|  |  |  |
| --- | --- | --- |
| **MA 3062** | | |
| Untersuchung und prototypische Umsetzung eines Lifelong Deep Neural Network Algorithmus | | |
| **Simon Kamm** | | |
|  | | |
| **Konzeption** | | |
|  | Prüfer: | Prof. Dr.-Ing. Michael Weyrich |
|  | Betreuer: | Benjamin Maschler, M.Sc. |
| Start: 29.04.2019 | | Abgabe: 29.10.2019 |
|  | |  |

**Dokument Versionsverwaltung**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Version | Autor | QS | Datum | Status | Änderungen |
| 0.1 | Kamm |  | 27.05.19 | in Bearb. | Erstellung |
| 1.0 | Kamm | Ms | 14.06.19 | Vorgelegt | Finalisierung Konzeption |
| 1.0 | Kamm | Ms | 24.06.19 | Akzeptiert | Durchsicht/Durchsprache Ms |

# Inhaltsverzeichnis

0 Inhaltsverzeichnis 2

1 Beschreibung der Konzeption 3

1.1 Modul A 3

1.1.1 AlexNet 4

1.1.2 VGG 4

1.1.3 ResNet 6

1.1.4 GoogLeNet/Inception 8

1.1.5 MobileNet 9

1.2 Modul B 11

2 Zusammenfassung 14

2.1 Modul A 14

2.2 Modul B 16

2.3 Lifelong DNN Algorithmus 17

3 Literaturverzeichnis 19

# Beschreibung der Konzeption

In diesem Dokument wird die konkrete Konzeption des Lifelong DNN (L DNN) Algorithmus beschrieben. Dafür werden unterschiedliche Architekturen untersucht, verglichen und schließlich eine konkrete Architektur ausgewählt, welche im weiteren Verlauf der Arbeit implementiert wird.

Der L DNN Algorithmus besteht aus zwei Modulen, Modul A und Modul B. Details zum Aufbau und den grundlegenden Ideen des L DNN Algorithmus sind in den „Grundlagen“ genannt. Für eine graphische Übersicht ist in Abbildung 1 die grobe Architektur des L DNN Algorithmus dargestellt.

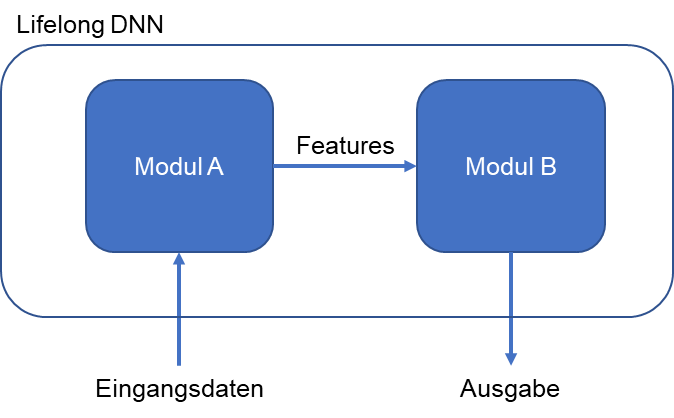


Abbildung 1: Graphische Darstellung des L DNN A

Im Folgenden werden nun unterschiedliche Architekturen für die einzelnen Module A und B aufgeführt, verglichen und schließlich für die in dieser Arbeit gestellten Anforderungen bewertet.

## Modul A

Modul A ist das langsame (oder nicht) lernende Modul innerhalb des L DNN Algorithmus. Es ist ein klassisches DNN, welches zur Extraktion von relevanten Features auf Basis von Eingangsdaten genutzt wird. Im Rahmen dieser Arbeit werden vortrainierte, feste DNN-Architekturen genutzt, da das Trainieren solcher DNN-Architekturen sehr zeit- und rechenaufwändig ist.

Es gibt eine Vielzahl an DNN-Architekturen, die heutzutage zur Feature-Extraktion eingesetzt werden. Jede Architektur erfüllt dabei unterschiedliche Anforderungen, beziehungsweiße nutzt unterschiedliche Ansätze, die für den einen oder anderen Anwendungsfall besser geeignet sind. Für Modul A innerhalb des L DNN Algorithmus ist wichtig, dass die extrahierten Features eine bestmögliche Klassifikation erlauben. Zudem soll das System auch in Echtzeitsystemen auf mobilen Endgeräten (z.B. Smartphone) funktionieren [1], [2]. Dafür muss Modul A Eingangsdaten schnell verarbeiten können und gleichzeitig darf der Speicher- und Rechenbedarf des Netzwerkes nicht zu groß sein, da sowohl Speicher als auch Rechenleistung auf mobilen Endgeräten limitiert ist.

Im Folgenden werden bekannte Klassifikator-Netzwerke auf Basis von DNN Architekturen vorgestellt und auf deren Einsetzbarkeit in dieser Arbeit verglichen. Als Grundlage für den späteren Vergleich und für die genannten Metriken wird der Bilddatensatz ImageNet [3] genutzt. Mit ca. 1,2 Millionen Bildern aus 1000 verschiedenen Klassen stellt ImageNet ein sehr komplexes Problem dar. Vortrainierte Netzwerke, welche zur Extraktion von Features genutzt werden, sind heutzutage auf diesem Datensatz trainiert, da er aufgrund seiner großen Anzahl an Trainingsbildern und Klassen eine gute Generalisierung der extrahierten Features gewährleistet.

### AlexNet

AlexNet war eines der ersten DNNs welches die Klassifikationsgenauigkeit auf ImageNet signifikant erhöhte. Im Rahmen der ILSVRC 2012 (*ImageNet Large Scale Visual Recognition Competition*) erreichte AlexNet einen Top-5 Klassifikationsfehler von 15,3%. Der zweitplatzierte dieses Wettbewerbs erreichte lediglich einen Top-5 Klassifikationsfehler von 26%. Es konnte eine erhebliche Steigerung der Klassifikationsgenauigkeit erzielt werden. Dabei ist AlexNet im Vergleich zu aktuellen Architekturen sehr simpel, mit lediglich fünf Convolutional, verschiedenen Max-Pooling sowie drei Fully Connected Schichten [4], [5]. Neuerungen im Vergleich zu damaligen DNNs war der Einsatz von ReLU-Aktivierungsfunktionen sowie die Regularisierung des Netzwerks mithilfe von Dropout [6]. Beide Methoden sind seitdem in nahezu allen DNN-Architekturen im Einsatz. Das Netzwerk besteht aus 60 Millionen Parametern und 650.000 Neuronen. In Abbildung 2 ist die Modellarchitektur des AlexNet graphisch dargestellt, bei welcher die fünf Convolutional und die drei Fully Connected (Dense) Schichten zu sehen sind.

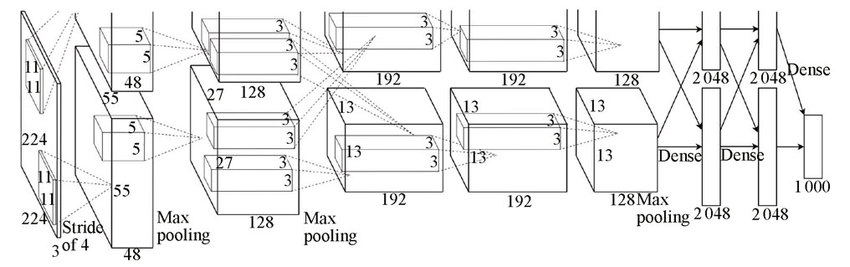


Abbildung 2: Modelarchitektur des AlexNet [5]

Die *Default*-Größe eines Parameters (Floating Point oder Integer) in TensorFlow ist mit 32-bit angegeben [7]. Unter der Annahme dieser Parametergröße benötigt AlexNet mit 60 Millionen Parametern 240 MB Speicher zur Sicherung des Modells. Um die Berechnungskomplexität vergleichen zu können wird die notwendige Anzahl an Operationen für ein einzelnes Eingangselement (z.B. einzelnes Bild) genutzt. Dabei wird lediglich der Vorwärtspfad in Betracht gezogen, wie es bei der späteren Anwendung (*Inference*) des Netzwerkes auch wäre. Diese Anzahl ist bekannt als Nummer von FLOPs (**F**loating **P**oint **OP**erations). AlexNet benötigt 727x106 FLOPs um alle notwendigen Berechnungen im Vorwärtspfad durchzuführen.

### VGG

Die sogenannten VGG-Architekturen entstammen der *Visual Geometry Group* (VGG) der Universität Oxford. Dabei gibt es einige bekannte Architekturen. Die am häufigsten genutzten sind das VGG-16 sowie das VGG-19 Netzwerk. Die Zahlen stehen dabei für die Anzahl an Gewichts-/Parameterschichten innerhalb des DNN. Von der grundsätzlichen Architektur ähnelt es dem bereits beschriebenen AlexNet. Der große Unterschied ist der Einsatz kleinerer Kernel-Filter in den *Convolutional* Operationen (3x3 Kernel anstatt z.B. 11x11 oder 5x5 Kernel in AlexNet). Zudem werden mehrere dieser *Convolutional* Schichten hintereinander gesetzt, wodurch die finale Funktion des DNN diskriminativer werden soll. Ein großer Vorteil ist die geringere Anzahl an Parametern. Unter der Annahme, dass der Eingang und Ausgang eines *Convolutional Stacks* mit drei 3x3 *Convolutional* Schichten Kanäle besitzt, werden Parameter benötigt. Wird lediglich eine einzelne *Convolutional* Schicht mit 7x7 Kernel-Filtern genutzt, werden Parameter benötigt [6]. Durch diese Parameterreduzierung können tiefere Netzwerke mit mehr Schichten aufgebaut werden ohne einen signifikant erhöhten Speicherbedarf zu haben. Durch den Anstieg der Tiefe des Netzwerks ist das Netzwerk fähig komplexere Features zu erlernen [8]. Unterschiedliche Architekturen der VGG-Netzwerke für den ImageNet Bilddatensatz mit 1000 Klassen und RGB Bildern der Größe 224x224 sind in Tabelle 1 dargestellt.

Tabelle 1:VGG-Netzwerk Architekturen [8]

|  |  |  |  |
| --- | --- | --- | --- |
| A  11 Gewichts-Schichten | B  13 Gewichts-Schichten | C  16 Gewichts-Schichten | D  19 Gewichts-Schichten |
| Eingangssignal (z.B. 224x224 RGB Bild) | | | |
| Conv3-64 | Conv3-64  Conv3-64 | Conv3-64  Conv3-64 | Conv3-64  Conv3-64 |
| Max-Pooling | | | |
| Conv3-128 | Conv3-128  Conv3-128 | Conv3-128  Conv3-128 | Conv3-128  Conv3-128 |
| Max-Pooling | | | |
| Conv3-256  Conv3-256 | Conv3-256  Conv3-256 | Conv3-256  Conv3-256  Conv3-256 | Conv3-256  Conv3-256  Conv3-256  Conv3-256 |
| Max-Pooling | | | |
| Conv3-512  Conv3-512 | Conv3-512  Conv3-512 | Conv3-512  Conv3-512  Conv3-512 | Conv3-512  Conv3-512  Conv3-512  Conv3-512 |
| Max-Pooling | | | |
| Conv3-512  Conv3-512 | Conv3-512  Conv3-512 | Conv3-512  Conv3-512  Conv3-512 | Conv3-512  Conv3-512  Conv3-512  Conv3-512 |
| Max-Pooling | | | |
| FC-4096 | | | |
| FC-4096 | | | |
| FC-1000 | | | |
| Softmax | | | |

Wie bereits geschrieben werden die Netzwerke anhand der Anzahl der Gewichtsschichten (*Convolutional* oder *Fully Connected* (FC) Schicht) benannt. So ist Netzwerk *C* als VVG-16 bekannt und Netzwerk D als VGG-19. Die häufig eingesetzten Feature-Extrahierer VGG-16 und VGG-19 werden im Folgenden bezüglich ihres Speicherbedarfes und der Klassifikationsgenauigkeit auf ImageNet untersucht. Dabei sind in Tabelle 2 die Werte der unterschiedlichen Kriterien für die einzelnen Netzwerke gelistet [8].

Tabelle 2: Vergleich von VGG-16 und VGG-19

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Netzwerk-Architektur | Anzahl Parameter | Speicher-bedarf  Parameter | Anzahl  FLOPs | Top-1 Klassifikationsfehler | Top-5 Klassifikationsfehler |
| VGG-16 | 138x106 | 552 MB | 16x109 | 25.6 % | 8.1 % |
| VGG-19 | 144x106 | 576 MB | 20x109 | 25.5 % | 8.0 % |

Die Anzahl der Parameter, und damit auch der Speicherbedarf, steigt mit der Anzahl an Schichten. Hinsichtlich des Klassifikationsfehlers auf ImageNet ist zu beobachten, dass durch die 3 zusätzlichen Schichten im VGG-19 Netzwerk ein nahezu vernachlässigbarer Genauigkeitsgewinn erreicht werden kann. Es werden 6 Millionen zusätzliche Parameter (24MB zusätzlicher Speicher) sowie 4x109 zusätzliche FLOPs benötigt, um den Top-1 und Top-3 Klassifikationsfehler um 0.1% zu verringern.

### ResNet

Aufgrund der bisher beschriebenen Modelle kann die Aussage getroffen werden, dass eine größere Tiefe des Netzwerks eine bessere Genauigkeit ermöglicht. Bei diesen Überlegungen muss jedoch berücksichtigt werden, dass bei der Backpropagation der Error-Vektoren durch das Netzwerk viele Schichten durchlaufen werden müssen. Dies führt dazu, dass der Error, der bei frühen Schichten ankommt, sehr klein oder gleich null ist und damit nur eine geringe oder gar keine Anpassung der Parameter stattfindet. Dieses Problem ist in der Literatur unter dem Namen *Vanishing Gradient* (Verschwindender Gradient) bekannt. Zusätzlich wurde bei sehr tiefen neuronalen Netzwerken das Degradations-Problem beobachtet. Mit einer zunehmenden Netzwerk-Tiefe findet eine Sättigung in Bezug auf die Genauigkeit des Netzwerkes statt. Ab einem gewissen Punkt degradiert diese Genauigkeit schließlich. Dieses Verhalten lässt darauf schließen, dass nicht alle Netzwerke beliebiger Architektur gleich zu trainieren sind.

Um diese genannten Probleme zu lösen, wurden *Deep Residual Networks*, häufig nur als ResNet (Residual Network) bezeichnet, eingeführt [9]. Diese ResNets nutzen dieselben grundlegende Schichten wie VGG-Netze, mit kleinen Kernel-Filter (3x3) und hauptsächlich *Convolutional* Schichten. ResNets bestehen zusätzlich aus einer Vielzahl von *Residual* Blöcken. Ein solcher *Residual* Block ist beispielhaft in Abbildung 3 dargestellt.

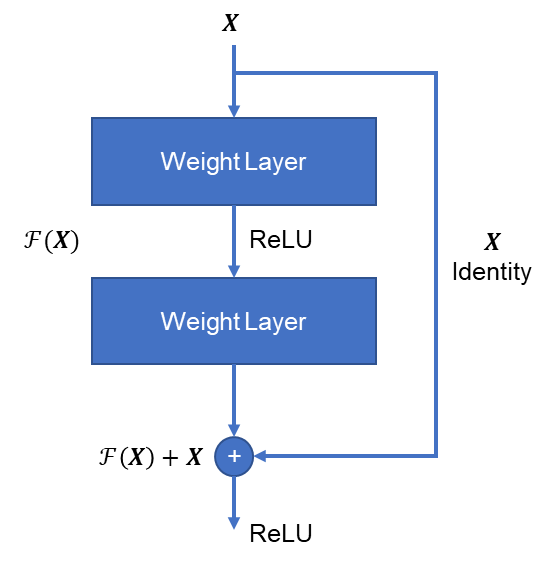


Abbildung 3: Beispielhafter Residual Block [9]

Diese *residual* Blöcke verwirklichen das gewünschte Verhalten des Eingangssignals mithilfe der non-linearen Verarbeitung und der Hinzunahme des ursprünglichen Eingangssignals **,** wodurch gilt: . Die Hinzunahme des Eingangssignals wird mithilfe von sogenannten *Shortcut Connections* realisiert. In den meisten ResNets entspricht die *Shortcut Connection* dabei dem Identity Mapping, in der Theorie kann diese Verbindung jedoch auch einen zusätzlichen Operator beinhalten. Mithilfe dieser Netzwerke können zum einen tiefe Netzwerke (mit mehr als 100 Schichten) effizient und schnell trainiert werden, zum anderen kann die Genauigkeit dieser Netzwerke weiter erhöht werden, da das Potenzial der zusätzlichen tiefen Schichten genutzt wird und keine Degradation stattfindet. Damit kann sowohl das Problem des *Vanishing Gradient*, als auch das *Degradation Problem* gelöst werden [9].

Ein Ausschnitt einer beispielhaften ResNet-Architektur ist in Abbildung 4 gegeben. Es sind exemplarisch nur die ersten Schichten dargestellt. Links ist als Referenz die Architektur des VGG-19 Netzwerks zu sehen. In der Mitte ist ein „normales“ DNN mit 34 Schichten gezeigt und rechts das dazu passende ResNet, ebenfalls mit 34 Schichten aber bestehend aus *residual* Blöcken.

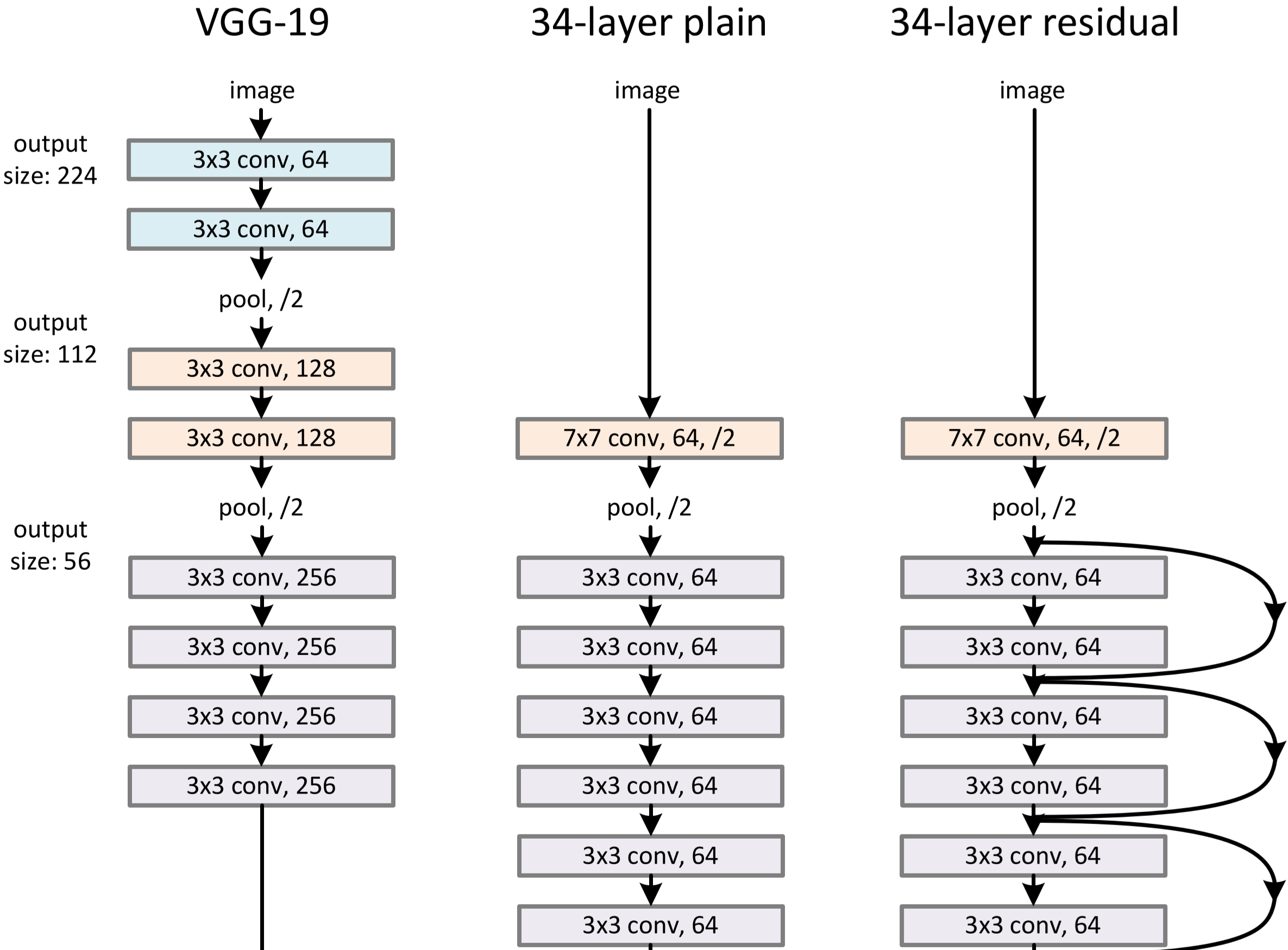


Abbildung 4: Beispielhafte Architektur VGG-19 (links), Standard-Architektur (mitte), ResNet (rechts) [9]

Für das Beispiel von 34 Schichten kann auf Basis des ImageNet Datensatzes der Top-1 Klassifikationsfehler von 28.54% im Falle des *plain* Netzwerks auf 25.03% mithilfe des ResNets reduziert werden. Die gesamte Modellarchitektur (Anzahl Parameter, Schichten, Multiplikationen…) ist dabei identisch, lediglich die *Shortcut Connections* sind zusätzlich eingefügt. Ein ResNet kann mit einer beliebigen Anzahl an Schichten aufgebaut werden. In dieser Arbeit werden als Referenzen eine Architektur mit 50 und 101 Schichten angegeben. Die Anzahl an Parameter, der benötigte Speicherbedarf bei Umsetzung in TensorFlow [7], sowie der Top-1 und Top-5 Klassifikationsfehler auf dem ImageNet Datensatz nach [9] sind in Tabelle 3 angegeben.

Tabelle 3: Vergleich von ResNet-50 und ResNet-100

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Netzwerk-Architektur | Anzahl Parameter | Speicher-bedarf Parameter | Anzahl FLOPs | Top-1  Klassifikationsfehler | Top-5  Klassifikationsfehler |
| ResNet-50 | 25.6x106 | 102 MB | 4x109 | 20.7 % | 5.3 % |
| ResNet-101 | 44.5x106 | 178 MB | 8x109 | 19.9 % | 4.6 % |

Im Vergleich zu den bereits eingeführten Architekturen (VGG oder ALexNet) kann hier mit weniger Speicherbedarf (geringerer Anzahl an Parametern) sowie deutlich weniger FLOPs dank der *Residual Blocks* eine bessere Genauigkeit auf dem ImageNet-Datensatz erreicht werden.

### GoogLeNet/Inception

Mithilfe der VGG-Architekturen konnte ein bemerkenswerter Sprung bei der Genauigkeit von DNNs auf dem ImageNet-Datensatz erzielt werden. Um diese VGG Netzwerke trainieren zu können bedarf es jedoch aufgrund der hohen Anforderungen für die Berechnung sehr teure Hardware (GPUs). An diesem Punkt setzt der Aufbau des GoogLeNet an, häufig auch als Inception Netzwerk bekannt [10], [11]. GoogLeNet nutzt die Idee, dass die meisten Aktivierungen in einem DNN entweder unnötig (Aktivierung gleich 0) oder aufgrund von Korrelationen redundant sind. Daher ist die effizienteste Architektur eines DNN ein Aufbau mit sparsen Verbindungen zwischen den Aktivierungen. Das bedeutet, dass nicht alle Ausgangskanäle mit allen Eingangskanälen in einer *Convolutional* Schicht verbunden sind, wie es typischerweise der Fall wäre [6]. GoogLeNet setzt dafür sogenannte Inception-Module ein, welche ein sparses CNN mit normalen Konstruktionen approximieren. Der Aufbau eines Inception-Moduls ist in Abbildung 5 dargestellt.

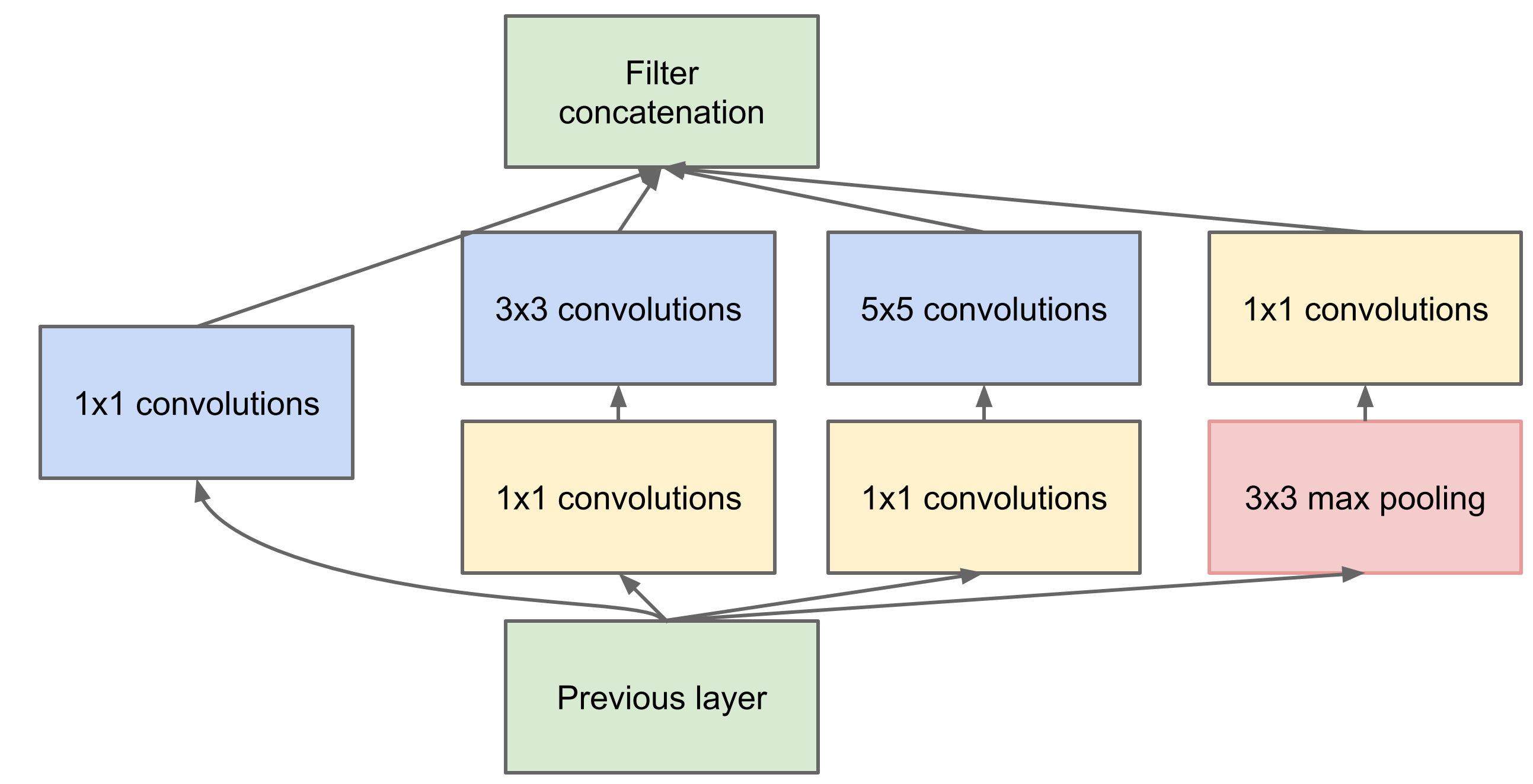


Abbildung 5: Inception-Modul [10]

Mithilfe dieser Architektur kann die Anzahl an benötigten Operationen (Multiplikation, Additionen) deutlich reduziert werden. Die 1x1 *Convolutions* dienen dabei der Dimensionsreduzierung der Kanäle, bevor größere Convolutional-Operationen (3x3 oder 5x5) durchgeführt werden. Als einfaches Beispiel kann dafür eine Berechnung von 192 Eingangskanälen an der 5x5 Convolution gesehen werden. Mit 32 Filtern würden Multiplikationen durchgeführt. Wird zuvor eine 1x1 Convolution mit 16 Filtern genutzt, sind nur Multiplikationen notwendig um dieselbe Ausgangsdimension zu erhalten. Mithilfe der Inception-Module kann ein sehr tiefes und weites Netz (große Eingangsdimensionen) aufgebaut werden. Zusätzlich zu diesen Änderungen verzichtet GoogLeNet auf Fully-Connected Schichten am Ende des Netzwerkes, und tauscht diese durch Pooling-Operationen aus. Dies reduziert die Anzahl an benötigten Parametern drastisch. Im AlexNet sind z.B. ca. 90% der Parameter in den Fully Connected Schichten enthalten [6].

Die dritte Version des Inception-Netzwerks (Inception-v3 [11]) besitzt 42 Schichten und erreicht einen Top-5 Klassifikationsfehler von 5.6% und einen Top-1 Klassifikationsfehler von 21.2%. Dabei werden 4.8x109 FLOPs ausgeführt und 24x106 Parameter genutzt. Unter der Annahme von 32-bit Variablen [7] ergibt das einen Speicherbedarf von 96 MB.

### MobileNet

Der Trend im Bereich DNN ist es, immer tiefere und komplexere Netzwerke zu gestalten, welche verbesserte Genauigkeiten für spezifische Aufgaben (z.B. ILSVRC) liefern. Als Folge von diesen Entwicklungen steigt häufig der Berechnungsaufwand und/oder der Speicherbedarf zur Umsetzung dieser Netzwerke. In vielen realen Anwendungen sind jedoch genau diese Themen kritisch, z.B. für Augmented Reality, autonome Fahrzeuge, Roboter und vieles mehr. In diesen Anwendungsgebieten sind echtzeitfähige Netzwerke notwendig, die auf kleinen Endgeräten (z.B. Smartphone oder Mikrocontroller) lauffähig sind, und keinen GPU-Server zur Verfügung haben. Aus diesen Anforderungen heraus wurde MobileNet [12] entworfen. Das Ziel der Architektur ist ein kleines Netzwerk mit geringer Latenz, dass für mobile Anwendungen genutzt werden kann. MobileNet nutzt dabei auch Inception-Module [10] um die Anzahl an Operationen zu reduzieren. Zusätzlich wird die *Depthwise Separable Convolution* eingesetzt. Dies ist eine Form der faktorisierten *Convolution*, bei dem die normale *Convolution* in eine *depthwise Convolution* und eine 1x1 *Convolution* (*pointwise* Convolution) zerlegt wird. Dadurch werden aus einer Schicht zwei Schichten, bei der die erste für die Filterung und die zweite für die Kombination der berechneten Filterausgänge verantwortlich ist [12]. Als Vergleich ist in Tabelle 4 der Unterschied für die Anzahl an FLOPs, der Anzahl der Parameter und der Genauigkeit auf dem ImageNet-Datensatz für ein MobileNet mit klassischen *Convolution*-Operatoren (Conv MobileNet) und einem MobileNet, welches *depthwise Convolutions* nutzt, gegeben.

Tabelle 4: *Depthwise Separable vs. Full Convolution MobileNet* [12]

|  |  |  |  |
| --- | --- | --- | --- |
| Modell | Top-1 Klassifikationsfehler | Anzahl FLOPs | Anzahl Parameter |
| Conv MobileNet | 28.3% | 4.87x109 | 29.3x106 |
| MobileNet | 29.4% | 0.57x109 | 4.2x106 |

Diese Ergebnisse zeigen, dass der Einsatz der d*epthwise Convolution* den Speicher- und Rechenaufwand drastisch reduziert, und der daraus resultierende Genauigkeitsverlust bei lediglich ca. 1% liegt, was angesichts der Reduzierung der Aufwände allgemein akzeptiert werden kann.

In [13] wird eine verbesserte Version 2 des MobileNets, MobileNet-V2, vorgestellt. Dabei werden zusätzlich zu den bereits beschriebenen Modulen neue Module eingesetzt, die zum einen den Speicherbedarf reduzieren und zum anderen die Performanz steigern sollen. Dies geschieht durch den Einsatz von *Inverted Residuals* mit *Linear Bottelenecks*. Die *Linear Bottlenecks* sorgen dafür, dass keine großen Tensoren zwischen den Schichten über *Shortcut* Verbindungen übergeben werden, sondern sorgen mithilfe einer linearen Transformation für ein *Bottleneck*. Durch die lineare Transformation soll sichergestellt werden, dass relevante Informationen nicht durch eine nicht-lineare Dimensionsreduktion verloren gehen. Die *Inverted Residual* Blöcke nutzen klassische *Shortcut* Verbindungen und deren Vorteile des verbesserten Gradienten-Flusses [9]. Das invertierende ist die Stelle, an denen diese Verbindungen eingesetzt werden. Typischerweise sind bei *Shortcut*-Verbindungen zwei hochdimensionale Blöcke verbunden, zwischen denen ein *Bottleneck* liegt. Bei den *Inverted Residual* Blöcken sind zwei *Bottlenecks* verbunden, zwischen denen ein höherdimensionaler Block liegt. Dadurch soll der Speicherbedarf weiter reduziert werden, da niederdimensionale Blöcke (die *Bottlenecks*) über die *Shortcut*-Verbindung weitergeleitet werden anstatt hochdimensionaler Tensoren. In Abbildung 6 ist die Architektur des MobileNet-V2 vereinfacht graphisch dargestellt. An dieser Abbildung lässt sich gut der Aufbau der beschriebenen *Inverted Residual* mit *Linear Bottlenecks* erkennen.

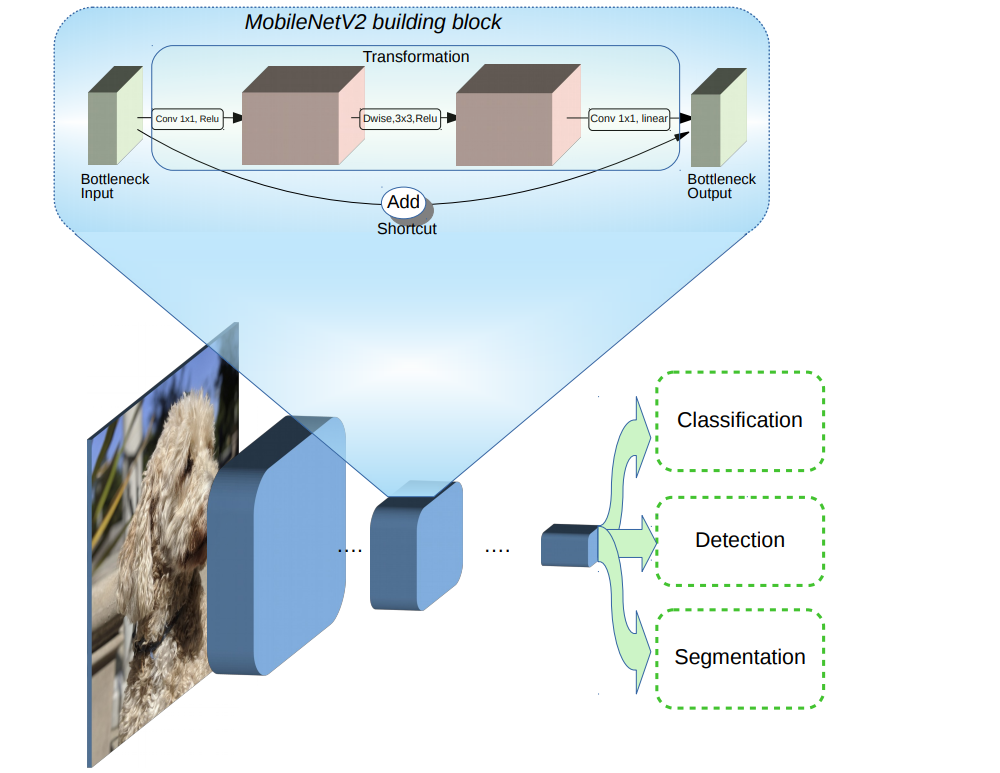


Abbildung 6: Schematische Übersicht über MobileNet-V2 Architekture [14]

MobileNet-V2 erreicht einen Top-1 Klassifikationsfehler von 28% und einen Top-5 Klassifikationsfehler von 9%. Das Modell besteht aus 3.5x106 Parametern (entspricht 14 MB Speicher) und führt 0.3x109 FLOPs pro Eingangsbild aus [13], [14].

## Modul B

Wie in den Grundlagen bereits beschrieben wird Modul B des L DNN Algorithmus mithilfe eines inkrementellen Klassifikators realisiert. Zu den grundlegenden Anforderungen eines inkrementellen Klassifikators, die ebenfalls in den Grundlagen beschrieben wurden, kommen in dieser Arbeit weitere spezifische Anforderungen. Es ist in dieser Arbeit relevant, dass der inkrementelle Klassifikator ohne Speicherung vorheriger Eingangsdaten arbeitet. Dafür gibt es zwei konkrete Gründe.

Zum einen sind Anwendungen interessant, bei denen die Algorithmen auf mobilen Endgeräten laufen. Diese haben nur begrenzten Speicher zur Verfügung und können nicht eine Vielzahl von Daten sichern. Zudem fallen bei Echtzeit-Anwendungen enorm viele Daten an, die nicht ohne einen großen Daten-Server gespeichert werden können. Ein Transfer dieser Daten auf einen Server würde jedoch einen erheblichen Kommunikationsaufwand darstellen.

Als zweiten Punkt kann der Einsatz in sicherheitsrelevanten Anwendungen genannt werden. Es kann dabei zwischen sicherheitskritischen Daten und persönlichen Daten unterschieden werden. Beide müssen bei einer Sicherung mit hohem Aufwand geschützt werden. Dies kann auf mobilen Endgeräten nicht immer sichergestellt werden. Bei einer Kommunikation zu einem Server muss wiederum eine sichere Verbindung gewährleistet werden.

Da diese Punkte hohe Aufwände verursachen, soll der inkrementelle Klassifikator nicht mit abgespeicherten Beispieldaten arbeiten, wodurch die genannten Probleme umgangen werden können.

Die beschriebenen Klassifikatoren, Incremental Classifier and Representation Learning (iCaRL) und Adaptive Resonance Theory (ART), erfüllen beide die in [15] gestellten Anforderungen an einen inkrementellen Klassifikator:

1. Er soll auf Basis eines Daten-Stream, in dem Sample der unterschiedlichen Klassen zu unterschiedlichen Zeitpunkten (zufällig) auftreten, trainierbar sein
2. Zu jedem Zeitpunkt muss ein funktionierender Multi-Klassen Klassifikator für die bereits gesehenen und damit bekannten Klassen verfügbar sein
3. Die Berechnungsanforderungen und der Speicherbedarf sollen beschränkt sein oder nur langsam ansteigen mit Bezug auf die Anzahl an bekannten Klassen

Jedoch benötigt iCaRL für die Berechnungen der Repräsentationen eine ausgewählte Menge an Exemplaren der jeweiligen Klassen. Diese Anzahl an gespeicherten Daten pro Klasse kann variieren, jedoch wird eine Anzahl von 10 bis 20 Exemplaren pro Klasse empfohlen [16]. Dies widerspricht den oben eingeführten Kriterien, keine spezifischen Daten sichern zu müssen. Zudem müssen für einen Austausch von Wissen zwischen einzelnen, verteilten Netzen diese gespeicherten Exemplare ebenfalls ausgetauscht werden. Aus diesem Grund wird in dieser Arbeit iCaRL nicht für die Umsetzung von Modul B genutzt.

ART-Netzwerke sichern ebenfalls Repräsentationen von Klassen, jedoch keine einzelnen Eingangsdaten von Klassen. Es wird eine Repräsentation pro Klasse angelegt, und inkrementell generalisiert, wenn neue Daten für diese Klasse eintreffen. Durch die Generalisierung der Daten können einzelne Eingangsdaten nicht rückverfolgt werden, weshalb die gestellten Anforderungen für Modul B erfüllt werden können.

Durch den Einsatz des in [1] vorgestellten „*Nothing I Know*“-Konzepts können neue, bisher unbekannte Klassen erkannt werden und ein neuer Knoten für diese Klasse erstellt werden. Bei diesem Punkt muss erwähnt werden, dass vor Beginn der Anwendung eine Anzahl an maximal möglichen Klassen festgelegt werden muss, da die Anzahl an verfügbaren Knoten (Neuronen) in einem ART-Netzwerk initial festgelegt wird. Somit können nicht beliebig viele Klassen inkrementell erlernt werden, aber es kann durch eine vorherige Abschätzung eine ausreichende Anzahl vorgegeben werden.

Es gibt eine Vielzahl an unterschiedlichen ART-Netzwerken, welche jeweils für verschiedene Anwendungen genutzt werden. In dieser Arbeit wird ein Fuzzy ARTMAP (FAM)-Netzwerk genutzt. Der Hauptvorteil besteht darin, dass durch den Einsatz der Fuzzy-Theorie anstatt binärer Vektoren (wie in einem klassischen ART1-Netzwerk) kontinuierliche Werte im Bereich zwischen 0 und 1 genutzt werden können. Zudem werden die Fuzzy-Operatoren eingesetzt. In einem FAM-Netzwerk ersetzt dabei der Fuzzy-AND Operator den binäre AND Operator. Der Fuzzy-AND Operator ist dabei wie folgend definiert:  [17].

Zusätzlich arbeitet ein typisches FAM-Netzwerk mit komplementärer Codierung (*complement coding*), wodurch ein ursprüngliches Eingangssignal umgewandelt wird zu , mit dem komplementären Eingangssignal , das an das originale Eingagssignal angehängt wird [17].

Nach [1] sind diese eben genannten Besonderheiten der Fuzzy-Operatoren und des *complement Coding* auch Probleme bei der Nutzung eines solchen Netzwerkes innerhalb des L DNN Algorithmus. Da DNNs als Feature Extrahierer zu sparsen Feature Repräsentationen tendieren, führt der Einsatz von komplementärer Codierung zu sehr hohen Aktivierungen im komplementären Teil. Kleine Unterschiede zwischen einzelnen Mustern lassen sich dann schwierig unterscheiden. Deswegen wird in [1] vorgeschlagen, den *complement Coding* Teil des FAM-Netzwerks nicht zu nutzen, was in dieser Arbeit übernommen wird. Als zusätzliche Maßnahme wird der Austausch der Fuzzy AND Logik durch das Skalarprodukt von zwei Vektoren vorgeschlagen. Dadurch kann erzielt werden, dass das Ergebnis der Vergleichsoperation normalisiert ist.

Aufbauend auf diesen Änderungen wird in dieser Arbeit eine weitere Adaption des FAM-Netzwerkes genutzt, das Simplified FAM (SFAM)-Netzwerk [18]. Das SFAM benötigt zur Initialisierung im Gegensatz zu den anderen ART-Netzwerk lediglich die Toleranzschwelle (*Vigilance Parameter*) . Wenn während des Trainings festgestellt wird, dass dieser Parameter zu niedrig ist und es deshalb zu Fehlklassifizierungen kommt, wird dieser Parameter automatisch nach oben angepasst. Dieser Fall wird *Category Mismatch* genannt und stellt die Besonderheit des SFAM-Netzwerkes dar [17]. Mit dieser Konzeption des Modul B können die gestellten Anforderungen in der Theorie bewältigt werden. Eine graphische Darstellung und eine kurze Zusammenfassung werden in Kapitel 2.2 gegeben.

# Zusammenfassung

In diesem Kapitel werden eine Zusammenfassung und finale Bewertung der unterschiedlichen Ansätze für die jeweiligen Module gegeben. Es wird auf Basis der eingesetzten Metriken und genannten Kriterien jeweils ein Ansatz pro Modul ausgewählt. Abschließend wird dann der gesamte Lifelong DNN Algorithmus mit den ausgewählten Modulen dargestellt.

## Modul A

Modul A hat wie in Kapitel 1.1 beschrieben unterschiedliche Kriterien, die ausschlaggebend für die Auswahl sind. Es gibt eine Vielzahl an möglichen Netzwerken, welche die Aufgaben von Modul A übernehmen können. Die konkrete Auswahl ist stark vom gewünschten Anwendungsfall und der gegebenen Hardwareumgebung abhängig. In Tabelle 5 sind alle vorgestellten DNN-Architekturen nochmals direkt miteinander verglichen.

Tabelle 5: Übersicht aller vorgestellten DNNs für Modul A

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Netzwerk-Architektur | Anzahl Parameter | Speicher-bedarf  Parameter | Anzahl  FLOPs | Top-1 Klassifikationsfehler | Top-5 Klassifikationsfehler |
| AlexNet | 60x106 | 240 MB | 0.7x109 | 36.7 % | 15.3 % |
| VGG-16 | 138x106 | 552 MB | 16x109 | 25.6 % | 8.1 % |
| VGG-19 | 144x106 | 576 MB | 20x109 | 25.5 % | 8.0 % |
| ResNet-50 | 25.6x106 | 102 MB | 4x109 | 20.7 % | 5.3 % |
| ResNet-101 | 44.5x106 | 178 MB | 8x109 | 19.9 % | 4.6 % |
| Inception-V3 | 24x106 | 96 MB | 4.8x109 | 21.6 % | 5.6 % |
| MobileNet-V2 | 3.5x106 | 14 MB | 0.3x109 | 28 % | 9 % |

Im Rahmen dieser Arbeit gibt es keine limitierenden Hardware-Begrenzungen. Dennoch soll berücksichtigt werden, dass der implementierte Lifelong DNN Algorithmus auch auf mobilen Endgeräten (z.B. Smartphone oder Mikrocontroller) lauffähig sein soll.

Zunächst wird deshalb der berechnete Speicherbedarf der Netzwerke verglichen. Hier gibt es eine große Schwankung zwischen 576 MB für VGG-19 und 14 MB für MobileNet-V2. Trotzdem kann gesagt werden, dass auch der Speicherbedarf eines VGG-19 Netzwerkes akzeptabel wäre, da Endgeräte heutzutage häufig die Möglichkeit besitzen ein zusätzliches Speichermodul (z.B. SD-Karte) einzusetzen, wodurch zusätzlicher Speicher gewonnen werden kann. Es muss jedoch auch in Relation gesetzt werden, dass z.B. auf Smartphones APPs nur selten größer als 300 MB sind, weshalb VGG-16 und VGG-19 in einigen mobilen Anwendungen nicht die optimale Wahl wäre.

Weiter wird die benötigte Anzahl an Operationen für die Bearbeitung eines einzelnen Eingangsbild verglichen. Auch hier gibt es große Unterschiede, von 20 Giga-FLOPs (GFLOPs) in einem VGG-19 bis zu 0,3 GFLOPs in MobileNet-V2. Dieser Punkt ist für mobile Anwendungen kritischer zu sehen als der Speicherbedarf. Der Grund liegt darin, dass bei vielen mobilen Anwendungen Echtzeitfähigkeit gefordert ist. Um dies zu gewährleisten, müssen die benötigten Operationen für die Bearbeitung von Eingangsdaten von dem vorhandenen Prozessor in einem gewissen Zeitraum (z.B. innerhalb einer Sekunde) abgearbeitet werden. Abhängig vom Anwendungsfall können dabei unterschiedliche Raten erforderlich sein. Im Fall von Bildverarbeitung wird in diesem Kontext von **F**rames **P**er **S**econd (FPS) gesprochen. Für einen konkreten Vergleich wird als Hardware ein in der Praxis häufig genutzter Mikrocontroller, der Raspberry Pi 3 Modul B [19], angenommen. In dem Kontext von FLOPs ist die Taktfrequenz der Hardware relevant, welche für diesen Mikrocontroller mit 1,2 GHz angegeben ist [19]. Unter der Annahme, dass nur die in der Tabelle aufgeführten FLOPs zur Bearbeitung des Eingangsbildes ausgeführt werden müssen, kann eine theoretische FPS-Rate berechnet werden. Mit einem VGG-19 würde ein Raspberry Pi 3 Modul B 0,06 FPS erreichen können. Die Bearbeitung eines Bildes würde somit 16,6 Sekunden brauchen. Mit dem ResNet-50 könnte eine FPS-Rate von 0,3 erreicht werden, und mit MobileNet-V2 nach diesen Überlegungen 4 FPS. Mit einem ResNet-50 würde die Bearbeitung eines Bildes ca. 3,3 Sekunden brauchen. Wenn dies noch als akzeptabel angenommen wird für die Anwendung, wäre auf Basis von diesem Kriterium lediglich AlexNet, ResNet-50 und MobileNet-V2 relevant für die spätere Nutzung.

Als letzten Punkt wird die Genauigkeit der Netzwerke verglichen. Die genannten Netzwerke werden im späteren Verlauf nicht als Klassifikatoren eingesetzt, aber die Klassifikationsgenauigkeit gibt eine gute Übersicht über die Fähigkeit, relevante Features zu extrahieren. Auch hier muss abhängig von der realen Anwendung entschieden werden. Für sicherheitskritische Anwendungen spielt die Genauigkeit eine große Rolle, während für sicherheitsunkritische Anwendungen die Genauigkeit eventuell geringer sein kann und dafür mehr Wert auf eine geringe Laufzeit gelegt wird. Im Rahmen dieser Arbeit wird das generelle Potenzial des Lifelong DNN Algorithmus untersucht und nicht die exakte Performanz von CNN-Architekturen. Deshalb kann auch eine geringere Genauigkeit auf dem ImageNet-Datensatz akzeptiert werden, wenn dafür die anderen, bereits genannten Kriterien die Erwartungen erfüllen.

Unter Berücksichtigung aller genannten Kriterien wird im weiteren Verlauf der Arbeit MobileNet-V2 als Modul A des Lifelong DNN Algorithmus eingesetzt. Dafür spricht die spezielle Architektur für mobile Anwendungen, die eine Echtzeit(nahe)-Bearbeitung von Eingangsdaten auch auf mobilen Endgeräten erlaubt. Dies wird sowohl durch den geringen Speicherbedarf mit lediglich 14 MB, als auch durch die geringe Anzahl von 0,3 GFLOPs, die pro Eingangsdaten ausgeführt werden müssen, gewährleistet. Dadurch kann dieses Netzwerk auch auf leistungsschwächeren Endgeräten laufen. Zudem kann trotz dieser Merkmale ein Top-5 Klassifikationsfehler von 9% auf dem ImageNet-Datensatz erreicht werden, was im Rahmen dieser Arbeit eine ausreichende Genauigkeit darstellt, die auch von vielen komplexeren Netzwerken nicht erreicht wird (z.B. AlexNet) oder nur knapp unterboten wird (z.B. VGG-16 und VGG-19).

Die Architektur des MobileNet-V2 Netzwerks ist in Tabelle 6 aufgeführt.

Tabelle 6: Architektur MobileNet-V2 [13]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | Operator | t | c | n | s |
| 224x224x3 | Conv2d | - | 32 | 1 | 2 |
| 112x112x32 | Bottleneck | 1 | 16 | 1 | 1 |
| 112x112x16 | Bottleneck | 6 | 24 | 2 | 2 |
| 56x56x24 | Bottleneck | 6 | 32 | 3 | 2 |
| 28x28x32 | Bottleneck | 6 | 64 | 4 | 2 |
| 14x14x64 | Bottleneck | 6 | 96 | 3 | 1 |
| 14x14x96 | Bottleneck | 6 | 160 | 3 | 2 |
| 7x7x160 | Bottleneck | 6 | 320 | 1 | 1 |
| 7x7x320 | Conv2d 1x1 | - | 1280 | 1 | 1 |
| 7x7x1280 | Avgpool 7x7 | - | - | 1 | - |

Dabei stellt jede Zeile eine Sequenz von einer oder mehreren identischen Schichten dar, die *n*-Mal wiederholt wird. Alle Schichten innerhalb einer Sequenz (Zeile) haben dieselbe Anzahl *c* an Ausgangskanälen. Die erste Schicht einer Sequenz hat dabei die *Stride s* und alle weiteren Schichten dieser Sequenz nutzen *Stride* 1. Alle *spatial Convolutions* nutzen 3x3 Kernel. Der Expansionsfaktor *t* wird auf die Eingangsgröße angewendet, um das Prinzip der linearen Bottlenecks zu nutzen.

Das Netzwerk besteht aus einer initialen *Convolutional* Schicht und darauffolgend 17 Bottleneck Schichten. Zum Schluss folgen eine 1x1 *Convolution* Schicht und eine Pooling-Schicht. Wenn das Netzwerk zur Klassifikation eingesetzt würde, würde noch eine zusätzliche Schicht folgen mit Softmax-Aktivierung. Da hier jedoch die Features nur genutzt werden, wird diese Schicht nicht genutzt. Damit ist der Ausgang von Modul A (und damit der Eingang von Modul B) ein Tensor mit den Dimensionen 1x1x1280, der wiederum direkt in einen Vektor der Größe 1280x1 übertragen werden kann.

## Modul B

Modul B besteht wie in Kapitel 1.2 beschrieben aus einem Simplified Fuzzy ARTMAP (SFAM) Netzwerk. Dieses Netzwerk kann initial mit bereits bekannten Klassen vortrainiert sein oder initial „leer“ sein (keine Klassen bekannt). Ausgehend davon werden neue Eingangsdaten klassifiziert. Falls die Eingangsdaten zu keiner bekannten Klasse passen, fallen diese Eingangsdaten in die Kategorie „Nothing I know“, und eine neue Repräsentation (neuer Knoten) wird im Netzwerk angelegt. Für diesen Schritt ist das Feedback des Benutzers notwendig, um das korrekte Klassenlabel zu erhalten. Die Architektur des genutzten SFAM-Netzwerks ist in Abbildung 7 dargestellt. Die einzelnen Bestandteile und deren Nutzen werden im Folgenden erläutert.

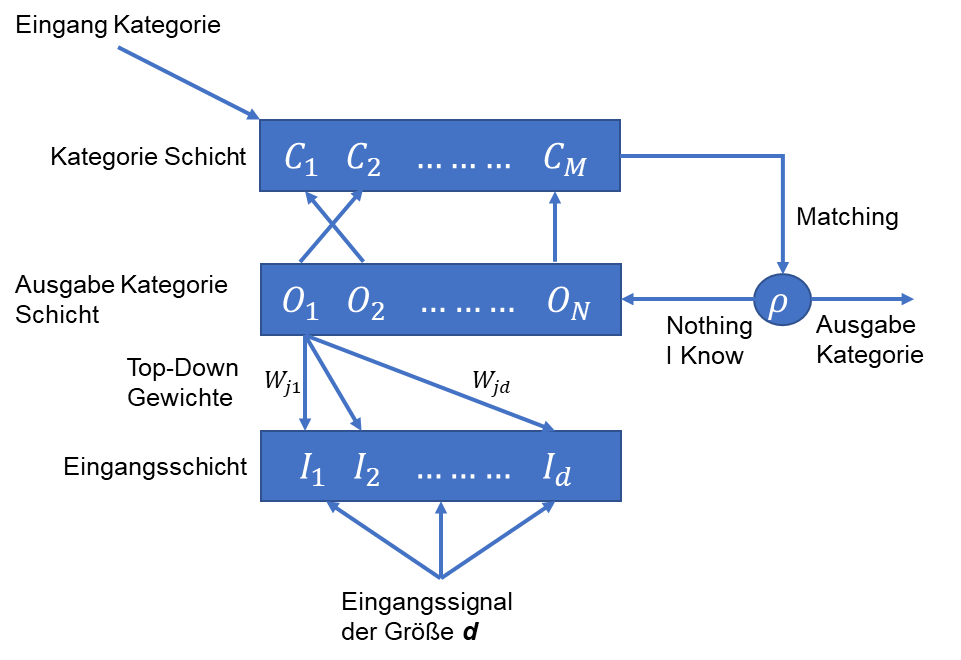


Abbildung 7: Architektur des SFAM-Netzwerks

Das Eingangssignal der Dimension (Größe) kommt an der Eingangsschicht an. Daraufhin werden die erlernten Gewichte der Kategorieknoten mit diesem Eingangssignal assoziiert. Diese Gewichte werden Top-Down Gewichte genannt, da sie von oben nach unten (Top-Down) mit den Eingangsdaten assoziiert werden. In dieser Arbeit wird wie in [1] vorgeschlagen das Skalarprodukt der Gewichte mit den Eingangsdaten für die Assoziation genutzt. Auf Basis dieser berechneten Assoziationen erfolgen dann die Aktivierungen der Ausgangsneuronen () in der Kategorie Schicht. Abhängig von den erlernten Gewichten können auch mehrere Kategorie-Knoten auf dieselben Ausgangsneuronen zeigen, wenn die einzelnen Kategorie-Knoten dieselbe Klasse repräsentieren. In der Kategorie Schicht wird während des Trainings (der Adaptionsphase) die korrekte Klasse hinzugefügt, um das richtige Label für ein korrektes adaptieren der Neuronen im Netzwerk zu haben.

Die berechneten Aktivierungen (*Matching*-Werte) werden dann mit dem *Vigilance* Parameter verglichen. Wenn dieser Schwellwert überschritten ist, wird die Gewinner-Kategorie/Klasse ausgegeben und somit die Eingangsdaten dieser Klasse zugeordnet. Bei einer Fehlklassifikation (Netzwerkausgabe entspricht nicht bekanntem Label) wird in einem SFAM-Netzwerk der Schwellwert automatisch nach oben angepasst, damit diese Fehlklassifikation in Zukunft nicht mehr auftreten kann. Dadurch werden die Kategorien feiner und es entstehen potenziell mehrere Kategorie-Knoten für dieselbe Klasse.

Wenn das Matching der Gewinner-Kategorie den Schwellwert unterschreitet, findet das *Nothing* *I know*-Konzept Anwendung. Da das höchste Matching nicht ausreichend ist, erkennt das Netzwerk, dass es dieses Eingangsmuster nicht kennt. In diesem Fall wird dann ein neuer Kategorie-Knoten für das gegebene Eingangsmuster angelegt. Der Benutzer muss in diesem Fall die korrekte Kategorie angeben, damit für spätere Eingangsdaten die passende Kategorie-Bezeichnung ausgegeben werden kann.

## Lifelong DNN Algorithmus

Aus den beiden beschriebenen Modulen A und B lässt sich nun der gesamte Lifelong DNN Algorithmus bauen. Er besteht aus einem vortrainierten Feature-Extrahierer, welcher durch das MobileNet-V2 realisiert wird. Dieses Netzwerk wird auf Basis des ImageNet-Datensatzes als Klassifikator vortrainiert. Die letzte Schicht wird dann entfernt, und die extrahierten Features an Modul B, den inkrementellen Klassifikator weitergeleitet. Dieses Modul wird durch ein SFAM-Netzwerk realisiert. Als Ausgabe des Modul B erhält der Benutzer das prädizierte Klassenlabel des Eingangsbildes. Die gesamte Architektur sowie Beispieldimensionen für ein 224x224x3 Eingangsbild sind in Abbildung 8 skizziert.

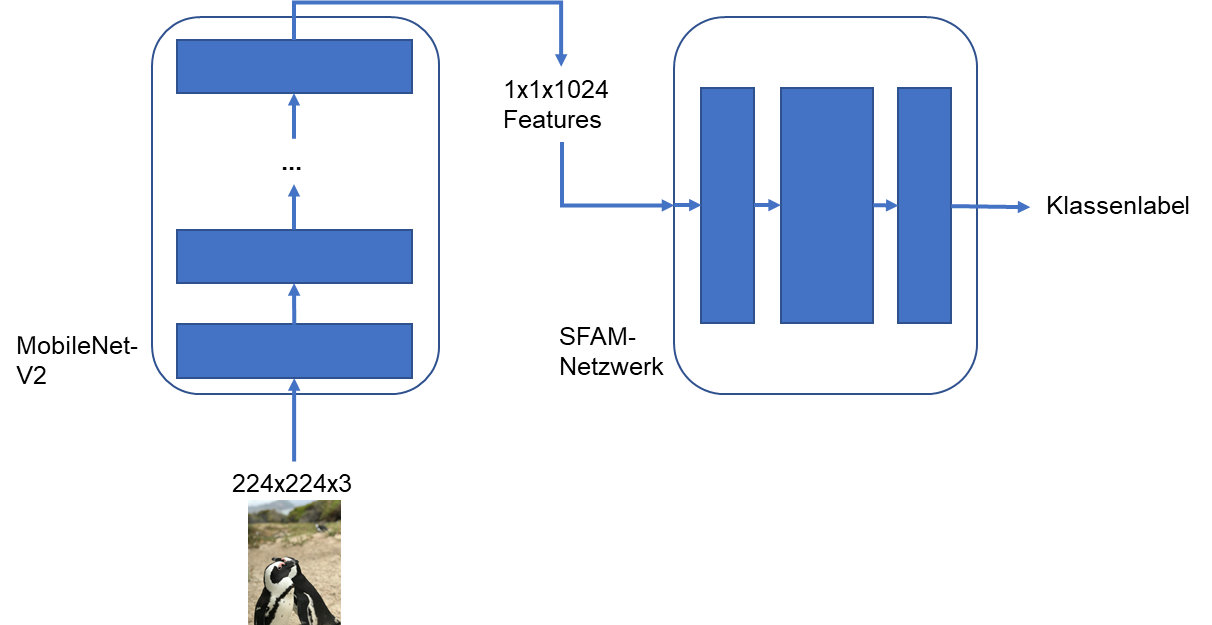


Abbildung 8: Gesamtarchitektur L DNN Algorithmus

Im weiteren Verlauf der Arbeit werden nun Experimente definiert, um die Performanz des Lifelong DNN Algorithmus sowie dessen Potenzials zu überprüfen. Daraufhin wird die dargestellte und erläuterte Konzeption des Lifelong DNN Algorithmus prototypisch implementiert und die definierten Testfälle werden durchgeführt.

# Literaturverzeichnis

|  |  |
| --- | --- |
| [1] | M. Luciw, S. Olivera, A. Gorshechnikov, J. Wurbs, H. M. Ames und M. Versace, „Systems and Methods to enable Continual, Memory-Bounded learning in Artificial Intelligence and Deep Learning Continuously operating Applications across networked Compute Edges“. United States of America Patent US 2018/0330238 A1, 15 November 2018. |
| [2] | Neurala Inc., „Lifelong Deep Neural Networks - Tech Summary,“ [Online]. Available: https://info.neurala.com/hubfs/docs/ Neurala\_LifelongDNNWhitepaper.pdf. [Zugriff am 7 Mai 2019]. |
| [3] | J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li und L. Fei-Fei, „ImageNet: A Large-Scale Hierarchical Image Database,“ *2009 IEEE Conference on Computer Vision and Pattern Recognition,* pp. 248-255, 2009. |
| [4] | M. A. E. Muhammed, A. A. Ahmed und T. A. Khalid, „Benchmark Analysis of Popular ImageNet Classification Deep CNN Architectures,“ *Proceedings of the 2017 International Conference On Smart Technology for Smart Nation, SmartTechCon 2017,* pp. 902-907, 2018. |
| [5] | A. Krizhevsky, I. Sutskever und H. G. E., „ImageNet Classification with Deep Convolutional Neural Networks,“ *Advances in Neural Information Processing Systems 25,* pp. 1097-1105, 2012. |
| [6] | K. Sinhal, „CV-Tricks,“ 2017. [Online]. Available: https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/. [Zugriff am 5 Juni 2019]. |
| [7] | Google Brain Team, „TensorFlow Guide,“ Google Brain, [Online]. Available: https://www.tensorflow.org/guide/tensors. [Zugriff am 29 Mai 2019]. |
| [8] | K. Simonyan und A. Zisserman, „Very Deep Convolutional Networks For Large-Scale Image Recognition,“ in *International Conference on Learning Representations 2015*, San Diego, 2015. |
| [9] | K. He, X. Zhang, S. Ren und J. Sun, „Deep residual learning for image recognition,“ *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* pp. 770-778, Dezember 2015. |
| [10] | C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke und A. Rabinovich, „Going deeper with convolutions,“ *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* pp. 1-9, Juni 2015. |
| [11] | C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens und Z. Wojna, „Rethinking the Inception Architecture for Computer Vision,“ *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* pp. 2818-2826, Dezember 2016. |
| [12] | A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto und H. Adam, „MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,“ *CoRR,* 2017. |
| [13] | M. Sandler, A. Howard, M. Zhu, A. Zhmoginov und L. C. Chen, „MobileNetV2: Inverted Residuals and Linear Bottlenecks,“ *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* pp. 4510-4520, 2018. |
| [14] | M. Sandler und A. G. Howard, „Google AI Blog,“ Google AI, 3 April 2018. [Online]. Available: https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html. [Zugriff am 5 Juni 2019]. |
| [15] | S. A. Rebuffi, A. Kolesnikov, G. Sperl und C. H. Lampert, „iCaRL: Incremental classifier and representation learning,“ *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017,* pp. 5533-5542, Januar 2017. |
| [16] | C. Lampert, „Institue of Science and Technology Austria,“ 25 August 2018. [Online]. Available: https://pub.ist.ac.at/~chl/talks/lampert-vsssw2018.pdf. [Zugriff am 07 Juni 2019]. |
| [17] | A. A.-m. Merten, „Adaptive Resonance Theory [ART] - Ein neuer Ansatz lernender Computer -,“ Universität Ulm, Ulm. |
| [18] | T. Kasuba, „Simplified Fuzzy Artmap,“ *AI Expert,* pp. 18-25, 1993. |
| [19] | Generation Robots, „Generation Robots,“ [Online]. Available: https://www.generationrobots.com/media/raspi3-datasheet.pdf. [Zugriff am 06 Juni 2019]. |