Erst bringen wir ihnen das Sehen bei, dann lernen sie, uns besser zu sehen. Wir Menschen werden erstmalig nicht die Einzigen sein, die die Welt auf diese Art entdecken und über sie nachdenken können. Wir werden Maschinen nicht nur für ihre Intelligenz nutzen, sondern auch mit ihnen zusammenarbeiten, und zwar auf Wegen, die wir jetzt noch gar nicht erfassen können.“ (*[3])

Die Möglichkeit einer Maschine beizubringen, dass sie versteht, was in einem Bild passiert, ist ein breites und faszinierendes Forschungsfeld, welches in vielen Bereichen, von autonomem Fahren über Robotik, bis hin zur Klassifizierung von Objekten und Gegenständen des alltäglichen Lebens Anwendung findet. In dieser Arbeit wird eine Pipeline zur Erkennung und Klassifizierung verschiedener Produkte anhand des Beispiels einer Autoklassifizierung vorgeschlagen. Das angestrebte Ziel ist es, ein theoretisches Grundverständnis für die angewendeten Methoden und Bausteine zu vermitteln, um damit den Aufbau des vorgeschlagenen Modells nachvollziehen und darauf aufbauen zu können. Dafür wird zunächst eine Übersicht über die Funktionsweise neuronaler Netze sowie die verwendeten Methodiken gegeben. Hierauf basierend wird eine Aufstellung für die Umsetzung der Autoklassifizierung präsentiert. Als Ausblick werden mögliche Optimierungen / Erweiterungen dieser Umsetzung aufgezeigt.

2 Grundlagen

In diesem Kapitel werden theoretische Grundlagen dieser Arbeit betrachtet. Es wird zunächst ein Überblick über neuronale Netzwerke präsentiert sowie genauer auf *Convolutional Neural Networks (CNN)* eingegangen. Hierzu werden die einzelnen Schichten dieser Struktur beschrieben und die Methodik dahinter analysiert. Darauf folgt die Beschreibung der Klassifizierungsmethode sowie Vorverarbeitung (Datenerweiterung) der Eingabe (Bilder).

2.1 Neuronale Netzwerke

Neural networks (NN) ermöglichen es, Modelle, welche sich an den biologischen Vorgängen im menschlichen Gehirn orientieren, zur Prognose / Vorhersage von Werten zu erstellen. Neuronen sind miteinander durch Synapsen verbunden. Die Stärke dieser Verbindung ist jedoch unterschiedlich ausgeprägt und kann sich beim Menschen durch neue Erfahrungen / Informationen verändern. Dieser Lernprozess wird auf künstliche *neural networks* übertragen, indem die Stärke und Verbindungen der Synapsen mittels eines Optimierungsverfahren so bestimmt werden, dass die Fehlerrate bei der Prognose / Vorhersage, z.B. durch Klassifizierung eines Bildes, minimiert wird. (vgl. [13] S. 5-7)

2.1.1 Künstliche Neuronen

Neuronen in einem künstlichen *neural network* haben folgenden Aufbau:

- **Propagierungsfunktion**: Bestimmt die Form, in welcher die Eingabe verarbeitet wird, z.B. durch Aufsummierung aller Eingänge.

  $$\sum_{n} \mathbf{X}n * \mathbf{W}n$$

  X0 – **X**n stehen hierbei für die Eingänge, **W**0 - **W**n für die Gewichtung der jeweiligen Verbindung (Synapsen).

- **Aktivierungsfunktion:** Berechnet den aktuellen Zustand eines Neurons. Dieser Zustand ist eine reelle Zahl aus einem bestimmten Intervall, wie z.B. [0 - 1]. Herfür finden verschiedene Funktionen wie die Sigmoidfunktion (2.6.1), Stufenfunktion etc. Anwendung.

- **Ausgabefunktion:** Legt den Ausgabewert des Neurons mittels der Aktivierungsfunktion fest.

- **Bias:** Gewichtung des Neurons.

2.1.2 Aufbau eines voll verbundenen Netzwerkes

Fully connected neural networks sind aus mehreren Schichten, welche **n** - Neuronen beinhalten, aufgebaut. Bei einer voll verbundenen Schicht ist jedes Neuron mit jedem Neuron in der vorherigen und folgenden Schicht verbunden. Dies ist in Abbildung 1 dargestellt. (vgl. [6])

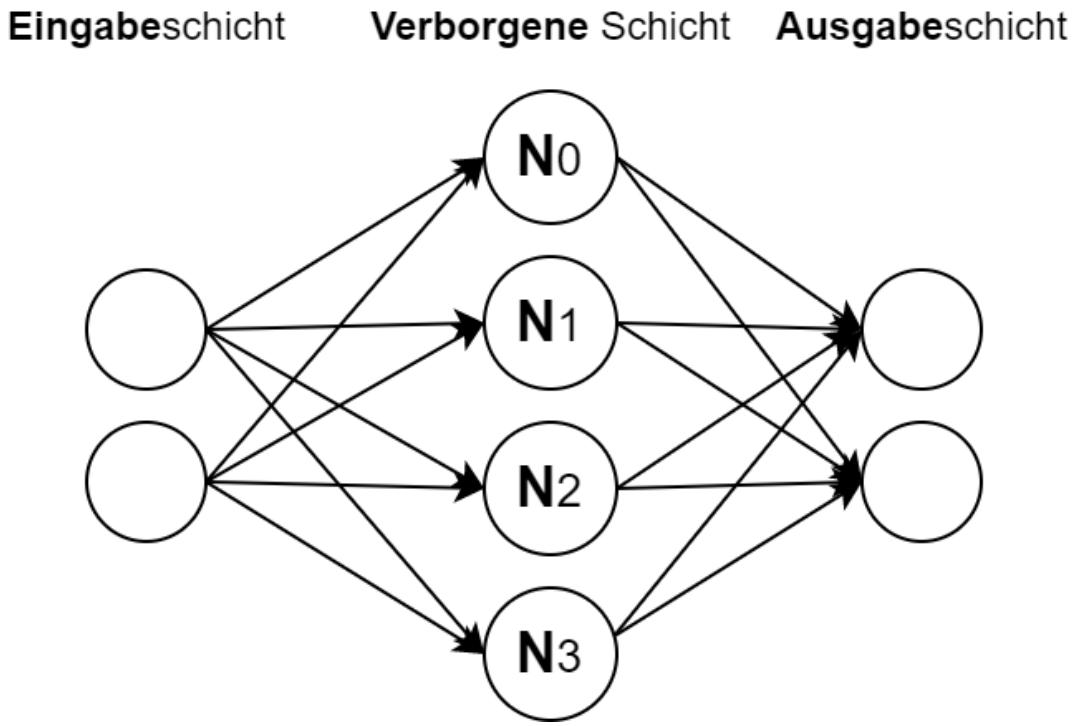


Abbildung 1 Darstellung eines voll verbundenen Netzwerkes.

Die Anzahl und Größe der Schichten variieren, basierend auf den Anforderungen an das Netzwerk. So wird die Größe der Eingabeschicht aufgrund der Anzahl der zu betrachtenden Werte bestimmt, wie z. B. die Anzahl der Pixel in einem Bild. Die Ausgabeschicht kann zur Klassifikation dieser Bilder verwendet werden, wobei jedes Neuron für eine Objektklasse steht. Die Anzahl und die Größe der Mittelschichten bzw. verborgenen Schichten folgt keiner klaren Anleitung und hat, basierend auf der Problemstellung, unterschiedliche Herangehensweisen.

2.1.3 Lernprozess

NN werden mit einem Datensatz, bestehend aus den betrachtenden Werten und deren Labels, welche jeweils die korrekte Ausgabe darstellen, trainiert. Während des Trainingsprozesses wird eine Vorhersage für jedes Label erstellt und deren Abweichung zum tatsächlichen Label mittels einer Verlustfunktion kalkuliert. Basierend auf dem Verlust / Fehler über den gesamten Datensatz werden die Gewichte der einzelnen Verbindungen so verändert, dass der Verlust minimal wird.

2.1.3.1 Verlustfunktion / Kostenfunktion

Die Verlustfunktion berechnet den Fehler eines einzelnen Trainingsbeispiels, während die Kostenfunktion den Durchschnittsverlust über den gesamten Trainingsdatensatz bestimmt. **O** ist die Ausgabe des Modells, **y** ist der Wert des Labels.

$$\mathbf{E} = \frac{1}{n}\sum (y_i - o_i)^2$$

Abbildung 2 Formel der Kostenfunktion. n [Anzahl der Trainingslabel], y [Wert des Trainingslabel], o [Vorhersage für Trainingslabel].

Die Kostenfunktion hängt demnach von allen Parametern (Gewichte und Bias) des *NN* ab. Veränderung der einzelnen Verbindungen wird daher auch eine Veränderung der Ausgabe der Kostenfunktion zur Folge haben.

2.1.3.2 Gradient Descent (Gradientenabstiegsverfahren)

Mittels dieses Verfahrens wird versucht, das Minimum der Kostenfunktion zu finden. Die Grundidee ist es, die Steigung der Funktion in Abhängigkeit der aktuellen Werte der Verbindungen / Gewichte zu berechnen. Hierzu werden zunächst alle Gewichte zufällig initialisiert und damit der Gradient bestimmt. Der Gradient ist ein Vektor / Array, welcher alle partiellen Ableitungen der Funktion beinhaltet (vgl. [2]).

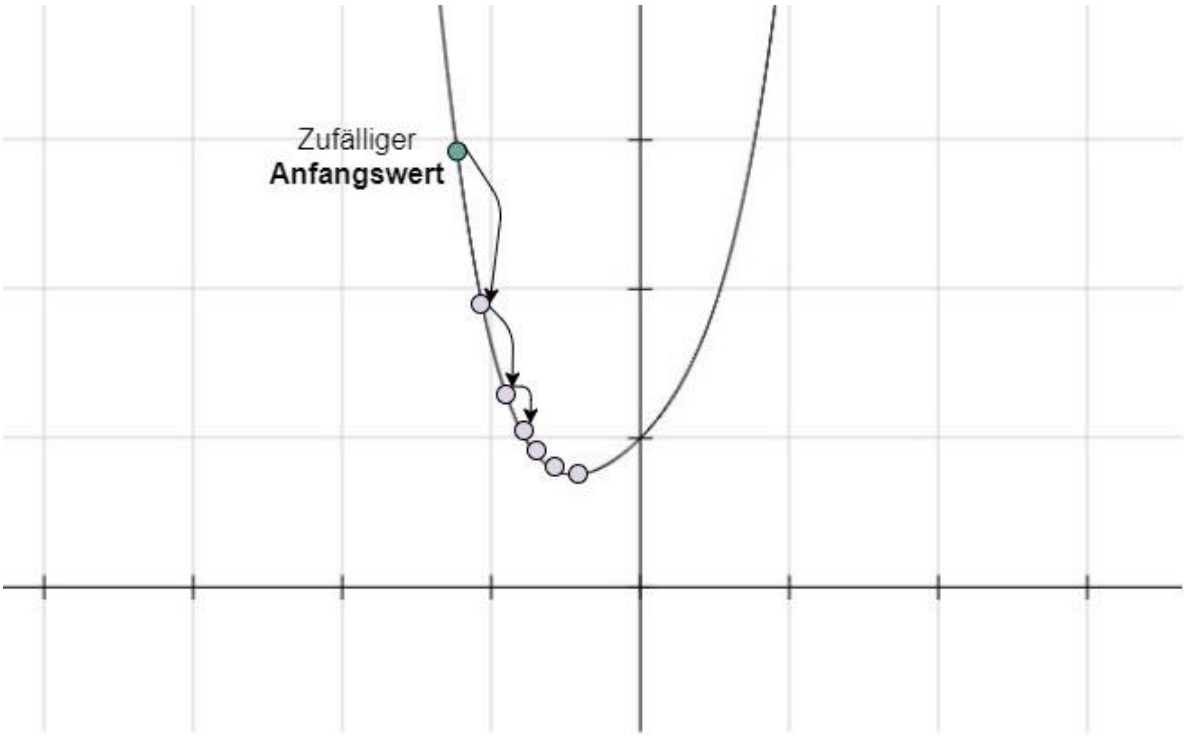


Abbildung 3 Beispiel des Gradientenabstiegsverfahren anhand einfacher Kostenfunktion mit einem Parameter.

Am Beispiel einer vereinfachten Kostenfunktion mit nur einem Parameter / Gewicht (Abbildung 3) resultiert daraus die Steigung am zufällig gewählten Anfangswert. Basierend auf der Richtung dieser Steigung wird nun die Verbindung bzw. das Gewicht erhöht oder verringert. Die Größe dieser Veränderung wird als adaptives Verfahren basierend auf der *learning rate* sowie der Stärke der Steigung umgesetzt. Dieser Prozess kann allerdings nicht garantieren, das globale Minimum einer komplexeren Funktion zu finden, sondern - basierend auf den Anfangswerten und *learning rate* nur ein lokales Minimum. Dadurch kann es bei mehrfacher Durchführung des Lernprozesses zu Abweichungen zwischen den erzielten Ergebnissen kommen.

2.2 *Convolutional neural networks*

CNN ist das bevorzugte Modell für jede Art der Bilderkennung. Hierbei sind die einzelnen Schichten nicht vollständig (siehe 2.1.2), sondern durch Konvolution (Faltung) miteinander verbunden, dadurch wird es dem Netzwerk ermöglicht, selbstständig wichtige Eigenschaften / Strukturen der Bilder zu erkennen.

2.2.1 Convolution (Faltung)

Hierbei werden Konvolutionsfilter, was einer Matrix kleiner als die der Eingangsmatrix (Bild) entspricht, über das Bild bewegt und die Summe der elementweisen Multiplikation aus Filter und Bild als Ausgabe gewertet. Die Koeffizienten dieser Filter werden im Trainingsprozess trainiert, so dass mittels dieser das Netzwerk in der Lage ist, lokale Merkmale zu erkennen. Für jeden Filter wird eine eigene Ausgabe / *feature map* erstellt. (vgl. [9], S.3-4). Mehrere Faltungsschichten hintereinander zu schalten, erlaubt es dieser Art Netzwerk, zunächst niederwertige Merkmale, wie z.B. Linien und in tieferen Schichten zunehmend abstraktere Objekte, zu erkennen.

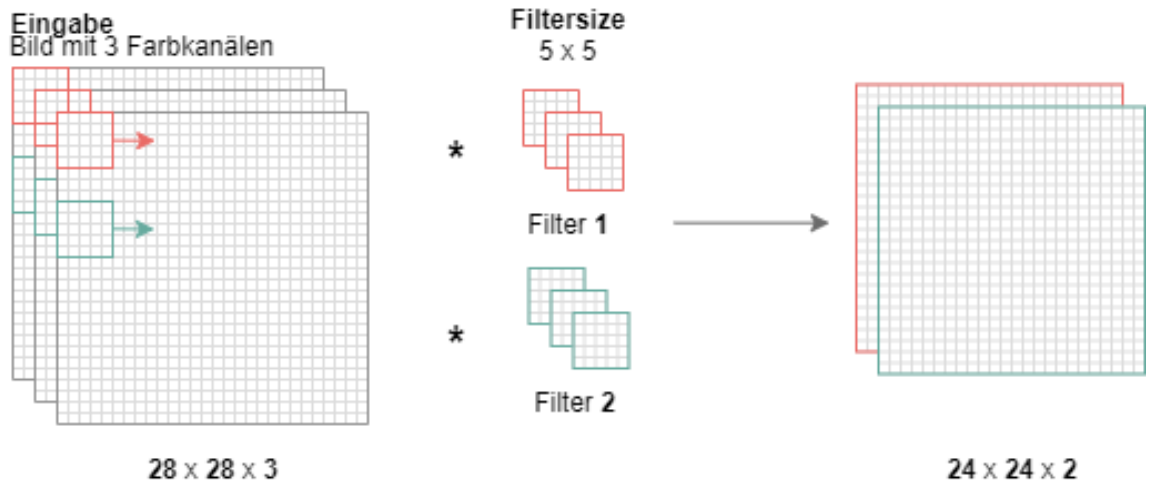


Abbildung 4 Beispiel einer Faltungsschicht. 2 Filter [5 x 5] resultieren bei einem Eingangsvolumen von 28 x 28 x 3 ein Ausgabevolumen 24 x 24 x 2 [Ohne Padding]. Jeder Filter erzeugt eine feature map.

2.2.1.1 Padding

Mittels Padding werden zusätzliche Pixel an den Rändern des Bildes erzeugt. Dieses Verfahren kann in Kombination mit einer Faltungsschicht verwendet werden. Damit wird es ermöglicht, keine Rand Pixel des eigentlichen Bildes durch den Faltungsprozess zu verlieren.

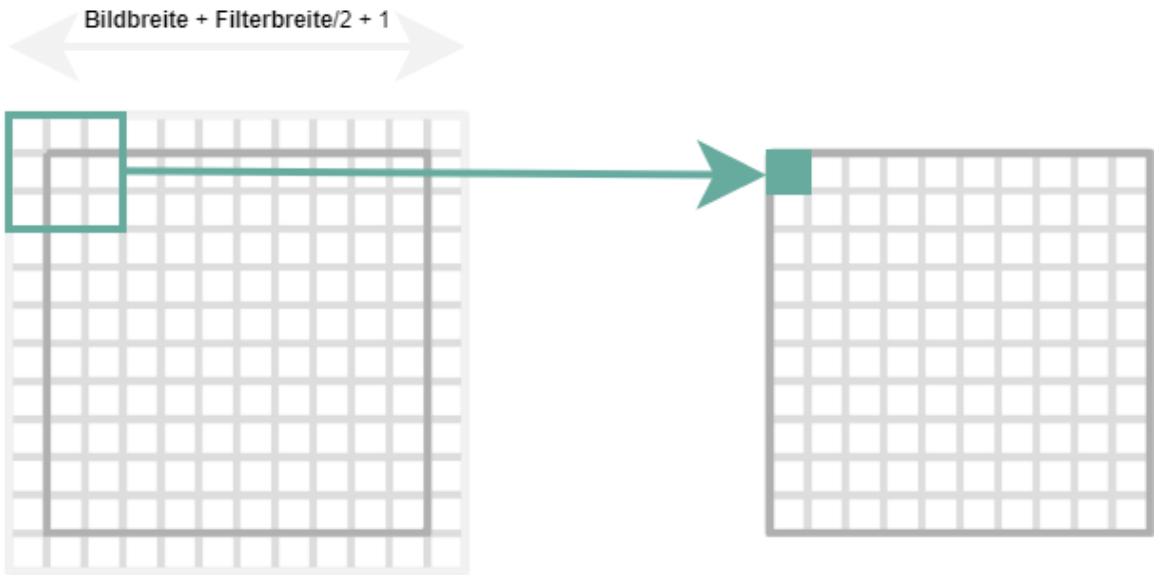


Abbildung 5 Erweiterung eines Bildes durch Padding mit einer 3 x 3 Filtermaske.

Die Werte der zusätzlichen Pixel werden durch verschiedene Verfahren definiert (vgl. [14], S.118–120):

- **Mirror:** Das Bild wird an den Rändern gespiegelt.
- **Min:** Pixelwerte werden als das Minimum definiert.
- **Max**: Pixelwerte werden als das Maximum definiert.
- **Continue**: Äußersten Pixelwert fortsetzen.

2.2.2 Pooling

Faltungsschichten sind dahingehend limitiert, dass sie die genaue Position eines erkannten Merkmales aufnehmen. Damit würde durch kleine Veränderungen der Position dieses Merkmales im Bild eine andere Ausgabe bzw. *feature map* kreiert werden, was es dem Netzwerk erschwert, Strukturen wiederzuerkennen. Um den Einfluss dieser kleiner Positionsveränderungen oder Rotation zu verringern, werden Pooling Schichten verwendet (vgl. [4]).

Pooling Schichten skalieren die Ausgabe, wobei wichtige strukturelle Elemente beibehalten werden, während andere Details, welche möglicherweise unwichtig für den Anwendungsfall sind, ignoriert werden. Hierbei gibt es verschiedene Methoden (vgl. [9], S.5-6):

- **Average Pooling:** Berechnet den Durchschnittswert.

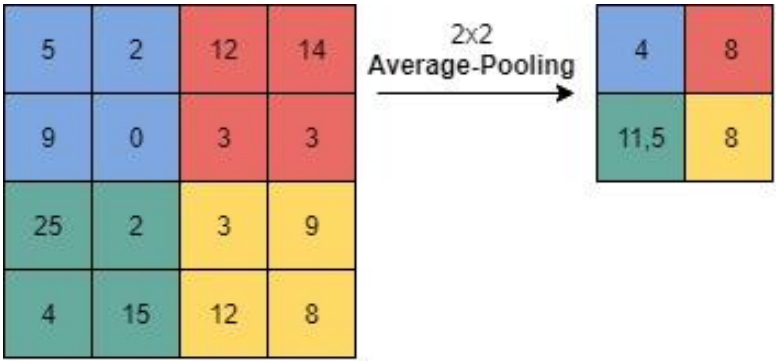


Abbildung 6 Average Pooling Beispiel 2x2.

- **Maximum Pooling:** Berechnet den Maximalwert.

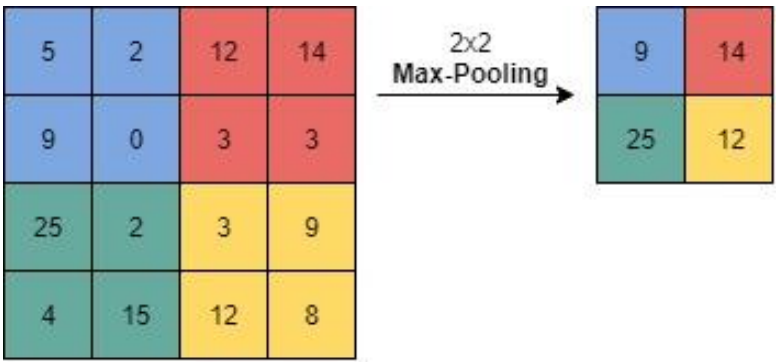


Abbildung 7 Max Pooling Beispiel 2x2.

2.2.3 Flatten

Für eine Klassifizierung wird eine oder mehrere voll verbundene Schichten am Ende des *CNN* angehängt. Um die *feature maps* in die notwendige Dimension zu transformieren und um die beiden Netzwerkarten zu verknüpfen, wird eine *flatten* Schicht dazwischengeschoben. Hierbei werden alle Elemente der *feature maps* in einem 1-dimensionalen Array zusammengefügt (vgl. [9], S.6).

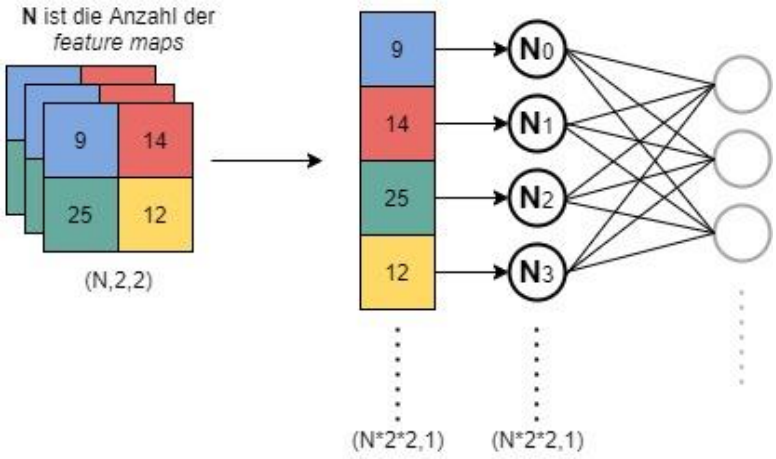


Abbildung 8 Verbindung der letzten Faltungsschicht mit einem voll verbundenen Netzwerk.

2.2.4 Dropout

Bei dem Trainingsprozess eines großen Netzwerkes mit einem limitierten / kleinen Datensatz kann es dazu kommen, dass das Netzwerk sich speziell an die Trainingsdaten anpasst, dadurch eine hohe Genauigkeit mit diesen Daten erzielt, jedoch bei neuen oder leicht veränderten Daten schlechte Ergebnisse liefert. Dieses Phänomen ist als Überanpassung (*Overfitting*) bekannt. Hierbei werden im Laufe des Trainingsprozesses einige Verbindungen deutlich stärker / wichtiger, während andere komplett ignoriert werden (vgl. [9], S.11). Um dieses Problem zu mindern, wird durch eine *Dropout* Schicht eine bestimmte Anzahl von Neuronen zufällig in jedem Layer ignoriert / ausgeschaltet. Damit muss das Netzwerk auch Neuronen betrachten, die möglicherweise ohne die Dropout Schicht ignoriert wurden.

2.3 Objekterkennung

Die Objekterkennung dient dazu, neben der Klassifizierung, eine Lokalisierung des erkannten Objektes durchzuführen. Hierfür finden verschiedene Verfahren Anwendung.

2.3.1 Grundnetzwerk

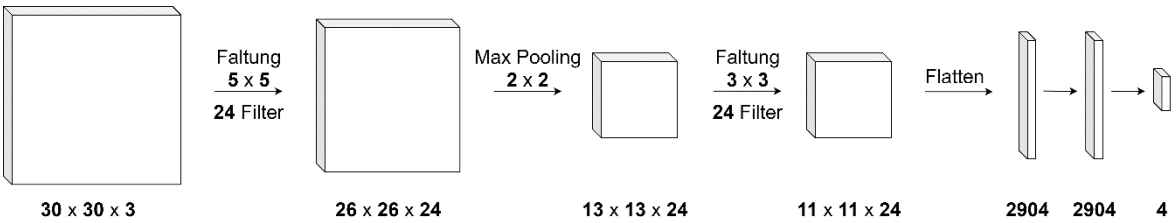


Abbildung 9 Aufbau eines CNN mit voll verbundenen Schichten zur Bildklassifizierung (vgl. [1]).

2.3.2 Schiebefenster (Sliding Window)

Bei dieser Methode wird ein Fenster mit einer festen Größe über das Bild geschoben und für jede Position / Bildbereich eine Klassifizierung durchgeführt. Die Klassifizierung erfolgt durch ein Netzwerk in Form des oben beschriebenen Grundnetzwerk, das im Anwendungsfall von Abbildung 10 darauf trainiert wurde, 4 verschiedene Kategorien zu identifizieren. Jeder Bildausschnitt wird dafür nacheinander an das Netzwerk übergeben und einzeln klassifiziert.

Abbildung 10 Bildklassifizierung mittels Schiebefenster. Bildquelle: Stanford Cars Dataset.

2.3.3 Erweiterung des Grundnetzwerk

Dieser Prozess kann durch eine Veränderung des Netzwerks optimiert werden. Hierfür werden die letzten voll verbundenen Schichten entfernt und durch **1** x **1** Faltungsschichten ersetzt. Dadurch entsteht ein Ausgabevolumen von **11** x **11** x **4**. Diese Ausgabe „unterteilt“ das Bild in 121 Sektionen mit jeweils 4 Kategorien. Dadurch werden alle Fensterpositionen simultan (in einem durchlauf) klassifiziert.

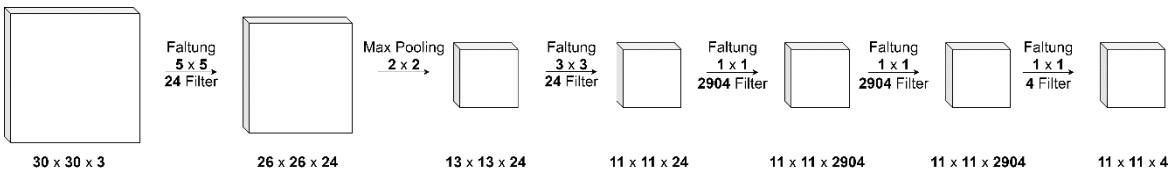


Abbildung 11 Erweiterung des Grundnetzwerks durch Ersetzung der voll verbundenen Schichten mit 1 x 1 Faltungsschichten (vgl. [1]).

2.3.4 Probleme der Schiebefenster Methode

Für jede Zelle der Ausgangsmatrix wird demnach ein Volumen mit den Dimensionen **1** x **1** x **4** ausgegeben, dass diese Zelle klassifiziert. Da mit einer festen Zellengröße [**11** x **11**] gearbeitet wird, können dadurch Objekte mit unterschiedlichen Skalierungen nur schwer lokalisiert werden. Zusätzlich ist jede Zelle dahingehend limitiert, dass diese nur einer Kategorie zugeordnet werden kann. (vgl. [5])

2.3.5 YOLO [You only look once] Algorithmus

Um eine genauere Positionsbestimmung der *bounding boxen* zu erstellen, wird das Ausgabevolumen erweitert. Dazu werden zusätzlich zu der Klassifizierung 5 weitere Parameter jedem Vektor hinzugefügt, siehe Abbildung 12. Die Parameter $\mathbf{b}_x$, $\mathbf{b}_y$, $\mathbf{b}_w$ und $\mathbf{b}_h$ bestimmen die Abmessungen und Position einer *bounding box*. $\mathbf{P}_c$ beschreibt, wie zuversichtlich das Netzwerk ist, dass sich ein Objekt der Klassen $\mathbf{c}_1$ – $\mathbf{c}_n$ im Bereich der *bounding box* befindet. Dieser Parameter wird bei den Trainingslabel daran fest gemacht, ob sich der Mittelpunkt des Objektes in dieser Zelle befindet. Die Trainingslabel für das erweiterte Grundnetzwerk mit den zusätzlichen Parametern aus Abbildung 12 müssen demnach in Form einer **11** x **11** x **9** Matrix erstellt werden. (vgl. [11])

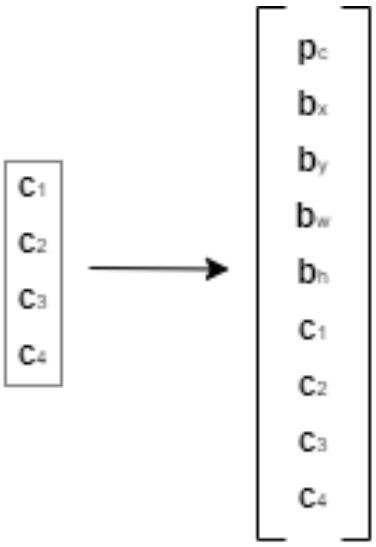


Abbildung 12 Erweiterung der Ausgabe um eine genauere Bestimmung der bounding boxen zu ermöglichen (vgl. [11]).

2.3.5.1 Mehrfacherkennung

Es kann jedoch dazu kommen, dass ein Objekt mit mehreren *bounding boxen* detektiert wurde. (Siehe Abbildung 13).

Abbildung 13 Erkanntes Objekt wird durch mehrere bounding boxen detektiert. Mehrere Zellen meinen den Mittelpunkt des Objektes erkannt zu haben und erzeugen eine hohe Zuversicht für die jeweilige Ausgabe.

Um die überflüssigen *bounding boxen* zu entfernen, wird das *non – max supression* Verfahren angewandt. Dieser Algorithmus wird für jede Klasse separat verwendet, indem zunächst grundsätzlich alle *bounding boxen* mit einer kleinen Zuversicht ($\mathbf{P}_C$) entfernt werden. Von den übriggebliebenen Boxen wird für jede Kategorie die Box mit der größten Zuversicht erfasst und die Überlappung mit allen anderen *bounding boxen* dieser Klasse kalkuliert. Ist die Überlappungsfläche größer **N**t (Schwellwert) wird diese ebenfalls entfernt. (vgl. [10]).

Ein Problem des bisherigen Aufbaus ist es, dass nur ein Objekt pro Zelle erkannt werden kann. Sollte z.B. ein Roller direkt vor einem Auto stehen und der Mittelpunkt dieser Objekte in dieselbe Zelle fallen, kann schon gar kein Trainingslabel mit den aktuellen Limitierungen für dieses Bild erstellt werden.

2.3.5.2 Anchor boxes

Anchor boxen erlauben es, Zellen mehrere Objekte zu erkennen. Hierzu wird das Ausgabevolumen von **11** x **11** x **9** auf **11** x **11** x **9*N** erweitert. Jeder Ausgabevektor besitzt demnach die Möglichkeit **N** - Boxen zu definieren. Jeder dieser **N** - Abschnitte wird im Voraus einer Grundform zugewiesen. $\mathbf{b}_x$, $\mathbf{b}_y$, $\mathbf{b}_w$ und $\mathbf{b}_h$ bestimmen nun nicht mehr die Position und Abmessungen der *bounding box,* sondern den Offset dieser zur festgelegten Grundform. Dadurch muss das Netzwerk keine Vorhersagen für ganz unterschiedliche Formen treffen, sondern nur kleine Abweichungen dieser bestimmen, was einen leichteren Trainigsprozess zur Folge hat. (vgl. [1])

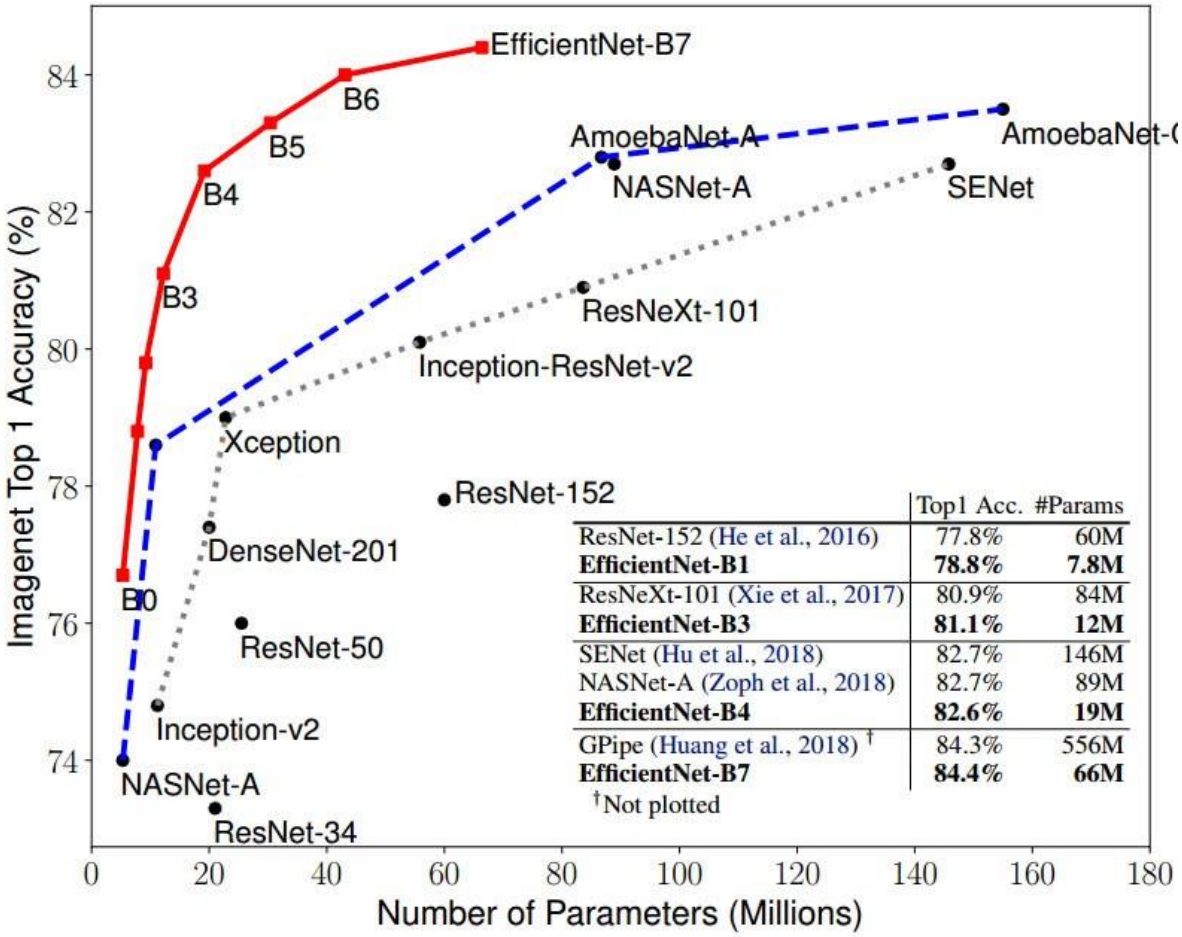
Abbildung 14 Zuweisung verschiedener Grundformen zum Ausgabevektor.

Die Trainingslabel werden demnach, basierend auf dem Formtyp mit der größten Überlappungsfläche, zwischen einer der Grundformen und der erstellten Trainings *bounding box* bestimmt. Ein Auto würde demnach sehr wahrscheinlich als Type **B** definiert werden. Die Grundformen sollten deshalb unter Bedacht des Anwendungsfalles erstellt werden.

2.4 EfficientNet

EfficientNet ist ein *CNN*, das mit dem imageNet Datensatz, basierend auf dem letzten Stand der Technik, eine hohe Genauigkeit im Vergleich mit anderen Netzwerkarten erzielt. Zum Beispiel wurde mit EfficientNet-B7 eine 84.4% Genauigkeit erzielt, obwohl das Netzwerk 8.4-mal kleiner und 6.1-mal schneller als das vorherige und weitverbreitete ResNet-50 ist. (vgl. [8])

Diese Netzwerkart wurde als neue Methode zur Skalierung von *CNN* von Google entwickelt und wird in dieser Arbeit mit dem ImageNet Datensatz als Basis Model verwendet.



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

Abbildung 15 Erzielte Genauigkeit verschiedener Netzwerkarten, basierend auf dem imageNet Datensatz.

Bildquelle https://medium.com/analytics-vidhya/efficientnet-the-state-of-the-art-in-imagenet-701d4304cfa3

2.5 Image Embedding

Image Embedding ist ein Verfahren, welches es Computer erlauben soll, ähnliche Bilder intuitiv zu erkennen. Dieses verwendet ein verändertes *CNN* als Merkmal Extraktor. Die Veränderung bezieht sich auf das Entfernen der voll verbundenen Schichten am Ende des Modells. Hierzu kann ein schon vortrainiertes Netzwerk, wie das EfficientNet - B5 Model mit dem ImageNet Datensatz, verwendet werden (vgl. [12]). Durch das Entfernen dieser Schichten resultiert die *flatten* Schicht als neue Ausgabeschicht. Die Methodik dahinter ist es, für jedes Bild im Trainingsdatensatz einen Vektor aus den durch die Filter erkannten Merkmale zu erstellen und durch logistische Regression oder andere Verfahren eine Separierung der Kategorien zu erzeugen, welche es ermöglicht, Vorhersagen / Klassifizierung für weitere Bilder zu erstellen. Im Kontext dieser Arbeit ist der dadurch resultierende Vorteil eine einfachere Anpassung an neue Daten. Da jedes Bild als Vektor, bei EfficientNet – B5, mit 2048 Werten, dargestellt wird, entsteht dadurch ein deutlich verringerter Speicherplatz. Dies bietet den Vorteil, dass beim Entfernen oder Hinzufügen neuere Produkte nicht jedes Bild erneut dem Netzwerk in einem Trainingsprozess gezeigt werden muss, sondern lediglich die bestehenden sowie neuen *embeddings* als Datensatz für eine Klassifizierungsmethode wie Logistische Regression, SVM, Neurales Netzwerk verwendet werden.

2.6 Logistische Regression

Logistische Regression (LG) ist eine Klassifikationstechnik, die versucht, basierend auf den Eigenschaften des Datensatzes, eine genaue Separierung der Kategorien und damit eine Klassifizierung zu ermöglichen. Unter Annahme eines einfachen Datensatzes, bestehend aus 2 Klassen, deren Trainingsdaten jeweils aus einem Array / Vektor mit 2 Eigenschaften / Werten dargestellt werden, wird basierend auf den Datenpunkten dieser Vektoren im 2-dimensionalen Raum eine Trennungsgerade an die Daten angepasst. Basierend auf dem Abstand zwischen Trennungsgerade und Vektor / Datenpunkt lässt sich eine Zuordnung / Klassifizierung weiterer Punkte mit der Definition (basierend auf Abbildung 16), welche einen positiven Distanzwert als Zuordnung zu Kategorie **A** (Blau) und einen negativen Distanzwert zu Kategorie **B** (Rot) wertet, ermöglichen. Die Distanz beschreibt die Zuversicht der Zuordnung, je größer die Distanz desto größer die Zuversicht. Hierbei kann diese, Werte in einem Wertebereich von negativ bis positiv unendlich annehmen. Da nicht die Distanz, sondern die Wahrscheinlichkeit der Zuordnung zu einer Klasse benötigt wird, welche üblicherweise in einem Zahlenraum von 0 bis 1 angeordnet ist, müssen die Distanzwerte in diesen Wertebereich überführt werden.

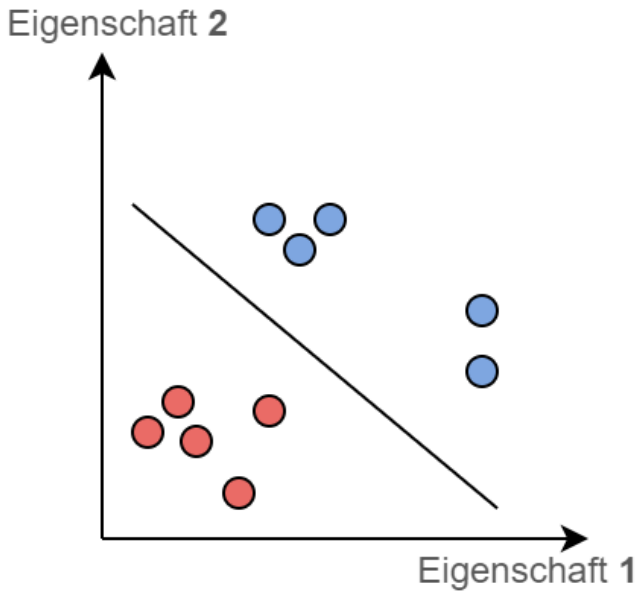


Abbildung 16 Anpassung einer Trennungsgerade zwischen zwei Kategorien (Blau & Rot). Jeder Datenpunkt ist durch zwei Eigenschaften / Parameter definiert.

2.6.1 Sigmoidfunktion

Diese Funktion besitzt die Eigenschaft für jeden reellen Wert **t** einen Wert zwischen 0 und 1 zuzuweisen.

$$\mathrm{sig}(t) = \frac{1}{1 + e^{-t}}$$

Abbildung 17 Sigmoidfunktion [Bildquelle Wikipedia]

Demnach wird für **t** die Distanz zur Trennungslinie eingesetzt und damit die Wahrscheinlichkeit zur Zugehörigkeit einer Kategorie bestimmt. Die Zugehörigkeit bezieht sich in Abbildung 16 auf die Kategorie mit positiven Distanzwerten (Blau). Demnach würde eine sehr hohe negative Distanz in einer sehr kleinen Wahrscheinlichkeit resultieren. Die Wahrscheinlichkeit bezogen auf Kategorie 2 (Rot) resultiert demnach aus (1 – **p**(Blau)).

2.6.2 Anpassung der Trennungslinie

Ziel ist es eine Trennungsgerade zu erstellen, bei welcher die Wahrscheinlichkeit für alle Trainingsdaten maximal wird. Jeder Trainingspunkt soll mit der größtmöglichen Wahrscheinlichkeit der entsprechenden Kategorie, basierend auf der Distanz, zugeordnet werden. Hierfür wird eine Funktion definiert, die basierend auf der Geradenfunktion mittels der Sigmoidfunktion, die Wahrscheinlichkeiten aller Trainingsdaten multipliziert. Mittels *gradient assent,* dem genauen Gegenteil von dem Gradientenabstiegsverfahren, kann das Maximum dieser Funktion und somit die Parameter der Trennungsgeradenfunktion, bei welcher das Produkt der Wahrscheinlichkeiten maximal wird, bestimmt werden.

2.6.3 One vs. Rest

Mit dem bisher betrachteten Modell ist eine binäre Klassifikation möglich. Nun wird darauf aufgebaut und eine Methode analysiert, die das Modell erweitert, um mehrere Klassen zu klassifizieren.

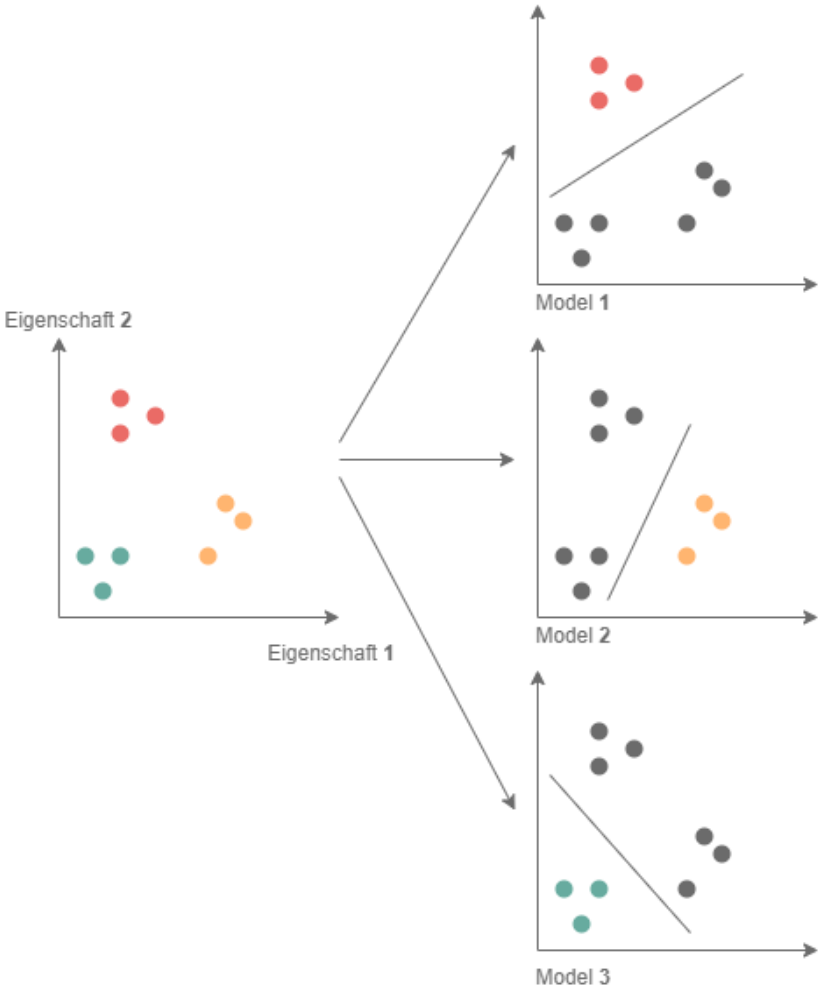


Abbildung 18 One Vs. Rest Methode mit 3 Klassen und 2 Parametern. Unterteilung in 3 Modelle.

Hierbei wird für jede Kategorie ein eigenes binär Modell erstellt, welches die weiteren Klassen als eine betrachtet. Im Vorhersageprozess liefert demnach jedes Modell die Wahrscheinlichkeit der Zugehörigkeit zur jeweiligen Kategorie. Das Modell mit der höchsten Wahrscheinlichkeit definiert die finale Klassifizierung. Bisher wurde zur vereinfachten Darstellung ein Datensatz mit nur 2 Eigenschaften / Parametern betrachtet. Erhöhung dieser, erhöht auch die Dimensionalität der Trennungsgerade.

2.7 Data Augmentation (Datenerweiterung)

Bei der Datenerweiterung wird die Vielfalt des Trainingsdatensatzes durch Transformationen erhöht. Zu beachten gilt, dass durch die Transformation keine Bilder entstehen sollten, welche das Modell im normalen Anwendungsfall nicht zu sehen bekommen würde. Hierbei gibt es verschiedene Verfahren die Anwendung finden:

2.7.1 Seitenverhältnis Anpassung

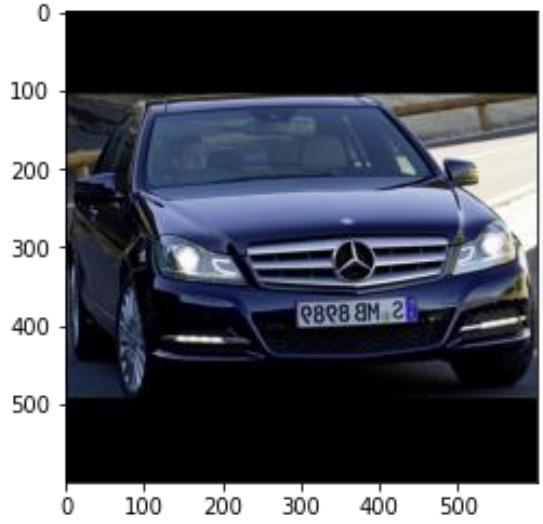

```
[62] from PIL import Image
     import matplotlib.pyplot as plt

     def load_image(path,width=456,height=456,keep_aspect_ratio=False):
       image = Image.open(path).convert('RGB')
       if keep_aspect_ratio:
         image_asr = np.zeros((width,height,3))
         new_height = int((image.height/image.width)*width)
         if new_height>height:
           new_height = height
         image = image.resize((width,new_height), Image.ANTIALIAS)
         image_asr[int((height-new_height)/2):new_height+int((height-new_height)/2),0:width] = np.array(image)
         return Image.fromarray((image_asr).astype(np.uint8))
       else:
         return image.resize((width,height), Image.ANTIALIAS)

     plt.figure()
     plt.imshow(load_image('example.jpg',keep_aspect_ratio=True))
```

Abbildung 19 Datenerweiterung [Seitenverhältnis Anpassung] inklusiver Umsetzung in Python [Google Colab] für Versuch 4.3. und Kapitel 3 (Produkterkennung Pipeline, Image preprocessing).

Neural networks haben eine festgelegte Größe, in welcher das Bild an das Netzwerk übergeben wird. Dafür muss dieses in den häufigsten Fällen angepasst werden. Es besteht die Möglichkeit, das Bild dafür über die volle Größe zu strecken (Abbildung 19 Links),

dadurch könnten jedoch Informationen über die Form des Objekts verzerrt werden. Eine Alternative ist, das Bild zu strecken, aber die Seitenverhältnisse beizubehalten (Abbildung 19 rechts).

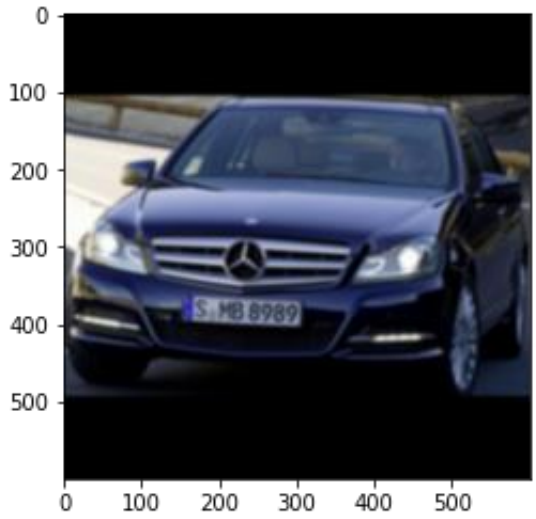
2.7.2 Spiegeln




```
[21] from PIL import Image,ImageOps,ImageFilter
     import matplotlib.pyplot as plt

     def loadImage(image_path):
         return Image.open(image_path)

     def mirror(image):
       return ImageOps.mirror(image.copy())

     plt.figure()
     plt.imshow(mirror(loadImage('example.jpg')))
```

Abbildung 20 Datenerweiterung [Spiegeln] inklusiver Umsetzung in Python [Google Colab] für Versuch 4.3. und Kapitel 3 (Produkterkennung Pipeline, Image preprocessing).

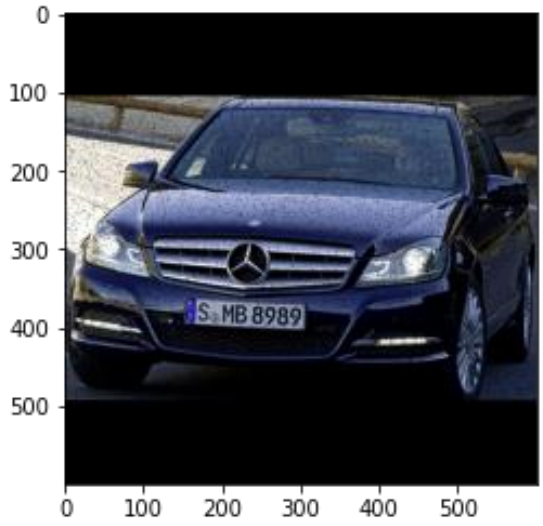
Gaussian blur




```
[19] from PIL import Image,ImageOps,ImageFilter
     import matplotlib.pyplot as plt

     def loadImage(image_path):
         return Image.open(image_path)

     def gaussian_blur(image,radius):
       return image.copy().filter(ImageFilter.GaussianBlur(radius))

     plt.figure()
     plt.imshow(gaussian_blur(loadImage('example.jpg'),2))
```

Abbildung 21 Datenerweiterung [Gaussian blur] inklusiver Umsetzung in Python [Google Colab] für Versuch 4.3. und Kapitel 3 (Produkterkennung Pipeline, Image preprocessing).

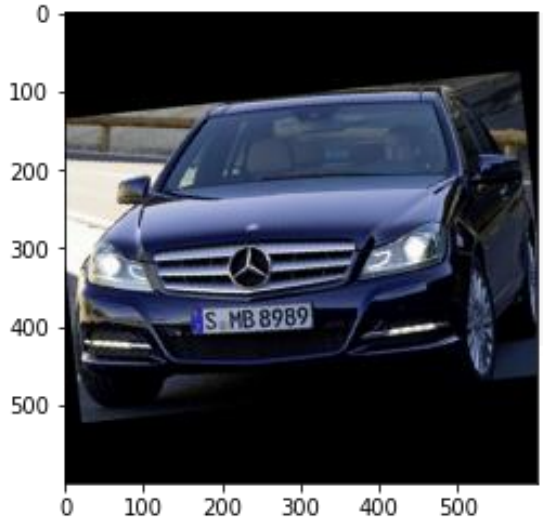
2.7.3 Noise




```
[18] from PIL import Image,ImageOps
     import random
     import matplotlib.pyplot as plt

     def loadImage(image_path):
         return Image.open(image_path)

     def add_noise(image,propability=0.1):
       image_ = image.copy()
       for x in range(image_.width):
         for y in range(image_.height):
           if random.random() <= propability:
             image_.putpixel((x,y),0)
       return image_

     plt.figure()
     plt.imshow(add_noise(loadImage('example.jpg')))
```

Abbildung 22 Datenerweiterung [Noise] inklusiver Umsetzung in Python [Google Colab] für Versuch 4.3. und Kapitel 3 (Produkterkennung Pipeline, Image preprocessing).

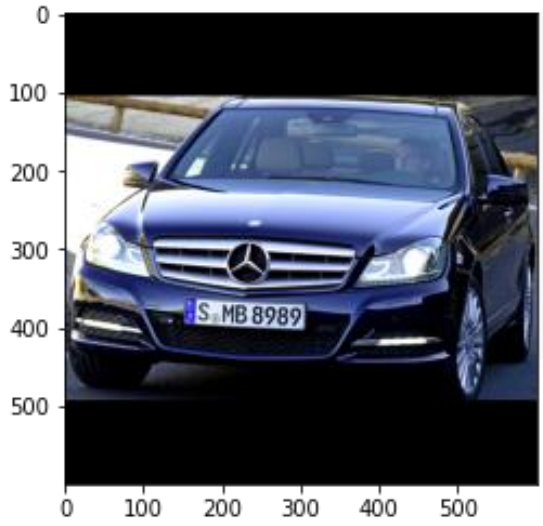
2.7.4 Rotation




```
[22] from PIL import Image,ImageOps,ImageFilter
     import matplotlib.pyplot as plt

     def loadImage(image_path):
         return Image.open(image_path)

     def rotate(image,angle):
       return image.copy().rotate(angle)

     plt.figure()
     plt.imshow(rotate(loadImage('example.jpg'),5))
```

Abbildung 23 Datenerweiterung [Rotation] inklusiver Umsetzung in Python [Google Colab] für Versuch 4.3. und Kapitel 3 (Produkterkennung Pipeline, Image preprocessing).

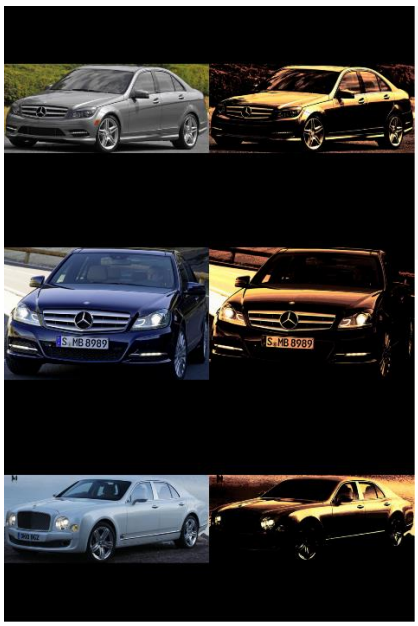
2.7.5 Kontrasterhöhung




```
[25] from PIL import Image,ImageOps,ImageFilter,ImageEnhance
     import matplotlib.pyplot as plt

     def loadImage(image_path):
         return Image.open(image_path)

     def enhance(image,factor=1.5):
       enhancer = ImageEnhance.Brightness(image.copy())
       return enhancer.enhance(factor)

     plt.figure()
     plt.imshow(enhance(loadImage('example.jpg')))
```

Abbildung 24 Datenerweiterung [Kontrasterhöhung] inklusiver Umsetzung in Python [Google Colab] für Versuch 4.3. und Kapitel 3 (Produkterkennung Pipeline, Image preprocessing).

2.7.6 Farbangleichung



```
[5]  def convert_to_sunset(image_path,palette_image_path='sunset_palette.jpg'):
       src_image = load_image(image_path)
       image = src_image.convert('P',
                                 palette=Image.ADAPTIVE,
                                 colors=256)
       palette_image = load_image(palette_image_path).convert(
           'P', palette=Image.ADAPTIVE, colors=150)
       image.putpalette(palette_image.getpalette())
       return image.convert('RGB')
```

Abbildung 25 Datenerweiterung [Farbangleichung] inklusiver Umsetzung in Python [Google Colab] für Kapitel 3 (Produkterkennung Pipeline, Image preprocessing)

3 Produkterkennung Pipeline

Aufbau einer Produkterkennungspipeline mit folgenden Anforderungen.

- Ein oder mehrere Produkte in einem Bild zu klassifizieren.
- Produkte hinzufügen sowie entfernen zu können.

3.1 Übersicht

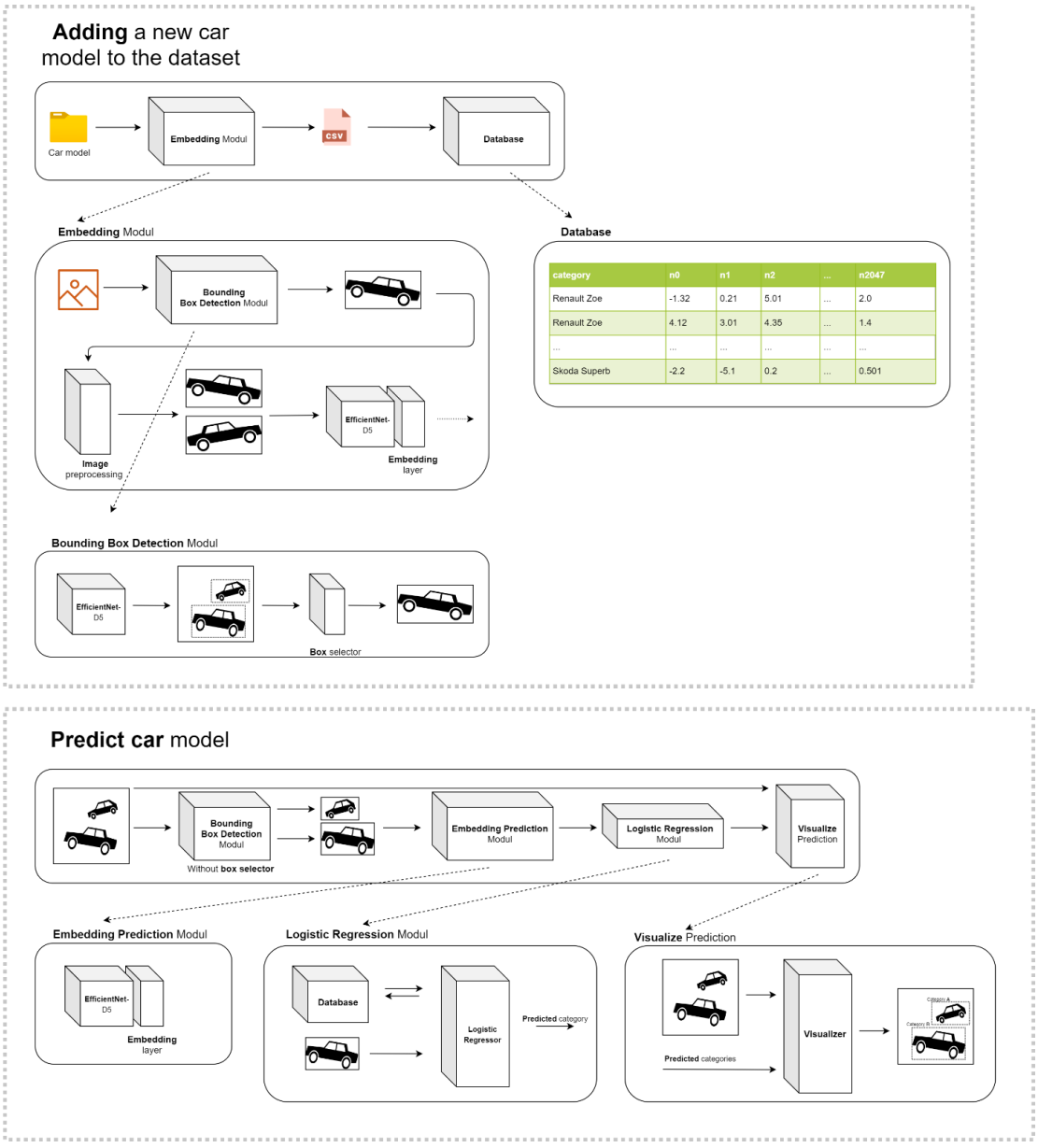


Abbildung 26 Übersicht der einzelnen Module der Produkterkennungspipeline anhand des Beispiels einer Autoklassifizierung

3.2 Bounding Box Detection Modul

Dieses Modul verwendet das EfficientNet-B5 (in der Abbildung als D5 beschrieben) Modell mit dem YOLO Algorithmus als Rückgrat, dadurch besteht die Möglichkeit, EfficientNet als komplett vortrainiertes Netzwerk zu integrieren. Basierend auf dem *ImageNet* Datensatz können damit 1000 verschiedene Klassen erkannt werden, wie z.B. Auto, Flugzeug, LKW etc. Sollte keine der Klassen für den Anwendungsfall passend sein, kann das Netzwerk auch mit einem eigenen Datensatz trainiert werden. Ziel dieses Modells ist es jedoch nicht, Objekte genau zu klassifizieren, sondern deren Überklassen wie Auto, Flugzeug, etc. zu erkennen.

Die Ausgabe wird in Form von Bildausschnitten, die das jeweilige Objekt beinhalten, realisiert. Der *Box Selector* wird zusätzlich noch im Embedding Prozess dazwischengeschaltet und selektiert die Box mit der größten Fläche. Dadurch entstehen 2 Anwendungsmodi für dieses Teilmodul.

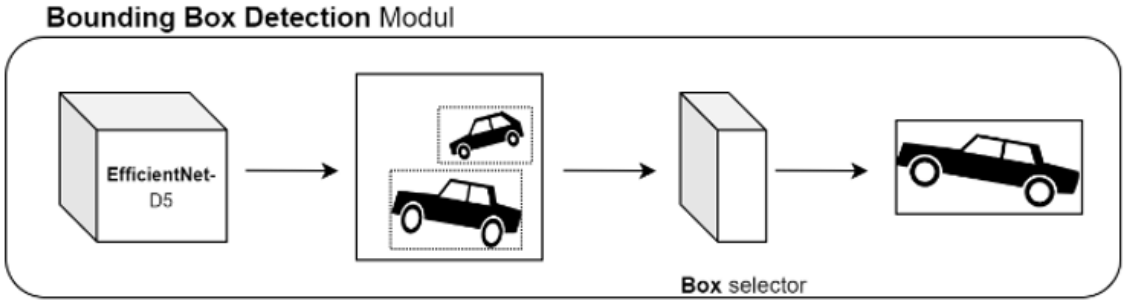


Abbildung 27 Aufbau der Bounding Box Erkennung für Produkte, im Beispiel Autos

3.3 Embedding Modul

Das *Embedding Modul* generiert für jedes Bild im Datensatz **N**-Vektoren, basierend auf den gewählten Möglichkeiten des *Image preprocessing* Modul (siehe Data Augmentation). Dazu wird zunächst jedes Bild im Datensatz an das Bounding Box Detection Modul mit zusätzlichem *Box Selector* übergeben. Dieser Schritt kann durch manuelle Erstellung der *bounding boxen* übersprungen werden. Es bietet sich jedoch an, dies in einem automatisierten Prozess ablaufen zu lassen, da Datensätze in vielen Fällen zu viele Bilder beinhalten, um dies manuell in einem realistischen Zeitrahmen durchzuführen.

Die Begründung, nicht einfach das gesamte Bild an das Embedding Modell zu übergeben, liegt darin, dass möglicherweise andere Objekte im Bildhintergrund einen Einfluss auf die Klassifizierungsgenauigkeit haben könnten. Da im Klassifizierungsprozess das *Bounding Box Detection Modul*, Bildausschnitte für jedes erkannte Fahrzeug erstellt und die *embeddings* ebenfalls im selben Format (Bildausschnitte) erstellt werden sollten, wird dadurch eine einheitliche Struktur geschaffen.

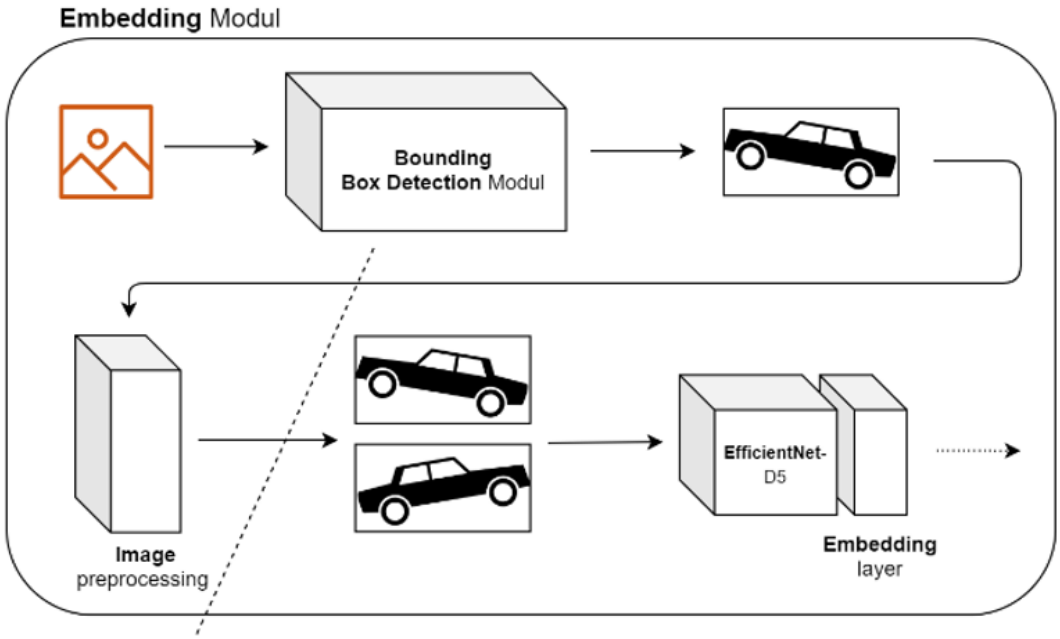


Abbildung 28 Implementierung eines Embedding Moduls anhand einer Autoklassifizierung

3.4 Produkte hinzufügen / entfernen

Da permanent neue Produkte entwickelt oder Veränderungen an deren Form vorgenommen werden, ist es sinnvoll, eine flexible Methodik zu implementieren, die es erlaubt, Produkte entfernen sowie hinzufügen zu können. Eine Änderung des Trainingsdatensatz erfordert zwangsweise eine Anpassung des Modells. Um diese Anpassung zu beschleunigen wird das *Image Embedding*-verfahren verwendet. (siehe 2.5)

3.4.1 Hinzufügen / Entfernen

Um ein Produkt hinzuzufügen, wird ein Ordner mit Bildern dieses Objektes erstellt. Für jedes dieser Bilder wird mittels des Embedding Modul ein oder mehrere Vektoren erstellt, welche in der Datenbank in Form einer CSV Datei gespeichert werden können. Die Datenstruktur beinhaltet die Kategorie sowie 2048 Werte. Die Anzahl der Werte basiert auf dem gewählten Embedding Modell. Durch Erstellung dieser zusätzlichen Daten muss eine erneute Anpassung des Klassifizierungsmodell (Logistic Regression Modul) durchgeführt werden. Durch Entfernen eines Produkts (Löschung aller Einträge der entsprechenden Kategorie in der Datenbank) muss ebenfalls eine erneute Anpassung erfolgen.

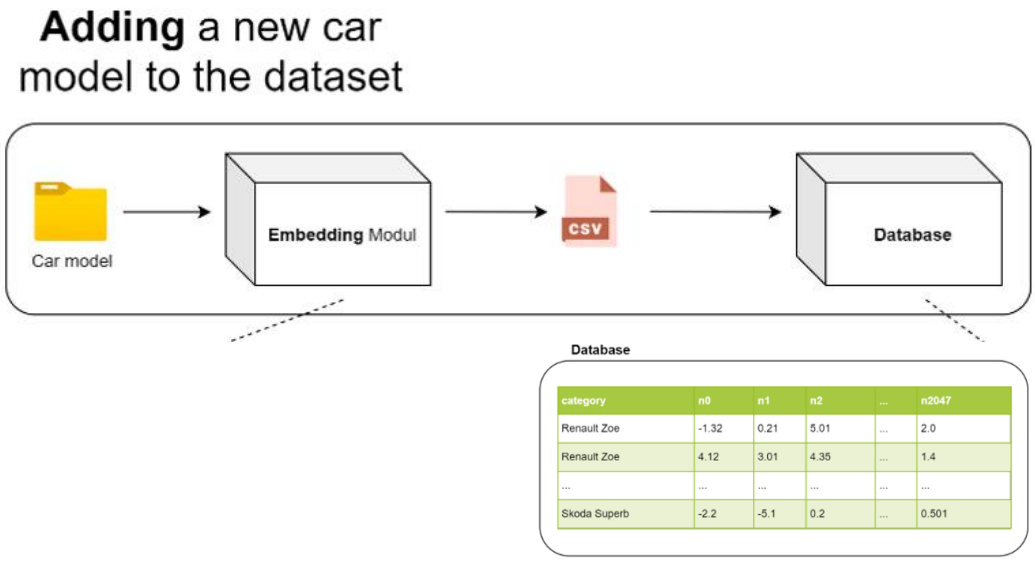


Abbildung 29 Umsetzung der Datenbankstruktur zur Ermöglichung, Produkte hinzufügen sowie entfernen zu können.

3.5 Logistic Regression Modul

Durch das *Logistic Regression Modul* wird die Vorhersage für ein Bildauschnitt, welcher nach den selben Prinzipien, wie die Bildauschnitte, die für den Embedding Prozess erstellt wurden, generiert. Die Vorhersage durch den *Logistic Regressor* fällt als Array aus, der die Wahrscheinlichkeit der Zugehörigkeit für jede Klasse beinhaltet.

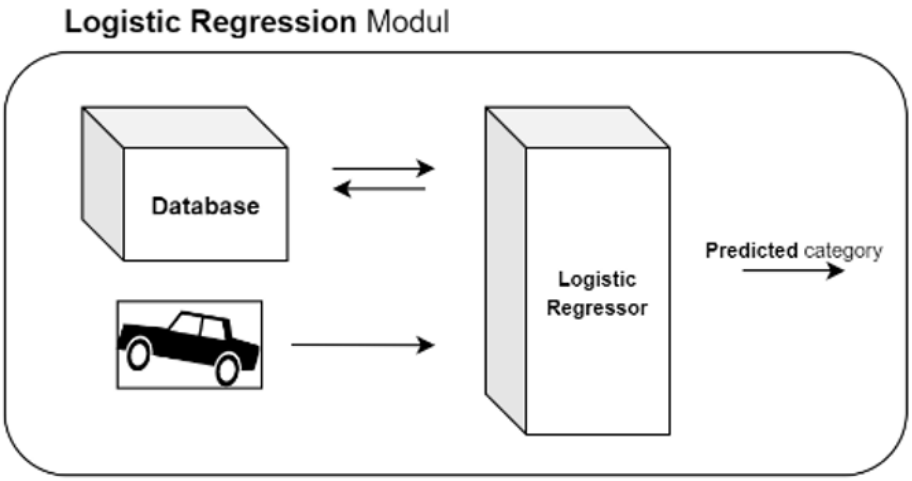


Abbildung 30 Umsetzung der Vorhersage durch Logistik Regression

Die Kommunikation mit der Datenbank entsteht dadurch, dass der *Logistic Regressor* an die Daten angepasst werden muss sowie bei Bearbeitung dieser Daten eine erneute Anpassung erfordert. Daher empfiehlt es sich, mehrere Produktänderungen zusammen durchzuführen.

3.6 Erkennung

Ein Gesamtüberblick über den Ablauf der Erkennung verschiedener Produkte in einem Bild ist in Abbildung 31 dargestellt.

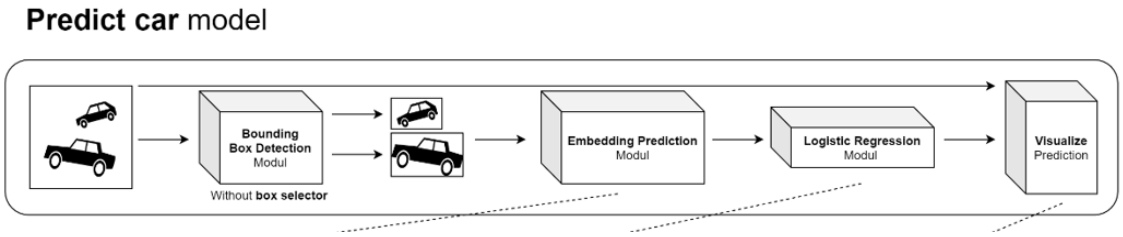


Abbildung 31 Ablauf Erkennung verschiedener Produkte in einem Bild

Im ersten Verarbeitungsschritt erstellt das oben beschriebene *Bounding Box Detection Modul* für jedes erkannte Produkt im gegebenen Bild einen Bildausschnitt, ohne die Box Selektion durchzuführen. Für jeden dieser Ausschnitte wird durch das *Embedding Modul* eine Vektordarstellung erstellt, welche durch das *Logistic Regression Modul* klassifiziert wird.

3.7 Darstellung der Vorhersagen

Im Ausgabeschritt werden die Vorhersagen der einzelnen Bildausschnitte in Textform auf dem Originalbild ausgegeben. Für jedes erkannte Objekt wird der Name der Kategorie sowie die Wahrscheinlichkeit der Zugehörigkeit angegeben.

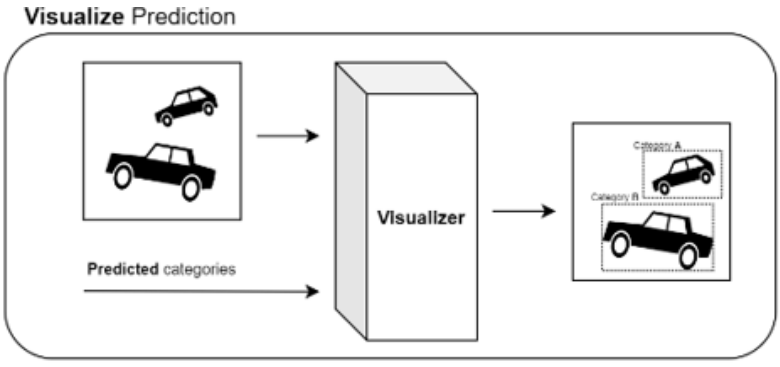


Abbildung 32 Visualisierung der Ausgabe

4 Experimente

Im Umfang dieser Arbeit wurden verschiedene Experimente, bezogen auf die Umsetzung und Gestaltung der Pipeline, durchgeführt. Zunächst wird auf die verwendeten Arbeitsmaterialien eingegangen sowie eine Übersicht über die Experimente gegeben. Die Ergebnisse werden im folgenden Teil analysiert und diskutiert.

4.1 Arbeitsmaterialien

Orange Data Mining, eine Open Source Software für maschinelles Lernen, wurde verwendet, um Experimente bezogen, auf Image Embedding durchzuführen. Diese Software erlaubt es, neben dem beschriebenen Verfahren der Logistischen Regression auch anderen Methoden wie SVM, Random Forest, Native Bayes, etc. zu evaluieren. Hierbei werden eine Vielzahl von Netzwerken für den Embedding - Prozess bereitgestellt, jedoch nicht EfficientNet.

Als Programmierumgebung wurde Google Colab mit Python verwendet. Dies bietet den Vorteil, schon alle benötigten Pakete für die Verwendung neuronaler Netze durch Tensorflow zu nutzen. Des Weiteren erlaubt es, eine GPU (Grafikkarte) einzubinden. Da *CNN* mit Filter, demnach Matrixoperationen, arbeiten, wird dadurch ein deutlicher Geschwindigkeitszuwachs gegenüber einer CPU erreicht.

Die Umsetzung der in Kapitel 3 beschriebenen Produkterkennung / Automodelerkennung ist über folgenden Link einzusehen.

https://github.com/PaSeitz/BA-Fahrzeugerkennung

4.2 Verschiedene Verfahren für Image Embedding

Das Ziel dieses Versuchsaufbau ist es zu analysieren, welches Verfahren zur Separierung und Zuordnung von Kategorien die höchste Genauigkeit erzielt. Im Versuch wird dazu ein Datensatz, unterteilt in 10 verschiedene Kategorien, betrachtet.

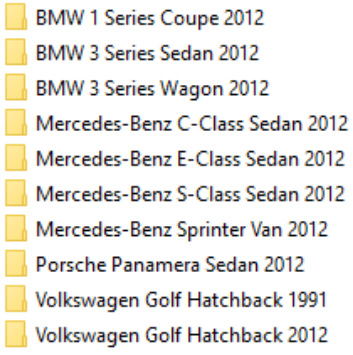


Abbildung 33 Datensatz zur Evaluierung der Genauigkeit verschiedener Verfahren für Image Embedding.

Für jede Klasse liegen zwischen 41 – 46 Bilder im Trainingsset und weitere 41 – 46 unterschiedliche im Testset vor. Jedes dieser Bilder wurde zugeschnitten, sodass das zu erkennende Objekt das gesamte Bild ausfüllt. Die Bilder wurden aus dem *Stanford Cars* Datensatz entnommen.

Abbildung 34 Format der verwendeten Bilder zur Evaluierung der verschiedenen Verfahren für Image Embedding.

4.2.1 Arbeitsmaterialien

- Orange Data Mining
- Standford Cars Dataset
- GIMP

4.2.2 Versuchsaufbau

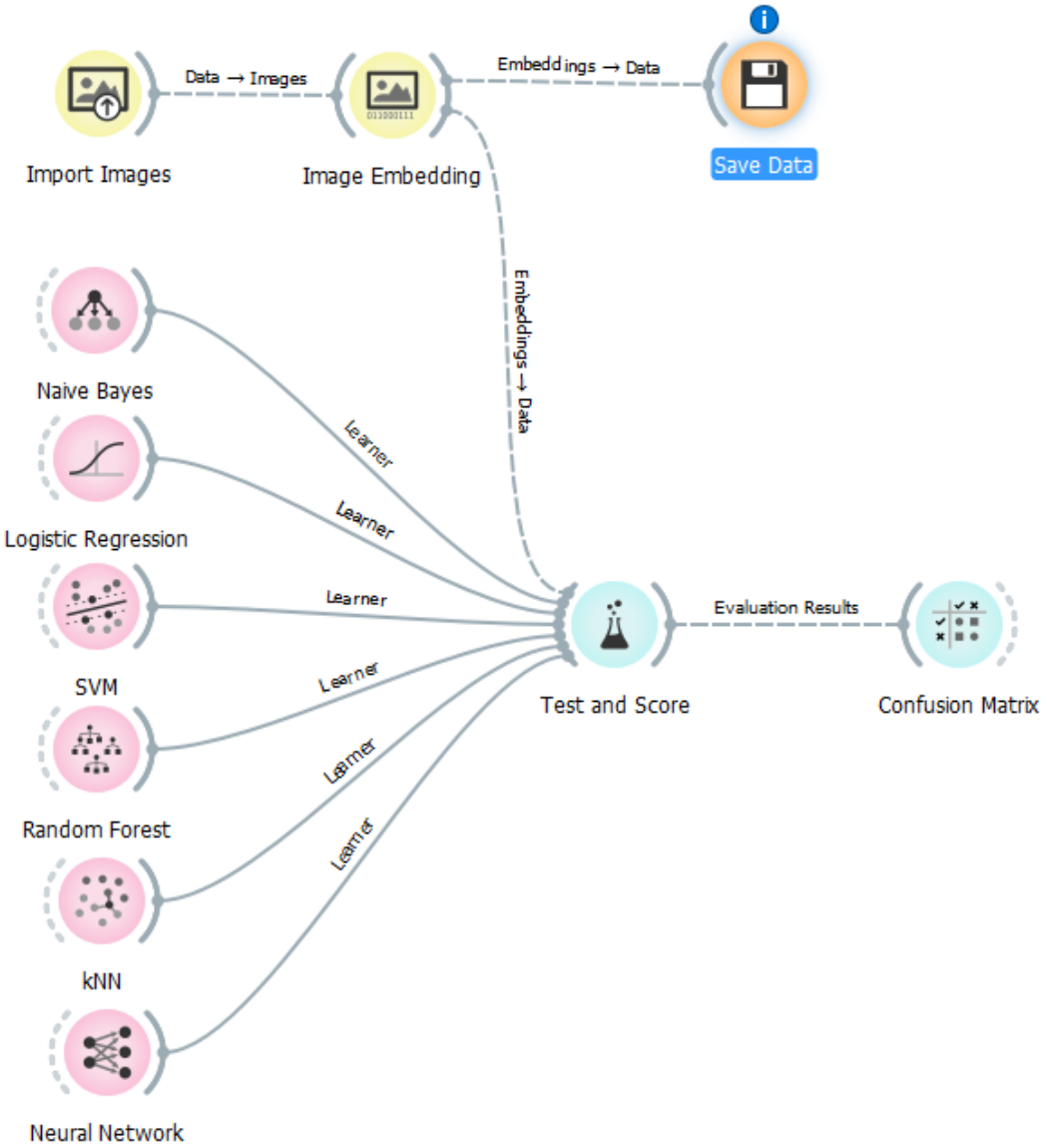


Abbildung 35 Versuchsaufbau in Orange zum Vergleich von Naive Bayes, Logistic Regression, SVM, Random Forest, kNN (Nächste Nachbar Klassifikation), Neural Network (1 versteckte Schicht mit 100 Neuronen).

4.2.3 Versuchsdurchführung

Zunächst wurden 10 verschiedene Kategorien / Ordner mit den entsprechenden Bildern aus dem *Stanford Cars* Datensatz ausgewählt. Hierbei wurden absichtlich Kategorien ähnlicher Modelle wie Mercedes-Benz S-Class Sedan 2012 und Mercedes-Benz C-Class Sedan 2012 gewählt, da diese selbst für den Menschen schwer zu unterscheiden sind.

Mittels GIMP einer Bildbearbeitungssoftware wurden die Bilder per Hand in die gewünschte Form transformiert (siehe Abbildung 34). Die Trainings sowie Testbilder werden zusammengefügt und mit einer 80/20 Teilung für das Training und die Evaluierung der Methoden verwendet. Die Bilder werden von Orange importiert und die Klassifizierungsgenauigkeit der verschiedenen Verfahren bestimmt. Als *Embedding Model* wurde VGG-16, VGG-19 und Inception v3, trainiert mit dem ImageNet Datensatz, verwendet.

4.2.4 Ergebnisse

4.2.4.1 Vergleich

VGG-19

Model	AUC	CA
kNN	0.920	0.648
SVM	0.934	0.646
Random Forest	0.867	0.550
Neural Network	0.945	0.727
Naive Bayes		0.629
Logistic Regression	0.963	0.764

VGG-16

Model	AUC	CA
kNN	0.909	0.622
SVM	0.927	0.645
Random Forest	0.862	0.558
Neural Network	0.946	0.713
Naive Bayes		0.654
Logistic Regression	0.965	0.751

Inceptionv3

Model	AUC	CA
kNN	0.935	0.706
SVM	0.977	0.795
Random Forest	0.882	0.603
Neural Network	0.982	0.823
Naive Bayes		0.720
Logistic Regression	0.983	0.826

Abbildung 36 Ergebnisse der Bestimmung der Klassifizierungsgenauigkeit verschiedener Verfahren für Image Embedding, durch VGG-16, VGG-19 und Inception v3. CA (Klassifizierungsgenauigkeit).

Logistic Regression (LG) hat mit einer Klassifizierungsgenauigkeit von 82.6% den höchsten Wert im Vergleich erreicht sowie in jedem Versuch die höchste Genauigkeit erzielt. Es ist außerdem erkennbar, dass das *Embedding Model* einen großen Einfluss auf die Genauigkeit aller Methoden besitzt, mit Inception v3 als Maßstab.

4.2.4.2 *Confusion Matrix (Konfusionmatrix) für Logistische Regression*

ID 0	**BMW** 1 Series Coupe 2012
ID 1	**BMW** 3 Series Sedan 2012
ID 2	**BMW** 3 Series Wagon 2012
ID 3	**Mercedes** C-Class Sedan 2012
ID 4	**Mercedes** E-Class Sedan 2012
ID 5	**Mercedes** S-Class Sedan 2012
ID 6	**Mercedes** Sprinter Van 2012
ID 7	**Porsche** Panamera Sedan 2012
ID 8	**VW** Golf Hatchback 1991
ID 9	**VW** Golf Hatchback 2012

	ID 0	ID 1	ID 2	ID 3	ID 4	ID 5	ID 6	ID 7	ID 8	ID 9
ID 0	81.5 %	11.5 %	0.6 %	0.5 %	4.7 %	0.6 %	0.0 %	2.2 %	0.5 %	1.7 %
ID 1	13.2 %	75.0 %	8.4 %	3.8 %	2.6 %	4.0 %	0.0 %	0.5 %	0.0 %	2.8 %
ID 2	2.0 %	4.7 %	74.3 %	3.2 %	1.0 %	1.7 %	0.0 %	2.2 %	6.3 %	5.1 %
ID 3	1.3 %	4.1 %	3.6 %	75.8 %	7.3 %	2.9 %	0.0 %	1.6 %	0.0 %	1.7 %
ID 4	0.0 %	0.7 %	3.0 %	6.5 %	73.1 %	5.7 %	0.0 %	2.2 %	0.5 %	0.0 %
ID 5	0.0 %	1.4 %	0.6 %	5.9 %	8.8 %	82.2 %	0.0 %	1.6 %	0.0 %	1.7 %
ID 6	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	99.4 %	0.0 %	0.0 %	0.0 %
ID 7	2.0 %	2.0 %	1.2 %	0.0 %	1.6 %	1.7 %	0.0 %	88.5 %	0.0 %	0.6 %
ID 8	0.0 %	0.7 %	3.0 %	1.1 %	0.0 %	0.0 %	0.0 %	0.0 %	92.7 %	2.8 %
ID 9	0.0 %	0.0 %	5.4 %	3.2 %	1.0 %	1.1 %	0.6 %	1.1 %	0.0 %	83.6 %

Abbildung 37 Confusion Matrix für Logistische Regression mit Inception v3.

4.2.5 Auswertung

Auf Basis der Ergebnisse aus Abbildung 36 hat sich Logistische Regression als das Modell mit der höchsten Genauigkeit für diesen Anwendungsfall herausgestellt. Zu beachten hierbei ist jedoch, dass verschiedene andere Verfahren, wie SVM und Neural Network nicht optimiert wurden. Somit sind diese Ergebnisse nur dahingehend aussagekräftig, dass sich Logistische Regression als robustes einfaches Verfahren, welches für diesen Anwendungsfall gute Ergebnisse liefert, erwiesen hat. Dies dient im Kontext dieser Arbeit als Entscheidungsbasis für die Auswahl Logistischer Regression als Klassifizierungsmethode. Da unterschiedliche Ergebnisse, basierend auf der Verwendung von VGG-16, VGG-19 oder Inception v3 auftreten, schließt sich daraus, dass das Embedding Model einen großen Einfluss auf die Genauigkeit der Klassifizierungsmodelle besitzt.

Anhand der *confusion matrix* (Siehe Abbildung 37) ist erkennbar, wie häufig ein Automodell richtig oder falsch klassifiziert wurde. Unter Betrachtung der Diagonale fällt auf, dass ID **6** (Mercedes Sprinter Van 2012) mit 99.4% einen deutlich über der Durchschnittklassifizierung (82.6%) liegenden Wert erzielt. Dies ist dadurch erklärbar, dass keine andere Autoklasse dieser ähnelt, dadurch fällt die Klassifizierung für den Menschen als auch das Modell deutlich leichter. Für ID 4 (Mercedes E-Class Sedan 2012) ist jedoch das genaue Gegenteil der Fall. Mit ID 5 (Mercedes S-Class Sedan 2012) ist ein Automodell im Datensatz, das dieser deutlich ähnelt und dementsprechend in 8.8% der Klassifizierung fälschlicherweise dieser Kategorie zugeordnet wird.

Zu beachten bei diesem Versuch ist, dass keine Datenerweiterung angewendet wurde, sondern lediglich die Originalbilder des Datensatzes benutzt wurden. Damit ist die Anzahl der Bilder mit 82 – 92 (Test- und Trainingsbilder) pro Klasse gering.

4.3 Auswirkung der Datenerweiterung auf die Genauigkeit der Bilderklassifizierung.

In diesem Versuch wird mit verschiedenen Verfahren der Datenerweiterung, deren Einfluss auf die erzielte Klassifizierungsgenauigkeit des *Image Embedding* Verfahren ermittelt.

4.3.1 Arbeitsmaterialien

- Standford Cars Dataset
- Google Colab (Python)
- scikit (LogisticRegression)
- Tensorflow

4.3.2 Versuchsaufbau

Basierend auf den Ergebnissen des Versuchs, verschiedene Verfahren für *Image Embedding (4.2)*, wurde folgender Versuchsaufbau gestaltet (Siehe Abbildung 38). Hierzu wurde Logistische Regression als Klassifizierer und EfficientNet – B5 anstelle Inception v3 verwendet. Dieser Versuchsaufbau kann somit nicht mit Orange umgesetzt werden, sondern wurde in Google Colab durchgeführt.

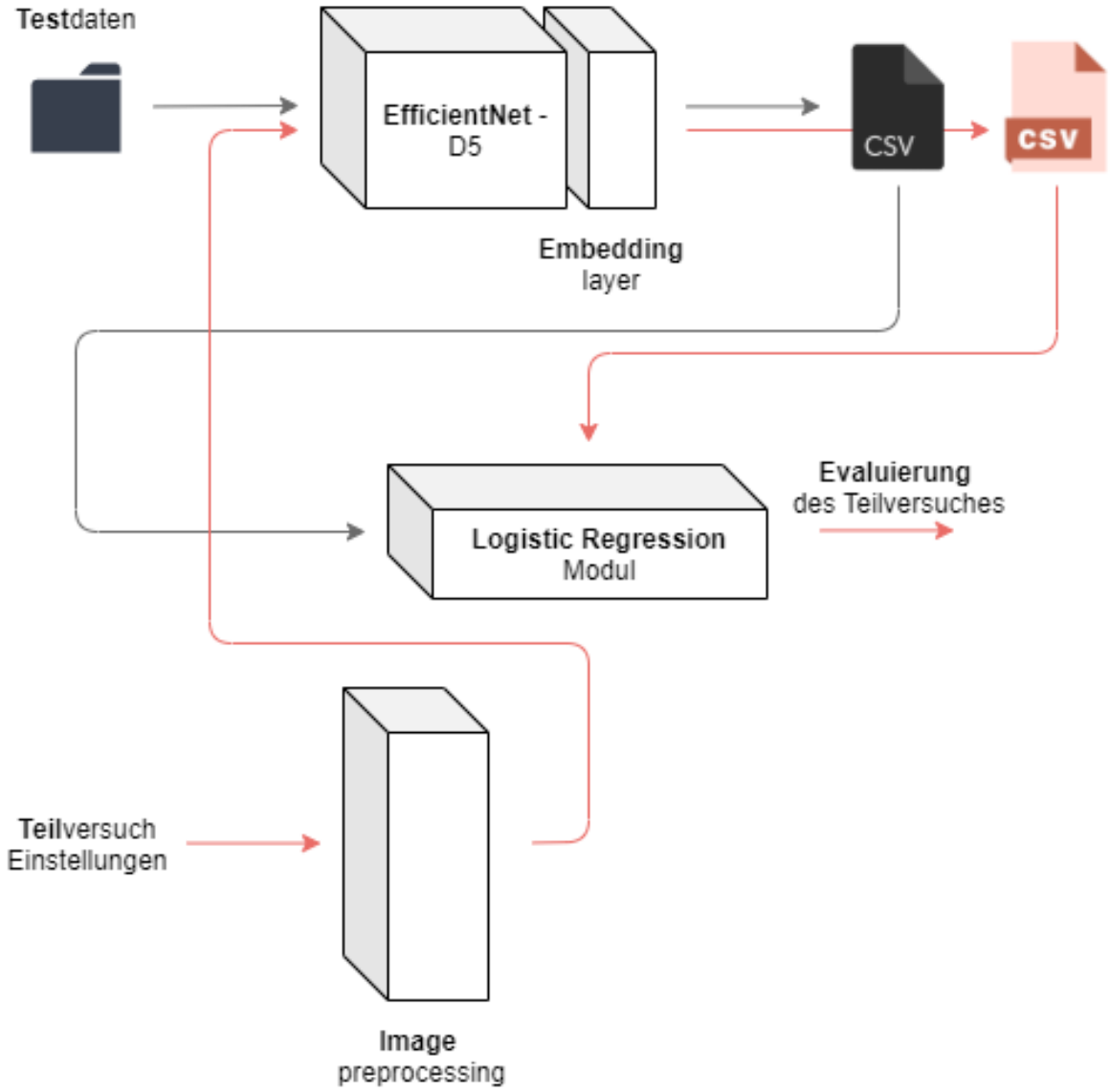


Abbildung 38 Versuchsaufbau zum Vergleich verschiedener Methoden der Datenerweiterung. EfficientNet – B5 mit Logistischer Regression.

4.3.3 Versuchsdurchführung

Ein Datensatz, bestehend aus denselben Kategorien, wie im Versuch, verschiedene Verfahren für *Image Embedding,* wurde erstellt. Dieser ist unterteilt in einen Test und Trainingsset mit jeweils unterschiedlichen Bildern derselben Fahrzeugklasse. In Python wurden dazu Funktionen geschrieben, welche die einzelnen Methoden der Datenerweiterung (siehe Datenerweiterung) implementieren. Hierzu sind verschiedene Teilversuche erstellt worden, welche die Grunddaten mit verschiedenen Methoden erweitern und mit dem Versuchsaufbau (siehe Abbildung 38) evaluieren.

4.3.4 Teilversuche

Abbildung 39 Übersicht über Teilversuche für Evaluierung verschiedener Verfahren der Datenerweiterung.

Jeder Teilversuch besteht aus den Originalbildern plus den zusätzlichen bearbeiteten Bildern. Jede Veränderung basiert auf dem Originalbild und wird nicht von der vorherigen Veränderung übernommen. Bei Teilversuch 1 werden die Seitenverhältnisse angepasst, hierbei ist zu beachten, dass die Testbilder ebenfalls auf dasselbe Format angepasst wurden.

4.3.5 Ergebnisse

Teilversuch ID	Erzielte Genauigkeit
1	83.72%
2	85.79%
3	86.39%
4	88.76%
5	90.53%

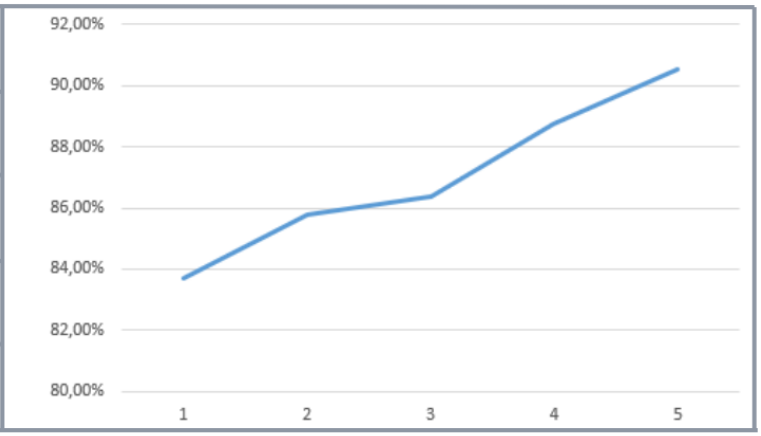


Abbildung 40 Ergebnisse der verschiedenen Teilversuche zur Datenerweiterung.

4.3.6 Auswertung

Teilversuch 1 hat im Vergleich zu der erzielten Genauigkeit der Daten ohne Anpassung der Seitenverhältnisse (84.32% erreichte Klassifizierungsgenauigkeit ohne jegliche Form der Datenerweiterung mit EfficientNet - D5) eine geringere Genauigkeit erzielt. Durch das Gleichbleiben der Seitenverhältnisse entstehen unterschiedliche große schwarze Ränder an den Seiten des Bildes, was bei einer limitierten Bildgröße (456 x 456), die ohnehin schon geringe Anzahl an Pixel weiter einschränkt und damit die Summe der Informationen im Bild reduziert. Auch wenn durch das Strecken des Bildes über die gesamte Eingangsgröße die Proportionen verzerrt werden, wurden damit bessere Klassifizierungsergebnisse erreicht. Im weiteren Versuchsverlauf wurde durch Ergänzungen weiterer Verfahren, eine höhere Vielfalt der Daten und somit eine steigende Genauigkeit erzielt.

5 Fazit

Unter Betrachtung der verschiedenen theoretischen Grundlagen bietet die in dieser Arbeit vorgeschlagene Objekterkennung, eine Basis für das Verstehen und Optimieren weiterer Schritte. Durch Durchführung der unterschiedlichen Versuche wurde ein Verständnis für verschiedene Arten der Datenerweiterung und deren Einfluss auf den Gesamterfolg (Erzielte Genauigkeit) des Modells sowie Informationen bezüglich verschiedener Klassifizierungsmöglichkeiten erlangt. Die hierbei gewählte Methode (Logistische Regression) basiert auf den Ergebnissen von dem Versuch, verschiedene Verfahren für Image Embedding (Kapitel 4.2), wobei sich dieses Verfahren als einfache und robuste Variante herausgestellt hat. Jedoch kann nicht definitiv festgelegt werden, dass es sich hierbei um die effizienteste oder genaueste Variante handelt. Ein tieferer Vergleich mit SVM und einem Neuronalen Netzwerk, der verschiedenen Aspekte, wie die Genauigkeit mit unterschiedlichen Parametern, sowie die Anpassungszeit bei Hinzufügen oder Entfernen von Produkten mit in Betracht zieht, würde genauere Aufschlüsse darüber liefern.

Aufgrund der Anforderung, Produkte hinzufügen sowie entfernen zu können, wurde mit *Image Embedding* ein Verfahren eingebunden, welches eine erneute Anpassung des Klassifizierungsmodells (Logistische Regression) erleichtert, da dadurch nur die alten und neu erstellten *embeddings*, anstatt dem gesamten Bilddatensatz, dem Modell gezeigt werden müssen.

Die damit entworfenen Pipeline ist, durch Erstellung verschiedener Teilmodule, in der Lage, eine Vielzahl von Produkten zu erkennen und zu klassifizieren. Dazu wurde ein *CNN* verwendet, welches als vortrainiertes Netzwerk, Auswahlmöglichkeiten für verschiedene Produktkategorien bereitstellt. Diese Auswahl beschränkt die aktuelle Umsetzung jedoch und ist demnach nicht für alle Anwendungsfälle anwendbar, jedoch für eine Autoklassifizierung ausreichend.

6 Ausblick

Bildverstehen ist ein sich rasant weiterentwickelndes Feld. Dadurch entstehen immer neue Verfahren / Methoden, welche sich durchaus interessant für mögliche Erweiterungen des bisher betrachteten Aufbaus erweisen könnten.

6.1 Erweiterung und Optimierung

In diesem Abschnitt werden einige Methoden und Überlegungen diskutiert, welche bei der Recherche dieser Arbeit aufgekommen sind, jedoch wegen zeitlicher Limitierung nicht weiter erläutert und betrachtet wurden.

Eine Erweiterung der erstellten Pipeline dieser Arbeit wäre Instance Segmentation (Instanz Segmentierung). Hierbei wird das Bild auf Pixelbasis klassifiziert und erkannte Objekte in Instanzen unterteilt, sodass Pixel genaue Masken der erkannten Objekte erstellt werden. Basierend darauf kann die genaue Form der Fahrzeuge erkannt werden, um möglicherweise, einen Einfluss von Hintergrundobjekten stark zu miniminieren. Mit einer Datenbank, welche die Formen der Fahrzeuge aus verschiedensten Blickwinkeln beinhaltet, kann dies z.B. durch die Überlappungsfläche mit in die Klassifizierungsmethode einfließen. Unter Annahme einer festmontierten in Fahrtrichtung blickenden Kamera würde sich die Anzahl der notwendigen Blickwinkel stark reduzieren, da Autos hauptsächlich von hinten oder vorne abgebildet werden.

Abbildung 41 Unterschied zwischen Objekt Erkennung (Links) und Instanz Segmentierung (Rechts). Bildquelle: Stanford Cars Dataset

Außerdem können verschiedene Aspekte des *Embedding Models* optimiert werden. Da dieses mit dem ImageNet Datensatz, welcher aus 14 Millionen Bilder besteht, trainiert wurde, ist es durchaus sinnvoll dies zu verwenden, jedoch lernt das Netzwerk dadurch Strukturen / Filter, welche für den Anwendungsfall möglicherweise uninteressant sind. Theoretisch wäre es demnach besser, das Netzwerk mit dem eigenen Datensatz für die entsprechenden Anwendungen selbst zu trainieren. Aber es ist unrealistisch, einen Datensatz mit vergleichbarer Anzahl Bilder, wie der ImageNet-Datensatz, zu erstellen. Da

jedoch *CNN* in den unteren Schichten weniger abstrakte Merkmale erkennt, bietet es sich an, lediglich die obersten Schichten des vortrainierten Netzwerkes mit dem eigenen Datensatz weiter zu trainieren, um das Netzwerk für die entsprechende Anwendung zu optimieren.

6.1.1 Generierung von Daten

Ein wichtiger Aspekt ist die Erstellung eines Datensatzes. Hierzu wurde in dieser Arbeit auf den Stanford Cars Datensatz zurückgegriffen. Dieser stellt schon eine Vielzahl von Autoklassen bereit, jedoch sind die neuesten Autos, Modelle von 2012, und die Anzahl der Bilder pro Klasse fällt mit 41 – 46 Bildern gering aus. Neben den Verfahren der Datenerweiterung gibt es weitere Möglichkeiten. Ein bekanntes Beispiel hierfür ist, die Einbindung von Nutzern, zur Erstellung und Vorverarbeitung (Label und *bounding boxen*) der Daten. Dieses Verfahren wird z.B. bei der Autorisierung auf manchen Webseiten verwendet. Dem Benutzer wird hierbei eine Auswahlmöglichkeit für verschiedenen Bildbereiche gegeben, woraus jene ausgewählt werden sollen, welche das entsprechende Objekt beinhalten. Eine weitere Möglichkeit ist es, Bilder künstlich zu generieren. Ein von NVIDIA entwickeltes Verfahren ist in der Lage, Gesichter, welche nicht nur keiner echten Person entsprechen, sondern auch täuschend echt aussehen, zu erzeugen. Hierbei wird eine Netzwerk Architektur, Generative Adversarial Networks kurz GAN, verwendet, welche aus einem Generator und Diskriminator aufgebaut ist. Der Generator erstellt basierend auf einem zufälligen Rauschen ein Bild, welches durch den Diskriminator als natürlich oder künstliches Bild klassifiziert wird. Dieser Prozess wird im Trainingsvorgang so lange wiederholt bis der Diskriminator idealerweise die künstlich erstellten Bilder als real klassifiziert. Die Möglichkeit mit diesem oder ähnlichen Methoden, eine Vielzahl an Bildern künstlich zu generieren, kann sich als attraktives Verfahren anbieten.