|  |  |  |
| --- | --- | --- |
| **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 |
|  |  |  |  |  |  |

*Bei der Erstellung eines Dokuments ist folgendes zu beachten:*

* *Wurden alle Anforderungen in der Konzeption umgesetzt?*
* *Ist die Beschreibung der Konzeption detailliert genug?*
* *Entspricht das Ergebnis den Erwartungen des Auftraggebers?*
* *Ist auf Basis der erstellten Konzeption die Realisierung eines Prototyps möglich?*

# 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 10

2 Zusammenfassung 11

2.1 Modul A 11

2.2 Modul B 11

3 Literaturverzeichnis 12

# Beschreibung der Konzeption

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

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

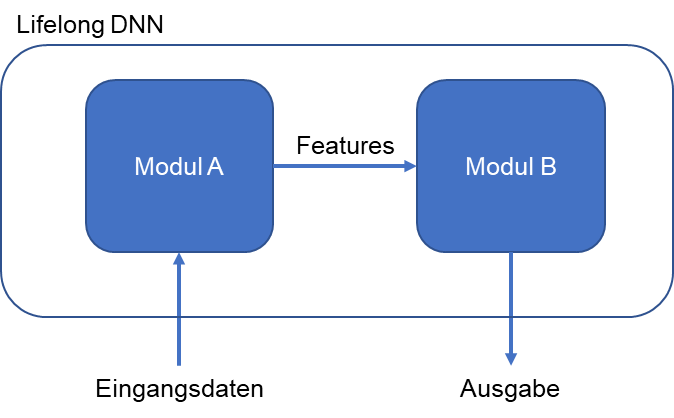


Abbildung 1: Graphische Darstellung des L DNN A

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

## 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 das Modul A innerhalb des L DNN Algorithmus ist wichtig, dass die extrahierten Features eine bestmögliche Klassifikation erlauben. Zum anderen soll das System auch fähig sein in Echtzeitsystemen auf mobilen Endgeräten (z.B. Smartphone) zu funktionieren [1], [2]. Dafür soll das 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 vorhanden ist.

Im Folgenden werden bekannte Klassifikator-Netzwerke auf Basis von DNN Architekturen vorgestellt und diese verglichen auf deren Einsetzbarkeit in dieser Arbeit. Als Grundlage für den späteren Vergleich und genannte 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 vortrainiert, da er aufgrund seiner großen Anzahl an Trainingsbildern und Klassen eine gute Generalisierung der extrahierten Features des Netzwerks erlaubt.

### AlexNet

AlexNet war eine der ersten DNNs welche die Klassifikationsgenauigkeit auf dem ImageNet Datensatz signifikant erhöhte. Im Rahmen der ILSVRC (*ImageNet Large Scale Visual Recognition Competition*) erreichte AlexNet einen Top-5 Klassifikationsfehler von 15,3%. Der zweitplatzierte dieses Wettkampfes erreichte lediglich einen Top-5 Klassifikationsfehler von 26%. Dadurch konnte eine erhebliche Steigerung der Klassifikationsperformanz erzielt werden. Dabei ist AlexNet im Vergleich zu aktuellen Architekturen sehr simpel, mit lediglich fünf Convolutional Schichten, verschiedenen Max-Pooling Schichten sowie drei Fully Connected Schichten [4], [5]. Neuerungen im Vergleich zu damaligen DNNs war der Einsatz von ReLU-Aktivierungsfunktionen sowie die Regularisierung mithilfe von Dropout. 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 Schichten zu sehen sind.

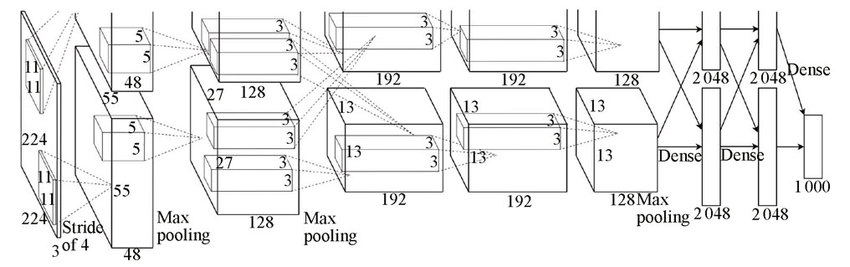


Abbildung 2: Modelarchitektur des AlexNet [5]

Die *Default*-Größe eines Parameters (Floating Point oder Integer) in TensorFlow ist mit 32-bit angegeben [6]. 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 in einem Netzwerk für ein einzelnes Element (z.B. einzelnes Bild) genutzt. Dabei wird lediglich der Vorwärtspfad in Betracht gezogen. Diese Anzahl ist bekannt als Nummer von FLOPs (**F**loating **P**oint **OP**erations). AlexNet benötigt 727x106 (727 Mega 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 unterschiedliche bekannte Architekturen, die am häufigsten genutzten dabei sind das VGG-16 sowie das VGG-19 Netzwerk. Die Zahlen stehen dabei für die Anzahl an Gewichtsschichten innerhalb des DNN. Von der grundsätzlichen Architektur ähnelt es deutlich dem bereits beschriebenen AlexNet. Der große Unterschied ist der Einsatz kleinerer Kernel-Filter in den *Convolutional* Operationen (3x3 anstatt z.B. 11x11 oder 5x5 in AlexNet). Zum einen werden mehrere dieser *Convolutional* Schichten hintereinander gesetzt, wodurch finale Funktion diskriminativer werden soll. Ein weiterer 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. Dadurch können tiefere Netzwerke aufgebaut werden mit mehr Schichten ohne einen deutlich erhöhten Speicherbedarf zu haben. Durch den Anstieg der Tiefe des Netzwerks ist das Netzwerk fähig komplexere Features zu erlernen [7]. Unterschiedliche Architekturen der VGG-Netzwerke für den ImageNet Bilddatensatz mit 1000 Klassen und RGB Bildern der Größe 224x224 sind in folgender Tabelle 1 dargestellt.

Tabelle 1:VGG-Netzwerk Architekturen [7]

|  |  |  |  |
| --- | --- | --- | --- |
| 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. VGG-16 und VGG-19 als häufig eingesetzte Feature-Extrahierer werden im Folgenden bezüglich ihrem Speicherbedarf und der Performanz untersucht. Dabei sind in Tabelle 2 die unterschiedlichen Kriterien nach [7] für die einzelnen Netzwerke gelistet.

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Netzwerk-Architektur | Anzahl Parameter | Speicherbedarf  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 steigt sinnvollerweise mit der Anzahl an Schichten, und damit auch der Speicherbedarf. 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 stattfinden kann. 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. Dieses Verhalten lässt darauf schließen, dass nicht alle Netzwerke beliebiger Architektur gleich zu lösen sind. Um diese Probleme zu lösen, wurden so genannte *Deep Residual Networks* eingeführt, häufig nur als ResNet (Residual Network) bezeichnet [8]. Diese ResNets nutzen dieselben grundlegende Schichten wie VGG-Netzwerke, mit kleinen Kernel-Filter (3x3) und hauptsächlich *Convolutional* Schichten. Diese ResNets bestehen aus einer Vielzahl von Residual Blocks, wie in Abbildung 3 dargestellt.

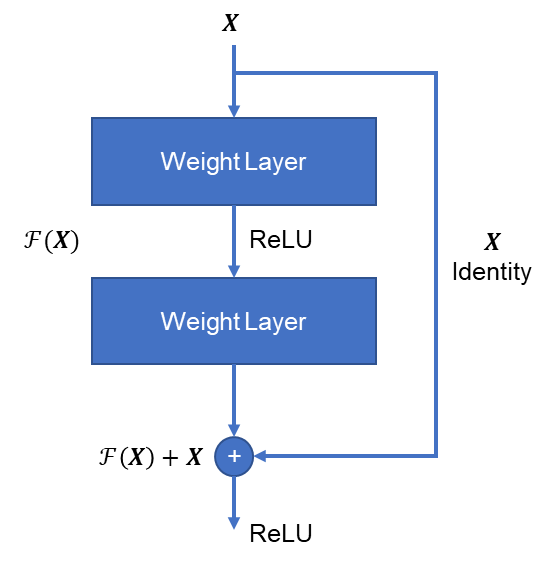


Abbildung 3: Beispielhafter Residual Block [8]

Diese *residual* Blocks verwirklichen das gewünschte Verhalten des Eingangssignals mithilfe der non-linearen Verarbeitung und der Hinzunahme des ursprünglichen Eingangssignals **,** wodurch gilt: . Diese Hinzunahme des Eingangssignals wird mithilfe von sogenannten *Shortcut Connections* realisiert. In den ursprünglichen 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 extrem tiefe Netzwerke (mit mehr als 100 Schichten) effizient und schnell trainiert werden, sowie die Genauigkeit dieser Netzwerke weiter erhöht werden, indem das Potenzial der zusätzlichen tiefen Schichten genutzt wird. Damit kann sowohl das Problem des *Vanishing Gradient*, als auch das *Degradation Problem* gelöst werden [8].

Ein Ausschnitt einer beispielhaften ResNet-Architektur ist in Abbildung 4 gegeben. Es sind exemplarisch nur die ersten Schichten dargestellt. Links ist dabei 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 *residual* Blöcken.

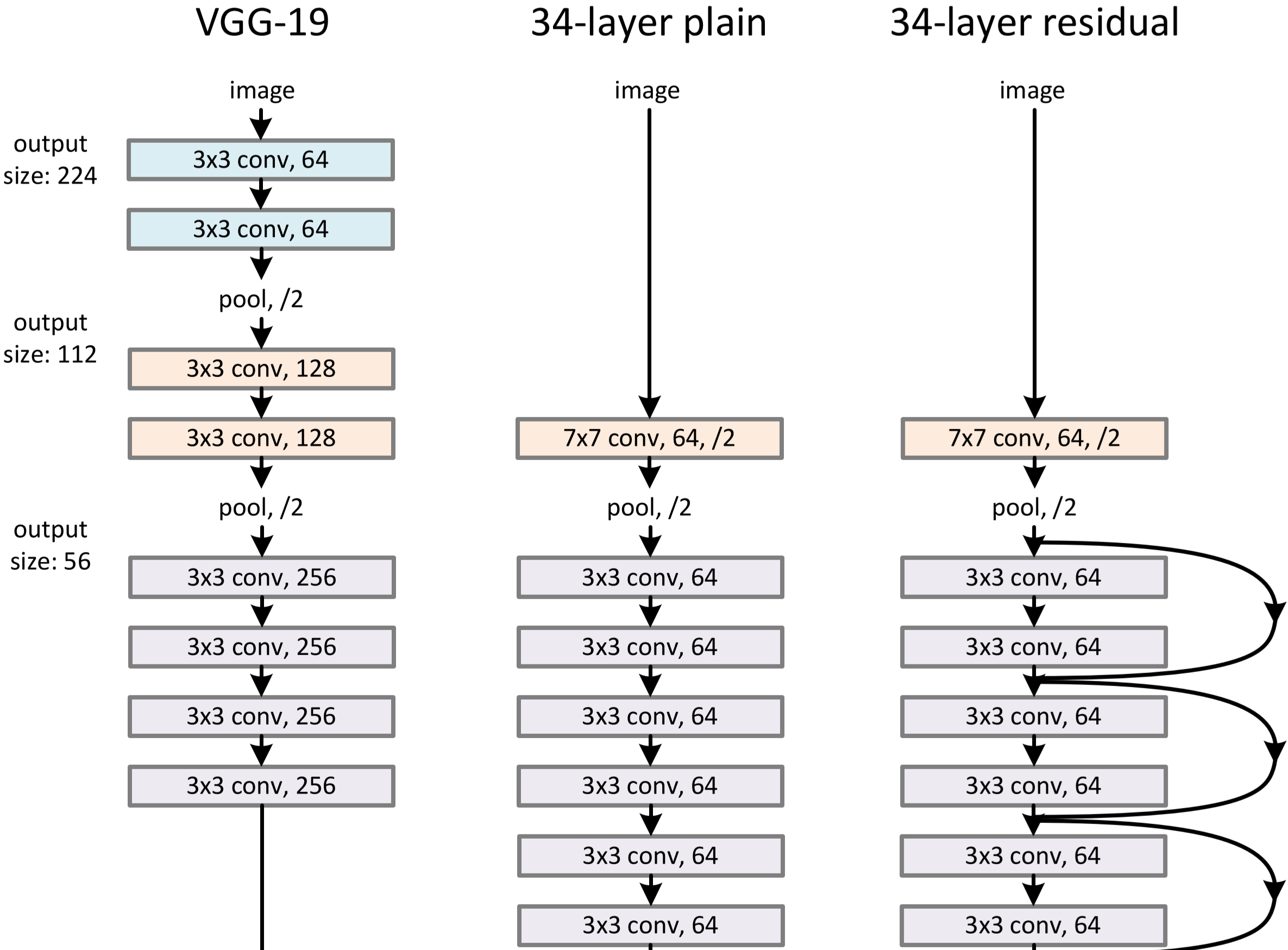


Abbildung 4: Beispielhafte Architektur VGG-19 (links), Standard-Architektur (mitte), ResNet (rechts) [8]

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 [6], sowie der Top-1 und Top-5 Klassifikationsfehler auf dem ImageNet Datensatz nach [8] sind in Tabelle 3 angegeben.

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Netzwerk-Architektur | Anzahl Parameter | Speicherbedarf 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 bereits eingeführten Architekturen (VGG oder ALexNet) kann hier mit weniger Speicherbedarf (geringerer Anzahl an Parametern) sowie deutlich weniger FLOPs mithilfe 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 sehr teurer Hardware (GPUs) aufgrund der hohen Anforderungen für die Berechnung. An diesem Punkt setzt der Aufbau des GoogLeNet an, häufig auch als Inception Netzwerk bekannt [9], [10]. Dabei nutzt GoogLeNet 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 [11]. GoogLeNet nutzt dabei die sogenannten Inception-Module, welche ein sparses CNN mit normalen Konstruktionen approximieren. Der Aufbau eines Inception-Moduls ist in Abbildung 5 dargestellt.

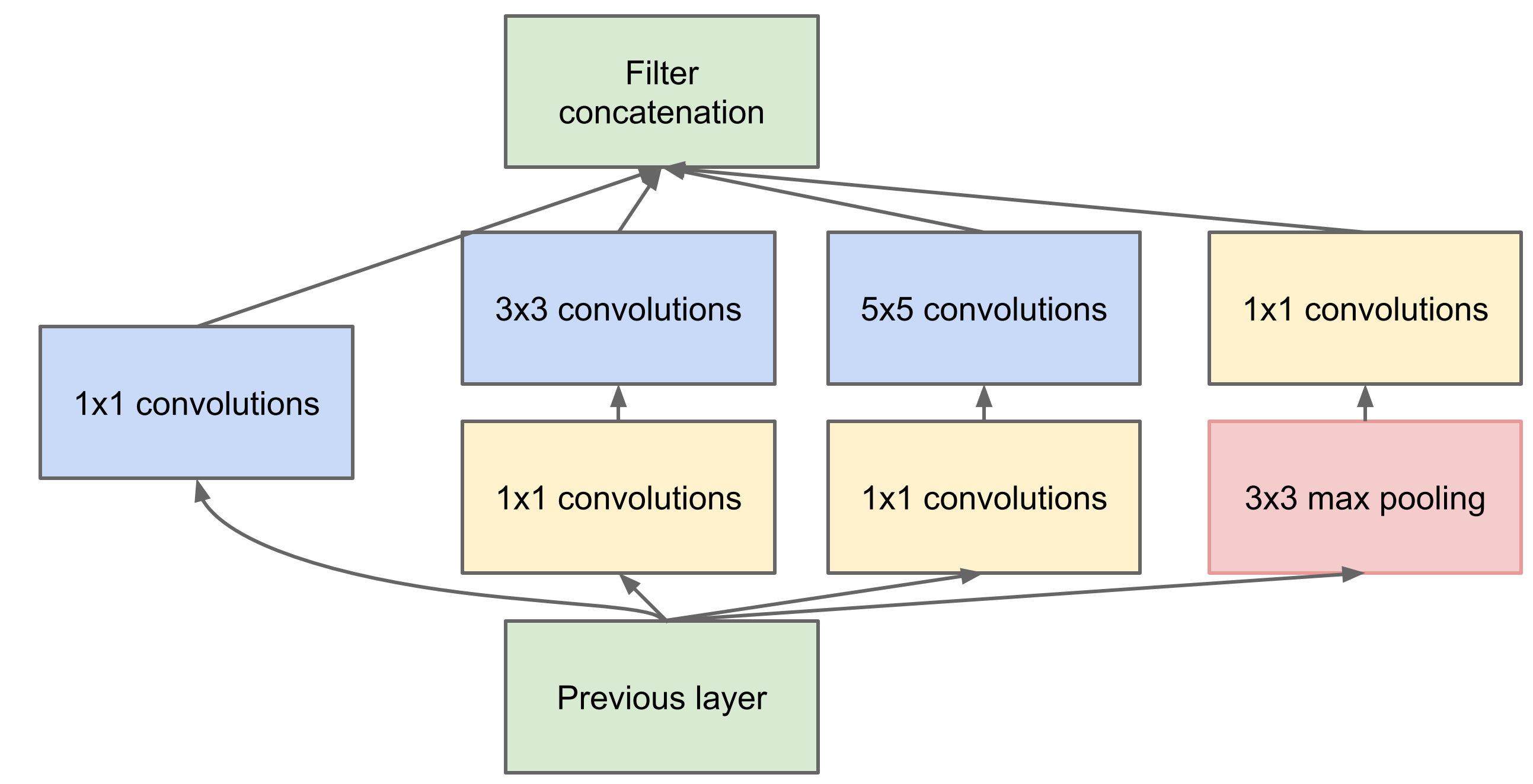


Abbildung 5: Inception-Modul [9]

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. Durch die Inception-Module kann ein sehr tiefes und weites Netz (große Eingangsdimensionen) genutzt 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 [11].

Die dritte Version des Inception-Netzwerks (Inception-v3 [10]) 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 [6] ergibt das einen Speicherbedarf von 96 MB.

### MobileNet

Der Trend im Bereich DNN ist es, immer tiefere und komplexere Netzwerke zu gestalten, die häufig auch verbesserte Genauigkeiten liefern für spezifische Aufgaben (z.B. ILSVRC). Als Folge von diesen Entwicklungen steigt häufig der Berechnungsaufwand und/oder der Speicherbedarf. In vielen realen Anwendungen sind jedoch genau diese Themen relevant, 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) lauffähig sind, und keinen GPU-Server zur Verfügung haben. Aus diesen Anforderungen heraus wurde das sogenannte MobileNet [12] entworfen. Das Ziel der Architektur ist ein kleines Netzwerk mit geringer Latenz das einfach für mobile Anwendungen genutzt werden kann. MobileNet nutzt dabei auch Inception-Module [9] 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 entsteht aus einer Schicht zwei Schichten, bei der die erste für die Filterung und die zweite für die Kombination der berechneten Filterausgänge eingesetzt wird [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.

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

|  |  |  |  |
| --- | --- | --- | --- |
| Modell | Top-1 Klassifikationsfehler ImageNet | 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 Performanzverlust bei ca. 1% liegt, was angesichts der Reduzierung der Aufwände allgemein akzeptiert werden kann.

In [13] wird eine verbesserte Version 2 des MobileNets vorgestellt, MobileNet-v2. Dabei werden zusätzlich zu den bereits beschriebenen Modulen neue Module hinzugefügt, die zum einen den Speicherbedarf weiter reduzieren sollen und zum anderen die Performanz weiter steigern sollen. Dies geschieht hauptsächlich durch den Einsatz von *Inverted Residuals* mit *Linear Bottelenecks*. Die *Linear Bottlenecks* sorgen dafür, dass keine großen Tensoren zwischen den Schichten 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 geht. Die *Inverted Residual* Blöcke nutzen klassische *Shortcut* Verbindungen und deren Vorteile des besser Gradientenflusses [8]. 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 beschrieben *Inverted Residual* mit *Linear Bottlenecks* erkennen.

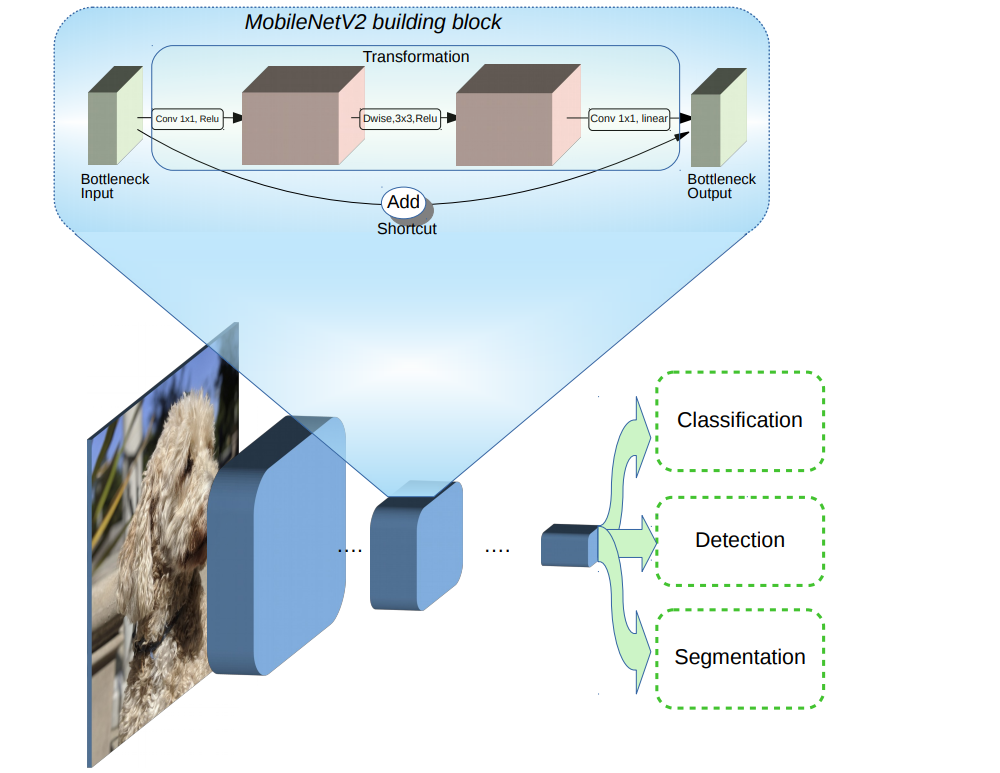


Abbildung 6: Schematische Übersicht über MobileNet-V2 Architekture [14]

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

# Zusammenfassung

*Hier soll das Ergebnis der Konzeption dargestellt werden.*

## Modul A

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Netzwerk-Architektur | Anzahl Parameter | Speicherbedarf  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 % |

## Modul B

# 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] | Google Brain Team, „TensorFlow Guide,“ Google Brain, [Online]. Available: https://www.tensorflow.org/guide/tensors. [Zugriff am 29 Mai 2019]. |
| [7] | K. Simonyan und A. Zisserman, „Very Deep Convolutional Networks For Large-Scale Image Recognition,“ in *International Conference on Learning Representations 2015*, San Diego, 2015. |
| [8] | 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. |
| [9] | 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. |
| [10] | 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. |
| [11] | K. Sinhal, „CV-Tricks,“ 2017. [Online]. Available: https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/. [Zugriff am 5 Juni 2019]. |
| [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]. |