BACHELORARBEIT

Transfer Learning in Bezug auf Edge Computing

ausgeführt am

Institut für
Information Systems Engineering
TU Wien

unter der Anleitung von

Univ. Prof. Mag. Dr. Ivona Brandic

durch

Benedikt Weber
Matrikelnummer: 01627753
Adolfstorgasse 1/8
1130 Wien

Wien, am 17. Juli 2023

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 17. Juli 2023

Benedikt Weber

Inhaltsverzeichnis

1 Einleitung **1**

2 Einführung in die Thematik **2**
2.1 Cloud, Fog und Edge Computing 2
2.1.1 Cloud Computing 2
2.1.2 Fog Computing 6
2.1.3 Edge Computing 6
2.2 Internet of Things (IoT) und 5G 8
2.3 Artificial Intelligence (AI) 10
2.3.1 Machine Learning 10
2.3.2 Deep Learning und CNNs 13
2.3.3 Transfer Learning 18

3 Edge Intelligence: Verwandte Arbeiten **21**
3.1 limitierte Ressourcen und Modellkompression 22
3.2 Datenaustausch und Learning Algorithmen 24

4 Methodik, Daten und Implementierung **28**
4.1 Beschreibung des Datensatzes 28
4.2 Bewertungsmetriken zur Analyse der Effektivität und Effizienz 29
4.3 Strukturelle Performance-Analyse des Transfer Learning Modells 30
4.4 Strukturelle Performance-Analyse des Keras Modells 31
4.5 Quantisierung 33

5 Analyse der Messungen **34**
5.1 Latenz 34
5.2 Energieverbrauch 34
5.3 Genauigkeit 35

6 Fazit **38**

Literaturverzeichnis **39**

1 Einleitung

Das Internet of Things (IoT) öffnet dem Menschen die Türen zu "intelligenten" Geräten und wird die zukünftige Interaktion von Mensch und Maschine grundlegend verändern. Smart Devices können die Effizienz und Produktivität von uns Menschen enorm steigern und einen großen Beitrag zur Energiewende leisten. Die sogenannte Edge Intelligence (EI) bildet eine Synergie aus Machine Learning und Edge Computing Paradigma und beschreibt eine Schlüsselkomponente des IoT. Um die Adaption der EI voranzutreiben, benötigt es Algorithmen für erhöhte Energieeffizienz und Effektivität der Modelle und IoT-Geräte. Transfer Learning verspricht eine schnelle und einfache Integration von Machine Learning in die Edge und eine effiziente und effektive Entwicklung der EI.

In dieser Bachelorarbeit wird ein Grundverständnis für Cloud/Edge Computing (2.1), IoT (2.2) und Machine/Deep Learning (2.3) vermittelt. Es werden die Vorteile einer EI (3) den Herausforderungen und Nachteilen gegenübergestellt und Transfer Learning (2.3.3) als möglicher Schlüsselfaktor der EI beleuchtet. Dabei orientiert sich die Arbeit vornehmlich am Buch "Deep Learning" von I. Goodfellow et al. [GBC16] und an den Forschungsartikeln "A survey on transfer learning" von S. J. Pan und Q. Yang [PY10], "A Survey on the Convergence of Edge Computing and AI for UAVs: Opportunities and Challenges" von P. McEnroe et al. [MWL22] und "OpenEI: An Open Framework for Edge Intelligence" von X. Zhang et al. [ZWL+19].

Um die Frage nach der Effizienz und Effektivität von Transfer Learning in Bezug auf Edge Computing zu beantworten, beinhaltet diese Arbeit einen praktischen Teil, in welchem Transfer Learning und die benutzerdefinierte Modellierung eines Deep Learning Systems in einem möglichst realistischen Szenario einer IoT Anwendung verglichen werden (siehe Kapitel 4). Konkret werden die 2 resultierenden Modelle darauf trainiert, Autokennzeichen in Bildern zu detektieren. Die für das Training verfügbaren Daten werden hierbei auf 1200 Bilder begrenzt.

Der Vergleich beinhaltet die Messung der Genauigkeit, der Geschwindigkeit und des Energieverbrauchs des jeweiligen Machine Learning Algorithmus. Die Messungen werden die Effizienz und Effektivität von Transfer Learning unterstreichen und die Relevanz dieser Machine Learning Methodik für zukünftige EI-Entwicklungen aufzeigen (5). Zudem wird die Effektivität der Post-Training Quantisierung demonstriert.

2 Einführung in die Thematik

Die folgenden Abschnitte der Arbeit befassen sich mit den Synergien und Herausforderungen, welche sich aus den Bereichen der künstlichen Intelligenz, des Internets der 5. Generation und des sogenannten Edge Computing ergeben. Um die Funktionsweise und das Zusammenspiel dieser Technologien zu veranschaulichen, werden die nächsten 3 Abschnitte diese Technologien beleuchten und so ein Fundament für die weiteren Kapitel der Arbeit legen.

2.1 Cloud, Fog und Edge Computing

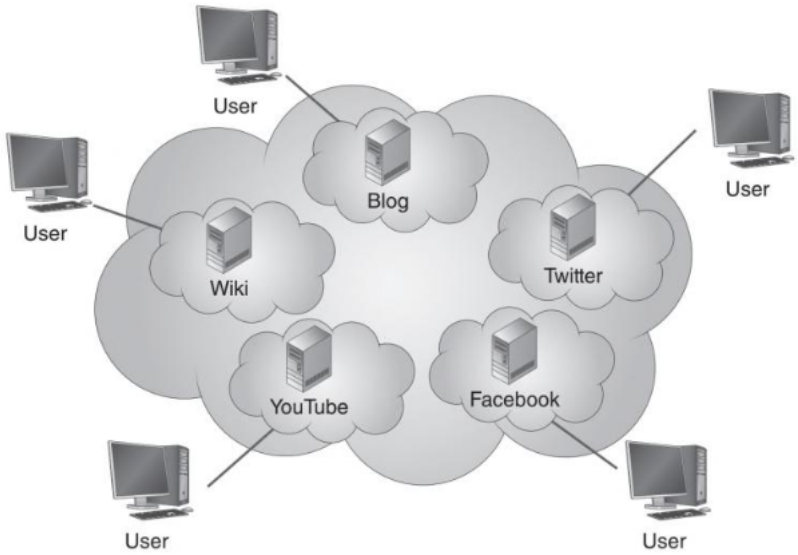
In dieser Sektion betrachten wir die Technologien rund um das Cloud Computing. Es werden die Entwicklung bis hin zum Fog und Edge Computing, die Chancen aber auch die Nachteile und Probleme dieser Technologien beleuchtet.

2.1.1 Cloud Computing

Wie A. Oludele et al. in ihrem Artikel "On the Evolution of Virtualization and Cloud Computing: A Review" erläutern, setzt sich Cloud Computing aus mehreren Technologien zusammen, welche alle ihren Ursprung Ende des 20. Jahrhunderts haben. Das Fundament des Cloud Computings bilden unter anderem die Einführung des Internets, die Herstellung symmetrischer Multiprozessorsysteme und die Implementierung von Virtualisierung. Ein Computer mit mehreren Prozessorkernen und die virtuelle Einteilung der verfügbaren Kapazitäten dieser Prozessoren ermöglichten die Parallelisierung des Arbeitsaufwandes und legten den Grundstein für Serverarchitekturen. Zudem eröffnete die kommerzielle Nutzung des Internets die Möglichkeit IT-Services in die Cloud, also auf jene zentralen Server, zu verlagern. Somit war es Nutzern möglich von ihren Rechnern aus auf die viel größeren Kapazitäten eines Cloud-Servers zurückzugreifen (vgl. [OOSC14], S. 40f).

Ein frühes Beispiel zur Veranschaulichung der Cloud Technologie und ihrer Verwendung, ist das virtuelle Musikgeschäft iTunes, welches Apple 2001 veröffentlichte. Mit einem iTunes-Konto hat der Benutzer auf Millionen von Songs und ähnliche Inhalte Zugriff. Man kann sich über mehrere Geräte mit der sogenannten iCloud verbinden, ohne die Daten lokal abspeichern zu müssen und unabhängig vom Betriebssystem des jeweiligen Gerätes (vgl. [K.13], S. 2f). Weitere Beispiele können in Abbildung 2.1 betrachtet werden.

Grundsätzlich wird zwischen 4 Kategorien von Cloud Services unterschieden: Software (engl. Software as a Service — SaaS), Plattform (PaaS), Infrastruktur (IaaS) und Identifikation (IDaaS). Bei SaaS wird die Applikation über den Webbrowser aufgerufen. Die Software selbst befindet sich mit samt ihren Programmen und Benutzerdaten in der Cloud. Unternehmen, die auf SaaS Lösungen zurückgreifen, müssen keine in-house Applikationen entwickeln und sich über Administration und Support keine Gedanken machen. Ein Beispiel



Quelle: [K.13, S. 4]

Abbildung 2.1: weitere Cloud Service Beispiele

für eine SaaS Anwendung ist Microsoft Office 365. PaaS stellen Entwickler*innen hingegen eine Reihe an Hardware- und Software-Ressourcen für die Entwicklung, den Build und das Deployment der eigenen Applikation zu Verfügung. Dies hat den Vorteil, dass Developer*innen keine Hardware kaufen beziehungsweise pflegen müssen und dass auch auf die Installation und Instandhaltung von Betriebssystemen und Datenbanken verzichtet werden kann. Prominente Beispiele hierfür sind Amazon Web Services (AWS) oder Microsoft Azure. Die Infrastruktur als ein Service ist vor allem für Unternehmen interessant, welche aus sicherheitstechnischen Gründen nicht auf PaaS zurückgreifen können. IaaS bieten Hardware-Ressourcen für firmeninterne Applikationen an. Der Kunde oder die Kundin ist somit nur noch für die Installation und das Management des Betriebssystems und Applikationssoftware verantwortlich und muss sich wie bei PaaS keine Gedanken über den Kauf und die Instandhaltung von Hardware machen. Auch hier kann beispielsweise auf AWS oder Microsoft Azure zurückgegriffen werden. IDaaS sollen den Klient*innen das Einloggen, das Passwort Management und ähnliches bei gleichzeitiger Gewährleistung der Sicherheit vereinfachen (vgl. [K.13], Kapitel 2, 3, 4, 5).

Cloud Computing hat viele Vorteile. Mit Hilfe der Virtualisierung kann dem*der Benutzer*in eine unlimitierte Skalierung der Ressourcen zugesagt werden. Das führt dazu, dass unerwartet hohe Belastungen der Nutzer*innen nicht zu Systemabstürzen auf Grund von limitierter Infrastruktur führen und dass Benutzer*innen keine hohen Anschaffungskosten für technische Infrastruktur aufbringen müssen, sondern an ihren Bedarf angepasst in die Cloud investieren können. Folglich können nicht nur die Anschaffungskosten, sondern auch die Betriebskosten erheblich gesenkt werden. Der*die Kund*in muss sich nicht um die Instandhaltung der Hardware oder Software kümmern und keine Computersysteme aufrüsten, sobald mehr Rechenleistung benötigt wird. In kürzester Zeit kann der Bedarf

des*der Kund*in von der Cloud gedeckt werden und es muss nur das gezahlt werden, was an Leistung verbraucht wird (vgl. [BSC+18], S. 5). Von dieser Kostenreduktion können besonders kleinere Unternehmen und Startups profitieren und die Innovation im Allgemeinen gefördert werden (vgl. [M.G14], S. 531f).

Wie J. Kris im Kapitel "Securing the Cloud" seines Buches "Cloud Computing" beschreibt, profitieren Cloud Anbieter*innen vom Economy-of-Scale Effekt, da sie ihren Service mehreren Kund*innen zur Verfügung stellen. Dadurch können die Anbieter*innen mehr Geld in die Sicherheit ihrer Systeme investieren. Die meisten Cloud Services haben eigene Teams für die Installation von Patches, um diese sofort deployen zu können. Das bedeutet wiederum, dass die Zeit, in der das System angreifbar ist, reduziert wird. Außerdem können Cloud Anbieter*innen durch die zusätzlichen finanziellen Mittel in Mitarbeiter*innen investieren, die sich nur mit der Prüfung und der Administration der System-Software befassen. Redundante Hardware und Software sichert die Cloud gegenüber Ausfällen ab (vgl. [K.13], Kapitel 9). Durch die Auslagerung der Services kann den Kund*innen somit auch die firmeninterne Verwaltung der Sicherheit vereinfacht werden.

Mit den Vorteilen der Skalierung und Virtualisierung gehen aber auch Herausforderungen einher. So müssen Cloud Anbieter*innen eine konstante Qualität des Services (Quality of Service — QoS) sicherstellen. Das bedeutet, dass der Service durchgängig erreichbar sein muss (Availability, Reliability) und durchgängig eine hohe Rechenleistung (Scalability) und Sicherheit (Security, Reliability) bieten muss. Die weite Bandbreite an Nutzungsmöglichkeiten und Kund*innen führt dazu, dass die Cloud zudem auf möglichst vielen Plattformen erhältlich sein muss. Beispielsweise Netflix darf nicht auf Geräte mit einem bestimmten Betriebssystem beschränkt sein, sondern muss neben Fernsehern auch für mobile Geräte oder Desktop PCs erreichbar sein. Die Cloud hat also mehr Anforderungen zu bewältigen als ein auf den*die Klient*in angepasstes Computersystem. Dieses Problem wird von sogenannten "cloud resource orchestration frameworks" (CROFs) angegriffen. Die Cloud Resource Orchestration beinhaltet die Selektion, das Deployment, das Monitoring und das Management von Software- und Hardware-Ressourcen zur Laufzeit. Um das Setup der Services für DevOps Manager*innen und Entwickler*innen leichter zu gestalten, bieten andere Firmen Frameworks (CROFs) an, die sich auf Gebiete wie das Deployment von Webapplikationen spezialisiert haben. Die steigende Anzahl an CROFs stellt uns jedoch vor neue Herausforderungen, da die starke Zunahme nicht mit einer Standardisierung einhergeht, sondern die Optionen der Entwickler*innen vervielfältigt. Dies macht wiederum die Selektion des richtigen CROFs schwierig (vgl. [KWR+16], S. 24f).

In den letzten Jahren ist die Nachhaltigkeit (Sustainability) von Technologie immer wichtiger geworden. Wurde zuvor noch die Performance-Steigerung in den Fokus gestellt, so wird der Energieeffizienz und dem ökologischen Fußabdruck neuer Technologien immer mehr Aufmerksamkeit geschenkt. Laut Greenpeace verbraucht Cloud Computing mehr Energie als die meisten Länder der Welt. Nur die USA, China, Russland und Japan übertreffen die Cloud noch. Doch mit der Reduktion des Energieverbrauchs geht bei Cloud Computing auch eine Reduktion des QoS einher. Punkte wie Availability, Reliability und Ressourcen Management benötigen viel Energie und sind für stabile Systeme unerlässlich. Nichtsdestotrotz sollte auch auf die Sustainability geachtet werden. Ein möglicher Ansatz ist es Deep Learning und künstliche Intelligenz in das Ressourcen Management der Cloud Anbieter*innen zu integrieren. Mit Hilfe einer künstlichen Intelligenz könnte der Workload opti-

mal auf die Server verteilt werden und so Energie eingespart werden (vgl. [BSC+18], S. 7f).

Um den Aspekt der Energieeffizienz aber auch Performance von Cloud Computing zu beleuchten, wird der Artikel "Extending Amdahl's Law for the Cloud Computing Era" von F. Díaz-del-Río et al. herangezogen. Amdahl's Law besagt, dass der Speedup S einer Berechnung die ursprüngliche Ausführungszeit durch die verbesserte Ausführungszeit ist. Die allgemeine Formel lautet:

$$S = \frac{1}{(1 - F) + F/S_f} \tag{2.1}$$

Dabei steht F für eine Sequenz des Programmes, welche parallelisiert werden kann, und (1 - F) für den nicht parallelisierbaren Code. S_f beschreibt die Anzahl der zur Verfügung stehenden Prozessoren, welche F gleichzeitig ausführen können. In Bezug auf Cloud Computing muss die Formel noch um 2 weitere Größen ergänzt werden: die Zeit, welche zum Senden der Daten an die Cloud benötigt wird (Z_S = Senden) und die Zeit, die benötigt wird, um das Ergebnis der Berechnung von der Cloud zu empfangen (Z_E = Empfang). Für den Speedup bedeutet das, dass im Idealfall durch Überlappung des Datentransfers und der Berechnung die Auswirkungen des Datenaustausches auf den Speedup erheblich gesenkt oder ignoriert werden können. Im schlechtesten Fall erfolgt die Berechnung jedoch erst, wenn die Cloud alle Daten empfangen hat, und die Cloud sendet dem*der Kund*in das Ergebnis, wenn die Berechnung vollkommen abgeschlossen ist. In diesem Fall muss die Zeit des Sendens und des Empfangens zu dem Speedup der Formel aus (2.1) addiert werden. Die daraus resultierende Formel lautet wie folgt:

$$S = Z_S + \frac{1}{(1 - F) + F/S_f} + Z_E \tag{2.2}$$

Amdahl's Law kann in Bezug auf Cloud Computing also folgender Maßen zusammengefasst werden:

Lokale Computersysteme haben eine limitierte Anzahl an verfügbaren Prozessorkernen. Ist die Berechnung nicht allzu komplex und sind genügend Kerne vorhanden, lohnt sich die Auslagerung der Berechnung in die Cloud nicht, da hier der Nachteil der Datenübertragung zum Tragen kommt. Je komplexer die Berechnung jedoch ist, desto weniger spielt der Datenaustausch eine Rolle und desto entscheidender wird die Rechenleistung. Für komplexe Operationen wird sich eine Auslagerung in die Cloud auszahlen, da der Skalierung durch die Virtualisierung keine Grenzen gesetzt sind und somit beliebig viele Kerne für die parallele Sequenz des Programmes herangezogen werden können.

Die Auslagerung hat neben der geringeren benötigten Rechenleistung des lokalen Systems aber noch einen weiteren Vorteil. Der Energieverbrauch einer Applikation ist definiert durch das Produkt der Leistung multipliziert mit der Zeit, die für die Berechnung benötigt wird. Betrachten wir Amdahl's Law für lokale Systeme, so haben wir einen Prozessorkern, der durchgehend Berechnungen durchführt, und mehrere Kerne, die für einen bestimmten Zeitabschnitt Energie für die Berechnung heranziehen. Da das Programm bei Cloud Computing jedoch in der Cloud und nicht lokal ausgeführt wird, ändert sich die Verteilung. Einzig für das Senden und Empfangen der Daten verbraucht die Applikation hierbei Energie. Je geringer also der Energieverbrauch beim Datenaustausch ausfällt, desto mehr

Energie kann lokal eingespart werden. Das ist vor allem für mobile Geräte und Embedded Systeme von Relevanz (vgl. [DdRSGSR16], S. 16ff).

Wie anhand von Amdahl's Law ersichtlich wird, ist Cloud Computing nicht für jede Anwendung geeignet. Die Abhängigkeit zum Internet und die damit einhergehende Latenz dürfen nicht außer Acht gelassen werden. Einerseits erhöht Cloud Computing die Auslastung des lokalen Netzwerkes, andererseits kann die lokale Berechnung schneller und damit effizienter sein, wenn die Latenz des Datentransfers im Verhältnis zur Komplexität der Berechnung hoch ist. Diese Umstände machen Cloud Computing für Echtzeitanwendungen uninteressant. Deswegen wurde das Cloud Computing Paradigma um 2 weitere Paradigmen ergänzt: dem Fog und dem Edge Computing.

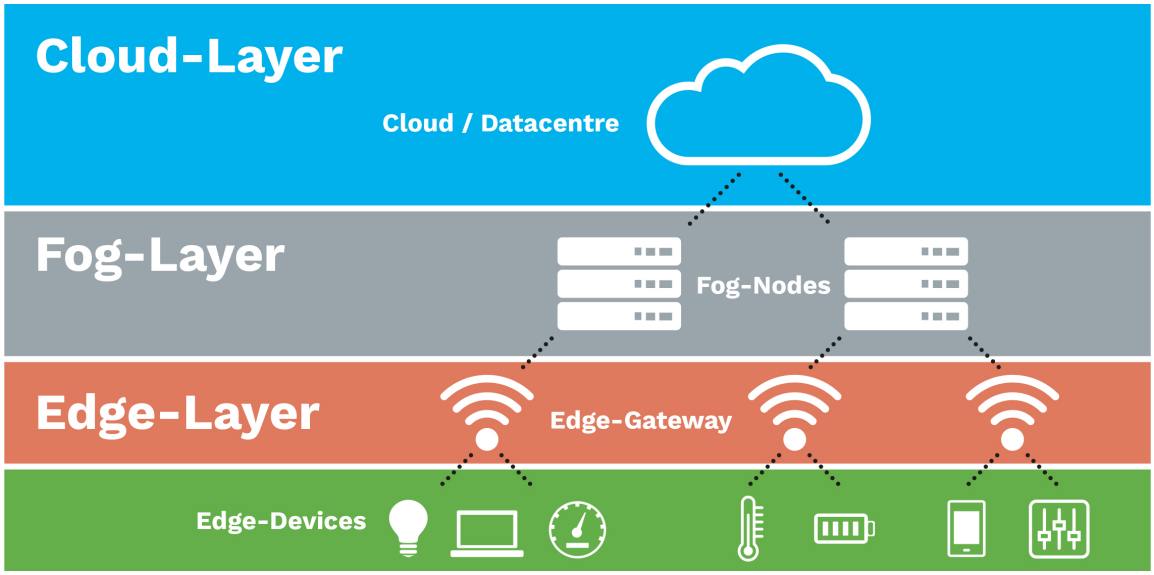
2.1.2 Fog Computing

F. Bonomi et al. beschreiben Fog Computing in ihrem Artikel "Fog Computing and its Role in the Internet of Things" wie folgt. Fog Computing befindet sich zwischen den traditionellen Cloud Computing Datenzentren und dem Endgerät. Wie beim Cloud Computing stellt Fog Computing ebenfalls Rechen-, Speicher- und Netzwerkressourcen zur Verfügung. Im Gegensatz zu den zentralen Datenzentren des Cloud Computings sind die Fog Computing Knotenpunkte dezentral und verteilt. Dadurch sinkt die Latenz des Netzwerkes. Die geographische Verteilung der Fog-Nodes kann beispielsweise bei Streaming-Diensten auch dann einen höheren QoS gewährleisten, wenn sich der*die Kund*in in einem Fahrzeug befindet und bewegt. Groß angelegte Sensornetzwerke zur Überwachung der Umwelt und das intelligente Stromnetz (Smart Grid) sind weitere Beispiele, welche eine größere Verteilung der Rechen- und Speicherkapazitäten notwendig machen. Aus der geografischen Verteilung folgt eine viel höhere Anzahl an Rechenzentren (= Fog Nodes), welche jedoch nicht dieselbe Leistung wie die zentralen Cloud Computing Server zur Verfügung stellen können. Fog Computing bietet Lokalität während Cloud Computing Globalisierung und Zentralisierung liefert (vgl. [BMZA12], S. 13ff).

2.1.3 Edge Computing

Gleich dem Fog Computing zielt Mobile Edge Computing auf geringe Latenz durch bewusste geographische Verteilung der Access Points (= Edge Server/Edge Nodes) ab. In beiden Paradigmen werden die Server am Rande des Internets, möglichst Nah am Endgerät, platziert. Daher wird beiden Herangehensweisen oftmals der Überbegriff Edge Computing gegeben. Zum besseren Verständnis bezeichnen wir in dieser Arbeit ausschließlich Mobile Edge Computing als Edge Computing Paradigma. Edge Computing unterscheidet sich von Fog Computing allein durch die noch nähere Lokalität der Nodes an den Endgeräten. Die Latenz wird also auf Kosten der Rechenleistung und Speicherkapazitäten weiter verringert. Die Anzahl der Server/Nodes wird gegenüber dem Fog Computing drastisch erhöht (siehe Abbildung 2.2). Dadurch werden einerseits, wie die Bezeichnung "Mobile Edge Computing" schon andeutet, mobile Anwendungen unterstützt, andererseits ermöglicht Edge Computing die Verwendung von real-time Anwendungen (vgl. [WES+19], S. 219f).

Zur Veranschaulichung der Vorteile des Edge Computings gegenüber dem Cloud Computing wird im Folgenden der Artikel "A Survey on the Convergence of Edge Computing and



Quelle: [Abb]

Abbildung 2.2: Schichten/Komponenten des Edge Computings. Der Fog-Layer ist optional.

AI for UAVs: Opportunities and Challenges" von P. McEnroe et al. herangezogen. UAVs sind unbemannte Flugfahrzeuge, auch Drohnen genannt. Für ferngesteuerte Drohnen kann Edge Computing essenziell sein. So erfordern Programme zur Kollisionsvermeidung des UAVs eine möglichst geringe Latenz, um gegebenenfalls rechtzeitig reagieren und ausweichen zu können.

Wie in Abbildung 2.2 ersichtlich ist, kann das Edge Computing in folgende 3 Komponenten unterteilt werden: dem Endgerät, dem Edge Gateway und dem Cloud Server. Das Endgerät stellt die Verbindung zum Edge Server her, welcher wiederum die Daten an den Cloud Server senden kann. Manchmal befindet sich der Edge Server direkt auf dem Endgerät. Je nachdem wie rechenintensiv die Berechnung ist und je nachdem wie wichtig die Latenz für die Berechnung ist, werden die Prozesse an die unterschiedlichen Nodes des Netzwerkes delegiert. Benötigt die Drohne eine hohe Rechenleistung, werden die Daten über das Edge Gateway an den Cloud Server gesendet und dort verarbeitet. Ist die Latenz von höherer Relevanz, dann kann das Endgerät entweder direkt die Berechnung durchführen, oder aber die Berechnung an die 2. Instanz, dem Edge Gateway, weiterleiten. Nachdem sich das Gateway in der Nähe des UAVs befindet, ist die Latenz in den meisten Fällen verkraftbar. Das Ziel dieses Paradigmas ist es also, die rechenintensiven Operationen auf die Cloud auszulagern und die Latenz-empfindlichen Operationen lokal durchzuführen. Das erhöht die Energieeffizienz aller Netzwerkteilnehmer und reduziert die Netzwerkauslastung. Für die Drohne bedeutet das, dass nur auf Echtzeit-Berechnungen spezialisierte und somit insgesamt weniger Hardware transportiert werden muss. So kann der lokale Energieverbrauch reduziert werden, da das UAV weniger Hardware tragen muss und dennoch Berechnungen in Echtzeit erfolgen können - wie in Amdahl's Law (2.1) bereits beschrieben wurde, spielt die benötigte Zeit einer Operation beim Energieverbrauch eine wichtige Rolle (vgl. [MWL22], S. 15435ff).

Doch auch die Herausforderungen des Mobile Edge Computing Paradigmas müssen beleuchtet werden. Um den Quality of Service zu gewährleisten, muss Edge Computing wie Cloud Computing die Sicherheit (Sercurity), den Datenschutz (Privacy) und die Skalierbarkeit (Scalability) der Ressourcen sicherstellen. Außerdem muss die Heterogenität (Heterogeneity) der Endgeräte unterstützt werden und der Dienst ununterbrochen verfügbar sein (Reliability). Im Zusammenhang mit der Reliability steht Edge Computing gegenüber dem Cloud Computing vor zusätzlichen Herausforderungen. Durch die Verteilung und Anzahl der Edge Server, welche QoS gewährleisten müssen, und durch die Mobilität der Nutzer*innen, welche sich ohne Verbindungsabbrüche in Echtzeit mit verschiedenen Edge Nodes verbinden, steigt der Wartungsaufwand der Edge Nodes enorm. Zudem steigen die Kosten für Techniken zur Fehlertoleranz, welche die durchgängige Verfügbarkeit der Ressourcen garantieren sollen. Diese Techniken erfordern zusätzliche Hardware und eine hohe Bandbreite des Netzwerks (vgl. [WES+19], S. 231f).

2.2 Internet of Things (IoT) und 5G

Um die steigende Anzahl an Netzwerkteilnehmern, die höhere Auslastung der Bandbreite und die Latenz-Anforderungen von Echtzeit-Anwendungen zu bewältigen, müssen neue Netzwerktechnologien entwickelt werden. Das Internet der 5. Generation soll diesen Anforderungen gewachsen sein und dem sogenannten “Internet of Things” (IoT) den Weg bereiten. Im Folgenden wird das IoT und der Zusammenhang dieser Technologie mit dem Edge Computing Paradigma erläutert. Um das “Internet der Dinge” zu beleuchten wird auf den Artikel “A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges” von G. A. Akpakwu et al. zurückgegriffen.

IoT verbindet eine Vielzahl an Geräte und lässt diese kommunizieren und miteinander interagieren. Von Waschmaschinen und Kühlschränken, mobilen Geräten, wie Tablets und Handys, bis zu Kameras und Sensoren, all diesen Geräten ermöglicht das IoT die Machine-to-Machine (M2M) Kommunikation. Die Anwendungsmöglichkeiten dieser Technologie sind riesig. Sogenannte “smart devices” könnten in jeder menschlichen Umgebung zum Einsatz kommen. “smart home”, “smart city” und “smart industry” sind nur einige wenige Begriffe, welche im Zusammenhang mit dem IoT stehen. Das Remote Monitoring der Industrie und von Agrarkulturen durch Drohnen und Roboter oder die automatisierte Verkehrsregulierung durch Sensoren (Smart Traffic) sind weitere Anwendungsgebiete.

Schätzungen zu Folge sind seit 2022 bereits über 12,5 Mrd. Geräte (vgl. [AIB+22], S. 7) über Mobilfunkstandards wie 2G, 3G und 4G, welche nicht auf das IoT optimiert sind, verbunden. Die steigende Anzahl der IoT-Geräte, stellt die Kapazitäten gängiger Netzwerkstandards vor große Herausforderungen. Da die Sensoren des Internet of Things auch an unzugänglichen Stellen platziert werden, müssen die Geräte über weite Gebiete erreichbar sein. Das Auswechseln von Batterien ist durch die Lokalität der Sensoren oftmals sehr umständlich und der Ausbau der Infrastruktur bzw. der Anschluss an das Stromnetzwerk nicht immer möglich, weshalb die Laufzeit der Geräte durch einen geringen Energieverbrauch maximiert werden muss. Somit muss sowohl die Internetverbindung als auch der ununterbrochene Betrieb der Geräte mit einem geringen Energieverbrauch einher gehen. Um die Anforderungen des IoT zu erfüllen, wurden neue Technologien zur Telekommunikation

entwickelt, welche sowohl über eine höhere Reichweite als auch über einen geringeren Energieverbrauch verfügen, als zuvor entwickelte Standards wie LTE (Long-Term Evolution). Diese Netzwerk-Technologien werden auch unter dem Akronym LPWAN (Low Power Wide Area Network) zusammengefasst. Der Verbindungsstandard LoRa (Long-Range), welcher in einem unlizenzierten Frequenzbereich für Industrie, Forschung und Medizinische Zwecke arbeitet, bietet sowohl eine hohe Reichweite als auch einen geringen Energieverbrauch und fällt unter die LPWAN Technologien. Ein weiteres Beispiel wäre Ingenu Random Phase Multiple Access (RPMA).

Das mobile Internet der 5. Generation, auch 5G genannt, soll weitere Abhilfe schaffen und den Mobilfunk an die Anforderungen des IoT anpassen. So soll 5G real-time Responses ermöglichen, weniger Energie verbrauchen und höhere Bandbreiten unterstützen. Das Problem von Hardware-basierten Netzwerkstrukturen wie dem Mobilfunk ist der manuelle Verwaltungsaufwand der Netzwerk-Installationen. Die Infrastruktur des mobilen Netzes sollte den ständig wachsenden Anforderungen an künftige Netzwerktechnologien gewachsen sein. Dennoch ist die konventionelle Infrastruktur veraltet und je schneller sich die Netzwerktechnologie weiterentwickelt, desto komplexer wird auch der manuelle Aufwand zukünftiger Konfigurationen. Deswegen werden Hardware-basierte Netzwerktechnologien nicht ausreichen, um die Anforderungen des Internets der Zukunft zu erfüllen. Um den Mobilfunk auf die Zukunft vorzubereiten, wird an Software-basierten Netzwerktechnologien wie dem "Software Defined Wireless Sensor Network" (SDWSN) geforscht, welche der Netzwerk-Hardware eine höhere Flexibilität ermöglichen (vgl. [ASHAM18], S. 3619ff).

Um die notwendigen Rechenkapazitäten für das IoT aufzubringen sind das Fog Computing Paradigma und das Mobile Edge Computing Paradigma essenziell. Damit die Endprozesse der Smart Devices durch das IoT optimiert werden können, müssen die Daten mehrerer Geräte aggregiert und analysiert werden. Dieser Datentransfer erfordert viel Energie und Ressourcen. Zudem würde ein ununterbrochener Datentransfer von Milliarden von Geräten das Netzwerk auslasten und die Latenz erhöhen. In Fog und Edge Computing werden genau diese Herausforderungen adressiert. Dadurch, dass die Rechenpower näher an das Endgerät gebracht wird, reduziert sich die Latenz, der Energieverbrauch, die Netzwerkauslastung und es werden Rechenkapazitäten eingespart. Auf Grund der Latenz- und Konnektivitätsanforderungen wären viele IoT Usecases ohne Fog/Edge Computing nicht möglich (vgl. [AIB+22], S. 27ff).

Ein weiterer Grundbaustein des IoT stellen künstliche Intelligenz bzw. Machine Learning Algorithmen dar. Die Sensoren oder, wie der Name schon sagt, "Smart" Devices sammeln ununterbrochen Informationen. Bei diesen Überwachungsdaten handelt es sich zumeist um sogenannte Heartbeat-Daten. Das bedeutet, das einen Großteil der Zeit keine relevanten Veränderungen detektiert werden. Diese Daten an ein Rechenzentrum zu senden und dort zu verarbeiten würde unnötig viele Ressourcen kosten. Um das zu verhindern kann Machine Learning eingesetzt werden, um die Heartbeat-Daten zu filtern und nur relevante Informationen an Server weiterzuleiten. Dies spart Energie, reduziert die Netzwerkauslastung und erlaubt dem Edge Device Entscheidungen in nahezu Echtzeit zu treffen (vgl. [SSS22], S. 2).

2.3 Artificial Intelligence (AI)

In dieser Sektion soll ein Grundverständnis für die Funktionsweise von Machine Learning Algorithmen aufgebaut werden. Dabei wird die Technologie im Allgemeinen und die Methodik des Deep Learning im Speziellen betrachtet. Dies soll als Fundament für den praktischen Teil der Arbeit dienen. Die folgenden 2 Abschnitte stützen sich hauptsächlich auf das Buch "Deep Learning" von I. Goodfellow et al.

2.3.1 Machine Learning

Mit Machine Learning bezeichnet man Algorithmen, die dazu im Stande sind, aus Daten zu lernen und so bessere Ergebnisse zu liefern. Um den Terminus "Lernen" in diesem Zusammenhang zu verstehen, muss man zuerst die Begriffe Task (T), Performance (P) und Experience (E) in Bezug auf Machine Learning erläutern.

Als ein Task wird eine Problemstellung bezeichnet, welche zu komplex ist, um sie mit herkömmlichen von Menschen designten Algorithmen zu lösen. Der Task selbst beschreibt dabei nicht den Prozess des Lernens, sondern wie das Machine Learning System, auch Modell genannt, ein "Sample" verarbeiten soll. Ein Sample wird formal als Vektor definiert, dessen Dimensionen für die Eigenschaften der Daten stehen. Dabei betrachtet man nur diejenigen Eigenschaften (= Features), die benötigt werden, um den Task lösen zu können.

Ziehen wir ein Beispiel heran, um T zu verdeutlichen: Die Daten bestehen aus unterschiedlichen Bildern. Diese Bilder haben Eigenschaften, welche beispielsweise Farbverläufe, Kontrast oder Helligkeit an gewissen Stellen des Bildes sein können. Wir wollen nun, dass unser Modell erkennt, auf welchem Bild sich ein Mensch befindet. Der Task lautet also "erkenne Menschen". Die Eigenschaften der Bilder, welche typischerweise einen Menschen charakterisieren, werden in Vektoren transformiert. Dabei steht ein Vektor für ein Bild und somit ein Sample. Der Algorithmus muss also nur jene Eigenschaften verarbeiten, welche für die Problemstellung "erkenne Menschen" relevant sind.

Es gibt viele unterschiedliche Aufgaben, die Machine Learning Algorithmen lösen können. Im genannten Beispiel muss das Modell die Bilder in 2 Kategorien einordnen: 1. auf dem Bild sind ein oder mehrere Menschen zu sehen oder 2. es sind keine Menschen zu sehen. Der Output des Modells ist eine Zuordnung zu einer der gewünschten Klassen. Der Algorithmus versucht also anhand der Eigenschaften die unterschiedlichen Kategorien zu erkennen und die Samples dementsprechend zuzuordnen. Diese Art der Problemstellung nennt man auch Klassifizierung. Ein weiteres Beispiel für ein Aufgabengebiet nennt sich Regression. Hierbei muss das Programm anhand der Eingabe eine Vorhersage in Form eines numerischen Wertes tätigen. Eingesetzt wird diese Form der Algorithmen, um zum Beispiel Preisentwicklungen am Finanzmarkt vorherzusagen (Algorithmic Trading). Anomaly Detection findet beispielsweise im Finanzsektor Anwendung, um mögliche betrügerische Transaktionen zu erkennen und zu unterbinden. Das Modell lernt Verhaltensmuster und markiert jene Transaktionen, welche aus diesem Muster ausbrechen. Bei Synthesis and Sampling lernt der Machine Learning Algorithmus anhand der Daten ähnliche neue Daten zu erstellen. Ein weiteres Beispiel nennt sich Denoising und wird unter anderem im Computergrafik Bereich verwendet. Unscharfe Bilder können so mittels Machine Learning ausgebessert und schärfer

gemacht werden. In der Computergrafik spricht man auch von AI-Supersampling. Machine Learning Modelle können aber noch viele weitere Aufgaben lösen.

Kommen wir nun zur Performance P. Im Machine Learning Kontext beschreibt die Performance eine messbare Größe, welche angibt, wie gut der Algorithmus seinen Task erledigt. Im Beispiel der Klassifizierung bedeutet das, dass überprüft wird, wie viele Bilder das Modell in die korrekte Kategorie eingeordnet hat. Man spricht auch von der Trefferquote (Accuracy). Umgekehrt kann man auch die Fehlerquote (Error Rate) messen. Die Werte dieser Größen befinden sich zwischen 0 und 1 und geben den Prozentsatz der jeweiligen Messung an. 1 bedeutet, dass 100% der Bilder richtig (Accuracy) bzw. falsch (Error Rate) zugeordnet wurden.

Die Experience (E) beschreibt, wie der Machine Learning Algorithmus aus den Daten lernt und den Datensatz "erfährt". Es gibt grundsätzlich 2 Arten von Machine Learning Algorithmen: Unsupervised und Supervised Machine Learning. Unsupervised Algorithmen erlernen aus den Daten statistische Verteilungen, welche für die Eigenschaften der Samples stehen, und unterteilen anhand dieser Verteilungen die Daten. Denoising Algorithmen versuchen beispielsweise jene Verteilung der Eigenschaften zu erkennen, welche zu den Verzerrungen der Daten geführt haben, um diese im weiteren Verlauf umzukehren. Unsupervised bedeutet also, dass dem Modell während des Lernvorganges nicht gezeigt wird, welches Ergebnis erwartet wird. Das Modell erlernt selbständig statistische Muster in den Daten zu erkennen und basiert seine Entscheidungen auf diese.

Im Kontrast dazu steht das Supervised Learning. Hier werden dem Modell beim Lernprozess/Training zusätzlich zu den Eigenschaftsvektoren des Datensatzes auch sogenannte "ground truth" Daten oder "labels" übergeben. Diese Labels markieren die Trainingsdaten und zeigen das von uns Menschen erwartete Ergebnis an.

Anhand des Task-Beispiels bedeutet das, dass wir die Bilder zuerst selbst der jeweiligen Klasse "Mensch im Bild" oder "kein Mensch im Bild" zuordnen müssen. Supervised Learning besagt, dass es eine Instanz (meist der*die Entwickler*in) gibt, die dem Machine Learning System zeigt, welches Ergebnis erwartet wird. Das Modell kann dann das erwünschte mit dem prognostizierten Ergebnis abgleichen und daraus lernen.

Wie ist der Learning-Begriff nun im Kontext des Machine Learnings zu verstehen? Ein Algorithmus lernt aus Experience (E), wenn seine Performance (P) in Bezug auf Task (T) zunimmt. Anhand des "Menschen erkennen" (T) Beispiels bedeutet das, dass die Accuracy (P) eines beispielsweise Supervised Machine Learning Algorithmus (E) zunimmt, wenn das Programm mit mehr Daten trainiert wird.

Jedoch kann ein Trainingsdatensatz nicht die gesamte Realität eines Tasks abbilden, weshalb ein Machine Learning Algorithmus auch zu bis dato unbekannten Daten Aussagen treffen muss. Die Schwierigkeit eines guten Modells liegt darin, dass auch bei jenen unbekannten Daten eine gute Performance erreicht wird. Man spricht auch von der "Generalisierung" eines Tasks.

Betrachten wir nochmals das Beispiel des "erkenne Menschen auf Bildern" Modells. Um den Task zu erlernen, muss das Modell mit Trainingsdaten gefüttert werden. Auch wenn dieser Datensatz aus Millionen von unterschiedlichen Bildern bestünde und Menschen aller

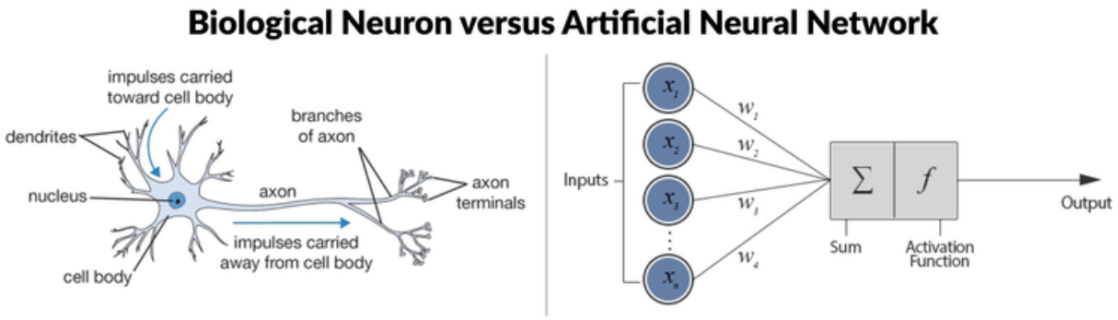
Quelle: [GBC16, S. 153]

Abbildung 2.3: Curse of Dimensionality: Je mehr Dimensionen der Eigenschaftsvektor hat, desto komplexer die Berechnung

Nationen, Kulturen, Größe, Alters usw. beinhalten würde, könnte selbst die Menge an Bildern nicht die unendlich vielen Kombinationen abbilden, die es in der Realität für Bilder mit Menschen gibt. Im schlechtesten Fall wäre der Datensatz "biased". Das bedeutet, dass der Algorithmus fälschlicherweise Eigenschaften der Daten für den Task als wichtig erachtet, welche keine Relevanz für T besitzen. So könnten alle Menschen des Trainingsdatensatzes eine Brille tragen und anstatt den Menschen zu erkennen, erlernt das Modell die Brillen zu erkennen. Würde man nun versuchen ein Bild eines Menschen ohne Brille zu klassifizieren, würde der Algorithmus dies der falschen Kategorie zuordnen, weil das Modell nicht den Menschen, sondern die Brille erkennt.

Um Generalisierungsfehler möglichst klein zu halten, unterteilt man die Daten in einen Trainings- und einen Testdatensatz. So kann nach dem Trainingsprozess anhand der Testdaten überprüft werden, ob die Trainingsperformance der Testperformance ähnelt. Ist die Trainingsperformance gut und die Testperformance signifikant schlechter, dann spricht man auch von "Overfitting" - der Algorithmus hat sich auf die Trainingsdaten spezialisiert und kann unbekannte Daten nicht korrekt zuordnen. Ist die Trainingsperformance schlecht, nennt man dies "Underfitting" - das Modell wurde mit zu wenig Daten trainiert. Ein gutes Machine Learning System liefert sowohl für die Trainingsdaten als auch die Testdaten eine hohe Genauigkeit.

Machine Learning Systeme besitzen oftmals sogenannte Hyperparameter. Der Algorithmus kann mit diesen Parametern an die eigenen Bedürfnisse angepasst werden und so auf den eigenen Task optimiert werden. Hyperparameter verändern das Verhalten des Modells. So kann beispielsweise die Anzahl der trainierbaren Parameter angepasst werden. Das Training bzw. der Lernprozess eines Modells ergibt sich aus der Gewichtung der Eigenschaften, welche für die Erfüllung des Tasks notwendig sind, sowie der Optimierung dieser Hyperparameter. Wie beim Optimieren der Gewichte, kann es beim Training der Hyperparameter zu Overfitting kommen. Damit dies nicht passiert, unterteilt man den Trainingsdatensatz in 2 vollkommen unabhängige Datensätze: den Trainingsdatensatz und den Validierungsdatensatz. Der Validierungsdatensatz ist zur Überprüfung des Generalisierungsfehlers der Hyperparameter zuständig. Nur wenn das Modell bei allen 3 Datensätzen eine hohe Accu-



Quelle: [bio]

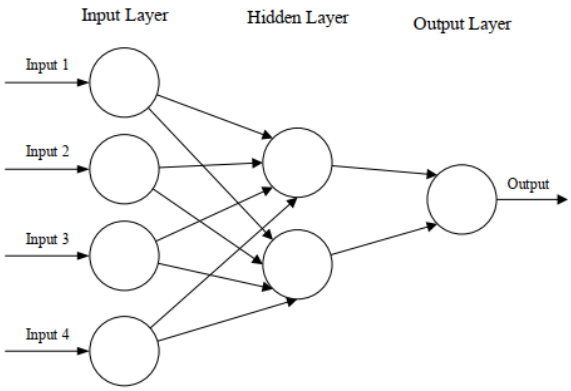
Abbildung 2.4: Auf der linken Seite wird der klassische Aufbau eines natürlichen Neurons dargestellt. Die Dendriten empfangen Signale, welche über den Zellkörper an den Nukleus weitergeleitet werden. Der Nukleus feuert, wenn eine bestimmte Bedingung der Eingänge erfüllt ist, einen Impuls ab, welcher wiederum über das Axon zum nächsten Neuron geleitet wird. Das künstliche Neuron auf der rechten Seite bildet diesen Vorgang mathematisch ab. Die Eingänge werden mit Gewichten (w_i) multipliziert und im "Zellkern" aufsummiert. Eine sogenannte Aktivierungsfunktion entscheidet dann, ob bzw. welcher Output generiert wird.

racy bzw. eine niedrige Error Rate aufweist, wird der Algorithmus auch im realen Einsatz gut abschneiden.

Traditionelle Machine Learning Algorithmen haben Schwierigkeiten komplexere Aufgaben wie Objekterkennung oder Spracherkennung zu lösen. Dies liegt daran, dass die Komplexität der Tasks mit der Anzahl der Eigenschaften, welche zum Lösen dieser notwendig sind, exponentiell ansteigt. Wie bereits erwähnt, beschreibt ein Vektor die für die Aufgabe relevanten Eigenschaften, wobei eine Dimension des Vektors für eine Eigenschaft der Daten steht. Je komplexer das Problem ist, desto mehr Eigenschaften benötigt der Algorithmus, um den Task zu lösen und umso mehr Dimensionen besitzt der Vektor. Man spricht auch vom "curse of dimensionality" (siehe auch Abbildung 2.3), also dem Fluch der Dimensionalität. Um dieses Problem zu umgehen, wurde eine weitere Art von Machine Learning Algorithmen entwickelt: das Deep Learning (vgl. [GBC16], Part I Kapitel 5).

2.3.2 Deep Learning und CNNs

Deep Learning wurde von biologischen Nervensystemen wie dem menschlichen Gehirn inspiriert. Deswegen nennt man diese Art der Machine Learning Algorithmen auch künstliche neuronale Netze (vgl. [ON15], S. 1). 1958 wurde von F. Rosenblatt in seinem Artikel "The perceptron: a probabilistic model for information storage and organization in the brain." das Perzeptron als Modell für die Speicherung von Informationen im menschlichen Gehirn vorgestellt (vgl. [Ros58], S. 386). Die einfachste Form eines Perzeptron Algorithmus wird auch "single-layer" Modell genannt und ähnelt im Aufbau einem menschlichen Neuron (vgl. [LLY+22], S. 6999). In Abbildung 2.4 wird der biologische Aufbau eines Neurons dem



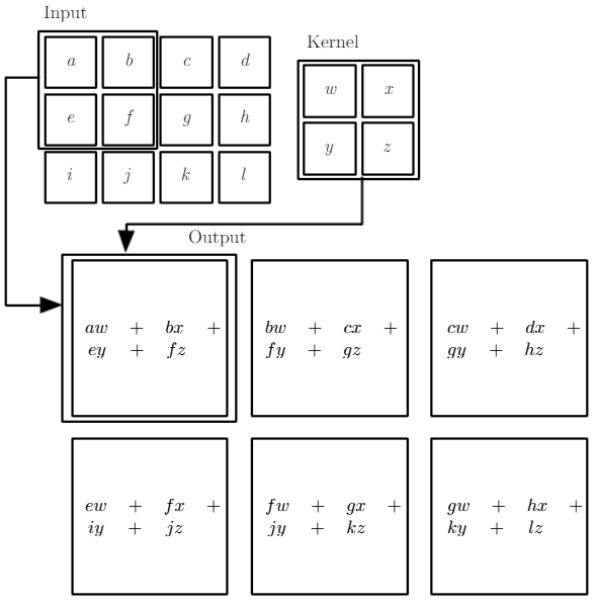
Quelle: [ON15, S. 2]

Abbildung 2.5: Ein einfaches 3-schichtiges FNN. Jede Kante stellt die Multiplikation des Wissens (Gewichte) mit den Eingängen dar. Jeder Knotenpunkt bildet die Aufsummierung der Ergebnisse und Aktivierung durch die Aktivierungsfunktion ab.

eines künstlichen Neurons gegenübergestellt.

Das Single-Layer-Modell besteht aus einem Knotenpunkt, dessen Input aus einer fixen Anzahl an Eingabewerten multipliziert mit Gewichten besteht. Die Ergebnisse aller Multiplikationen werden im Perzeptron aufsummiert (siehe Abbildung 2.4). Die Eingabe beschreibt also eine Matrixmultiplikation aus den Eigenschaften der Daten, welche von den Eingabewerten abgebildet werden, und den Gewichten des Perzeptrons, welche das gespeicherte Wissen des künstlichen neuronalen Netzes darstellen. Wie bei natürlichen Neuronen bestimmt ein Schwellenwert, auch Aktivierungsfunktion genannt, ob das Neuron feuert und wenn ja, welchen Wert die Ausgabe besitzt (vgl. [Est97], S. 3f). Es gibt eine Vielzahl an Möglichkeiten ein Single-Layer-Modell aufzubauen. So kann das Modell unterschiedlich viele Eingänge bzw. Features akzeptieren oder die Aktivierungsfunktion kann variieren und dadurch unterschiedliche Funktionalitäten erfüllen (vgl. [BHR14], S. 149f). Trainiert werden neuronale Netze, indem die Gewichte der Kanten angepasst werden. Diese bilden wie bereits erwähnt das Wissen eines Perzeptron ab. Dadurch werden die Eigenschaften der Daten wie bei herkömmlichen Machine Learning Algorithmen unterschiedlich bewertet (vgl. [Est97], S. 4f).

Grundsätzlich ist ein Perzeptron nicht auf einen Knotenpunkt beschränkt, sondern kann ein Netzwerk aus Knoten bilden. Solche Machine Learning Modelle nennt man auch “multilayer perceptrons” oder “feedforward neural networks” (vgl. [GBC16], S. 164). Bei diesen Multi-Layer-Modellen werden die Knoten in Schichten angereiht und miteinander vernetzt. Ein einfaches Beispiel eines 3-schichtigen Feedforward Neural Networks (FNN) besteht aus einem Input-Layer, einem sogenannten Hidden-Layer und einem Output-Layer (siehe Abbildung 2.5). In jedem der Knotenpunkte werden die Eingaben zusammengerechnet und über eine Aktivierungsfunktion die Ausgaben ermittelt. In den Kanten des Multi-Layer-Perzeptrons werden die Werte mit Gewichten multipliziert und an das nächste künstliche Neuron weitergeleitet. Je nach Schicht des Netzwerkes variiert die Aktivierungsfunktion

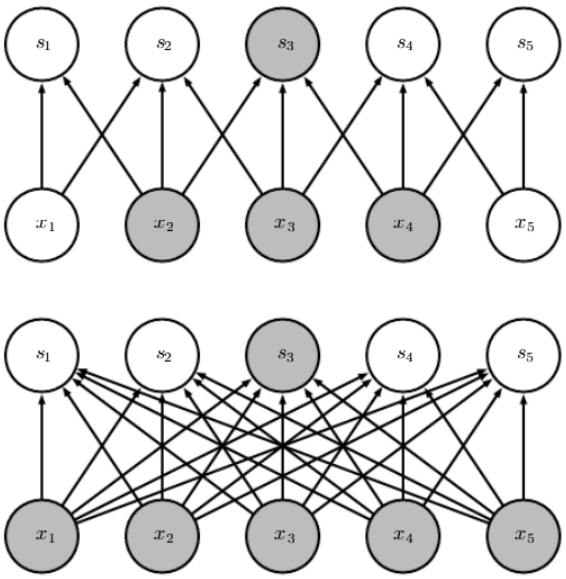


Quelle: [GBC16, S. 330]

Abbildung 2.6: Der Input eines CNNs sind die Pixelwerte des Bildes. Der Kernel eines Convolution Layers wird beim Modellieren des CNNs definiert und kann unterschiedliche Funktionen innehaben. In diesem Beispiel betrachtet ein Kernel 4 Pixel und multipliziert und summiert die Werte mit seinen eigenen Werten auf, wodurch ein Output generiert wird. Danach wird der Kernel verschoben und betrachtet die nächsten 4 Pixel.

der Knotenpunkte und die Vernetzung der Kanten und je nach Architektur des neuronalen Netzes können unterschiedliche Tasks verrichtet werden. Weitere Beispiele für künstliche neuronale Netzwerke sind “Restricted Boltzmann Machines” oder “Recurrent Neural Networks” (vgl. [ON15], S. 2). Für diese Arbeit wollen wir uns auf sogenannte Convolutional Neural Networks (CNN) konzentrieren.

CNNs werden hauptsächlich für die Mustererkennung in Bilddaten verwendet. Ihre Netzwerkarchitektur ist auf das Format der Bilder und auf den eigenen speziellen Task der Objekterkennung optimiert. Dadurch kann die Größe und Komplexität des CNNs reduziert werden und es müssen insgesamt weniger Parameter gesetzt werden (vgl. [ON15], S. 3). Der Name “Convolutional” Neural Network bezieht sich auf die mathematische Operation der Faltung. CNNs sind neuronale Netzwerke, welche zumindest einen Layer besitzen, der eine Faltungsoperation statt einer Matrixmultiplikation durchführt. Eine Faltung im Kontext der CNNs beschreibt eine Multiplikation und Addition der einzelnen Eingabewerte mit einem sogenannten Kernel. Dieser Kernel stellt ein Array von unterschiedlichen Werten dar und weist meist eine geringere Dimension als die Eingabe auf. Die Dimensionen des Kernels definieren, wie viele Werte betrachtet und multipliziert werden. Je nach Format und Werten des Kernels kann dieser unterschiedliche Vorteile und Funktionen, wie beispielsweise horizontale oder vertikale Kantendetektierung, mit sich bringen. In Abbildung 2.6 wird die



Quelle: [GBC16, S. 332]

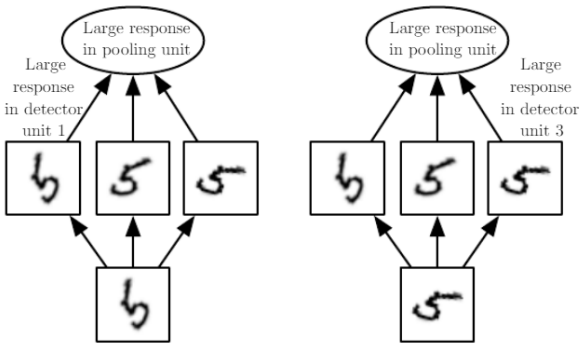
Abbildung 2.7: Die farblich markierten Knotenpunkte x_1 und x_5 des unteren neuronalen Netzwerkes müssen durch die Faltung nicht mit s_3 verbunden werden, da die Faltung diese Pixel bereits in ihrer Berechnung inkludiert. Daraus resultiert das obere CNN.

Funktionsweise der Faltung demonstriert.

Die Faltung hat gegenüber der Matrixmultiplikation einige Vorteile. Der erste große Vorteil ist die Reduktion der notwendigen Gewichte und dadurch trainierbaren Parameter, auch "sparse interactions" genannt. Anstatt jeden Inputwert mit einem eigenen Gewicht zu multiplizieren, wird bei der Faltung auf den Kernel zurückgegriffen, welcher eine kleinere Dimension als die Eingaben aufweist. Man nennt den Kernel deshalb auch "sparse weights". Bei der Faltung wird also nicht nur der Trainingsaufwand reduziert, sondern auch die Effizienz der Berechnung erhöht. Dieser Vorgang der Berechnung (die Klassifizierung von Daten) wird Inferenz genannt.

Ein weiterer Vorteil nennt sich "parameter sharing". Wie in Abbildung 2.6 zu sehen ist, wird der Kernel mehrere Male auf ein und dasselbe Pixel des Bildes angewandt. Dies führt dazu, dass der Parameter in die Gewichtung weiter entfernt liegender Knotenpunkte des Netzwerks mit einbezogen wird und nicht eigens mit diesen Knotenpunkten verbunden werden muss. Auch Parameter Sharing reduziert somit die Komplexität des Modells erheblich und erhöht folglich die Geschwindigkeit der Inferenz. Wie Abbildung 2.7 zeigt, kann die Anzahl der Kanten eines künstlichen neuronalen Netzwerkes durch die Faltung reduziert werden.

Neben dem Convolutional Layer ist der sogenannte "pooling layer" für CNNs charakteristisch. Pooling fasst die Ausgabe mehrerer Neuronen einer Bildregion durch eine anfangs definierte Operation zusammen. Ein "max pooling layer" leitet beispielsweise nur den höchsten Wert aller betrachteten Neuronen weiter. Pooling reduziert die Größe und Komplexität



Quelle: [GBC16, S. 338]

Abbildung 2.8: Ein CNN muss 2 unterschiedlich transformierte 5er als solche erkennen. Die Faltung des 1. Bildes führt zu 3 Ausgaben, von denen die erste Einheit den höchsten Wert ausgibt, da ihr Muster dem 5er am meisten ähnelt und diesen als solche erkennt. Im 2. Versuch detektiert die 3. Einheit der Faltung den 5er. Der Max-Pooling Layer gibt nur die Ergebnisse der höchsten Detektor-Einheit aus, wodurch in beiden Fällen der 5er trotz Rotation erkannt wird.

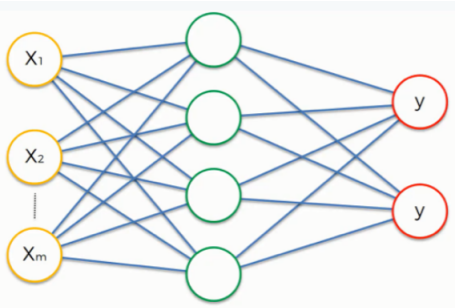
des CNNs weiter. Außerdem führt die Operation dazu, dass das neuronale Netz invariant bezüglich kleinerer Transformationen der Muster im Bild wird (vgl. [GBC16], Part II Kapitel 9). Die Funktionsweise der Pooling Schicht wird in Abbildung 2.8 demonstriert.

Abgeschlossen werden CNNs mit einem "fully-connected layer". Diese Schicht verbindet ein jedes Neuron mit allen Neuronen der vorherigen Schicht und allen Neuronen des Output-Layers, welcher schlussendlich die Klassifizierung durchführt (siehe Abbildung 2.9). Sinn dahinter ist es, alle Bildregionen der einzelnen Neuronen zu vereinen und so nach den gewünschten Mustern im gesamten Bild zu suchen. Nachteil einer fully-connected Neuronenschicht ist die Anzahl an Verbindungen und dadurch Parametern, welche das Training komplizierter gestalten und verlangsamen (vgl. [SAS17], S. 5).

Die Deep Learning Technologie bringt auch einige Herausforderungen mit sich. Wie Y. Guo et al. in ihrem Artikel "Deep learning for visual understanding: A review" zusammenfassen, sind Deep Learning Modelle sehr komplex und die zugrunde liegende Technologie wird noch nicht vollends verstanden. So liegt beispielsweise für Entwickler*innen die Schwierigkeit darin herauszufinden, welche Modell-Architektur die besten Ergebnisse für ihren Task liefert. Wie viele Schichten soll ein neuronales Netzwerk haben oder wie viele Neuronen pro Schicht liefern das beste Ergebnis für die eigene Problemstellung?

Große künstliche neuronale Netzwerke erreichen eine bessere Performance als kleine Modelle. Damit einher geht aber auch das Risiko für Overfitting, steigende Ressourcenanforderungen - bezogen auf die Rechenleistung, die Trainingsdatenmenge und den Energieverbrauch - sowie längere Inferenz-Zeiten (= Latenz). Eine weitere Herausforderung ist es also ein Deep Learning System zu entwickeln, welches effizient im Training und der Inferenz ist und dennoch gute Ergebnisse liefert. Speziell für real-time Anwendungen ist eine schnelle Inferenz von großer Bedeutung.

Die benötigten Ressourcen stellen besonders in Bezug auf das Training der neurona-



Quelle: [ful]

Abbildung 2.9: Die X_i Knotenpunkte beschreiben den Input der in grün gekennzeichneten Fully-Connected Schicht. Jeder X-Knoten wird mit jedem grünen Knoten verbunden. Der Fully-Connected Layer generiert wiederum Ausgaben, welche im roten Output-Layer aggregiert werden. Auch die y-Knoten sind dabei mit jedem grünen Knotenpunkt verbunden. Die Ausgabe der y_i gibt die Klassifizierung des CNNs an.

len Netze eine Herausforderung dar. Deep Learning benötigt viele Daten um die Modelle trainieren zu können. Oftmals handelt es sich um Supervised Learning, wodurch die Trainingsdaten zuvor manuell aufbereitet werden müssen und sehr teuer werden können. Hinzukommt die Problematik, dass für manche Tasks die Datenerhebung entweder nicht in dem Ausmaß, indem es notwendig wäre, oder gar überhaupt nicht möglich ist. "Data Augmentation" Algorithmen sind ein Ansatz um diese Probleme zumindest teilweise zu lösen. Diese Algorithmen variieren bestehende Bilddaten durch Transformationen wie Rotation oder Stauchung hinreichend stark, um die Trainingsperformance zu verbessern und gleichzeitig das Overfitting-Risiko klein zu halten (vgl. [GLO+16], S. 43f).

2.3.3 Transfer Learning

"Transfer Learning" (TL) beschreibt einen weiteren Ansatz um die Herausforderungen der CNN-Technologie zu lösen. TL ermöglicht die Entwicklung performanter Modelle ohne große Trainingsdatensätze. Wie der Name schon sagt, wird dazu Wissen transferiert. Das bedeutet, dass wir ein bereits trainiertes Deep Learning System, welches eine gute Performance aufweist und dessen Aufgabenbereich unserem eigenen Anwendungsfall ähnelt, für unsere eigenen Zwecke adaptieren können. Um zu verstehen, warum das möglich ist, ziehen wir das Beispiel aus "A survey of transfer learning" von K. Weiss et al. heran. Angenommen 2 Menschen wollen Klavier spielen lernen. Einer von beiden hat keinen musikalischen Hintergrund, muss diese neue Fähigkeit also von Grund auf erlernen. Der zweite Mensch hingegen hat bereits gelernt auf der Gitarre zu spielen. Die zweite Person kann bereits erlerntes Wissen für den Task "Klavier spielen" verwenden und dadurch effizienter lernen als die 1. Person, die beispielsweise zusätzlich Noten lesen lernen muss. Selbiges gilt für Deep Learning Systeme. Ein Modell, welches bereits gelernt hat, ein bestimmtes Objekt in einem Bild zu erkennen, hat gelernt, Muster in Bildern zu erkennen. Weicht die Domäne eines anderen Tasks nicht allzu sehr ab, kann das Modell mittels Transfer Lear-

ning auf den neuen Task umgelernt werden und die erlernte Mustererkennung auch auf den neuen Bilderkennungstask angewendet werden. Auf das Beispiel der Menschen bezogen bedeutet das, dass der zweite Mensch nur in der Musikdomäne Wissen transferieren kann oder die Tätigkeit des Gitarre Spielens für eine andere ähnliche Aktion verwenden kann. Wichtig für das Transfer Learning ist, dass die Aufgabenstellungen Ähnlichkeiten aufweisen, ein Bilderkennungsmodell beispielsweise keine Spracherkennungstasks erlernen soll (vgl. [WKW16], S. 1f). Nichtsdestotrotz kann nicht immer Wissen transferiert werden, wenn sich 2 Domänen ähneln. Ein anderes Beispiel aus "A Comprehensive Survey on Transfer Learning" von F. Zhuang et al. ist das Erlernen der Sprachen Spanisch und Französisch. Obwohl sowohl Spanisch als auch Französisch romanische Sprachen sind und einen ähnlichen Aufbau besitzen, werden Menschen, welche die eine Sprache beherrschen, nicht unbedingt effizienter beim Erlernen der zweiten Sprache sein. Man nennt den ursprünglichen Task eines Transfer Learning Modells auch "source" Task und die zu erlernende neue Aufgabe "target" Task (vgl. [ZQD+21], S. 44).

Je nach Setting der Aufgabenstellungen und Problemdomänen, wird Transfer Learning in unterschiedliche Kategorien unterteilt. Supervised Transfer Learning wird in induktives und transduktives Transfer Learning unterteilt. Induktives Transfer Learning beschreibt ein Setting, bei dem die Domänen (z.B. Musik) der Aufgaben dieselben sind, aber sich die Tasks (z.B. Gitarre/Klavier spielen) unterscheiden. Für den induktiven Wissenstransfer sind Trainingsdaten inklusive Ground Truth Daten der neuen Aufgabenstellung notwendig, um das Modell an den Target Task anzupassen. Jedoch fällt die notwendige Datenmenge um ein Vielfaches geringer aus als beim herkömmlichen Lernprozess eines CNNs. Das transduktive Transfer Learning beschreibt eine Ausgangsituation, bei der der Source- und der Target-Task dieselben sind, die Domänen sich jedoch unterscheiden. Hierfür werden keine zusätzlichen Trainingsdaten benötigt. Die 2 genannten Kategorien können je nach Trainingsumständen noch weiter unterteilt werden. Die 3. Kategorie heißt unsupervised Transfer Learning. Hier müssen die Domänen oder Tasks ebenfalls Ähnlichkeiten aufweisen, es werden jedoch keine Labels benötigt.

Welche Art von Wissen kann transferiert werden? Es gibt insgesamt 4 Herangehensweisen, welche vom Setting des Transfer Learnings abhängen. Diese 4 Transfer Learning Methoden lauten "Instance-transfer", "Feature-representation-transfer", "Parameter-transfer" und "Relational-knowledge-transfer". Instance-transfer gewichtet manche der markierten Daten aus der Source-Domäne neu und passt das Modell-Training so an den Target-Task an. Es kann sowohl im induktiven als auch im transduktiven Transfer Learning angewendet werden. Feature-representation-transfer beschreibt eine Vorgehensweise, bei der ein Set an Eigenschaften gesucht wird, welches sowohl die Source- als auch die Target-Domäne in den Daten "gut" repräsentiert. Die Unterschiede zwischen den Domänen sollen minimiert werden und dadurch die Fehlerrate der Klassifizierung oder Regression des Modells möglichst klein gehalten werden. Diese Methode kann für das induktive, das transduktive aber auch das unsupervised Transfer Learning verwendet werden. Beim Parameter-transfer werden Parameter gesucht, welche sowohl das Source- als auch das Target-Modell teilen und somit wiederverwendet werden können. Parameter-transfer findet nur im induktiven Transfer Learning Anwendung. Auch der Relational-knowledge-transfer kann nur beim induktiven Wissenstransfer verwendet werden. Im Gegensatz zu den ersten 3 genannten Herangehensweisen ist diese Methode für relationale Domänen entwickelt worden. Ein Beispiel einer

solchen Domäne sind Social Media oder andere Netzwerkdaten. Die Annahme ist, dass die Daten zusammenhängen und nicht unabhängig sind.

Transfer Learning verspricht eine effiziente und effektive Methode zu sein ein künstliches neuronales Netzwerk zu trainieren. Modelle, welche mittels Wissenstransfer bereits trainierter Machine Learning Systeme entwickelt werden, weisen zumeist einen geringeren Generalisierungsfehler auf und können den Entwicklungs- und Trainingsaufwand immens reduzieren. Nichtdestotrotz birgt Transfer Learning auch das Risiko des sogenannten negativen Transfers. Wie anhand des Französisch-Spanisch Beispiels gezeigt wurde, kann es passieren, dass vermeintlich ähnliche Problemstellungen keinen Wissenstransfer ermöglichen. Transfer Learning kann hier zu schlechteren Ergebnissen führen, als wenn kein Wissenstransfer stattfinden würde. Es gilt die Übertragbarkeit von Wissen zwischen Domänen und Tasks noch genauer zu studieren (vgl. [PY10], S. 3ff).

3 Edge Intelligence: Verwandte Arbeiten

Die Limitationen des Edge Computing Paradigmas einschließlich der Hardware-Kapazitäten (limitierte Rechenleistung und Speicherallokation) und der Netzwerkanforderungen (Echtzeitanwendungen und hohe Bandbreiten) erfordern ein optimiertes Ressourcenmanagement. Die Integration von Artificial Intelligence (AI) in das Edge Computing Paradigma kann viele dieser Herausforderungen adressieren. AI-Modelle können für die Optimierung des "computation offloading", also der Delegation rechenintesiver Operationen, verwendet werden. Dadurch kann die Netzwerkauslastung reduziert werden (vgl. [SLY19, SGAS20]). Andererseits ermöglicht die Ausführung von Machine Learning Systemen auf den Endgeräten die Einsparung lokaler Ressourcen und die Entscheidungsfindung komplexer Fragestellungen in Echtzeit, wodurch weniger Daten über das Netzwerk transferiert werden müssen. Machine Learning Algorithmen auf mobilen Geräten nennt man auch Edge Intelligence (EI) (vgl. [ZWL+19], S. 1840).

Ein autonomes Fahrzeug generiert durch seine Sensoren und Kameras ungefähr 1 Gigabyte an Daten pro Sekunde (vgl. [ZWL+19], S. 1840). Die EI verarbeitet diese Daten direkt auf dem Endgerät und entscheidet darüber, welche Daten relevant sind und welche Daten an Server weitergeleitet werden sollen. Dadurch wird die Menge der transferierten und gespeicherten Daten erheblich reduziert und die Netzwerkauslastung sowie der Energieverbrauch verringert. Zudem ist die Distanz, über die der Datenaustausch stattfindet, durch das Edge Computing Paradigma viel kleiner als beim Cloud Computing. All dies führt dazu, dass die Sicherheit, der Datenschutz und die Zuverlässigkeit einer EI gegenüber einer Cloud AI immens steigen, die Latenz verringert werden kann und Echtzeitanwendungen ermöglicht werden (vgl. [MWL22], S. 15438f). Um die Vorteile der EI und die mit der Technologie einher gehenden Herausforderungen zu verdeutlichen, werden als Beispiel wie in Kapitel 2.1.3 UAV-Systeme betrachtet und der Artikel "A Survey on the Convergence of Edge Computing and AI for UAVs: Opportunities and Challenges" von P. McEnroe et al. herangezogen.

Die EI-Technologie ermöglicht einer erhebliche Reduktion des lokalen Energieverbrauchs. Da AI-Chips eine andere Architektur aufweisen und mit "near-memory" oder "in-memory" Datenströmen arbeiten, ermöglichen sie dem Endgerät die Daten zu verarbeiten, ohne dass diese zwischen Speicher und Prozessor transferiert werden müssen. Außerdem werden die Machine Learning Algorithmen meist als 8- oder 16-Bit-Berechnung ausgeführt, was gegenüber 32-Bit weitere Energieeinsparungen ermöglicht. Qualcomm behauptet einen optimierten Edge AI Chip entwickelt zu haben, der 25 mal weniger Energie verbraucht als herkömmliche Computerchips (vgl. [MWL22], S. 15439). Für Drohnen ist der Energieverbrauch von großer Bedeutung, da er festlegt, wie lange die Drohne in der Luft bleiben kann und welche Ausrüstung transportiert werden kann.

Außerdem können der Datenschutz und die Sicherheit durch die Edge Intelligence profitieren. Durch Trainingstechniken wie dem "Federated Learning" können die Modelle direkt

auf den Drohnen trainiert werden. Dadurch müssen sensible Daten der Sensoren und Kameras nicht an einen zentralen Server weitergeleitet werden, sondern nur die Modell-Updates (beispielsweise mittels Parameter-transfer) mit einem zentralen Modell synchronisiert werden. Dieses kann sich entweder auf einer anderen Drohne befinden oder auf einem Edge Server am Boden synchronisiert werden (siehe Abbildung 3.3).

Um den "communication overhead" weiter zu reduzieren, können Algorithmen entwickelt werden, welche die Updates der Modelle planen. Man nennt diesen Vorgang auch "aggregation frequency control". Die Reduktion des Datentransfers führt zu einem geringeren Energieverbrauch.

Auch die Edge Intelligence Technologie hat jedoch noch einige Herausforderungen zu meistern. Edge Nodes sind in ihren Ressourcen limitiert und beschränken dadurch auch die Machine Learning Algorithmen. Komplexe künstliche neuronale Netze liefern, wie in Kapitel 2.3.2 aufgezeigt, die besten Ergebnisse. Sie benötigen jedoch auch mehr Energie und Rechenleistung und die Latenz der Inferenz ist gegenüber einfacheren Modellen erhöht. Drohnen können nur eine beschränkte Menge an Hardware transportieren. Dennoch ist eine hohe Accuracy, eine Entscheidungsfindung in Echtzeit und ein geringer Energieverbrauch essenziell, um beispielsweise Kollisionskurse frühzeitig zu erkennen und zu vermeiden. Eine performante Edge Intelligence ist somit eine Grundvorraussetzung für autonome UAV-Systeme.

Eine weitere Herausforderung stellen Trainingsalgorithmen dar. Da die Modelle auf die Drohnen verteilt sind, die Daten sensibel sein können und der Datentransfer mit hohen Energie- und Netzwerkskosten einher geht, müssen Algorithmen entwickelt werden, die möglichst effizient, sicher und schnell alle Modelle auf dem neuesten Stand halten (vgl. [MWL22], S. 15438ff). In den folgenden Kapiteln werden die Herausforderungen der Edge Intelligence Technologie genauer betrachtet und Lösungsansätze verwandter Arbeiten diskutiert.

3.1 limitierte Ressourcen und Modellkompression

Es gibt mehrere Möglichkeiten, um das Problem limitierter Hardware-Kapazitäten zu umgehen. Beispielsweise können Modelle parallelisiert werden, Hardware kann auf bestimmte Verwendungszwecke spezialisiert werden (Stichtwort AI-Chips) oder die Komplexität und Größe der neuronalen Netzwerke kann reduziert werden und so Ressourcen eingespart werden. Da einfache Machine Learning Systeme oftmals schlechter performen, wurden Modell-Kompressions-Techniken entwickelt, welche performante Modelle in kleinere und sparsamere aber immer noch performante neuronale Netze konvertieren (vgl. [MWL22], S. 15438).

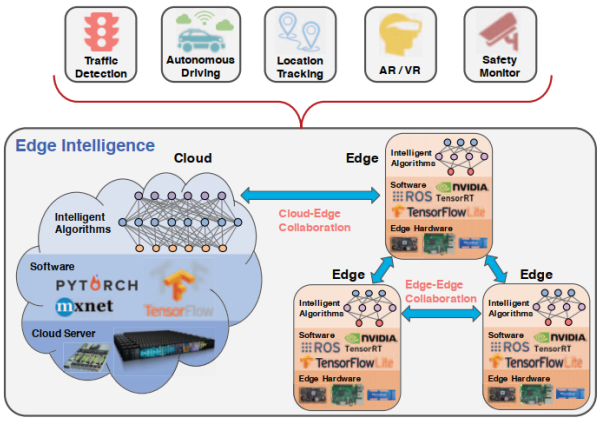
Die Quantisierung, als erstes Beispiel eines solchen Komprimierungs-Algorithmus, reduziert die Komplexität eines neuronalen Netzes dadurch, dass die Wertebereiche der Modell-Parameter wie Gewichte, Aktivierungsfunktionen oder Kernels verkleinert werden. Es ist vergleichbar mit dem Sampling eines kontinuierlichen Signals. Um eine Schallwelle digitalisieren zu können, muss das Signal in regelmäßigen Abständen abgetastet werden. Dadurch kann eine gute Repräsentation des Originalsignals erreicht werden. Beim Sampling wird also die Zeitachse eines Signals diskretisiert. Die Quantisierung betrachtet nicht die zeitliche

Achse, sondern die Amplitude (vgl. [WKL96], S. 353). Anstatt 32-Bit-Gleitkommazahlen werden für die Inferenz des neuronalen Netzwerks beispielsweise nur noch 8-Bit ganze Zahlen verwendet. Dadurch werden die numerischen Einheiten der Modell-Parameter kompakter, aber auch ungenauer (vgl. [SGKW21], S. 1). Die Quantisierung kann in "quantization-aware training" (QAT) und "post-training quantization" (PTQ) Techniken unterteilt werden. QAT ist präziser und führt zu besseren Ergebnissen, kann jedoch nicht für jeden Task angewendet werden. Das trifft vor allem auf Anwendungsfälle zu, bei denen die Trainingsdaten sensibel sind oder nicht erhoben werden können. Auf die PTQ trifft dies nicht zu. Außerdem ist die Anwendung der PTQ in der Praxis einfacher, da kein Re-Training von Nöten ist. Post-Training-Quantisierung wird deswegen trotz daraus resultierender signifikant schlechterer Genauigkeit für 4- und weniger Bit-Algorithmen immer beliebter.

Jüngste Entwicklungen haben eine performante Post-Training-Quantisierung auf 4-Bit Integer Modelle ermöglicht. I. Hubara et al. haben in ihrem Artikel "Accurate Post Training Quantization With Small Calibration Sets" 3 Optimierungsmethoden der PTQ vorgestellt, welche 4-Bit-Modellen eine hohe Accuracy erlaubt (vgl. [HNH+21], S. 4466ff). Y. Nahshan et al. haben in ihrem Artikel "Loss aware post-training quantization" eine Methode vorgestellt, welche schichtweise alle quantisierbaren Parameter betrachtet und die Quantisierung anhand von "loss functions" optimiert. Dadurch kann in manchen Modellen eine mit 4-Bit-Quantisierung vergleichbare Performance erreicht werden und gleichzeitig eine Accuracy auf vergleichbarem Niveau des originalen Modells erhalten bleiben. Diese Methode benötigt zudem keine spezielle Hardware zur Quantisierung, wie es bei der ein oder anderen PTQ der Fall ist (vgl. [NCB+21], S. 3245ff).

"Pruning" beschreibt eine weitere Methode zur Komprimierung von Deep Learning Algorithmen. Dabei werden Redundanzen im Modell ausfindig gemacht und eliminiert, wodurch die Größe des Modells abnimmt, und die Effizienz erhöht wird. Mögliche Redundanzen stellen Parameter im Modell dar, welche den Wert 0 haben und somit für die Inferenz nicht von Bedeutung sind (vgl. [GOX20], S. 1508). In "To prune, or not to prune: exploring the efficacy of pruning for model compression" haben M. Zhu und S. Gupta die Wirksamkeit von Pruning überprüft. Dabei haben sie große neuronale Netzwerke unterschiedlicher Architekturen trainiert und danach mittels Pruning komprimiert. Die Modelle wurden dann einfacheren Modellen, welche denselben Task erlernten und nach dem Pruning der komplexeren Modelle eine vergleichbare Größe aufwiesen, gegenübergestellt und die Effektivität des Prunings analysiert. Die komprimierten neuronalen Netzwerke, egal welche zugrunde liegende Netzwerkstruktur diese hatten, übertrafen die weniger komplexen Modelle und konnten somit die Effektivität dieser Kompressionsmethodik bestätigen (vgl. [ZG17], S. 1ff). J. Guo et al. haben ein einheitliches Framework für Pruning vorgestellt, welches signifikante Einsparungen bei der Anwendung von multidimensionalen künstlichen neuronalen Netzwerken verspricht (vgl. [GOX20], S. 1508ff).

Als 3. Beispiel einer Kompressionstechnik künstlicher neuronaler Netze beleuchten wir die "low rank factorization". Wie beim Pruning wird bei der Faktorisierung versucht Redundanzen innerhalb eines Modells zu reduzieren. Im Gegensatz zum Pruning betrachtet die "low rank factorization" jedoch nicht die einzelnen Parameter, sondern viel mehr die schichtweisen Operationen des neuronalen Netzwerkes. Speziell die Faltungsoperationen



Quelle: [ZWL+19, S. 1841]

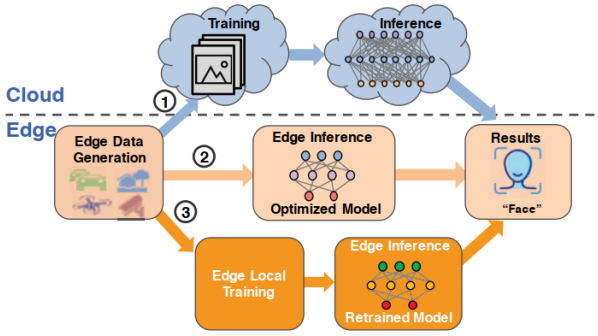
Abbildung 3.1: Auf Grund der unterschiedlichen Edge Hardware müssen neue Software Frameworks entwickelt werden, welche die Intergration der künstlichen Intelligenz in die Edge vereinfacht.

des Convolutional Layers von CNNs sind sehr rechenintensiv. Die Faktorisierung kann hier beispielsweise die Dimensionen der Filter auf wenige Faktoren reduzieren und so einen signifikanten Speed-Up bei fast gleichbleibender Accuracy erreichen. Die Faktorisierung ist komplex und rechenintensiv und erfordert ein Re-Training des Modells, was diese Art der Komprimierung für die meisten Edge Computing Anwendungen unterinteressant macht (vgl. [CWZZ18], S. 129f).

Nicht nur die limitierte Rechenleistung der Endgeräte erschwert die Entwicklung der EI. Auch die Vielfältigkeit eingesetzter Hardware und die Verteilung der Daten führt zu Problemen. X. Zhang et al. haben in ihrem Artikel “OpenEI: An Open Framework for Edge Intelligence” ein Framework vorgestellt, welches die Entwicklung und Einbindung einer Edge Intelligence in diverse Edge Devices vereinfachen soll. Auch Google bietet inzwischen Tensorflow Lite an. Diese Bibliothek soll die Integration von Machine Learning Algorithmen in das Edge Computing Paradigma vereinfachen. Nichtsdestotrotz stellen die unterschiedlichen Anforderungen der Edge Devices und der Modelle die Entwickler*innen immer noch vor Herausforderungen (vgl. [ZWL+19], S. 1840ff). Abbildung 3.1 visualisiert mögliche Anwendungsfälle einer EI und bildet die Zusammenarbeit der einzelnen Komponenten (Hardware, Software) in der Cloud ab.

3.2 Datenaustausch und Learning Algorithmen

Nicht nur die Hardware der Edge Nodes und Architektur der Machine Learning Algorithmen können die Edge Intelligence Technologie limitieren. Besonders rechenintensiv ist im Deep Leaning das Trainieren der Modelle und besonders ressourcenaufwändig ist im Edge Computing der Datentransfer. Die Edge Intelligence erfordert neue Trainingsalgorithmen und Kommunikationsprotokolle, um diese Probleme zu lösen. In diesem Kapital werden wir auf mögliche Lösungsansätze eingehen.



Quelle: [ZWL+19, S. 1842]

Abbildung 3.2: Diese Abbildung visualisiert die 3 unterschiedlichen Möglichkeiten der Integration eines Machine Learning Systems in die Edge.

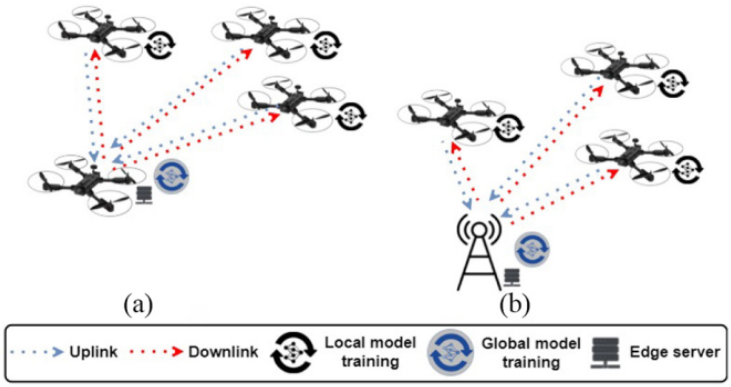
Wie in "OpenEI: An Open Framework for Edge Intelligence" von X. Zhang et al. aufgezeigt wird, gibt es 3 Möglichkeiten, wie die Datenströme innerhalb des Edge Computing Paradigmas fließen können (siehe Abbildung 3.2 durch die Nummern gekennzeichnet):

1) Die Daten jedes einzelnen Edge Devices werden an einen Cloud-Server weitergeleitet und ein zentrales Modell mit diesen Daten trainiert. Das Modell in der Cloud führt dann die Inferenz durch und leitet die Ergebnisse an die Endgeräte weiter. Diese Vorgehensweise ist weitverbreitet.

2) Als zweites können die Daten direkt auf den Edge Devices mittels Machine Learning verarbeitet werden. Dazu wird das Modell, welches zuvor in der Cloud trainiert wurde, auf die Edge Node geladen und dort die Inferenz durchgeführt.

3) Bei der 3. und letzten Möglichkeit wird nicht nur die Inferenz lokal durchgeführt, sondern das Modell auch lokal trainiert. Das Modell wird lokal re-trainiert und dadurch optimiert. Erst danach wird die Inferenz ausgeführt. Dadurch können Modelle im Edge Computing Paradigma trotz sensibler Daten an ihren Anwendungsfall optimiert werden (vgl. [ZWL+19], S. 1842f). Man nennt diese Vorgehensweise auch dezentrales Machine Learning.

Im Beispiel des Drohnensystems aus "A Survey on the Convergence of Edge Computing and AI for UAVs: Opportunities and Challenges" von P. McEnroe et al. wurde Federated Learning als möglicher Lösungsansatz einer dezentralen Modell-Trainingsmethode angesprochen. Im Gegensatz zu den herkömmlichen Methoden, bei denen die Daten an einem zentralen Punkt aggregiert werden und ein Modell trainiert wird, wird ein und dasselbe Modell auf allen Edge Devices, in dem Fall allen Drohnen, ausgeführt und durch die lokal aufgenommenen Daten trainiert. Als nächstes werden die Updates aller lokalen Modelle an einen Server weitergeleitet und dort ein zentrales/globales Modell aktualisiert. Es handelt sich also auch um eine Form des Tranfer Learnings, da das neu generierte Wissen der lokalen Modelle an das globale Modell weitergeleitet wird. Dieses Modell wird nach der Aktualisierung wiederum an alle UAVs gesendet und so die Updates aller lokalen Modelle synchronisiert. In Abbildung 3.3 wird dieser Vorgang bildlich dargestellt.



Quelle: [MWL22, S. 15439]

Abbildung 3.3: Der Server, welcher von den verteilten Modellen angesprochen wird, kann sich sowohl auf einer Drohne, als auch am Boden befinden. Auf ihm wird das globale Modell aktualisiert und das zentrale Update an die einzelnen Drohnen gesendet.

Diese Methode des Learnings erhöht die Sicherheit und den Datenschutz einer Edge Intelligence, da keine Rohdaten, sondern nur die Updates der Modelle im Netzwerk transferiert werden. Zudem reduziert Federated Learning die Latenz und den Communication Overhead des Internets. Besonders in Bezug auf UAVs, bei denen Latenz und eine hohe Netzwerkauslastung zu Abstürzen führen können, kann Federated Learning somit ein Schlüsselelement werden (vgl. [MWL22], S. 15438ff). Außerdem reduziert Federated Learning die Zeit, welche für das Trainieren der Modelle und deren Inferenz benötigt wird, signifikant. Der Artikel "Federated Learning in Edge Computing: A Systematic Survey" von H. G. Abreha et al. wirft einen Blick auf den momentanen Stand der Entwicklung und zeigt anhand von aktuellen wissenschaftlichen Arbeiten Implementierungsmöglichkeiten, Vorteile aber auch Nachteile und Herausforderungen dieser Technologie auf (vgl. [AHS22], S. 3ff).

Auch Transfer Learning kann dabei helfen die Edge Intelligence voranzutreiben. Wie im Transfer Learning Kapitel (2.3.1) bereits erwähnt wurde, ist die Erhebung der Daten manchmal sehr kostspielig oder gar nicht möglich. Transfer Learning reduziert den Trainingsaufwand und liefert oftmals vergleichbare, wenn nicht bessere Ergebnisse, da der Generalisierungsfehler durch den Wissenstransfer minimiert werden kann (vgl. [PY10], S. 1ff). In diesen Fällen kann Transfer Learning EI unterstützen. Auch im Federated Learning findet Transfer Learning Anwendung und ist oftmals für den Wissenstransfer und das Training des zentralen Modells zuständig (vgl. [AHS22], S. 3ff). L. Valerio et al. haben in ihrem Artikel "Hypothesis Transfer Learning for Efficient Data Computing in Smart Cities Environments" ein Transfer Learning Framework für dezentrale Machine Learning Algorithmen vorgestellt. Das Framework soll wie beim Federated Learning, die Modelle lokal trainieren, nur die Modell-Updates transferieren und so die Auslastung des Netzwerkes minimieren (vgl. [VPC16], S. 1ff).

Aber nicht nur Trainingsalgorithmen müssen an das Edge Computing Paradigma angepasst werden. Um die Kommunikation zwischen den Edge Nodes so effizient wie möglich zu gestalten, sind auch optimierte Datentransfer-Algorithmen von Nöten. Wie A. Aral et al. in ihrem Artikel "Staleness Control for Edge Data Analytics" festhalten, wurden heutige sogenannte "EDA" Techniken nicht für den Datenaustausch verteilter Machine Learning Modelle in nahezu Echtzeit entwickelt. Die bestehenden Techniken berücksichtigen somit keine dynamischen Synchronisationsperioden. Damit die Modelle durchgängig richtige Entscheidungen treffen können, sind derartige Synchronisationsperioden jedoch essenziell. Der von A. Aral et al. entwickelte Lösungsvorschlag nennt sich "Staleness Control for Edge Data Analytics" (SCEDA). SCEDA beschreibt einen auf Reinforcement Learning basierten Algorithmus. Dieser lernt, die Dringlichkeit von Edge Node Updates zu erkennen und ermöglicht dadurch eine effiziente dynamische Synchronization der Modelle (vgl. [AEKB20], S. 2ff).

4 Methodik, Daten und Implementierung

In dieser Bachelorarbeit wollen wir nun die Effizienz und Effektivität induktiven Transfer Learnings in Bezug auf das Edge Computing Paradigma prüfen. Dazu wurde die Inferenz zweier unterschiedlicher Modelle auf 2 verschiedenen Edge Devices - einer Coral Edge TPU, welche 8-Bit-Operationen unterstützt, und einem Raspberry Pi 3+, welches für 16- und 32-Bit-Berechnungen verwendet wurde - ausgeführt und die Inferenz-Geschwindigkeit, der Energieverbrauch und die Accuracy gemessen bzw. verglichen. Das erste Modell wurde bereits auf Bilderkennungstasks trainiert und per Transfer Learning mittels Parametertransfer auf unseren Anwendungsfall optimiert. Das zweite künstliche neuronale Netzwerk wurde neu modelliert und trainiert. Wie im Kapitel "Edge Intelligence: Verwandte Arbeiten" (3) bereits erwähnt wurde, soll der Datentransfer zwischen den Edge Nodes auf Grund des Energieverbrauchs, der Netzwerkauslastung und der Sicherheits/Datenschutz-Problematik minimiert werden. Das führt dazu, dass die Erhebung von Daten sehr kostspielig oder manchmal gar unmöglich ist. Für die Simulation eines solchen Anwendungsfalles wurde der Datensatz auf 1200 Bilder limitiert. Der Task der beiden Modelle lautet "Erkennung von Autokennzeichen".

Für die Implementierung wurde auf Python und Jupyter Notebook zurückgegriffen. Der Code wurde in einem GitHub Repository gespeichert und kann unter folgendem Link eingesehen werden: [Git].

In GitHub sind zudem Python-Scripts zu finden. "util.py" diente als Unterstützung für den Download der Bilder und die grafische Darstellung des Modell-Trainings. Die übrigen 3 Python-Dateien "evaluate_quantized_models.py", "execute_models.py" und "execute_models_on_tpu.py" wurden für die Inferenz der Modelle auf den Edge Devices implementiert. Das Transfer Learning wurde auf Basis der Model Maker Bibliothek von TensorFlow Lite durchgeführt (siehe [TFM]). Für die Modellierung des CNNs wurde Keras (siehe [Ker]) herangezogen. Die Modelle können wie die Resultate der Inferenz-Messungen in GitHub eingesehen werden.

4.1 Beschreibung des Datensatzes

Für die Bilddaten wurde auf die Open Images V6 Datenbank zurückgegriffen (siehe [OID]). Es wurden insgesamt 1200 Bilder heruntergeladen, welche in 1000 Trainingsbilder und jeweils 100 Test- und Validierungsbilder unterteilt wurden. 50% der Bilder beinhalten Autokennzeichen. Um zu verhindern, dass die Modelle die Bilder anhand falscher Eigenschaften detektieren, also "biased" werden (siehe Kapitel 2.3.1), wurden die Bilder der anderen Hälfte "kein Autokennzeichen" aus mehreren unterschiedlichen Klassen gebildet (Äpfel, Katzen und Hunde). Für den Download wurde die von Google empfohlene Bibliothek "fiftyone" (siehe [501a, 501b]) verwendet. Die Daten beinhalten bereits eine korrekte Zu-

weisung zu den zugehörigen Klassen, sodass die Labels nur noch mittels Python-Code extrahiert werden mussten.

Induktives Transfer Learning erfordert einen kleinen Datensatz des Zieltasks, um das Modell an die neue Aufgabenstellung anzupassen. Die Bilddaten mussten den Anforderungen des Model Maker Frameworks, welches für das Transfer Learning verwendet wurde, angepasst werden. EfficientNet-Lite-Modelle akzeptieren Bilder im Format 300 zu 300 Pixel. Zudem wurden die Bildwerte von einem 0-255 ganzzahligen Wertebereich in Gleitkommazahlen zwischen 0-1 skaliert. Um die Daten an das Modell füttern zu können, mussten die Bilder und Labels in das Bibliotheks interne Format der DataLoader-Klasse transformiert werden. Dazu wurde eine eigene Funktion "get_data_in_DataLoader_format" geschrieben, welche sich an den GitHub-Code der DataLoader-Klasse orientiert (siehe [MMD]). Um die Modelle möglichst gut vergleichen zu können wurden die Daten für das Training des neu modellierten neuronalen Netzwerks an das 300x300 Pixel Format der EfficientNet-Lite-Modelle angepasst und ebenfalls in den Zahlenraum 0-1 skaliert.

Als letzten Schritt wurden die Daten mit einem Data Augmentation Algorithmus erweitert. Wie bereits im Kapitel "Deep Learning und CNNs" (2.3.2) erwähnt wurde, sind Data Augmentation Algorithmen eine Möglichkeit, um das Problem der limitierten verfügbaren Datenmenge zu minimieren. Die Daten werden mittels Transformationen verändert. So sinkt das Risiko von Overfitting, selbst wenn Bilder mehrfach für das Training herangezogen werden. Model Maker bietet eine eingebaute Funktion für die Erweiterung der Daten an. Damit die Ergebnisse der Modelle möglichst vergleichbar bleiben, wurde jedoch ein eigener Algorithmus entwickelt, welcher auf dem Preprocessing-Layer der Keras-Bibliothek aufbaut (siehe [Dat]). So kann sichergestellt werden, dass die Trainingsdaten durch dieselben Transformationen variiert werden.

4.2 Bewertungsmetriken zur Analyse der Effektivität und Effizienz

Für den Vergleich der beiden Modelle werden folgende Metriken herangezogen: die Accuracy, der Energieverbrauch und die Latenz der Inferenz. Wie in "Edge Intelligence: Verwandte Arbeiten" (3) beschrieben wurde, ist sowohl der Energieverbrauch als auch die Ausführungsgeschwindigkeit im Edge Computing Paradigma von großer Bedeutung. Manche Anwendungsfälle erfordern eine Inferenz in beinahe Echtzeit. Zudem spielt der Energieverbrauch gerade in Bezug auf schwer zugängliche IoT-Geräte, welche über Batterien betrieben werden, eine wichtige Rolle. Diese Metriken dürfen aber, wie in "Edge Intelligence: Verwandte Arbeiten" aufgezeigt wurde, nicht ohne die Genauigkeit eines neuronalen Netzes betrachtet werden. Je komplexer ein Modell ist, desto höher ist seine Trefferquote und desto geringer der Generalisierungsfehler. Mit der Größe und Anzahl der Parameter eines CNNs geht jedoch auch ein erhöhter Energieverbrauch und eine höhere Latenz der Inferenz einher, da mehr Berechnungen innerhalb des Modells notwendig sind, um zu einem Ergebnis zu kommen. Aus diesem Grund wird zusätzlich die Accuracy betrachtet und in Relation mit den 2 anderen Größen gesetzt.

4.3 Strukturelle Performance-Analyse des Transfer Learning Modells

Die Model Maker Bibliothek von TensorFlow Lite wendet Transfer Learning standardmäßig auf das EfficientNet-Lite-0-Modell an. Grundsätzlich gibt es jedoch die Möglichkeit zwischen 5 unterschiedlichen EfficientNet-Lite-Modellen auszuwählen und eigene Modelle können in das Transfer Learning Framework von Model Maker eingebunden werden (siehe [TFM]). Die EfficientNet-Lite-Modell-Serie ist bereits für Edge Computing optimiert und bietet ein gutes Fundament für den Anwendungsfall dieser Bachelorarbeit. Um eine möglichst hohe Accuracy zu erhalten, wurde für das Transfer Learning EfficientNet-Lite-4 ausgewählt (siehe [Effb, Effa]).

Für das Training wurden die Hyperparameter wie folgt eingestellt: Die Größe der Epochen wurde auf 30 gesetzt. Eine Epoche gibt an, wie oft der gesamte Trainingsdatensatz für das Training herangezogen wird. Ein Wert von 30 bedeutet somit, dass das CNN zu jedem Bild 30-mal Prognosen tätigt, mit den Labels abgleicht und daraus lernt. Je öfters die Trainingsbilder betrachtet werden, desto höher ist die Wahrscheinlichkeit, dass das Modell auf den Datensatz spezialisiert wird und der Generalisierungsfehler steigt. Nachdem Data Augmentation implementiert wurde, wird das Bild bei jedem Durchlauf den definierten Transformationen entsprechend angepasst und verändert, wodurch das Overfitting-Risiko sinkt. Für die "batch_size" wurde ebenfalls ein Wert von 30 festgelegt. Die Batch-Größe gibt an, wie viele Bilder betrachtet werden, bevor die Parameter des Netzwerkes angepasst werden. Für das Training wurden also die Ergebnisse von 30 Bildern in einem Lernschritt evaluiert. Weitere Parameter für das Transfer Learning des EfficientNet-Lite-4-CNNs waren ein Trainingsdatensatz, ein Validierungsdatensatz um mögliches Overfitting zuerkennen, "train_whole_model", "shuffle" und "use_augmentation" (siehe [hyp]). Im Daten-Vorverarbeitungsschritt wurden die Daten bereits gemischt, somit konnte der "shuffle" Parameter ignoriert werden. "use_augmentation" aktiviert den Model Maker internen Data Augmentation Algorithmus, weshalb dieser ebenfalls auf "false" gesetzt wurde. Beim Setzen des Parameters "train_whole_model" auf "true" würde das gesamte EfficientNet-Lite-4-System an den neuen Task angepasst werden. Wird stattdessen "false" übergeben, werden nur die letzten Layer des CNNs angepasst, welche für die Klassifizierung der Bilder zuständig sind. Für den ersten Trainingsversuch wurde diese Funktion ausgeschaltet.

Beim ersten Versuch des Modell-Trainings konnte festgestellt werden, dass bei der Validierung ab der 10. Epoche keine signifikant besseren Ergebnisse erzielt wurden, während die Trefferquote in Bezug auf die Trainingsdaten weiter anstieg. Dies ist ein Indiz dafür, dass ein Epochenwert größer 10 zu Overfitting des Modells führt. Die Evaluierung der Testdaten ergab eine Accuracy von 89%. Nachdem EfficientNet-Lite-4 innerhalb der ersten 3-4 Epochen bereits eine Accuracy von 89% aufwies deutet dies daraufhin, dass das gesamte Modell für das Re-Training angepasst werden muss, um bessere Ergebnisse erzielen zu können.

Für die Optimierung des Modells wurden somit folgende Hyperparameter angepasst und das Training erneut durchgeführt: Die Epochen wurden auf 10 herabgesetzt und die "train_whole_model" Funktion wurde aktiviert. Die Evaluierung der Testdaten führte zu einem Ergebnis von 94% Accuracy.

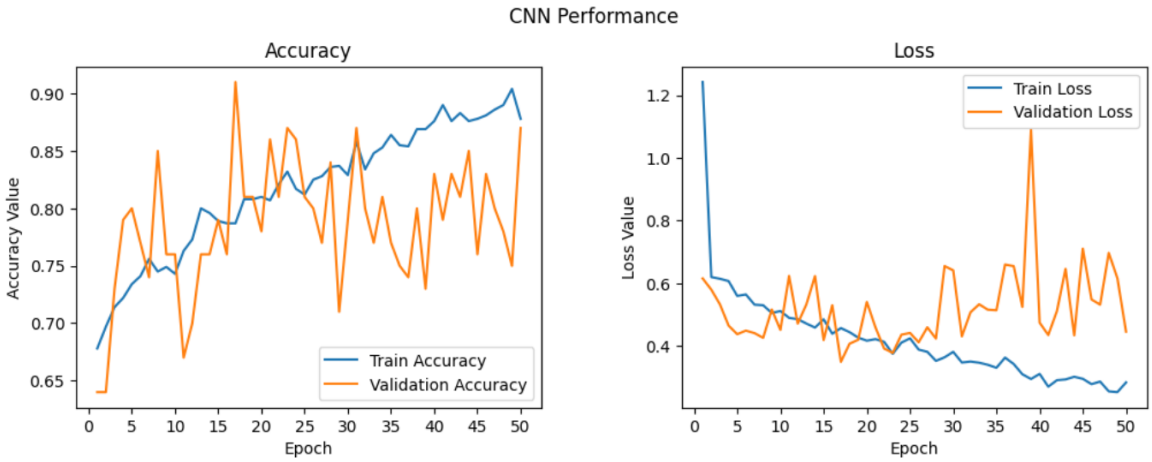


Abbildung 4.1: Die linke Grafik stellt die Genauigkeit des Modells bezogen auf die Epochen und die Trainings- bzw. Validierungsdaten des ersten Trainingsversuchs dar. Auf der rechten Seite ist die Fehlerrate des Trainings bzw. der Validierung zu sehen. Aus diesen Grafiken lässt sich eine Performance des CNNs ableiten.

4.4 Strukturelle Performance-Analyse des Keras Modells

Als Grundlage für die Implementierung wurde die Bibliothek Keras verwendet (siehe [Ker]). Als Inputformat wurde dasselbe Format wie beim EfficientNet-Lite-4-Modell gesetzt. Wie im Kapitel "Deep Learning und CNNs" (2.3.2) erläutert wurde, bestehen Convolutional Neural Networks primär aus Convolution Layers gefolgt von Pooling Layers. Für das erste Training des neuronalen Netzwerks wurde zuerst eine einfache Architektur gewählt bestehend aus 4 Convolution Layer jeweils gefolgt von 1 MaxPooling Layer. Ein Flatten Layer (siehe [fla]) reduziert die Dimensionen der letzten Ausgabe und bringt diese in das richtige Format. Für die Klassifizierung der Bilder sind 2 Dense Layer zuständig (siehe [den]). Diese sind vergleichbar mit Fully-Connected Layern. Das Modell wurde als Binary Cross (siehe [bin]) Modell kompiliert und somit für Klassifizierungstasks mit 2 Klassen ausgelegt.

Das erste Training des Modells wurde mit einer unveränderten "batch_size" von 30 und einem Epochenwert von 50 durchgeführt. Die Epoche wurde hoch angesetzt, um das Verhalten des Modells analysieren zu können. Wie an der Trainingshistorie (siehe Abbildung 4.1) erkannt werden kann, erhöht sich die Genauigkeit des CNNs bei der Validierung ab der 32. Epoche nicht, während die Trainings-Accuracy weiter zunimmt. Dies deutet auf ein erhöhtes Risiko für Overfitting ab Epoche 32 hin. Diese These wird auch von der Error-Rate unterstützt (im Bild "Loss"). Die Fehlerquote des Trainingsdatensatzes verringert sich kontinuierlich, während sich die Fehlerrate der Validierung von der des Trainings bereits ab der 25 Epoche abhebt. Insgesamt konnte bei der Validierung ein Wert von 87% Accuracy erreicht werden.

Um den Generalisierungsfehler des Modells zu minimieren, wurden einige Umstrukturierungsversuche der Modell-Architektur unternommen (siehe Abbildung 4.2 und 4.3).

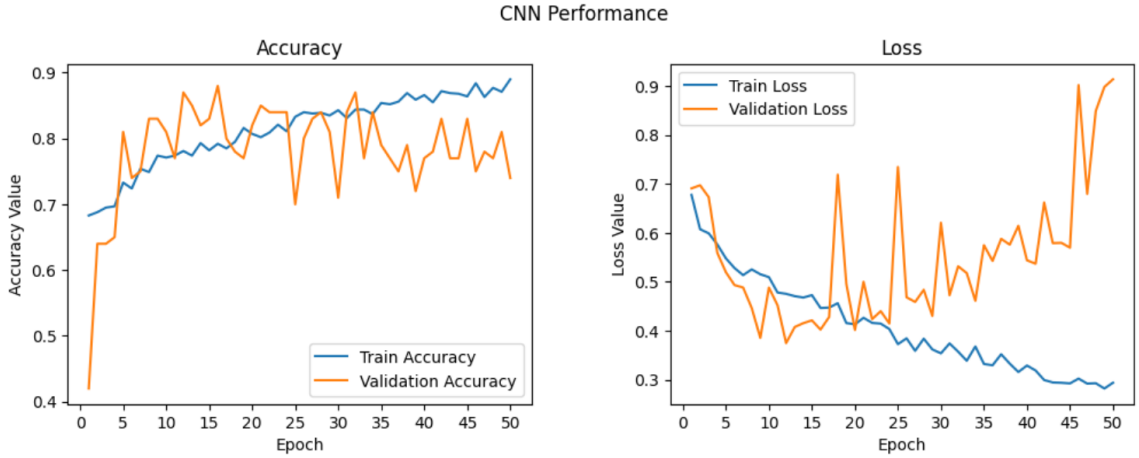


Abbildung 4.2: Ein zusätzlicher Convolution Layer mit derselben Anzahl an Ausgabewerte (128) wie zuvor, führte ebenfalls zu Overfitting, wenngleich die allgemeine Performance gegenüber dem ersten Modell anstieg und vor allem keine so starke Schwankungsbreite aufwies.

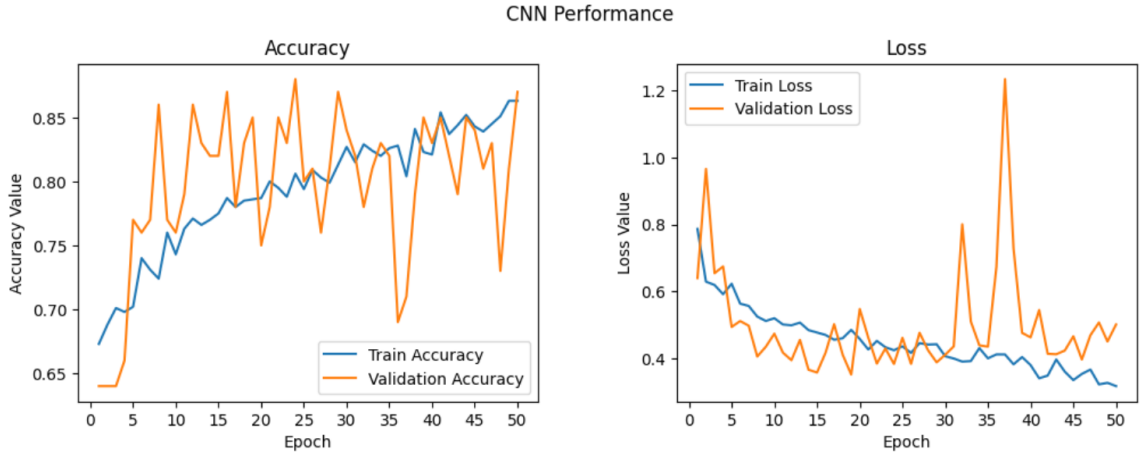


Abbildung 4.3: In diesem Beispiel wurde statt eines 128 Filter Outputs der zusätzlichen Faltungsschicht ein 256-Filter-Output gewählt, die Ausgabe also verdoppelt. Dadurch konnte, wie durch die Validierungsgenauigkeit und dem Validation-Loss ersichtlich wird, der Generalisierungsfehler gesenkt werden. Anhand der Trainingshistorie kann zudem erkannt werden, dass bei einer Epochen Anzahl von 35 das Overfitting-Risiko des Modells gering ist.

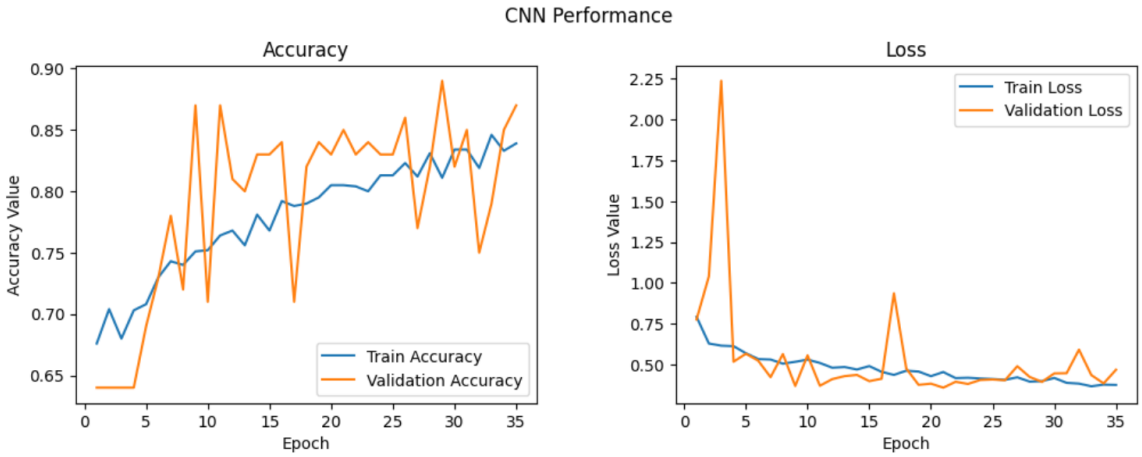


Abbildung 4.4: Die Trainingshistorie des finalen Keras-Modells

Das optimierte Modell wurde mit einen zusätzlichen Convolution Layer mit einer Output-Größe von 256 Filtern erweitert und der Epochenwert auf 35 gesenkt. Das Training des Modells resultierte ebenfalls in einer Validation-Accuracy von 87% jedoch konnte der Generalisierungsfehler minimiert werden und so das Modell insgesamt eine bessere Performance erreichen (siehe Abbildung 4.4).

4.5 Quantisierung

Die trainierten Modelle haben nun gelernt, Autokennzeichen in Bildern zu detektieren. Um sie für die Ausführung auf den Edge Devices (Coral Edge TPU und Raspberry Pi 3+) vorzubereiten, mussten die Modelle an die Hardware der Endgeräte angepasst werden und komprimiert werden. Wie im Kapitel "limitierte Ressourcen und Modellkompression" (3.1) beschrieben wurde, kann ein künstliches neuronales Netz auf unterschiedliche Arten komprimiert werden. Eine der gängigsten Methoden ist die Post-Training-Quantisierung (PTQ). Sie erfordert kein Re-Training des Modells und nur einen sehr kleinen Kalibrierungsdatensatz, um die Wertebereiche des Modells bestmöglich abzutasten. Nachdem TensorFlow Lite eine integrierte Funktion für die PTQ anbietet, wurde auf diese Methode zurückgegriffen (siehe [tf_a, tf_b]).

Auch die Exportfunktion der Model Maker Bibliothek bietet die Option der Post-Training-Quantisierung an. Für die Komprimierung musste nur eine "quantization_config" Variable mit den gewünschten Bit-Bereichen und ein Kalibrierungsdatensatz angegeben werden (siehe [mm]). Die Modelle werden automatisch in das TensorFlow Lite interne Format "tflite" formatiert, welches mit den meisten Edge Devices kompatibel ist.

Für das benutzerdefinierte CNN mussten TFLiteConverter Objekte definiert werden, welche das Modell quantisiert und in das gewünschte Format bringen. Zudem war für die 8-Bit-Quantisierung ein Kalibrierungsdatensatz notwendig.

Für die Messungen und Evaluierung der Modelle wurden diese in 3 unterschiedliche Wertebereiche quantisiert: 32-Bit-Float-Zahlen, 16-Bit-Float und 8-Bit-Integer.

5 Analyse der Messungen

Um die Messungen durchführen zu können wurden Python Scripts geschrieben, welche auf das Raspberry Pi 3+ geladen wurden und mittels Bash-Befehl ausgeführt wurden. Die 32- bzw. 16-Bit-Modelle wurden auf dem Pi getestet, während die Inferenz der 8-Bit-Modelle auf eine Coral Edge TPU (siehe [cora]) delegiert wurde. Für die Inferenz der 8-Bit-Modelle war es erforderlich, die Modelle an die Systemanforderungen (siehe [corb]) anzupassen und neu zu kompilieren. Zudem musste ein eigenes Script geschrieben werden, da für die Ausführung auf die API der PyCoral-Bibliothek (siehe [pyc]) zurückgegriffen werden musste. Für die Inferenz wurden 100 Bilder der Testdatenbank Open Images V6 heruntergeladen.

5.1 Latenz

Die Latenz beschreibt die Dauer der Inferenz eines Bildes, also die Zeit, welche für die Prognose eines einzelnen Bildes benötigt wird. Die Inferenzgeschwindigkeit ist abhängig von der Anzahl an Operationen und somit indirekt von der Größe eines CNNs. Während das Keras Modell eine einfache Struktur mit 5 Faltungs-Schichten und insgesamt 6.816.161 Parametern aufweist, besteht das EfficientNet-Lite-4-Modell aus insgesamt 11.840.498 Parametern. Somit war zu erwarten, dass die Keras-Modelle schneller sein werden. Diese Erwartung wurde von den Messungen bestätigt. Mit 1,627 (32 Bit), 1,55 (16 Bit) und 1,423 Sekunden (8-Bit) pro Bild waren die EfficientNet-Modelle deutlich langsamer als die neu entwickelten Modelle der Keras-Bibliothek, welche eine Latenz von 0,193 (32-Bit), 0,18 (16-Bit) und 0,227 (8-Bit) Sekunden aufwiesen (siehe Abbildung 5.1). Auch eine Beschleunigung der Modelle durch die Post-Training-Quantisierung konnte durch die verringerte Laufzeit der 16- bzw. 8-Bit-Modelle bestätigt werden.

5.2 Energieverbrauch

Um den Energieverbrauch der Modelle während der Inferenz zu messen, wurden die Edge Devices an einen Stromzähler angeschlossen und die Pymeas-Bibliothek in die Scripte integriert. Pymeas stellt eine Verbindung zum Stromzähler her, nimmt für ungefähr eine Sekunde den Stromverbrauch des gesamten Edge Devices auf und errechnet daraus einen Durchschnitt des in der Sekunde verbrauchten Stroms in Wattstunden. Die Bibliothek tastet somit Schritt für Schritt den Energieverbrauch ab und speichert die Ergebnisse in einer Datei ab.

Die Messergebnisse der EfficientNet-Lite-Modelle (siehe Abbildung 5.2) zeigen deutlich, dass auch der Energieverbrauch stark mit der Quantisierung und somit Modell-Größe korreliert. Der Schluss liegt nahe, dass ein größerer Wertebereich der Parameter zu komplexeren

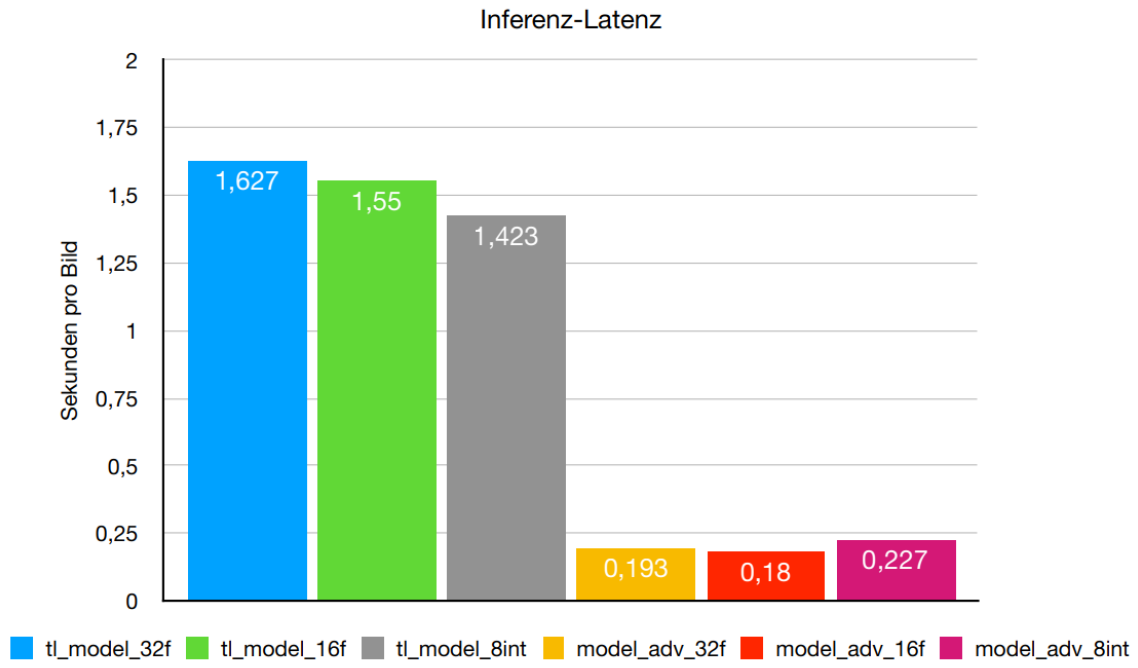


Abbildung 5.1: Die erhöhte Latenz der EfficientNet-Lite-4-Modelle ist auf die höhere Komplexität und Größe der Modelle zurückzuführen.

Berechnung und dadurch zu einer erhöhten Auslastung und Wärmeentwicklung der Computerchips der Edge Devices führt.

Im Gegensatz dazu weisen die Ergebnisse der Keras-Modelle keine solche Korrelation auf. Dies könnte erneut auf die Architektur der Modelle zurückzuführen sein, welche nicht wie die EfficientNet-Lite-Modell-Serie für das Edge Computing optimiert wurden. Des Weiteren handelt es sich um eine Momentaufnahme des Stromvebrauchs der Edge Devices. Diese könnte auch durch äußerliche Faktoren beeinflusst werden und die Ergebnisse verfälschen. So wurden die Messungen der Transfer Learning Modelle vor den Keras-Modellen ausgeführt, was zu einer allgemein erhöhten Wärmebildung während der Inferenz der Keras-Modelle führen kann. Beim Vergleich der 32-Bit-Modelle lässt sich jedoch feststellen, dass auch die Anzahl an Parametern eines CNNs in Bezug zum Energieverbrauch eine Rolle zu spielen scheint.

5.3 Genauigkeit

Die 32-Bit-Modelle konnten bei der Inferenz eine Genauigkeit von 92% für das EfficientNet-Lite-4-Modell und 80% für das neu modellierte CNN erreichen. Das entspricht einem Verlust von 2% respektive 7% gegenüber der Validierungsanalyse. Die 16-Bit-Modelle konnten ebenfalls 92% und 80% Accuracy erzielen und bestätigen die Beobachtung der 32-Bit-Modelle. Selbst die auf 8-Bit quantisierten Modelle konnten auf der Coral Edge TPU die

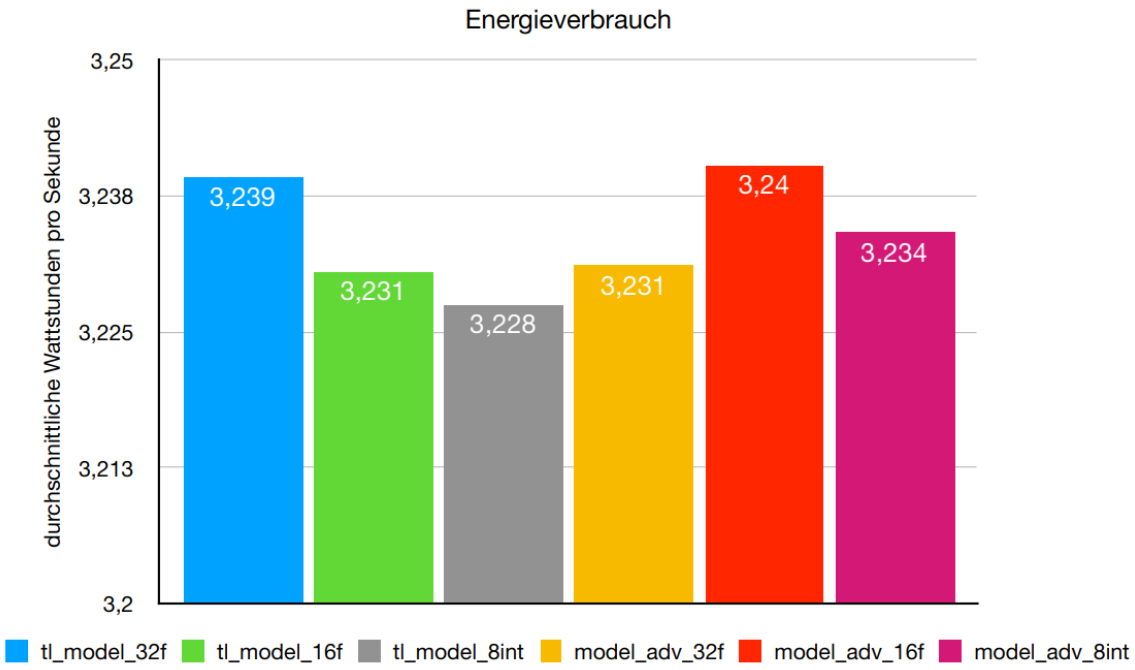


Abbildung 5.2: Für das Messergebnis wurde das arithmetische Mittel der Abtastungsergebnisse in Wattstunden berechnet.

Performance halten und 92% bzw. das von Grund auf trainierte CNN 81% erreichen. Der Umstand, dass die Modelle, welche mittels Transfer Learning trainiert wurden, einen deutlich geringeren Verlust der Accuracy im Vergleich zu der Trainingsanalyse verzeichnen, deutet daraufhin, dass Transfer Learning den Generalisierungsfehler reduzieren kann. Der allgemeine Leistungsunterschied zwischen den beiden Modellarchitekturen ist einerseits auf die unterschiedliche Architektur der CNNs und dadurch Komplexität und Anzahl an Parametern zurückzuführen, andererseits wurde EfficientNet-Lite-4 bereits auf Bilderkennung trainiert und kann somit mehr Wissen in den neuen Task miteinbringen.

Wie zuvor bereits erwähnt, deuten die einzelnen Ergebnisse daraufhin, dass die Latenz der Inferenz, der Energieverbrauch des Modells pro Sekunde und die Accuracy eines Modells stark von der Architektur und Größe des zugrunde liegenden neuronalen Netzes abhängen. Die Genauigkeit eines CNNs bildet im Gegensatz zu den anderen Größen jedoch eine positive Korrelation. Aus diesem Grund müssen die Metriken in Relation zueinander betrachtet werden, um eine finale Aussage über die Effizienz und Effektivität der Modelle tätigen zu können.

Der Energieverbrauch der Inferenz eines Modelles steht in Abhängigkeit zur insgesamt benötigten Zeit. Deshalb wurde, um die Messungen in Relation setzen zu können, der Energieverbrauch pro Bild berechnet, also die Latenz mit dem Energieverbrauch pro Sekunde multipliziert. Die Ergebnisse der Messungen (siehe Abbildung 5.3) zeigen deutlich auf, dass der Energieverbrauch in Korrelation mit der Accuracy eines Modells steht. Diese

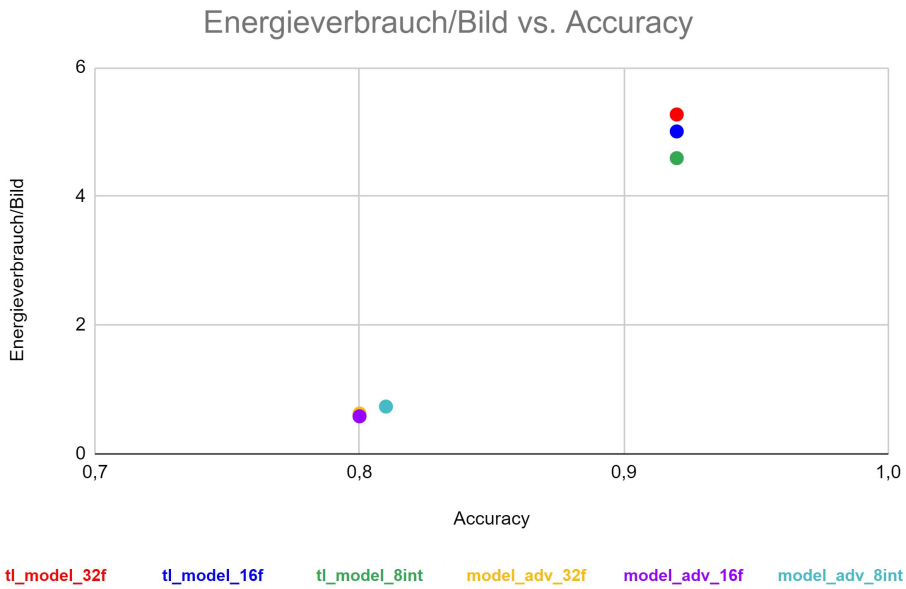


Abbildung 5.3: Die Modelle, welche eine höhere Genauigkeit besitzen, weisen einen erhöhten Energieverbrauch auf. Dies liegt and der komplexeren Architektur des CNNs, welche für eine höhere Accuracy notwendig ist. Um diesem Energieeffizienz- und Geschwindigkeitsverlust zu entgegnen, müssen die Techniken der Modellkompression optimiert werden.

Beobachtung bestätigt die Messungen und Forschungsergebnisse der im Kapital “Edge Intelligence: Verwandte Arbeiten” (3) referenzierten Artikel und unterstreicht die Relevanz von Modellkompressions-Techniken und der Methodik des Transfer Learnings für das Edge Computing Paradigma.

6 Fazit

Das Edge Computing Paradigma und die aufstrebende Technologie der künstlichen Intelligenz weisen viele Synergien auf. Während Machine Learning die Anwendungsmöglichkeiten des Edge Computing erweitert und dessen Effizienz steigert, kann die Edge Intelligence der Latenz einer zentralen Cloud basierten AI entgegenwirken und sowohl die Sicherheit als auch den Datenschutz steigern (siehe Kapitel 3). Dezentrale AI-Anwendungen und das IoT ermöglichen dem Menschen eine enorme Effizienzsteigerung und dadurch Energieeinsparung in so gut wie allen Bereichen und Prozessen des Lebens. Unter anderem wird das IoT die Energiewende unterstützen können und zur Nachhaltigkeit unserer Industrie beitragen können.

Wie steht es nun um die Effektivität und Effizienz von Transfer Learning? Die vorgenommenen Messungen können die Beobachtungen bestehender Fachliteratur bestätigen. Der Energieverbrauch eines Modells ist primär von dessen Architektur abhängig. Transfer Learning ermöglicht durch Frameworks wie Model Maker den Zugriff auf bereits optimierte CNNs wie der EfficientNet-Lite-Serie. Dies erlaubt die Entwicklung effizienter Modelle, ohne eine hohe Fachkompetenz besitzen zu müssen. Es kann jedoch nicht ausgeschlossen werden, dass ein auf den Task spezialisiertes neuronales Netzwerk mit einer maßgeschneiderten Modell-Architektur durch geringere Redundanzen des Modells eine höhere Effizienz aufweisen kann. Neben der Modell-Architektur spielen Modellkompressions-Algorithmen eine wichtige Rolle bei der Effizienzsteigerung.

Auch die Effektivität von Transfer Learning konnten die Messungen der Accuracy bestätigen. Der geringe Genauigkeitsverlust bei der Inferenz unbekannter Bilder deutet auf einen geringen Generalisierungsfehler hin. Transfer Learning ermöglicht außerdem durch Bibliotheken wie Model Maker eine sehr effektive Entwicklung. Die Modellierung eines benutzerdefinierten CNNs ist sehr komplex, erfordert eine hohe Fachkompetenz und beinhaltet dementsprechend hohe Kosten. Deswegen kann Transfer Learning die Adaption von EI fördern. Wenn die Datenerhebung für das Training der Modelle sehr kostspielig oder der Datentransfer auf Grund der Sensibilität der Daten nicht möglich ist, kann Transfer Learning eine Grundvoraussetzung für EI bilden (siehe Kapitel 2.3.2).

Dennoch müssen auch die Nachteile betrachtet werden. Transfer Learning bietet zwar eine einfache Möglichkeit KI zu integrieren, aber speziell in Anwendungen wie der in den Kapiteln 2.1.3 und 3 genannten Kollisionserkennung von UAVs spielen sowohl die Latenz und der Energieverbrauch als auch die Genauigkeit eine große Rolle. Spezialisierte CNNs werden somit immer von Nöten sein.

Schlussendlich zeigen die Resultate, dass Transfer Learning eine wichtige Rolle bei der Entwicklung von EI spielt. Die Methode ist jedoch nicht für jeden Task geeignet und kann zu negativem Transfer führen (2.3.3). Transfer Learning ist effektiv und kann auf effizienten Modellen aufbauen. Modellkompressions-Algorithmen können die Effizienz weiter steigern und Transfer Learning zu einer Schlüsselkomponente der EI machen.

Literaturverzeichnis

[501a] Fiftyone. https://docs.voxel51.com/. Accessed: 2023-06-20.

[501b] Fiftyone dokumentation. https://docs.voxel51.com/user_guide/dataset_zoo/datasets.html#open-images-v6. Accessed: 2023-06-20.

[Abb] Die schichten des cloud, fog und mobile edge computings. https://www.rielloupsamerica.com/news/12846-will-the-edge-computing-revolutionize-the-cloud-s-paradigm. Accessed: 2023-06-24.

[AEKB20] A. Aral, M. Erol-Kantarci, and I. Brandić. Staleness control for edge data analytics. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 4(no. 2):1–24, 2020.

[AHS22] H. G. Abreha, M. Hayajneh, and M. A. Serhani. Federated learning in edge computing: A systematic survey. *Sensors*, vol. 22(no. 2), 2022.

[AIB+22] O. Ali, M. K. Ishak, M. K. L. Bhatti, I. Khan, and K. Kim. A comprehensive review of internet of things: Technology stack, middlewares, and fog/edge computing interface. *Sensors*, vol. 22(no. 3):0–43, 2022.

[ASHAM18] G. A. Akpakwu, B. J. Silva, G. P. Hancke, and A. M. Abu-Mahfouz. A survey on 5g networks for the internet of things: Communication technologies and challenges. *IEEE Access*, vol. 6:3619–3647, 2018.

[BHR14] N. Brunel, V. Hakim, and M. J. Richardson. Single neuron dynamics and computation. *Current Opinion in Neurobiology*, vol. 25:149–155, 2014. Theoretical and computational neuroscience.

[bin] Binary crossentropy dokumentation. https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy. Accessed: 2023-06-22.

[bio] biologische vs. künstliches neuron. https://www.datacamp.com/tutorial/deep-learning-python. Accessed: 2023-06-25.

[BMZA12] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, page 13–16, New York, NY, USA, 2012. Association for Computing Machinery.

[BSC+18] R. Buyya, S. N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, B. Varghese, E. Gelenbe, B. Javadi, L. M. Vaquero, M. A. S. Netto, A. N. Toosi, M. A. Rodriguez, I. M. Llorente, S. D. C. D. Vimercati, P. Samarati, D. Milojicic, C. Varela, R. Bahsoon, M. D. D. Assuncao, O. Rana, W. Zhou, H. Jin, W. Gentzsch, A. Y. Zomaya, and H. Shen. A manifesto for future generation cloud computing: Research directions for the next decade. *ACM Comput. Surv.*, vol. 51(no. 5), nov 2018.

[cora] Coral kompatibilität dokumentation. https://coral.ai/docs/edgetpu/models-intro/#compatibility-overview. Accessed: 2023-06-23.

[corb] Coral systemanforderungen dokumentation. https://coral.ai/docs/edgetpu/compiler/#system-requirements. Accessed: 2023-06-23.

[CWZZ18] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, vol. 35(no. 1):126–136, 2018.

[Dat] Keras data augmentation dokumentation. https://www.tensorflow.org/tutorials/images/data_augmentation. Accessed: 2023-06-20.

[DdRSGSR16] F. Díaz del Río, J. J. Salmerón García, and J. L. Sevillano Ramos. Extending amdahl's law for the cloud computing era. *Computer*, vol. 49(no. 2):14–22, 2016.

[den] Keras dense layer dokumentation. https://keras.io/api/layers/core_layers/dense/. Accessed: 2023-06-22.

[Effa] Efficientnet-lite blog. https://blog.tensorflow.org/2020/03/higher-accuracy-on-vision-models-with-efficientnet-lite.html. Accessed: 2023-06-22.

[Effb] Efficientnet-lite github. https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet/lite?utm_source=www.tensorflow.org&utm_medium=referral. Accessed: 2023-06-20.

[Est97] M. D. Estebon. Perceptrons: An associative learning network. *Virginia Tech.—1997*, 1997.

[fla] Keras flatten layer dokumentation. https://keras.io/api/layers/reshaping_layers/flatten/. Accessed: 2023-06-22.

[ful] Fully connected layer. https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-4-full-connection. Accessed: 2023-06-25.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[Git] Bachelor arbeit: Transfer learning github. https://github.com/e01627753/bachelor-thesis-transfer-learning/tree/main. Accessed: 2023-06-20.

[GLO+16] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew. Deep learning for visual understanding: A review. *Neurocomputing*, vol. 187:27–48, 2016. Recent Developments on Deep Big Vision.

[GOX20] J. Guo, W. Ouyang, and D. Xu. Multi-dimensional pruning: A unified framework for model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1508–1517, 2020.

[HNH+21] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D. Soudry. Accurate post training quantization with small calibration sets. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume vol. 139 of *Proceedings of Machine Learning Research*, pages 4466–4475. PMLR, 18-24 Jul 2021.

[hyp] Model maker hyperparameters dokumentation. https://www.tensorflow.org/lite/models/modify/model_maker/image_classification#change_the_training_hyperparameters. Accessed: 2023-06-22.

[K.13] Jamsa K. *Cloud Computing*. Jones Bartlett Learning LLC, 1. edition, 2013.

[Ker] Keras api dokumentation. https://keras.io/api/layers/. Accessed: 2023-06-20.

[KWR+16] A. Khoshkbarforoushha, M. Wang, R. Ranjan, L. Wang, L. Alem, S. U. Khan, and B. Benatallah. Dimensions for evaluating cloud resource orchestration frameworks. *Computer*, vol. 49(no. 2):24–33, 2016.

[LLY+22] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33(no. 12):6999–7019, 2022.

[M.G14] Avram M.G. Advantages and challenges of adopting cloud computing from an enterprise perspective. *Procedia Technology*, vol. 12:529–534, 2014. The 7th International Conference Interdisciplinarity in Engineering, INTER-ENG 2013, 10-11 October 2013, Petru Maior University of Tirgu Mures, Romania.

[mm] Model maker ptq dokumentation. https://www.tensorflow.org/lite/models/modify/model_maker/image_classification#customize_post-training_quantization_on_the_tensorflow_lite_model. Accessed: 2023-06-22.

[MMD] Tensorflow lite model maker dataloader code. https://github.com/tensorflow/examples/blob/master/tensorflow_examples/lite/model_maker/core/data_util/image_dataloader.py?utm_source=www.tensorflow.org&utm_medium=referral#L53-L106. Accessed: 2023-06-20.

[MWL22] P. McEnroe, S. Wang, and M. Liyanage. A survey on the convergence of edge computing and ai for uavs: Opportunities and challenges. *IEEE Internet of Things Journal*, vol. 9(no. 17):15435–15459, 2022.

[NCB+21] Y. Nahshan, B. Chmiel, C. Baskin, E. Zheltonozhskii, R. Banner, A. M. Bronstein, and A. Mendelson. Loss aware post-training quantization. *Machine Learning*, vol. 110(no. 11-12):3245–3262, 2021.

[OID] Open images v7 datenbank. https://storage.googleapis.com/openimages/web/visualizer/index.html?type=detection&set=train&c=%2Fm%2F01jfm_. Accessed: 2023-06-20.

[ON15] K. O'Shea and R. Nash. An introduction to convolutional neural networks. *CoRR*, vol. abs/1511.08458, 2015.

[OOSC14] A. Oludele, E. C. Ogu, K. Shade, and U. Chinecherem. On the evolution of virtualization and cloud computing: A review. *Journal of Computer Sciences and Applications*, vol. 2(no. 3):40–43, 2014.

[PY10] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, vol. 22(no. 10):1345–1359, 2010.

[pyc] Pycoral api dokumentation. https://coral.ai/docs/reference/py/. Accessed: 2023-06-23.

[Ros58] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, vol. 65(no. 6):386, 1958.

[SAS17] S. Saad, M. T. Abed, and A. Saad. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[SGAS20] A. Shakarami, M. Ghobaei-Arani, and A. Shahidinejad. A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. *Computer Networks*, vol. 182:107496, 2020.

[SGKW21] G. Shomron, F. Gabbay, S. Kurzum, and U. Weiser. Post-training sparsity-aware quantization. *Advances in Neural Information Processing Systems*, vol. 34:17737–17748, 2021.

[SLY19] W. Sun, J. Liu, and Y. Yue. Ai-enhanced offloading in edge computing: When machine learning meets industrial iot. *IEEE Network*, vol. 33(no. 5):68–74, 2019.

[SSS22] R. Sharma, S. Sikhwal, and S. Shrivastava. Edge computing: An overview. *Journal of Analysis and Computation (JAC)*, vol. 16, 2022.

[tf_a] Tensorflow model optimierung Übersicht. https://www.tensorflow.org/lite/performance/model_optimization. Accessed: 2023-06-22.

[tf_b] Tensorflow ptq dokumentation. https://www.tensorflow.org/lite/performance/post_training_quantization. Accessed: 2023-06-22.

[TFM] Tensorflow lite model maker dokumentation. https://www.tensorflow.org/lite/api_docs/python/tflite_model_maker. Accessed: 2023-06-20.

[VPC16] L. Valerio, A. Passarella, and M. Conti. Hypothesis transfer learning for efficient data computing in smart cities environments. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–8, 2016.

[WES+19] Z. K. Wazir, A. Ejaz, H. Saqib, Y. Ibrar, and A. Arif. Edge computing: A survey. *Future Generation Computer Systems*, vol. 97:219–235, 2019.

[WKL96] B. Widrow, I. Kollar, and Ming-Chang L. Statistical theory of quantization. *IEEE Transactions on Instrumentation and Measurement*, vol. 45(no. 2):353–361, 1996.

[WKW16] K. Weiss, T. M. Khoshgoftaar, and D. Wang. A survey of transfer learning. *Journal of Big Data*, vol. 3(no. 9):0–40, 2016.

[ZG17] M. Zhu and S. Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression, 2017.

[ZQD+21] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, vol. 109(no. 1):43–76, 2021.

[ZWL+19] X. Zhang, Y. Wang, S. Lu, L. Liu, L. Xu, and W. Shi. Openei: An open framework for edge intelligence. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1840–1851, 2019.