wirtschafts
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Fakultät für
Wirtschafts-
wissenschaften
Business School

Zusammenstellung, Installation und Konfiguration eines Personal Computers für Deep Learning Projekte

Stefan Selle

Professor für Wirtschaftsinformatik
Fakultät für Wirtschaftswissenschaften
Hochschule für Technik und Wirtschaft des Saarlandes

Saarbrücken, 30.01.2018

Kurzfassung

Deep Learning (DL) ist eine Technik des maschinellen Lernens. Nach dem Vorbild der Natur werden Künstliche Neuronale Netzwerke (KNN) trainiert, um bspw. Bilder oder Sprache in digitaler Form zu analysieren. Für das autonomen Fahren von Automobilen ist diese Technik der Künstliche Intelligenz (KI) essentiell. Die KNN sind so groß bzw. tief, d.h. sie bestehen aus sehr vielen, sogenannten verborgenen Schichten von Neuronen, dass spezielle Lernverfahren und sehr viel Rechenleistung zum Trainieren notwendig sind. Moderne Computer besitzen heutzutage starke Grafikkarten, weil die Anforderungen an grafische Anwendungen, insbesondere im Segment der Computerspiele, gestiegen sind. Diese Grafikkarten haben einen Grafikprozessor (engl. *Graphics Processing Unit (GPU)*), um die sehr anspruchsvollen Berechnungen durchzuführen. Neben dieser primären Aufgabe haben sich mittlerweile aber auch zwei andere Anwendungsbereiche für Grafikkarten etabliert: Das *Mining* von Kryptowährungen wie bspw. *Bitcoin* und das *Deep Learning*.

In dieser Arbeit wird gezeigt, wie man einen Personal Computer (PC) zusammenstellt, installiert und konfiguriert, sodass dieser für *Deep Learning* Projekte eingesetzt werden kann. Ausgehend von der Auswahl geeigneter Hardware-Komponenten, wird sehr detailliert der Installations- und Konfigurationsprozess schrittweise beschrieben. Zunächst wird die BIOS-Firmware des Mainboards aktualisiert und Änderungen in den Einstellungen vorgenommen. Dann wird das Betriebssystem mittels vorbereiteten, bootfähigen USB-Stick installiert und angepasst. Des Weiteren werden spezielle Grafikkarten-Treiber, nützliche Tools sowie eine ausgewählte DL-Bibliothek installiert und konfiguriert. Die dabei verwendete Software ist entweder *Open Source* oder *Freeware*. Abschließend wird eine Beispielanwendung benutzt, um das System zu validieren und die *Performance* festzustellen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AI Artificial Intelligence

API Application Programming Interface

APT Advanced Packaging Tool

ATX AT Extended

BIOS Basic Input Output System

CD Compact Disc

CPU Central Processing Unit

DDR Double Data Rate

DL Deep Learning

DVD Digital Versatile Disc

DVI Digital Visual Interface

ERP Enterprise Resource Planning

EULA End User License Agreement

FAT File Allocation Table

GB Gigabyte

GCC Gnu Compiler Collection

GHz Gigahertz

GPU Graphics Processing Unit

HD High Definition

HDMI High Definition Multimedia Interface

IBM International Business Machines

IGD Integrated Graphics Device

IP Internet Protocol

KI Künstliche Intelligenz

KNN Künstliches Neuronales Netzwerk

LAN Local Area Network

LTS Long Time Support

LVM Logical Volume Manager

MB Megabyte

MCTS Monte Carlo Tree Search

MHz Megahertz

MSI Micro-Star International

NAS Network Attached Storage

NLP Natural Language Processing

PC Personal Computer

PCI Peripheral Component Interconnect

PCIe PCI Express

PEG PCI Express Graphics

PPA Personal Package Archive

RAM Random Access Memory

RL Reinforcement Learning

SATA Serial AT Attachment

SMI System Management Interface

SSD Solid-State-Drive

SSH Secure Shell

TDP Thermal Design Power

UEFI Unified Extensible Firmware Interface

URL Uniform Resource Locator

USB Universal Serial Bus

UUID Universally Unique Identifier

WD Western Digital

1 Einleitung

Im März 2016 unterlag der südkoreanische Profispieler Lee Sedol einer Maschine beim Brettspiel Go mit 1:4 Spielen unter Turnierbedingungen [Wik18a]. Dies ist deshalb so bemerkenswert, weil Go eine deutlich höhere Komplexität als Schach aufweist. Knapp 20 Jahre zuvor, im Mai 1997, schlug der IBM-Computer *Deep Blue* den damaligen Weltmeister Garri Kasparow bei einem Schachturnier von 6 Partien mit 3,5 zu 2,5 [Wik18b]. *Deep Blue* war ein Supercomputer mit enormer paralleler Rechenleistung von 30 Knoten mit jeweils 16 optimierten Schachprozessoren, der 200 Millionen Stellungen pro Sekunde berechnen konnte. Das Schachbrett hat eine Größe von 8 x 8 Felder, also 64 Felder. Das Spiel startet mit 32 Figuren in 6 Rollen mit definierten Zugregeln. Die Zahl der möglichen Stellungen wird auf 10^{43} geschätzt [Sha50]. Im Laufe des Schachspiels nimmt die Komplexität ab, weil Figuren geschlagen und vom Brett genommen werden. Schachcomputer funktionieren folgendermaßen: Im ersten Schritt erstellen sie einen großen Spielbaum mit den möglichen nächsten Zügen als Äste. Diese verzweigen dann weiter, indem die möglichen Antwortzüge des Gegners betrachtet werden usw. Im zweiten Schritt wird dieser Baum nach dem besten nächsten Zug durchsucht, wobei eine Bewertungsfunktion angewendet wird. Zur Konstruktion der Bäume und zur Suche in diesen Bäumen werden spezielle Algorithmen entwickelt und angewendet. *Deep Blue* hat gezeigt, dass es mit sehr großer Rechenleistung möglich ist, diese Algorithmen so effizient auszuführen, dass die Maschine den Menschen beim Schachspiel besiegen kann. Go wird dagegen auf einem 19 x 19 Brett gespielt, das sind also insgesamt bereits 172 Felder. Ein Feld kann leer oder mit einem schwarzen oder weißen Spielstein besetzt sein. Es gibt also bereits $3 \cdot 10^{172}$ Möglichkeiten. Das sind mehr Kombinationen als es Atome im Universum gibt. Aufgrund dieser Komplexität ist es nicht sinnvoll, Go-Computer analog zu Schachcomputern zu entwickeln. Es wird somit ein völlig neuer Ansatz benötigt.

Das Computersystem, das Lee Sedol bezwungen hat, heißt *AlphaGo* und wurde von DeepMind entwickelt [Dee18]. Mittlerweile gehört DeepMind zur Alphabet-Gruppe, die aus Google hervorgegangen ist. *AlphaGo* benutzt tiefe Künstliche Neuronale Netzwerke (KNN), um aus einer Datenbank mit 30 Millionen Stellungen von bereits gespielten Go-Partien zwischen Experten, geeignete Zugkandidaten zu bestimmen. Diese Netze werden mit dem überwachten Lernen (engl. *Supervised Learning*) trainiert. Aufgrund der Größe bzw. Tiefe dieser KNN spricht man auch von *Deep Learning (DL)*. Des Weiteren kommt das bestärkende Lernen (engl. *Reinforcement Learning (RL)*) zum Einsatz. Hier spielt *AlphaGo* sehr viele Partien Go gegen sich selbst, macht dabei Fehler und lernt wiederum aus diesen. Dadurch wird das System sehr leistungsstark. Eine zweite Klasse von KNN wird benutzt, um den Gewinner des Spiel zu prognostizieren. Diese Netze werden ebenfalls mittels *Reinforcement Learning* trainiert. Außerdem verwendet *AlphaGo* einen neuen Monte Carlo Suchalgorithmus (engl. *Monte Carlo Tree Search (MCTS)*), um aus der Kombination dieser beiden Klassen von KNN schließlich den nächsten Spielzug zu bestimmen. Das verteilte System von *AlphaGo* benutzt 40 parallele Suchprozesse, die 1.202 Prozessoren (engl. *Central Processing Unit (CPU)*) und 176 Grafikprozessoren (engl. *Graphics Processing Unit (GPU)*) verwenden [Sil+16]. In diesem Anwendungsbeispiel übertrifft die Künstliche Intelligenz (KI) (engl. *Artificial Intelligence (AI)*) bereits die des Menschen.

Generell ist *Deep Learning* als KI-Methode besonders gut geeignet, wenn in sehr großen Datenmengen (*Big Data*) nach Mustern gesucht werden muss. Deshalb wird diese Technik häufig für die Bild- oder Spracherkennung eingesetzt. Anwendungen sind bspw. das autonome Fahren von Automobilen oder die digitalen Assistenten als Smartphone-Apps. Wie *Deep Learning* genau funktioniert ist nicht Bestandteil dieser Arbeit. Interessierten Lesern sei das Standardwerk von Ian Goodfellow, Yosuah Bengio und Aaron Courville empfohlen [GBC16].

Das Ziel dieser Arbeit ist die Beschreibung der Zusammenstellung, Installation und Konfiguration eines Computers, der für einfache *Deep Learning* Projekte in der Forschung und Lehre an der Hochschule für Wirtschaft und Technik des Saarlandes (htw saar) eingesetzt werden kann. Das Herzstück eines solchen DL-Rechners ist nicht der Prozessor sondern seine Grafikkarte. Denn auf dieser lassen sich sehr rechenintensive Berechnungen parallel durchführen, ideal also für *Deep Learning*. Es stellt sich also die Frage: Was für eine Grafikkarte sollte man hierzu verwenden? Mit genau dieser Frage hat sich bereits ausführlich Slav Ivanov in seinem Blogbeitrag beschäftigt [Iva17a]. Bei ausreichend vorhandenen Budget ist seine eindeutige Empfehlung: GTX 1080 Ti von Nvidia. In einem seiner früheren Blogbeiträge stellt Ivanov dar, wie man einen DL-Rechner zusammenschraubt und einrichtet [Iva17b]. Die verwendete Software ist komplett *Open Source* oder *Freeware* und deshalb fallen keine Lizenzgebühren für deren Nutzung an. Diese Arbeit orientiert sich sehr stark an die Beschreibungen Ivanovs. Allerdings gibt es einige gravierende Unterschiede. Die Hardware ist nicht exakt die gleiche. Mittlerweile liegen aktuellere Versionen der verwendeten Software vor. Das führt teilweise zu Kompatibilitätsproblemen und einem anderen Vorgehen bei der Installation und Konfiguration. Die Beschreibung hierzu ist außerdem wesentlich detaillierter als bei Ivanov. Generell benötigt man Basiswissen zu Computerhardware und zum Umgang mit dem Betriebssystem Linux. Zwar werden einige der Linux-Kommandos kurz erklärt, dennoch werden diese Kenntnisse beim Leser vorausgesetzt.

Des Weiteren ist diese Arbeit auch als Dokumentation und Anleitung gedacht. Denn es hat sich gezeigt, dass die Installation und Konfiguration des DL-Rechners mehrfach vorgenommen werden musste. Viele Softwareprogramme wurden installiert und konfiguriert. Auch Programme, die in dieser Arbeit gar nicht beschrieben werden, wurden getestet. Aufgrund der Komplexität und Abhängigkeiten zwischen den installierten Versionen, kam es häufig zu Fehlern und Problemen – entweder schon während der Installation oder erst später bei der Benutzung der Programme. Diese Probleme konnten meistens durch extensive Internetrecherchen und intensives Ausprobieren gelöst werden. Aber eben nicht immer. Manchmal blieb dann auch ein System dabei zurück, das nicht mehr richtig funktionierte. Die einzige Lösung bestand dann darin, das System wieder auf den ursprünglichen Zustand zurückzusetzen. Genau hierbei hat diese Arbeit in Form als Dokumentation sehr geholfen. Mit einem neuen, frisch aufgesetzten System war es wieder sehr viel leichter möglich, die nächsten Schritte zu meistern. Als Ergebnis ist somit diese sehr nützliche Anleitung entstanden, um einen DL-Rechner in nur einem Schritt zu installieren und zu konfigurieren.

Die vorliegende Arbeit ist in sieben Kapiteln gegliedert. Nach der Einleitung werden in Kapitel 2 die einzelnen, ausgewählten Hardware-Komponenten kurz vorgestellt. In Kap. 3 wird beschrieben, wie die Firmware des *Basic Input Output System (BIOS)* des Mainboards aktualisiert wird und welche Einstellungen im BIOS vorzunehmen sind. In Kap. 4 wird gezeigt, wie man das Linux-Betriebssystem Ubuntu 16.04. LTS auf den Rechner

installiert. Damit die externe Grafikkarte einwandfrei funktioniert und sinnvoll für *Deep Learning* genutzt werden kann, müssen spezielle Nvidia-Treiber und andere Tools (CUDA 9.1, CuDNN 7.0.5) installiert werden. Dies wird in Kap. 5 beschrieben. Außerdem wird die Python 3.6-Distribution Anaconda 5.0.1 verwendet. In Kap. 6 wird dann die DL-Bibliothek TensorFlow 1.5.0 eingerichtet. Eine Beispielanwendung wird ausgeführt, um die *Performance* des Systems zu zeigen und *Benchmarks* im Vergleich zwischen GPU zu CPU zu messen. Im letzten Kapitel wird dann die Arbeit zusammengefasst und ein kurzer Ausblick gegeben.

2 Hardware

Im Mittelpunkt der Hardware eines Rechners mit dem Einsatzzweck *Deep Learning* steht seine Grafikkarte. In der Einleitung wurde bereits erwähnt, dass diese auf dem Grafikprozessor GTX 1080 Ti von Nvidia basieren sollte. Die anderen Hardware-Komponenten wurden dann so ausgesucht, dass sie miteinander kompatibel sind und ein insgesamt leistungsstarker, aber noch bezahlbarer Rechner entsteht. In Tab. 2.1 sind die einzelnen Komponenten des DL-Rechners dargestellt.

Nr.	Komponente	Hersteller	Bezeichnung (und Merkmale)	Preis	Anteil
1	Gehäuse	Corsair	Carbide 270R Window Tower	57,00	2,6 %
2	Netzteil	Corsair	RM1000X (1000 Watt)	149,00	6,9 %
3	Mainboard	MSI	Z270 Gaming Pro Carbon	139,00	6,5 %
4	Prozessor	Intel	Core i7-7700K (4 x 4,2GHz)	269,00	12,5 %
5	CPU-Kühler	Alpenföhn	Brocken Eco	28,00	1,3 %
6	Speicher	Corsair	DIMM 64GB DDR4-2400 Kit	536,00	24,9 %
7	SSD	Samsung	960 EVO (SSD 250GB, M.2)	107,00	5,0 %
8	Festplatte	WD	WD40EFRX (4TB, SATA 600)	111,00	5,2 %
9	Grafikkarte	Gigabyte	AORUS GeForce GTX 1080 Ti	755,45	35,1 %
	Gesamt			2.094,45	

Tabelle 2.1: Hardware-Komponenten

Die Preise sind als Nettopreise in Euro auf Basis eines Angebots vom 04.10.2017 der Firma Alternate GmbH angegeben. Für eine Montage und Konfiguration der Komponenten werden weitere 80 Euro fällig. Weiterhin kommen Fracht-, Porto- und Transaktionsgebühren hinzu. Schließlich muss noch die Umsatzsteuer von 19 Prozent berücksichtigt werden. Der Gesamtpreis für den kompletten Rechner beträgt dann 2.663,41 Euro. Grafikkarte, Arbeitsspeicher und Prozessor sind die mit Abstand teuersten Komponenten im DL-Computer. In den folgenden Abschnitten werden alle ausgewählten Komponenten detailliert betrachtet. Um den Computer zu konfigurieren, werden natürlich noch Eingabegeräte wie Tastatur und Maus und ein Monitor als Ausgabegerät benötigt.

Gehäuse Das Tower-Gehäuse Corsair Carbide 270R Window (vgl. Abb. 2.1) wurde gewählt, weil es für ein Mainboard und Netzteil des Formfaktors ATX konzipiert ist und einen CPU-Kühler bis zu einer Höhe von 170 Millimetern und Grafikkarten bis zu einer Länge von 370 Millimetern Platz bietet [Alt18b]. Somit lassen sich die anderen Komponenten (vgl. Tab. 2.1) dort problemlos unterbringen. Das Seitenfenster ist nicht unbedingt notwendig. Für einen Aufpreis von 10 Euro gegenüber dem Basismodell hat man hierdurch allerdings die Möglichkeit, die verbauten und teilweise beleuchteten Komponenten im Einsatz zu bestaunen.

Abbildung 2.1: Gehäuse Corsair Carbide 270R Window [Gei18b]

Netzteil Das modular aufgebaute PC-Netzteil Corsair RM1000X (vgl. Abb. 2.1) wurde passend zum Gehäuse (s.o.) gewählt. Das Netzteil ist trotz seiner maximalen Leistung von 1000 Watt aufgrund der Verwendung von geräuscharmen Kondensatoren und Transformatoren sowie der speziellen Konstruktion des Lüfters extrem leise. Die Geräuschentwicklung beträgt nur 23,2 Dezibel bei 100 Prozent Last. Der zertifizierte 80-PLUS-Gold-Wirkungsgrad reduziert Betriebskosten und übermäßige Wärme (Energieeffizienz bis zu 90 Prozent) [Alt18d]. Das Netzteil ist bereits so dimensioniert, dass auch zukünftige Hardware-Erweiterungen, wie bspw. der Einsatz einer zweiten Grafikkarte, mit ausreichend Strom versorgt werden können.

Abbildung 2.2: Netzteil Corsair RM1000X [Gei18d]

Mainboard Das Mainboard Z270 Gaming Pro Carbon (vgl. Abb. 2.9) des Herstellers Micro-Star International (MSI) ist, wie der Name bereits verrät, eigentlich für Computerspieler entwickelt worden. Es basiert auf dem Intel-Z270-Chipsatz und unterstützt Intel-Prozessoren für den Sockel 1151. Es besitzt den ATX-Formfaktor und verfügt über vier DDR4-Slots zur Aufnahme von maximal 64 GB Arbeitsspeicher. Besonders erwähnenswert sind die drei PCIe-3.0x16-Slots, welche für Grafikkarten benutzt werden können. Außerdem gibt es neben sechs SATA3-Anschlüssen auch zwei M.2-Anschlüsse für den Anschluss von schnellen SSDs. Standard sind 8-Kanal-Sound, eine Gigabit-LAN-Schnittstelle sowie mehrere USB-3-Ports. Eine CPU-abhängige Grafiklösung mit HDMI- und DVI-D-Anschluss ist bereits ebenfalls mit an Bord [Alt18g].

Abbildung 2.3: Mainboard MSI Z270 Gaming Pro Carbon [Gei18g]

Prozessor Der Prozessor Core i7-7700K von Intel basiert auf der 14-nm-Architektur *Kaby Lake*. Er enthält eine Quad-Core-CPU für den Sockel 1151 (vgl. Abb. 2.4) und passt damit perfekt zum ausgewählten Mainboard (s.o.), Mit einer Frequenz von 4,2 GHz wird jeder der vier Kerne getaktet. Eine Übertaktung (engl. *Overclocking*) auf 4,5 GHz ist bei einer Leistungsaufnahme von maximal 130 Watt TDP ebenfalls möglich. Der Prozessor verfügt über 8 MB Level-3-Cache und unterstützt u.a. Standards wie DDR4-2400. Die enthaltene GPU Intel HD Graphics 630 läuft im normalen Takt mit 350 MHz, maximal bis 1150 MHz [Alt18f].

Abbildung 2.4: Prozessor Intel Core i7-7700K [Gei18f]

CPU-Kühler Um die Wärme des Prozessors abzuführen, wird eine Kühlung benötigt. In diesem Fall muss noch keine Wasserkühlung installiert werden, es genügt ein normaler Kühler mit Radiator und Lüfter. Der Alpenföhn Brocken Eco (vgl. Abb. 2.5) ist u.a. für den Sockel 1151 geeignet und kann für CPUs mit einer Leistung von maximal 160 Watt TDP eingesetzt werden. Des Weiteren ist er sehr leise, d.h. er verursacht bei 1.500 U/min nur eine Geräuschentwicklung von 27,3 Dezibel. Mit einer Höhe von 150 Millimetern passt er perfekt in das ausgewählte ATX-Gehäuse [Alt18a].

Abbildung 2.5: Kühler Alpenföhn Brocken Eco [Gei18a]

Arbeitsspeicher Als Arbeitsspeicher (engl. *Random Access Memory (RAM)*) kommen Module von Corsair zum Einsatz (vgl. Abb. 2.6). Diese sind passend für das Mainboard und den Prozessor (s.o.) ausgewählt worden, d.h. solche mit der Spezifikation DDR4-2400 (PC4-19200) und einer Spannung von 1,2 V. Es sind vier Speichermodule mit jeweils 16 GB, zusammen also 64 GB. Dies ist auch die maximale Kapazität, die das Mainboard aufnehmen kann. Weil das spätere Nachrüsten von Arbeitsspeicher aufgrund unterschiedlicher Timings und Latenzzeiten der Module aus unterschiedlichen Bauserien zu Problemen im Betrieb führen kann, wurde direkt das Maximum an Speicher gekauft. Außerdem wurde Markenspeicher statt No-Name-Speicher gewählt, auch wenn dieser etwas teurer ist. Die Vengeance LPX Serie von Cosair wurde speziell für eine anspruchsvolle Nutzung entwickelt und bietet dafür auch eine hohe Performance, Stabilität und Verlässlichkeit. Ein weiteres Feature ist XMP 2.0 [Alt18c].

Abbildung 2.6: Arbeitsspeicher Corsair DIMM DDR4-2400 Vengeance LPX [Gei18c]

Solid-State-Drive Für das Speichern des Betriebssystems und der wichtigsten Programme wird eine sehr schnelle Festplatte benötigt. *State-of-the-Art* ist hier das sogenannte Solid-State-Drive (SSD). Das Samsung 960 EVO ist ein solches SSD (vgl. Abb. 2.7). Der Speicherplatz von 250 GB ist ausreichend. Statt einer konventionellen SATA-Schnittstelle wird eine PCIe-Schnittstelle in Verbindung mit dem NVMe-Protokoll (M.2) verwendet. Hierdurch ergeben sich Transferraten von bis zu 3.200 MB/s für Lese- und 1.500 MB/s für Schreibvorgänge [Alt18h].

Abbildung 2.7: SSD Samsung 960 EVO [Gei18h]

Festplatte Für die Datenspeicherung wird eine konventionelle Festplatte benötigt. Die WD40EFRX aus der Red-Serie von Western Digital (WD) (vgl. Abb. 2.8) ist eigentlich speziell für NAS-Systeme konstruiert worden. Sie ist ausgiebig für den 24/7-Dauerbetrieb getestet worden und hierfür ausgelegt. Somit garantiert sie eine hohe Datensicherheit. Es ist eine klassische Platte im 3,5-Zoll-Format, mit SATA-600-Schnittstelle und 4 TB Kapazität. Der Cache beträgt 64 MB, die maximale Lautstärke 28 Dezibel bei 5.400 U/min [Alt18i].

Abbildung 2.8: Festplatte WD40EFRX [Gei18i]

Grafikkarte Das Herzstück des DL-Computers ist die Grafikkarte. Die Gigabyte AORUS GeForce GTX 1080 Ti (vgl. Abb. 2.9) ist eine High-End-Grafikkarte mit einem schnellen Grafikprozessor von Nvidia. Die GPU GTX 1080 Ti verfügt über 3584 Streamprozessoren und taktet mit 1594 MHz (Boost Clock bis 1708 MHz), der 11 GB große GDDR5X-Speicher ist über einen 352-Bit-Speicherkontroller angebunden und arbeitet mit 5505 MHz (effektive Datenrate 11010 MHz). Die Karte besitzt einen PCIex16-Steckplatz. Der durchschnittliche Stromverbrauch wird mit 375 Watt angegeben. Die Karte besitzt die Abmessungen 142 mm x 55 mm x 293 mm, sodass sie problemlos in das ausgewählte Gehäuse passt. Sie unterstützt bereits die Multi-GPU-Technologie SLI-HB, falls eine weitere Karte nachgerüstet werden sollte [Alt18e].

Abbildung 2.9: Grafikkarte Gigabyte AORUS GeForce GTX 1080 Ti [Gei18e]

Der komplett montierte Rechner ist in Abb. 2.10 zu sehen.

Abbildung 2.10: Kompletter DL-Rechner

3 BIOS

Bevor das Betriebssystem auf der neuen Hardware (vgl. Kap. 2) installiert wird, sollte zunächst das *Basic Input Output System (BIOS)* aktualisiert und konfiguriert werden. Es handelt sich dabei um eine Firmware, die sich im nicht-flüchtigen Speicher des Mainboards befindet und direkt nach dem Anschalten des Computers ausgeführt wird. Insbesondere werden dabei die am Mainboard angeschlossenen Komponenten (Prozessor, Hauptspeicher, Festplatten, Grafikkarte, Tastatur, Maus, Monitor usw.) initialisiert, sodass diese anschließend vom Betriebssystem (vgl. Kap. 4) benutzt werden können. Somit stellt das BIOS die wichtige Schnittstelle zwischen Hardware und Betriebssystem dar. Inzwischen ist das BIOS durch das sogenannte *Unified Extensible Firmware Interface (UEFI)* abgelöst worden. Trotzdem benutzt man noch immer den Begriff BIOS. Beim Hersteller MSI heißt das UEFI zum Mainboard Z270 Gaming Pro Carbon bspw. Click BIOS 5.

3.1 BIOS-Update

Der DL-Rechner hat eine integrierte Grafikkarte und eine externe Grafikkarte mit Nvidia-GPU. Somit stehen an der Rückseite des Gehäuses verschiedene Anschlüsse für den Monitor zur Verfügung. Das BIOS-Update funktioniert nur, wenn der Monitor an der externen Grafikkarte angeschlossen ist. Ggf. muss man also hierzu im BIOS zunächst einstellen, dass genau diese Grafikausgabe präferiert wird.

Auf der Support-Webseite des Mainboard-Herstellers MSI (URL: https://de.msi.com/Motherboard/support/Z270-GAMING-PRO-CARBON) kann man herausfinden, welches die neuste Firmware für das BIOS der Z270 Gaming Pro Carbon ist [MSI18b]: Version 1.6 (7A63v16) vom 01.12.2017. Es ist außerdem eine englischsprachige Anleitung als Word-Dokument hinterlegt, in der beschrieben steht, wie man bei dem Update vorgehen muss [MSI18a]. Zunächst lädt man die Archiv-Datei 7A63v16.zip (Größe 6,99 MB) von der Webseite herunter. Dann entpackt man das ZIP-Archiv und kopiert den Ordner inklusive der Dateien auf einen FAT32-formatierten USB-Stick. Diesen USB-Stick schließt man an den DL-Rechner an und startet diesen. Während des Startvorgangs muss die `Enf`-Taste bzw. `Del`-Taste gedrückt gehalten werden, um in das Setup-Menü des MSI Click BIOS 5 zu kommen.

Im oberen rechten Bereich wird u.a. angezeigt, welche aktuelle Versionsdatei installiert ist: *E7A63IMS.150* (vgl. Abb. 3.1). Die letzten Ziffern geben an, dass es sich um Version 1.5 handelt. Unter dem Menüpunkt `Settings\Systemstatus` kann man sich zusätzlich vergewissern. Dort wird unterhalb der BIOS-Version auch noch das Erstellungsdatum angezeigt.

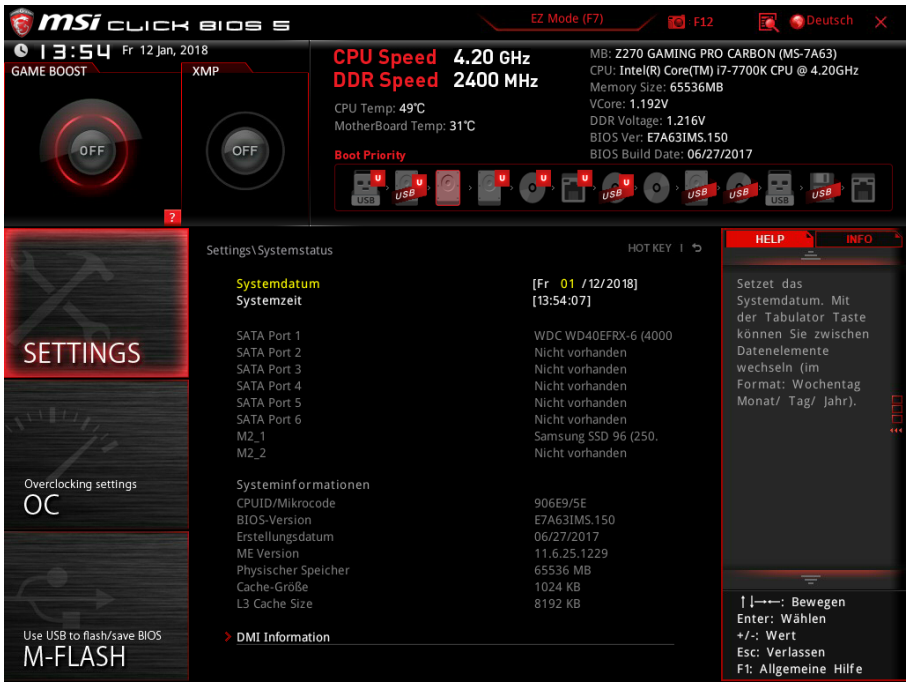

Abbildung 3.1: MSI BIOS: Systemstatus vor dem Update

Nun wechselt man zur Funktion `M-Flash`, indem man die Schaltfläche unten links klickt. Man bekommt einen Hinweis, dass das System neu gestartet wird und dabei der Flash-Modus geladen wird (vgl. Abb. 3.2).

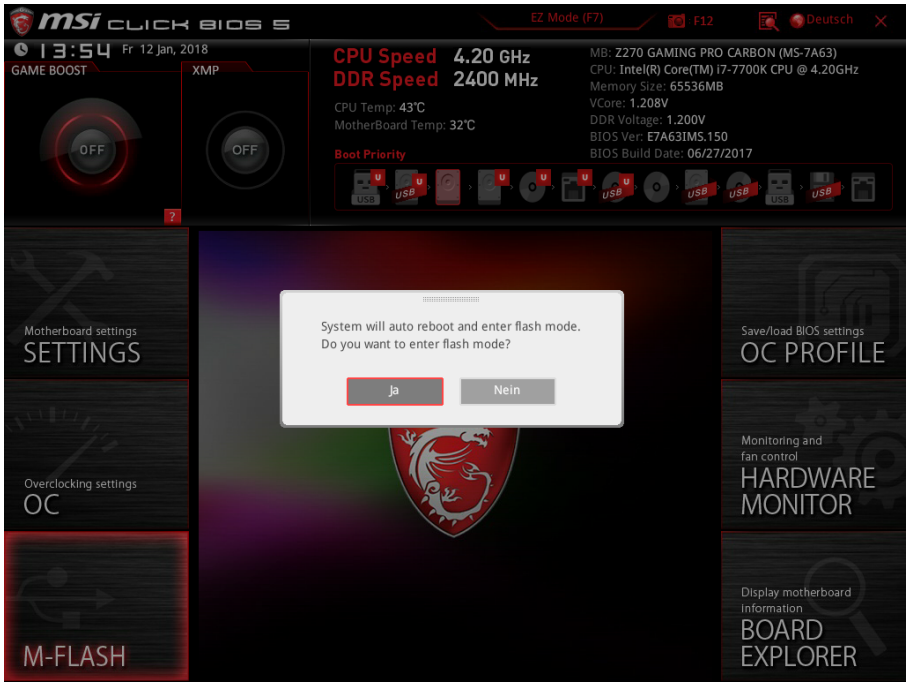

Abbildung 3.2: MSI BIOS: M-Flash

Bestätigt man mit Klick auf `Ja`, dann kann man nach dem Neustart den USB-Stick auswählen und im Dateisystem navigieren, bis man die Datei E7A63IMS.160 findet. Diese

wählt man aus und startet das Update. Während das Update läuft, darf man den Rechner weder ausschalten noch den USB-Stick herausziehen (vgl. Abb. 3.3).

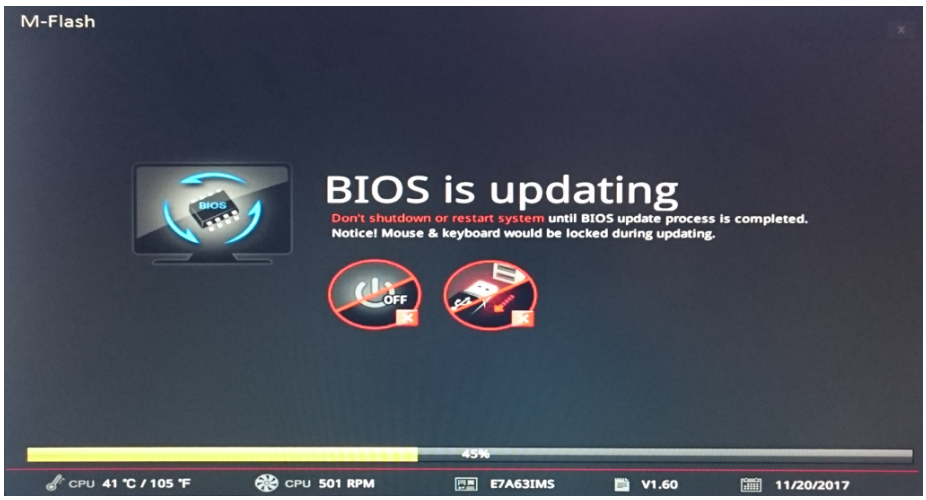

Abbildung 3.3: MSI BIOS: Update

Nach dem Update sollte man sich vergewissern, dass die neue Version installiert ist, indem man wieder den Menüpunkt `Settings\Systemstatus` aufruft (vgl. Abb. 3.4).

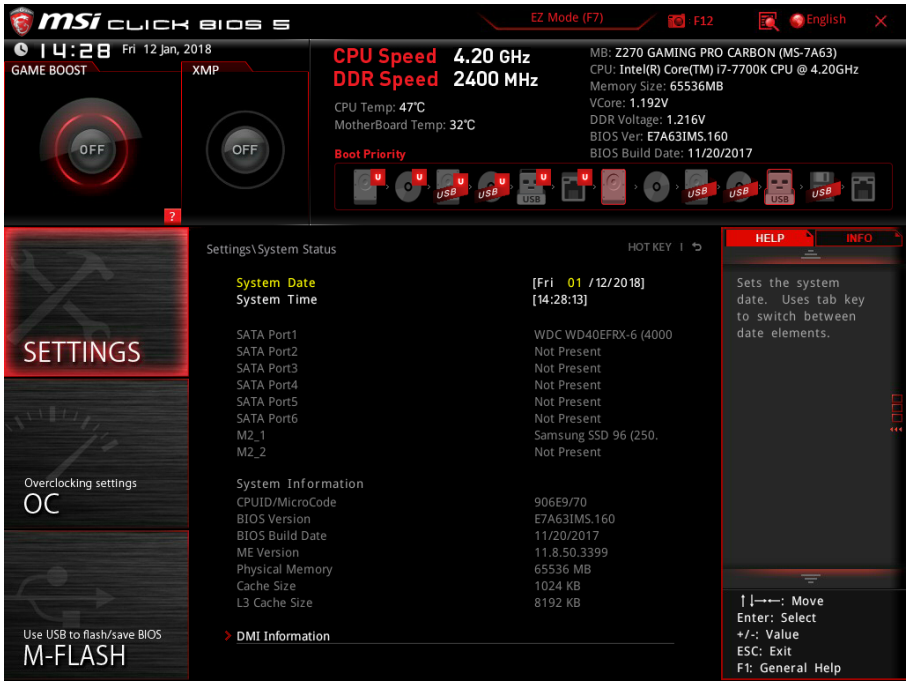

Abbildung 3.4: MSI BIOS: Systemstatus nach dem Update

Wie man sieht, wurde bei dem Update auch die Sprache auf Englisch geändert. Dies lässt sich leicht korrigieren, indem man oben rechts auf die Sprache klickt und in der Nachschlageliste Deutsch auswählt (vgl. Abb. 3.5).

Abbildung 3.5: MSI BIOS: Auswahl der Sprache

3.2 BIOS-Konfiguration

Im BIOS lassen sich nun einige Einstellungen hinsichtlich der eingebauten Hardwarekomponenten vornehmen. In den folgenden Abbildungen 3.6 bis 3.19 sind die aktuellen bzw. geänderten Einstellungen dargestellt. Die gegenüber den Standardeinstellungen veränderten Werte werden jeweils kurz kommentiert.

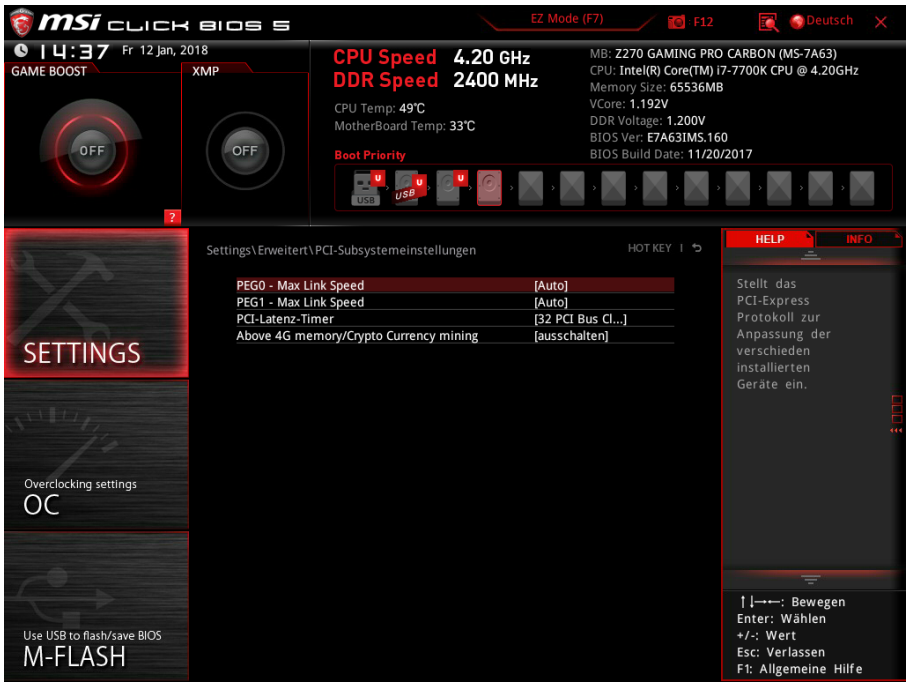

Abbildung 3.6: MSI BIOS: PCI-Subsystemeinstellungen

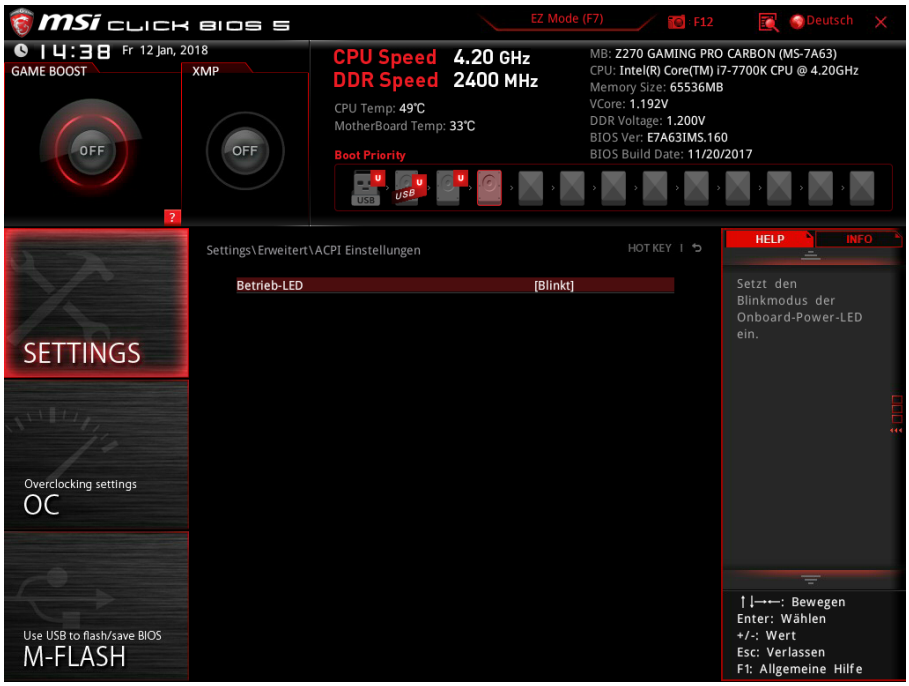

Abbildung 3.7: MSI BIOS: ACPI-Einstellungen

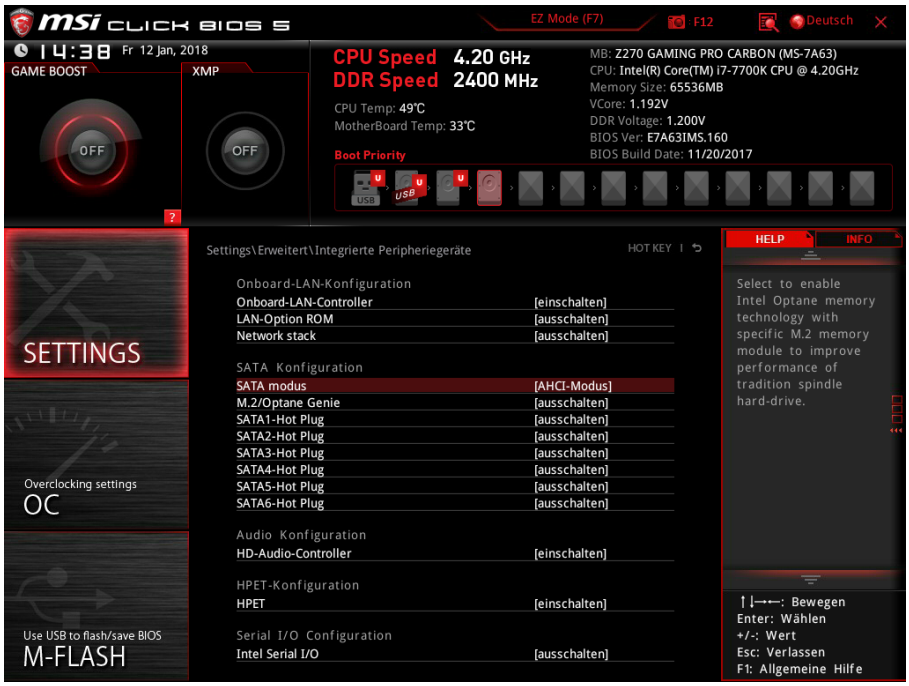

Abbildung 3.8: MSI BIOS: Integrierte Peripheriegeräte

Im Menü `Integrierte-Grafik-Konfiguration` wird als `Initial-Grafikadapter` *PCI Express Graphics (PEG)* statt *Integrated Graphics Device (IGD)* eingestellt (vgl. Abb. 3.9). Damit erfolgt die Grafikausgabe standardmäßig auf die externe Grafikkarte mit der GPU von Nvidia. Leider bereiten die Grafikkarten-Treiber des Herstellers Nvidia sehr viele Probleme in Linux-Betriebssystemen. Deshalb empfiehlt es sich, zusätzlich die Grafik auch durch die interne Grafikkarte ausgeben zu lassen. Hierzu wird der IGD-Mehrfachmonitor einge-

schaltet. Als `geteilter integrierte Grafik-Speicher` wird der mögliche Maximalwert 1024M eingestellt. Die im Intel-Prozessor integrierte Grafikkarte benutzt somit 1 GB des Hauptspeichers. Dieser ist mit insgesamt 64 GB sehr großzügig dimensioniert, sodass hier problemlos der Maximalwert verwendet werden kann.

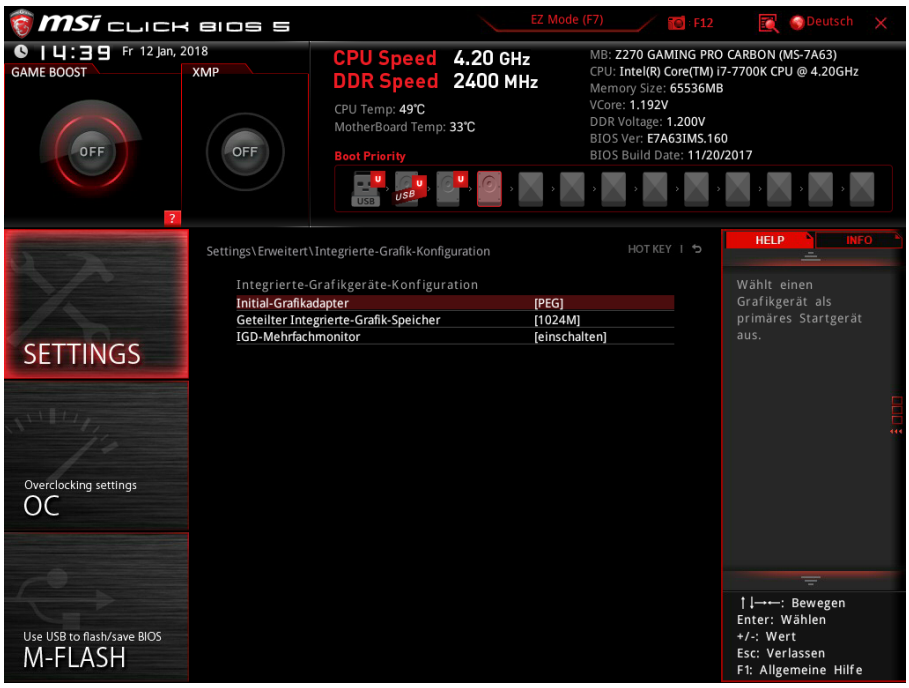

Abbildung 3.9: MSI BIOS: Integrierte-Grafik-Konfiguration

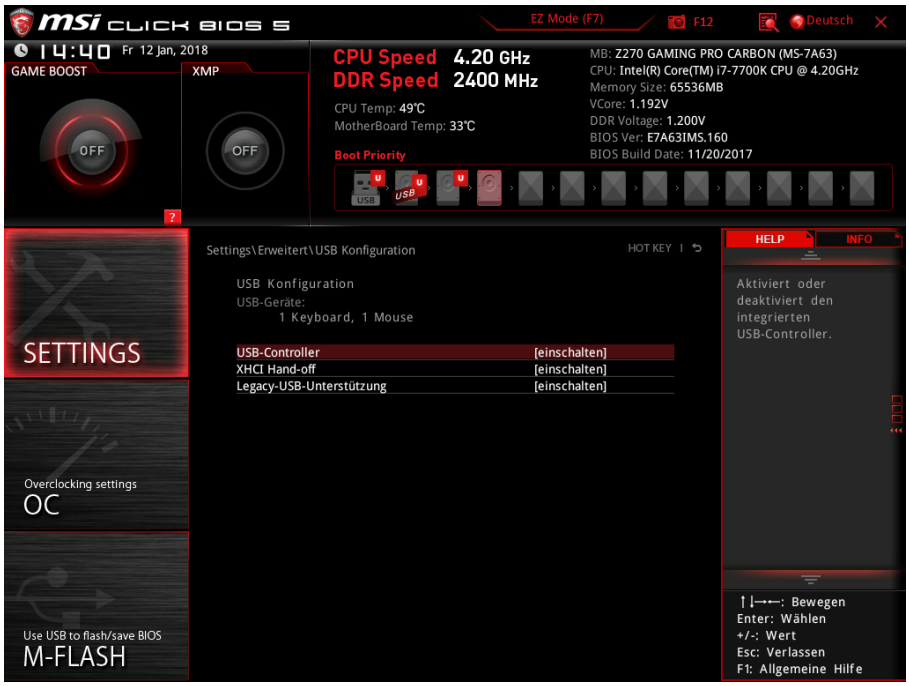

Abbildung 3.10: MSI BIOS: USB-Konfiguration

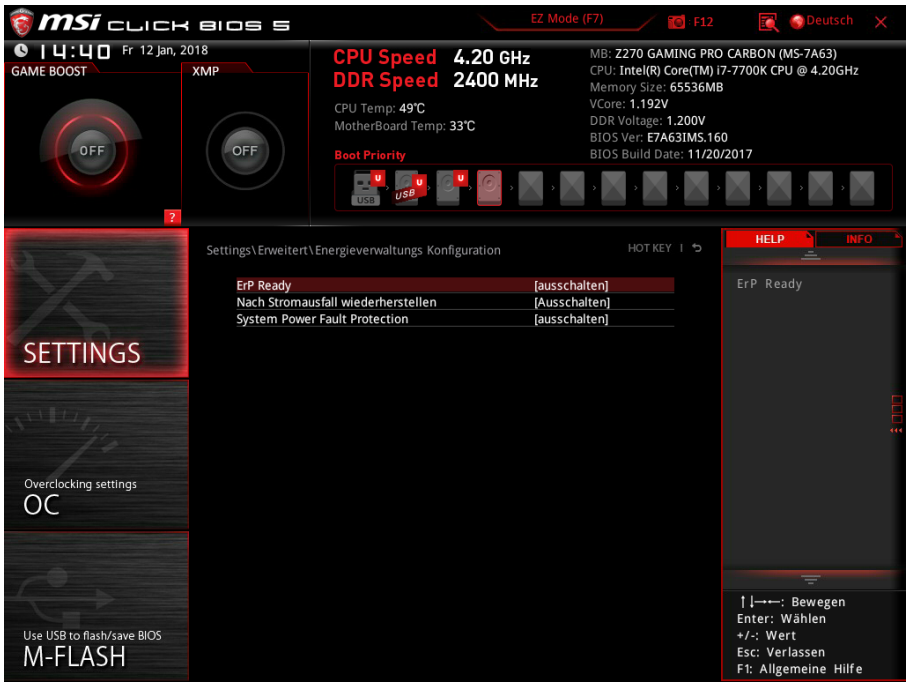

Abbildung 3.11: MSI BIOS: Energieverwaltungs-Konfiguration

Weil auf dem DL-Rechner kein Windows als Betriebssystem installiert wird, können alle entsprechenden Einstellungen hierzu ausgeschaltet werden (vgl. Abb. 3.12).

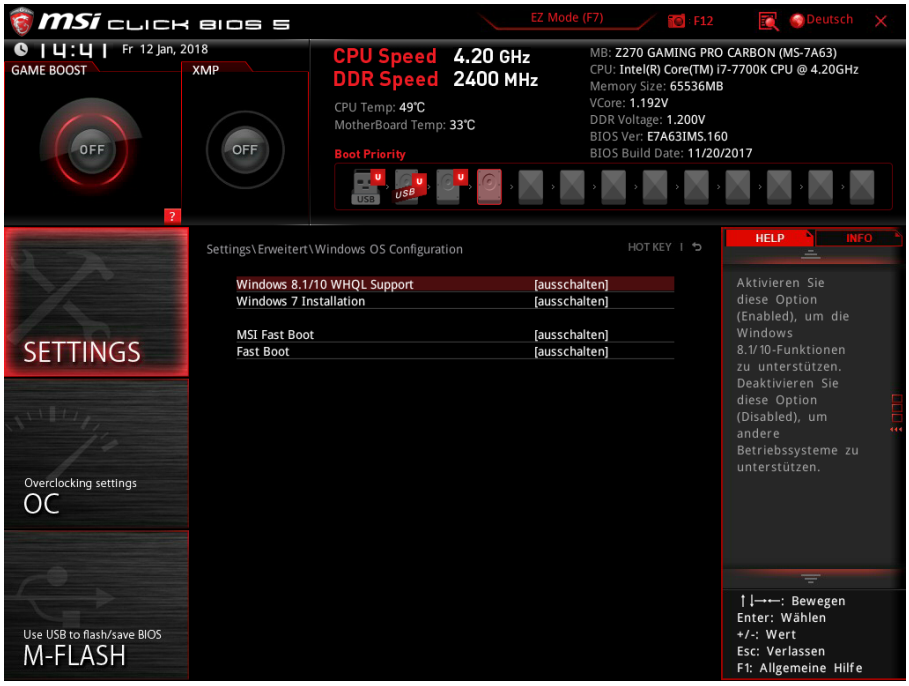

Abbildung 3.12: MSI BIOS: Windows OS Konfiguration

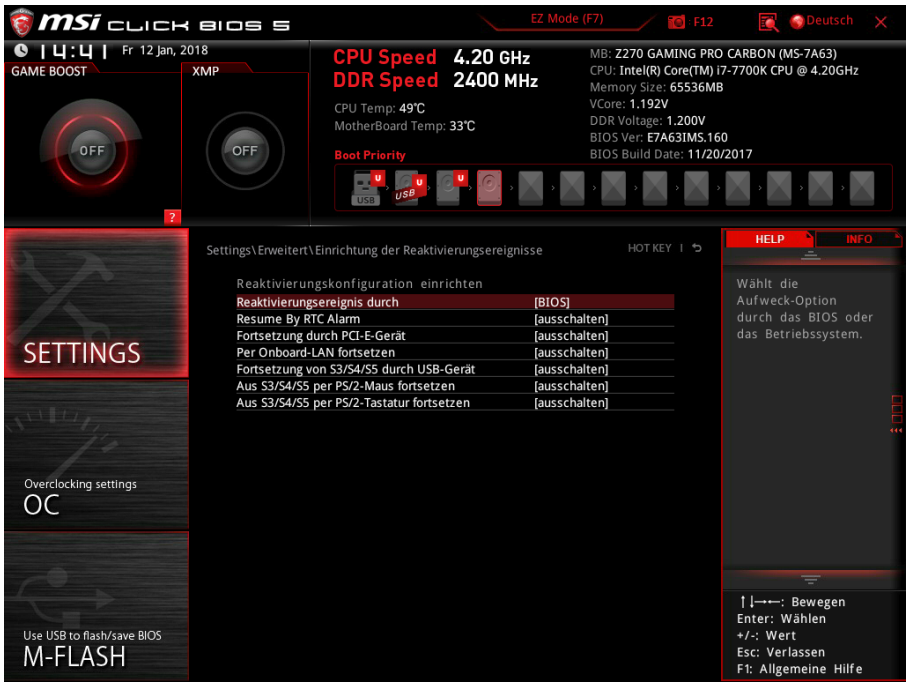

Abbildung 3.13: MSI BIOS: Einrichtung der Reaktivierungsereignisse

In der Boot-Konfiguration kann die Vollbild-Logoanzeige ausgeschaltet werden. Dadurch wird beim Start nicht das Logo zu dem MSI-Mainboard angezeigt. Die Boot-Reihenfolge sollte dann so geändert werden, dass zunächst externe USB-Medien angesprochen werden. Dies ist notwendig, damit Linux von einem bootfähigen USB-Stick aus installiert werden kann. Das Betriebssystem soll dann auf dem Samsung-SSD installiert werden soll. Somit bildet dieser Eintrag der letzte in der Auswahlliste.

Abbildung 3.14: MSI BIOS: Boot-Konfiguration

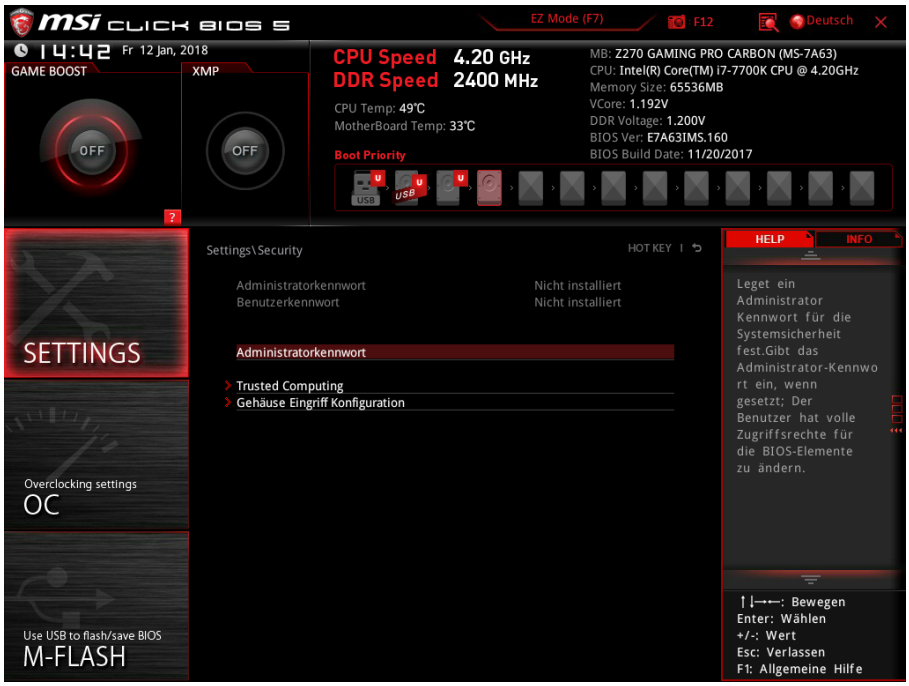

Abbildung 3.15: MSI BIOS: Security

Im Menü `Speichern und Verlassen` lassen sich die aktuellen Einstellungen dauerhaft speichern.

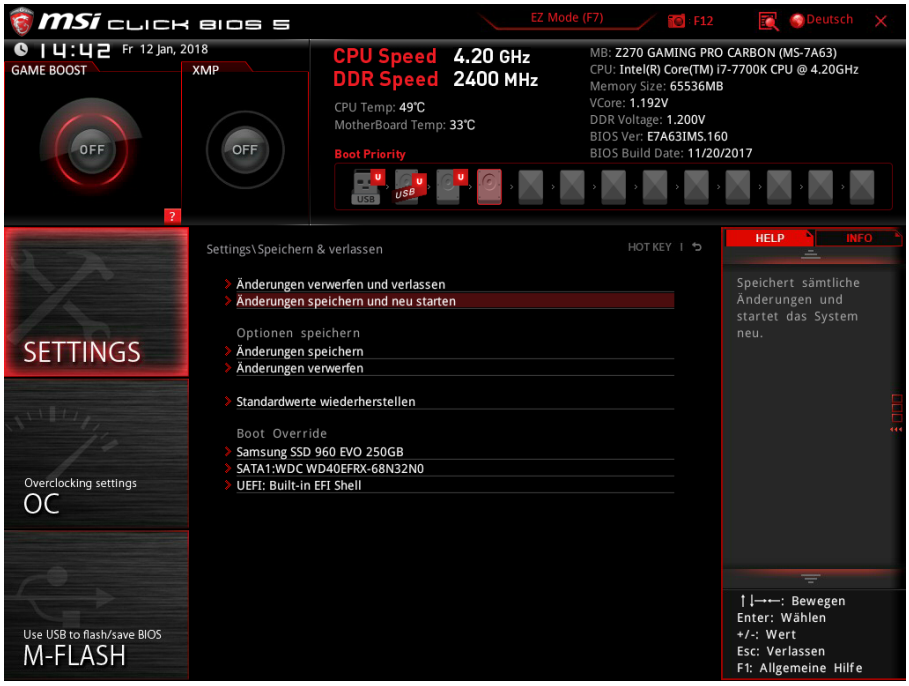

Abbildung 3.16: MSI BIOS: Speichern und Verlassen

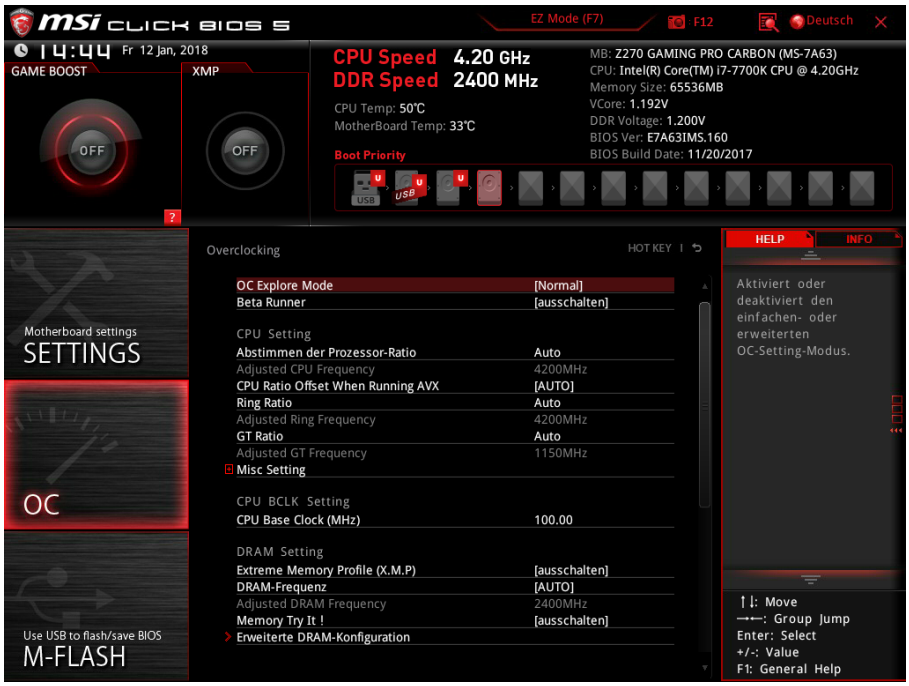

Abbildung 3.17: MSI BIOS: Overclocking

Mit der Funktionstaste F7 kann zwischen der Experten-Ansicht und Standard-Ansicht (vgl. Abb. 3.18) gewechselt werden.

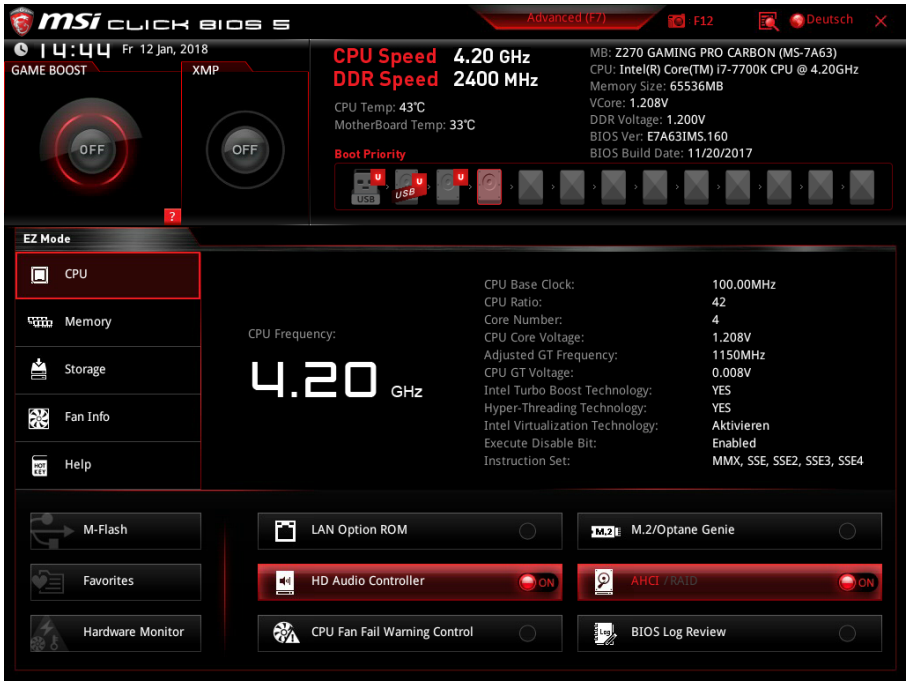

Abbildung 3.18: MSI BIOS: Standard-Ansicht

Mit dem Hardware-Monitor (vgl. Abb. 3.19) lässt sich der Systemzustand in Echtzeit überwachen.

Abbildung 3.19: MSI BIOS: Hardware-Monitor

4 Betriebssystem

In diesem Kapitel wird beschrieben, wie die Linux-Distribution Ubuntu 16.04 LTS als Betriebssystem auf dem DL-Rechner installiert wird. Früher wurde die Betriebssystem-Software normalerweise von CD oder DVD installiert. Mittlerweile kann man hierfür aber auch einen bootfähigen USB-Stick verwenden. Der DL-Rechner verfügt auch gar nicht über ein CD- oder DVD-Laufwerk. Im Abschnitt 4.1 wird daher zunächst beschrieben, wie ein solcher bootfähiger USB-Stick für die Installation vorbereitet wird. Im Abschnitt 4.2 wird dann das Vorgehen der Installation von Ubuntu schrittweise erklärt.

4.1 USB-Stick vorbereiten

Zunächst wird das zur Installation benötigte Ubuntu-Image aus dem Internet unter der URL http://releases.ubuntu.com/16.04/ heruntergeladen [Ubu18a]. Es wird die 64-Bit-Desktop-Version von Ubuntu 16.04.3 LTS (*Xenial Xerus*) verwendet, weil mehr als 4 GB RAM auf dem Mainboard verbaut sind und diese auch benutzt werden sollen. Außerdem unterstützt der Intel-Prozessor die 64-Bit-Version des x86-Instruktionssatzes. Die Datei mit dem Namen `ubuntu-16.04.3-desktop-amd64.iso` hat eine Größe von 1.550.400 kB, also ca. 1,5 GB. Die Endung ISO gibt an, dass es sich um ein Abbild einer CD bzw. DVD handelt (s.o.).

Das kostenlose Programm `Linux Live USB Creator` in der aktuellen Version 2.9.4 wird benutzt, um den USB-Stick vorzubereiten. Es kann unter der URL https://www.heise.de/download/product/linuxlive-usb-creator-90060 aus dem Internet heruntergeladen werden [Lau18]. Die Datei mit dem Namen `linuxlive_usb_creator_2.9.4.exe` hat eine Größe von 6.016 kB, also ca. 6 MB. Die Endung `EXE`, engl. für *executable*, gibt an, dass es sich um eine ausführbare Datei handelt. Per Doppelklick kann diese gestartet werden und das Programm wird installiert. Mit Hilfe der Abbildungen 4.1 bis 4.5 wird der Installationsverlauf dargestellt. Es muss nichts eingegeben oder geändert werden, man braucht nur die bereits aktiven Schaltflächen anzuklicken.

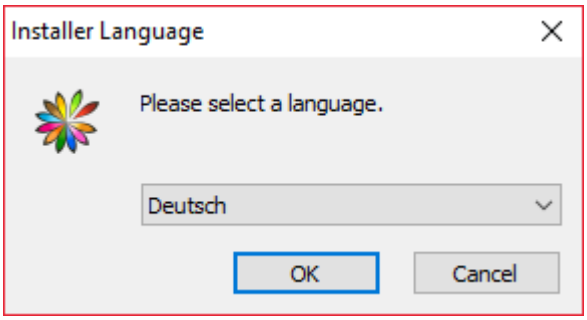

Abbildung 4.1: Linux Live USB Creator installieren: Schritt 1

Abbildung 4.2: Linux Live USB Creator installieren: Schritt 2

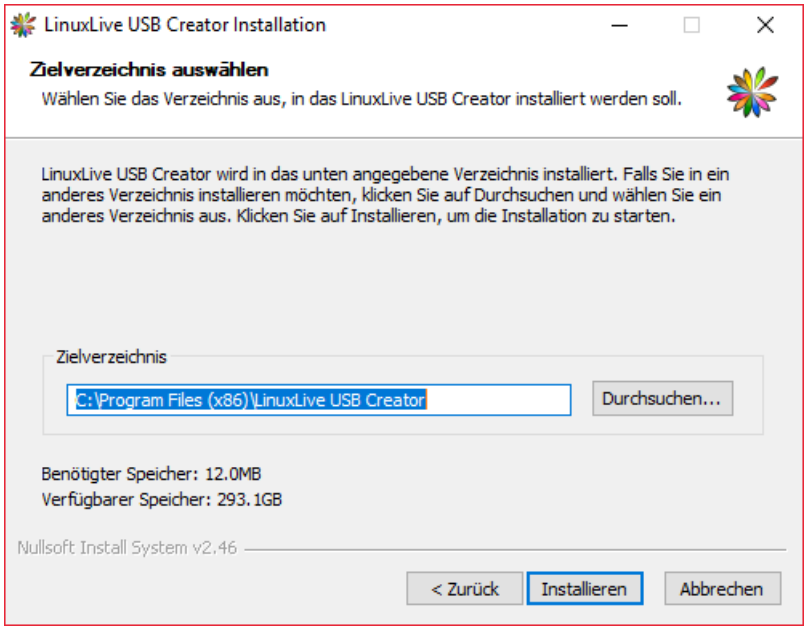

Abbildung 4.3: Linux Live USB Creator installieren: Schritt 3

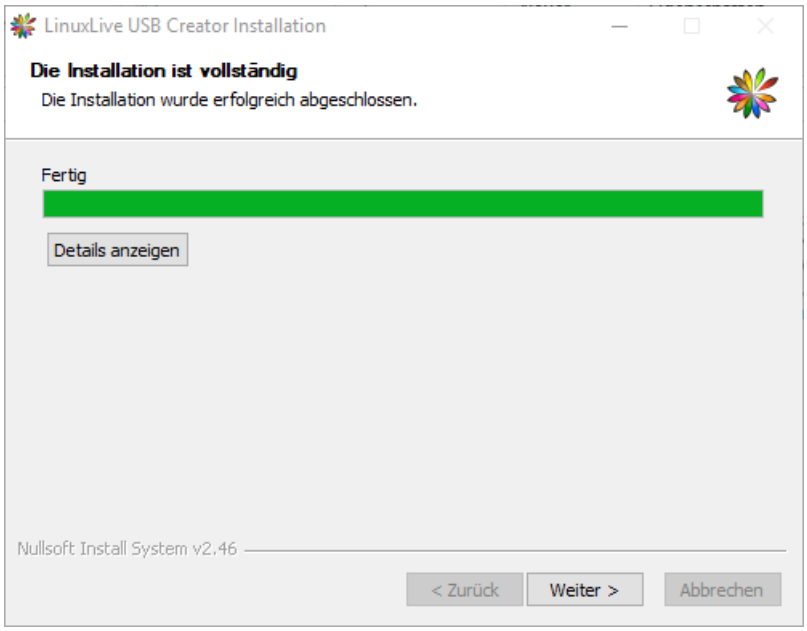

Abbildung 4.4: Linux Live USB Creator installieren: Schritt 4

Abbildung 4.5: Linux Live USB Creator installieren: Schritt 5

Nach der Installation startet das Programm `Linux Live USB Creator`. Es erscheint eine transparente Grafik, die in fünf Bereiche (Schritte 1 bis 5) unterteilt ist (vgl. Abb. 4.6).

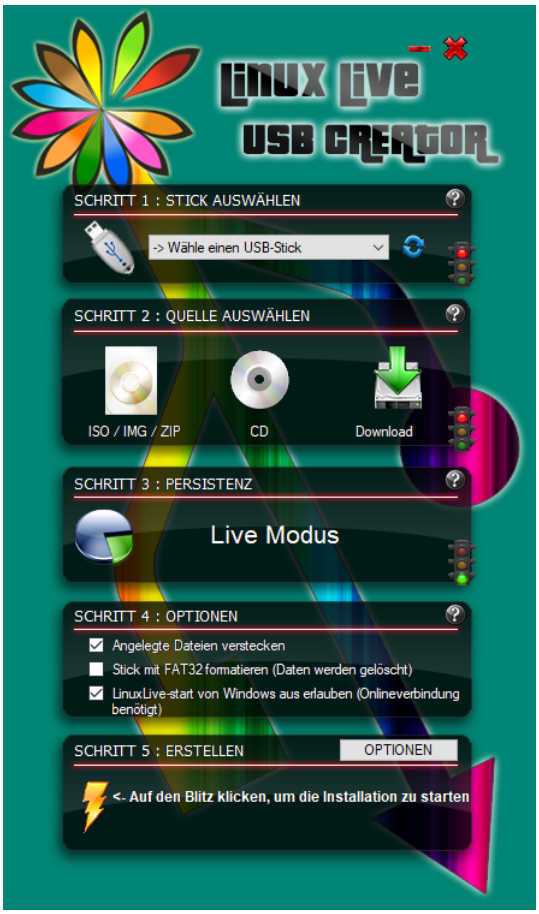

Abbildung 4.6: Linux Live USB Creator

Schritt für Schritt werden nun Einstellungen vorgenommen, die in Tab. 4.1 kurz dargestellt sind. Die folgenden Abbildungen 4.7 und 4.8 zeigen den Verlauf, um den USB-Stick

Schritt	Name	Einstellung, Wert, Aktion
1	Stick auswählen	D: KINGSTON-FAT32 - 14.4 GB
2	Quelle auswählen	ISO [`linuxlive_usb_creator_2.9.4.exe`]
3	Persistenz	4090 MB (Persistenter Modus)
4	Optionen	Alles aktivieren
5	Erstellen	Auf den Blitz klicken

Tabelle 4.1: Linux Live USB Creator: Einzelne Schritte

vorzubereiten. Im fünften und letzten Schritt muss dann nur noch auf das Blitz-Symbol geklickt werden, um den USB-Stick zu formatieren und zu bespielen.

Abbildung 4.7: USB-Stick vorbereiten: Schritt 1 und 2

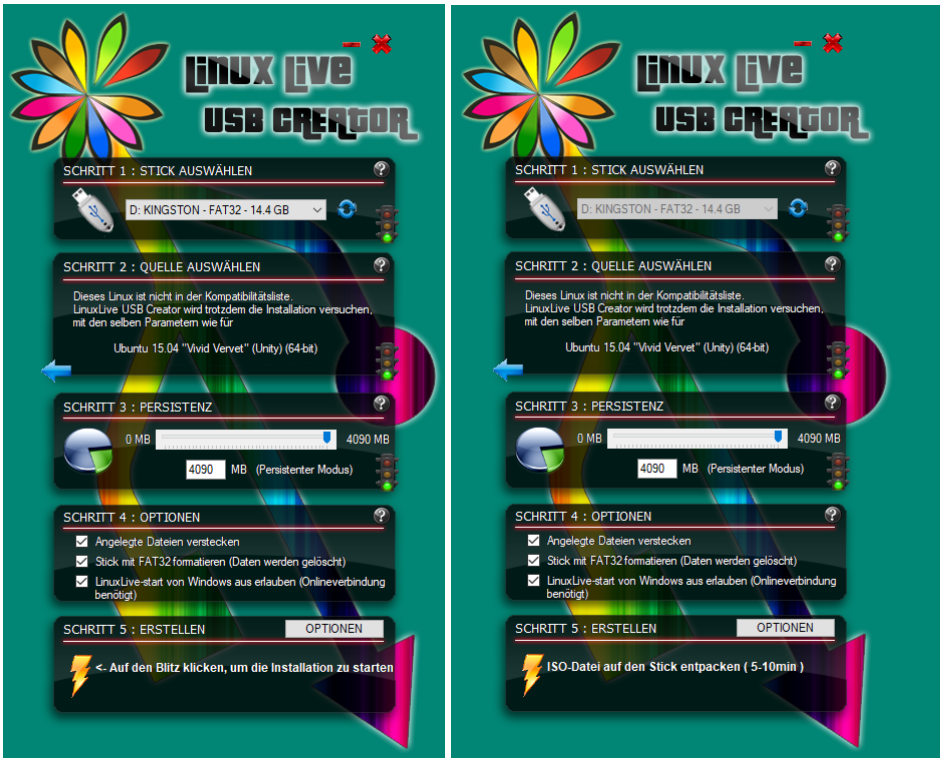

Abbildung 4.8: USB-Stick vorbereiten: Schritt 3 und 4

4.2 Ubuntu installieren

Damit Ubuntu installiert werden kann, muss die integrierte Grafikkarte und nicht die externe Grafikkarte benutzt werden. Hierzu sind im BIOS (vgl. Kap. 3) entsprechende Einstellungen vorzunehmen. Weil während der Installation keine Bildschirmfotos (engl. *Screenshots*) erstellt wurden, wird der Installationsprozess im Folgenden rein textuell beschrieben. Der zuvor erstellte USB-Stick (vgl. Kap. 4.1) wird nun am DL-Rechner angeschlossen und das System nach den BIOS-Änderungen wieder neu gestartet. Wenn der USB-Stick korrekt gebootet wird, dann meldet sich als erstes ein textbasiertes Programm namens GNU GRUB in der Version 2.02.

Schritt 1: GNU GRUB
«Install Ubuntu» auswählen.
«Enter» drücken.
Auf dem Monitor erscheinen nun im Installationsverlauf mehrere verschiedene Grafik-Fenster, in denen in der Titelleiste jeweils *Install (as superuser)* angezeigt wird.

Schritt 2: Welcome
«Deutsch» als Sprache auswählen.
«Weiter» klicken.

Schritt 3: Installation von Ubuntu wird vorbereitet
«Herunterladen der Aktualisierungen, während Ubuntu aktualisiert wird» aktivieren.
«Installation von Drittanbieter-Software für Grafik...» nicht aktivieren.
«Weiter» klicken.
Ggf. erscheint nun eine Warnung.
«Weiter im UEFI-Modus» klicken.

Schritt 4: Installationsart
«Festplatte löschen und Ubuntu installieren» auswählen.
«LVM bei der neuen Ubuntu-Installation verwenden» aktivieren.
«Weiter» klicken.
Hinweis: Mit dem Logical Volume Manager (LVM) lassen sich leicht Änderungen der Partitionen von vornehmen.

Schritt 5: Festplatte löschen und Ubuntu installieren
Laufwerk wählen: «/dev/nume0n1 250GB Unknown» auswählen.
«Jetzt installieren» klicken.
Ggf. erscheint nun die Meldung «Änderungen auf die Festplatten schreiben?».
«Weiter» klicken.

Schritt 6: Wo befinden Sie sich?
«Berlin» auswählen.
«Weiter» klicken.

Schritt 7: Tastaturbelegung
«Deutsch» auswählen.
«Weiter» klicken.

Schritt 8: Wer sind Sie?
Ihr Name: «Stefan Selle» eingeben.
Name Ihres Rechners: «selle-ki» eingeben.
Wählen Sie einen Benutzernamen: «stefan» eingeben.
Ein Passwort auswählen: «********» eingeben.
Passwort wiederholen: «********» eingeben.
«Passwort zum Anmelden abfragen» auswählen.
«Weiter» klicken.
Statt der Sternchen muss natürlich ein sinnvolles Passwort eingegeben werden.

Schritt 9: Installation Nun startet die eigentliche Installation. Diese kann einige Minuten in Anspruch nehmen. Am Ende erscheint die Meldung «Sie müssen jetzt den Rechner neu starten, um das System zu benutzen».
«Jetzt neu starten» klicken.
Nun muss auch der USB-Stick wieder entfernt werden. Beim Neustart muss außerdem die Ent-Taste gedrückt gehalten werden, um ins BIOS zu kommen. Hier muss ggf. noch die Boot-Reihenfolge angepasst werden (vgl. Abb. 4.9).

Abbildung 4.9: BIOS: Boot-Reihenfolge ändern

Nach dem Speichern der Änderungen und dem Neustart erscheint dann Ubuntu mit dem Login-Bildschirm. Nach Eingabe des bei der Installation vergebenen Passworts kommt man ins System und es wird die grafische Desktop-Arbeitsumgebung angezeigt (vgl. Abb. 4.10).

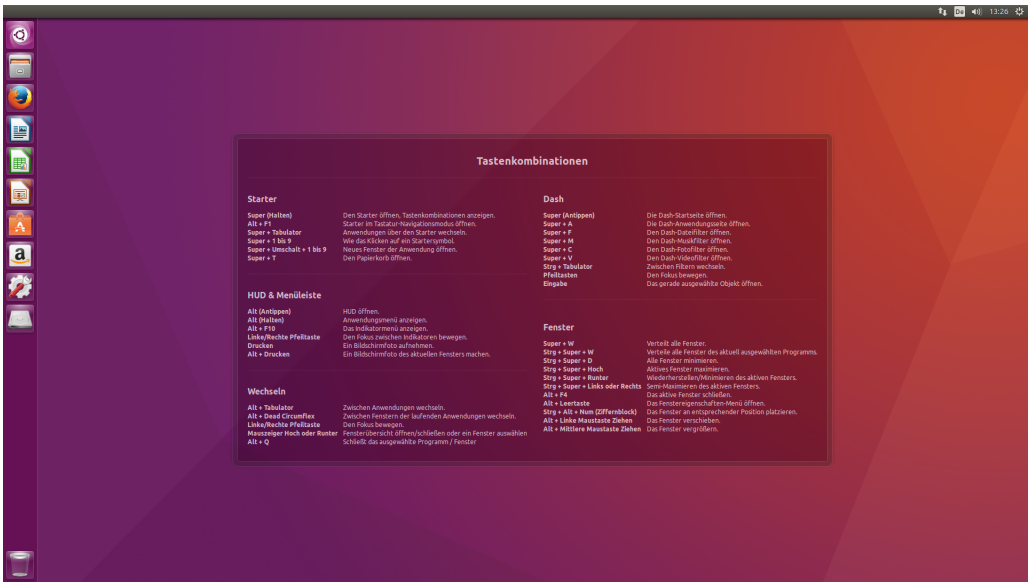

Abbildung 4.10: Ubuntu-Desktop

In den folgenden Kapiteln wird sehr häufig mit der Bash-Shell in einem Terminal-Fenster (vgl. Abb. 4.11) gearbeitet. Dieses kann man starten, indem man das Kontextmenü benutzt, d.h. die rechte Maustaste klickt, und dann die entsprechende Funktion aufruft. Alternativ ist auch die Tastenkombination `STRG + ALT + F1` bis `F6` möglich.

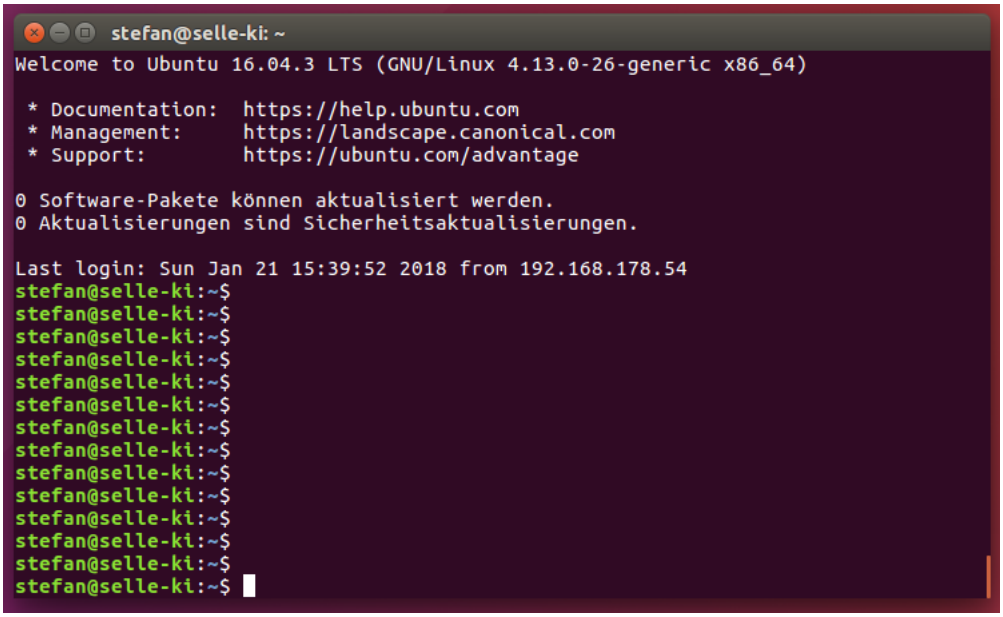

Abbildung 4.11: Terminal-Fenster mit Bash-Shell

5 Treiber und Tools

In diesem Kapitel wird beschrieben, wie Ubuntu 16.04 LTS konfiguriert wird, sodass ausgewählte DL-Softwarepakete einfach installiert und benutzt werden können. Alle Anpassungen werden in der Konsole, d.h. im Terminal-Fenster mit der Bash-Shell durchgeführt [Ubu18d]. Meistens muss das Schlüsselwort `sudo`, engl. für *substitute user do*, den Befehlen vorangestellt werden, um diese als *Root*, d.h. als Administrator mit allen Berechtigungen, auszuführen [Ubu18as].

5.1 Basis-Konfiguration

Ubuntu aktualisieren Es ist sinnvoll, das Betriebssystem regelmäßig zu pflegen und warten. Neue, verbesserte Programmversionen erhöhen die Stabilität und Sicherheit des Systems. Insbesondere vor der Installation von neuen Programmen, sollte man zunächst das Betriebssystem auf dem neuesten Stand bringen. Mit dem Befehl `apt-get` lassen sich Softwarepakete hinzufügen, entfernen, installieren, aktualisieren usw. [Ubu18r]. APT steht für *Advanced Packaging Tool* [Ubu18b].

```
# Paketlisten neu einlesen
sudo apt-get update
# Bereits installierte Softwarepakete auf neue, verbesserte Versionen aktualisieren
sudo apt-get upgrade
```

Listing 5.1: Ubuntu aktualisieren (apt-get)

Neue Verzeichnisse anlegen Es werden zwei neue Verzeichnisse `/install` und `/data` erstellt. Das erste wird zur Installation von Software verwendet, das zweite als Speicherplatz für Datenanalysen. Unter Ubuntu gibt es bereits das Verzeichnis `/opt` (für optional), in dem normalerweise Programme installiert werden, die nicht *Open Source* sind oder die an der Paketverwaltung APT (s.o.) vorbei installiert werden sollen [Ubu18ao]. Im Unterschied zum Verzeichnis `/opt` bekommt das Verzeichnis `/install` über alle Gruppen die volle Berechtigung, sodass Installationsprobleme aufgrund fehlender Berechtigungen direkt entgegen gewirkt werden kann. Im späteren, produktiven Betrieb sollten dann die Berechtigungen wieder geändert werden oder das Verzeichnis `/opt` verwendet werden.

Verzeichnisse werden mit dem Befehl `mkdir`, engl. für *make directory*, angelegt [Ubu18ak]. Die Berechtigung lässt sich mit dem Befehl `chmod`, engl. für *change mode*, ändern [Ubu18u]. Dabei werden normalerweise drei Ziffern als Option im Oktal-Modus angegeben. Die erste Ziffer bezieht sich auf den Besitzer, die zweite auf die Gruppe und die dritte auf alle anderen Benutzer. Jede Ziffer ist eine Linearkombination aus 1 (für Ausführen), 2 für (Schreiben) und 4 (für Lesen). Bspw. bedeuten die Ziffern `754`, dass der Besitzer die volle Berechtigung hat, während Benutzer der gleichen Gruppe keine Schreibrechte und alle übrigen Benutzer nur Leserechte haben. Der `chmod`-Befehl kann auf einzelne Dateien und auf Verzeichnisse angewendet werden, auch rekursiv, falls man die Option `-R` verwendet. Mittels `chown`-Befehl, engl. für *change owner*, lässt sich der Besitzer einer Datei oder eines Verzeichnisses bei Bedarf ändern [Ubu18v].

```
# Neues Verzeichnis zur Installation von externer Software anlegen
sudo mkdir /install
# Umfassende Berechtigung für alle Benutzer für dieses neue Verzeichnis vergeben
sudo chmod -R 777 /install
# Neues Verzeichnis für die Daten anlegen
sudo mkdir /data
```

Listing 5.2: Neue Verzeichnisse anlegen (mkdir)

Daten-Festplatte einrichten Die 4TB-Festplatte (engl. *Hard Disk*) soll als Datenspeicher genutzt werden und deshalb auf das neu angelegte Verzeichnis `/data` (s.o.) eingehängt (engl. *mount*) werden [Ubu18am]. Hierzu muss das Gerät (engl. *Device*, abgekürzt *dev*) zunächst mittels `fdisk` partitioniert [Ubu18ac] und die Partition(en) formatiert werden [Ubu18g]. Als Dateisystem wird `ext4` verwendet. Dieses Journal gilt als *State-of-the-Art* im Linux-Bereich [Ubu18ab]. Jede Partition bekommt intern eine eindeutige Identifikationsnummer, die sogenannte *Universally Unique Identifier (UUID)* [Ubu18n]. Diese Nummer kann mittels `blkid`-Kommando abgefragt werden [Ubu18s]. Sie wird benötigt, damit die Partition bei jedem Neustart des Computers automatisch auf das Verzeichnis `/data` gemountet wird. Die Informationen hierzu müssen in der Konfigurationsdatei `fstab` eingetragen werden [Ubu18ae]. Diese Datei enthält die Tabelle (engl. *Table*, abgekürzt *tab*) zu dem Dateisystem (engl. *File System*, abgekürzt *fs*).

```
# Formatierungstool installieren
sudo apt-get install e2fsprogs
# Alle angeschlossenen Festplatten auflisten
sudo fdisk -l
# Festplatte partitionieren (exemplarisch für das Device sda)
sudo fdisk /dev/sda
# n für neue Partition eingeben
# Festplatte mit dem Dateisystem ext4 formatieren
sudo mkfs.ext4 /dev/sda1
# Abfragen, welche UUID die Partitionen haben
sudo blkid
# Konfigurationsdatei editieren
sudo vi /etc/fstab
```

Listing 5.3: Daten-Festplatte einrichten (fdisk mkfs etc.)

Mit dem Befehl `vi` öffnet sich ein einfacher Editor [Ubu18q]. Es wird die Konfigurationsdatei `fstab` geladen, an deren Ende der folgende Eintrag hinzugefügt wird.

```
UUID=***        /data    ext4            defaults        0       0
```

Listing 5.4: Konfigurationsdatei /etc/fstab

Die Sternchen müssen dabei natürlich durch die UUID (s.o.) ersetzt werden. Nachdem diese Änderungen gespeichert wurden (z.B. mit den Tastaturbefehlen `:w` oder `ZZ`), sollte ein Neustart des Rechners vorgenommen werden. Nach dem Neustart kann man sich bspw. mit dem einfachen Befehl `df`, engl. für *disk free*, vergewissern, dass die 4TB-Platte nun korrekt auf das Verzeichnis `/data` gemountet ist [Ubu18x]. Weitere, nützliche Linux-Kommandos [Ubu18e] im Umgang mit Dateien und Verzeichnissen sind: `cd` (engl. für *change directory*) [Ubu18t], `ls` (engl. für *list*) [Ubu18aj], `du` (engl. für *disk usage*) [Ubu18z], `cp` (engl. für *copy*) [Ubu18w], `mv` (engl. für *move*) [Ubu18an], `rm` (engl. für *remove*) [Ubu18ar], `ln` (engl. für *link*) [Ubu18ai], `find` [Ubu18ad], `which` [Ubu18au], `more` [Ubu18al] und `less` [Ubu18ah].

```
# Anzeigen, wie viel Platz auf dem Dateisystem verfügbar ist (df = disk free)
df
# Ins Verzeichnis /data wechseln (cd = change directory)
cd /data
# Inhalt eines Verzeichnisses anzeigen lassen (list). Mit Option -l für die Langform
ls -l
# Den belegten Plattenplatz nach Verzeichnissen gruppiert ausgeben (du = disk usage).
# Mit Option -h für menschenlesbare Zahlen (-human-readable)
du -h
# Dateien / Verzeichnisse kopieren (cp = copy)
cp <source> <destination>
# Dateien / Verzeichnisse verschieben oder umbenennen (mv = move)
mv <source> <destination>
# Dateien / Verzeichnisse löschen (rm = remove)
rm <file>
# Eine (symbolische) Verknüpfung erstellen (ln = link)
ln -s <source> <destination>
# Nach Dateien / Verzeichnissen suchen
find <file>
# Nach ausführbaren Programmen suchen
which <command>
# Dateiinhalte ausgeben (v1)
more <file>
# Dateiinhalte ausgeben (v2)
less <file>
```

Listing 5.5: Einige nützliche Linux-Befehle

Download-Tools installieren Mittels des Befehls `wget` lassen sich aus dem Terminal Dateien aus dem Internet herunterladen [Ubu18at]. Git ist ein nützliches Werkzeug zur Versionsverwaltung von Software [Ubu18j]. Mit Hilfe des Befehls `git` lässt sich die aktuellste Version eines Softwareprojektes von der Plattform *GitHub* herunterladen.

```
# wget installieren
sudo apt-get install wget
# git installieren
sudo apt-get install git
```

Listing 5.6: Download-Tools installieren (wget und git)

5.2 GPU-Konfiguration

CUDA installieren CUDA ist eine Software von Nvidia, um auf deren Grafikprozessoren Rechnungen parallelisiert durchzuführen. Auf der Nvidia-Webseite werden zwei Möglichkeiten vorgestellt, wie das CUDA Toolkit v9.1.85 unter Ubuntu 16.04 LTS installiert werden kann [Nvi17]: *Paket Manager Installation* oder *Runfile Installation*. Die Variante per APT installiert automatisch Nvidia-Treiber für die GPU GTX 1080 Ti. Leider funktionieren diese aber nicht korrekt und es kommt infolge dessen immer wieder zu Programmabstürzen (vgl. Abb. 5.1).

In der zweiten Variante dagegen, kann man während des Installationsvorgangs wählen, dass man die Grafikkarten-Treiber nicht installieren möchte. Diese lassen sich dann nachträglich auch manuell installieren. Damit diese Variante aber zum Erfolg führt, müssen einige notwendige Vorarbeiten erledigt werden.

Abbildung 5.1: Absturzbericht zum Grafikkarten-Treiber

```
# Alle Softwarepakere von Nvidia entfernen
sudo apt-get --purge remove nvidia-*
# In das Installationsverzeichnis wechseln (cd = change directory)
cd /install
# Die Installationsdatei (runfile) herunterladen
wget https://developer.nvidia.com/compute/cuda/9.1/Prod/local_installers/
    cuda_9.1.85_387.26_linux
# Konfigurationsdatei /etc/modprobe.d/blacklist-nouveau.conf anlegen
sudo vi /etc/modprobe.d/blacklist-nouveau.conf
```

Listing 5.7: Nvidia-Installation vorbereiten

Im `vi`-Editor müssen nun zwei Zeilen zur noch leeren Konfigurationsdatei hinzugefügt werden.

```
blacklist nouveau
options nouveau modeset=0
```

Listing 5.8: Konfigurationsdatei /etc/modprobe.d/blacklist-nouveau.conf

Nouveau ist der *Open Source* Grafikkarten-Treiber für Nvidia-GPUs. Diese schwarze Liste sorgt dafür, dass keine solchen Grafikkarten-Treiber automatisch vom System installiert werden. Jetzt muss noch die Boot-Umgebung für den Linux-Kernel aktualisiert und das System neu gestartet werden.

```
# Boot-Umgebung aktualisieren
sudo update-initramfs -u
# System neu starten
sudo reboot
```

Listing 5.9: Änderungen abschließen

Nach dem Neustart (engl. *Reboot*) sollte man sich sicherheitshalber nicht an der grafischen Desktop-Umgebung von Ubuntu anmelden. Stattdessen drückt man die Tasten-Kombination `STRG + ALT + F1`. Dadurch kommt man direkt auf ein Terminal und kann sich anmelden. Mit den folgenden Befehlen lässt sich nun CUDA installieren.

```
# In das Installationsverzeichnis wechseln (cd = change directory)
cd /install
# Installationsskript als eigene Shell ausführen (sh = shell)
sudo sh cuda_9.1.85_387.26_linux.run
```

Listing 5.10: CUDA installieren

Der Installationsvorgang erfolgt interaktiv in sieben Schritten, die nachfolgend exemplarisch dargestellt sind. Zunächst muss dem Endbenutzer-Lizenzvertrag (engl. *End User License Agreement (EULA)*) zugestimmt werden, anschließend muss man darauf achten, die Grafikkarten-Treiber nicht zu installieren usw.

Schritt 1: Bestätigung der EULA
«Do you accept the previously read EULA?»
«acceppt» eingeben.

Schritt 2: Nvidia-Treiber nicht installieren
«Install NVIDIA Accelerated Graphics Driver for linux-x86_64 387.26?»
«n» für Nein (engl. *no*) eingeben.

Schritt 3: CUDA-Toolkit installieren
«Install the CUDA 9.1 Toolkit?»
«y» für Ja (engl. *yes*) eingeben.

Schritt 4: Verzeichnis festlegen
«Enter Toolkit Location (default: /usr/local/cuda-9.1)»
«ENTER» drücken, um das angegebene Standardverzeichnis zu verwenden.

Schritt 5: Symbolischen Link anlegen
«Do you want to install a symbolic link at /usr/local/cuda?»
«y» für Ja (engl. *yes*) eingeben.

Schritt 6: Beispiele installieren
«Install the CUDA 9.1 samples?»
«y» für Ja (engl. *yes*) eingeben.

Schritt 7: Beispiel-Verzeichnis angeben
«Enter CUDA samples location»
«/data» eingeben.

Nach der Installation sollte man noch die Umgebungsvariablen anpassen [Ubu18o]. Mit dem Kommando `env`, engl. für *environment*, lassen sich diese anzeigen. Zum Ändern muss die Datei `.bashrc` im Home-Verzeichnis editiert werden.

```
# Aktuelle Umgebungsvariablen anzeigen lassen (env = environment)
env
# Bash-Konfiguration editieren
vi ~/.bashrc
```

Listing 5.11: Umgebungsvariablen (env)

Am Ende der Datei sollten die folgenden Zeilen hinzugefügt werden:

```
# PATH and LD_LIBRARY_PATH
export PATH=/usr/local/cuda/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
```

Listing 5.12: Konfigurationsdatei .bashrc

Diese neuen Umgebungsvariablen bewirken, dass CUDA-Programme wie bspw. `nvcc` von überall aus aufgerufen werden können, weil nun in `PATH` der Pfad zu dem ausführbaren Programm enthalten ist. Damit der Programmaufruf auch funktioniert, müssen die geänderten Umgebungsvariablen neu geladen werden. Dies kann man erreichen, indem man auf dem grafischen Desktop ein neues Terminal-Fenster öffnet oder mit `STRG + ALT + F1` bis `STRG + ALT + F6` in ein Terminal wechselt. In dem bestehenden Terminal kann aber auch der `source`-Befehl benutzt werden, um das Neuladen der Umgebungsvariablen zu erreichen.

```
# .bashrc-Datei neu einlesen und ausführen
source ~/.bashrc
# Aktuelle Umgebungsvariablen anzeigen lassen
env
# Nvidia CUDA Compiler aufrufen, um Installation zu testen
nvcc --version
```

Listing 5.13: CUDA-Test

Als Ergebnis sollte nun folgende Ausgabe erscheinen:

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2017 NVIDIA Corporation
Built on Fri_Nov__3_21:07:56_CDT_2017
Cuda compilation tools, release 9.1, V9.1.85
```

Nvidia-Treiber installieren Mit dem Befehl `lspci` kann man sich einen Überblick über Geräte verschaffen, die über PCI am Rechner angeschlossen sind [Ubu18i]. Spezielle Optionen geben dabei an, dass sowohl Nummern als auch Namen (`-nn`) und die Kernel-Treiber (`-k`) angezeigt werden. Das Ergebnis lässt sich dann per *Pipe*-Operator (Symbol `|`) an einen anderen Befehl weiterleiten [Ubu18p]: Mittels `grep`-Kommando wird im Ergebnis nach Begriffen wie VGA, Kern, 3D und Display gesucht [Ubu18af]. Dabei kommen sogenannte reguläre Ausdrücke (engl. *Regular Expressions*) zum Einsatz. Normalerweise wird als Ergebnis dann immer die Textzeile angezeigt, in der der Suchbegriff vorkommt. Die Option `-i` ignoriert bei der Suche die Groß- und Kleinschreibung, während `-A2` bewirkt, dass noch zwei weitere Textzeilen ausgegeben werden.

```
# Grafikkarten auflisten
lspci -nnk | grep -i "VGA\|'Kern'\|3D\|Display" -A2
```

Listing 5.14: Grafikkarten anzeigen (lspci und grep)

Als Ergebnis sollte man somit eine Übersicht über die im Rechner verbauten Grafikkarten bekommen.

```
00:02.0 VGA compatible controller [0300]: Intel Corporation Device [8086:5912] (rev 04)
      DeviceName:  Onboard IGD
      Subsystem: Micro-Star International Co., Ltd. [MSI] Device [1462:7a63]
--
01:00.0 VGA compatible controller [0300]: NVIDIA Corporation Device [10de:1b06] (rev a1)
      Subsystem: Gigabyte Technology Co., Ltd Device [1458:3752]
      Kernel driver in use: nouveau
```

Mit dem *Advanced Packaging Tool* kann man nach Nvidia-Treibern suchen (engl. *search*). Das Ergebnis lässt sich wieder per *Pipeline* an den `grep`-Befehl weiterleiten. Diesmal wird die erweiterte Variante (engl. *extended*, Option `-E`) benutzt und nur die gefundene, passende Zeichenkette und nicht die ganze Textzeile ausgegeben (Option `-o`, engl. für *only matching*). Der reguläre Ausdruck benutzt, dass die Versionen der Treiber von Nvidia hinter dem Namen als Ziffernfolge angegeben ist.

```
# Nach Grafikkarten-Treiber von Nvidia suchen
apt search nvidia | grep -E -o 'nvidia\-[0-9]+(\/|([ ]+\-))'
```

Listing 5.15: Grafikkarten-Treiber (apt search und grep)

Ergebnis:

```
nvidia-304/
nvidia-331/
nvidia-340/
...
nvidia-375/
nvidia-384/
nvidia-387/
```

Mit dem *Debian Package Manager*, d.h. dem Befehl `dpkg`, kann man ebenfalls feststellen, welche Nvidia-Pakete vorhanden sind, wobei die Option `-l` diese Softwarepakete auflistet [Ubu18y]. Dabei wird der Status, die Version und eine Kurzbeschreibung der Pakete als Liste ausgegeben. Beim Status bedeutet `i` als erstes Zeichen, dass man dieses Paket installieren kann und `i` als zweites Zeichen, dass das Paket bereits vollständig installiert ist. Ist das erste Zeichen ein `u`, dann ist der Status unbekannt. Ein `n` als zweites Zeichen steht für nicht installiert.

```
# Softwarepakete von Nvidia auflisten
dpkg -l nvidia*
```

Listing 5.16: Nvidia-Softwarepakete (dpkg)

Ergebnis:

```
Gewünscht=Unbekannt/Installieren/R=Entfernen/P=Vollständig Löschen/Halten
| Status=Nicht/Installiert/Config/U=Entpackt/halb konFiguriert/
         Halb installiert/Trigger erWartet/Trigger anhängig
|/ Fehler?=(kein)/R=Neuinstallation notwendig (Status, Fehler: GROSS=schlecht)
||/ Name           Version      Architektur  Beschreibung
+++-==============-============-============-================================
in  nvidia-387     387.26-0ubun amd64        NVIDIA binary driver - version 38
in  nvidia-387-dev 387.26-0ubun amd64        NVIDIA binary Xorg driver develop
un  nvidia-common  <keine>      <keine>      (keine Beschreibung vorhanden)
un  nvidia-current <keine>      <keine>      (keine Beschreibung vorhanden)
un  nvidia-driver- <keine>      <keine>      (keine Beschreibung vorhanden)
un  nvidia-legacy- <keine>      <keine>      (keine Beschreibung vorhanden)
un  nvidia-libopen <keine>      <keine>      (keine Beschreibung vorhanden)
un  nvidia-libopen <keine>      <keine>      (keine Beschreibung vorhanden)
in  nvidia-modprob 387.26-0ubun amd64        Load the NVIDIA kernel driver and
un  nvidia-opencl- <keine>      <keine>      (keine Beschreibung vorhanden)
in  nvidia-opencl- 387.26-0ubun amd64        NVIDIA OpenCL ICD
un  nvidia-persist <keine>      <keine>      (keine Beschreibung vorhanden)
in  nvidia-prime   0.8.2        amd64        Tools to enable NVIDIA's Prime
in  nvidia-setting 387.26-0ubun amd64        Tool for configuring the NVIDIA g
un  nvidia-setting <keine>      <keine>      (keine Beschreibung vorhanden)
un  nvidia-smi     <keine>      <keine>      (keine Beschreibung vorhanden)
un  nvidia-vdpau-d <keine>      <keine>      (keine Beschreibung vorhanden)
```

Für Nvidia-Grafikkarten werden normalerweise unter Ubuntu die Nouveau-Treiber als *Open Source* installiert. Diese bereiten jedoch Probleme und wurden deshalb bereits auf eine schwarze Liste gesetzt (s.o.). Um nun die Installation von den proprietären Nvidia-Treibern zu ermöglichen, kann man Softwarepakete von solchen Drittanbietern mit Hilfe des sogenannten *Personal Package Archive (PPA)* den regulären Ubuntu-Paketen hinzufügen [Ubu18k]. Mit den folgenden Kommandos wird dies realisiert und dann der neuste proprietären Nvidia-Treiber installiert.

```
# Grafikkarten-Treiber von Drittanbieter ermöglichen
sudo add-apt-repository ppa:graphics-drivers/ppa
# Paketlisten neu einlesen
sudo apt-get update
# Aktuellen Treiber installieren (Version 387)
sudo apt-get install nvidia-387
```

Listing 5.17: Grafikkarten-Treiber installieren

Das *System Management Interface (SMI)* von Nvidia ist ein Kommandozeilenwerkzeug (Befehl `nvidia-smi`), mit dem man Informationen zu GPUs abrufen kann und sich außerdem der aktuelle Zustand überfachen lässt.

```
# Informationen zu GPUs abrufen
nvidia-smi
```

Listing 5.18: Grafikkarten-Treiber testen (smi)

Ergebnis:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 387.34                 Driver Version: 387.34                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 108...  Off  | 00000000:01:00.0 Off |                  N/A |
|  0%   32C    P5    24W / 250W |    200MiB / 11172MiB |      2%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1077      G   /usr/lib/xorg/Xorg                           157MiB |
|    0      1861      G   compiz                                        41MiB |
+-----------------------------------------------------------------------------+
```

Im negativen Fall bekommt man die folgende Meldung:

```
NVIDIA-SMI has failed because it couldn't communicate with the NVIDIA driver.
Make sure that the latest NVIDIA driver is installed and running.
```

In diesem Fall sollte man einen anderen der Treiber aus der Liste ausprobieren. Wenn man nach der erfolgreichen Installation wieder den `lspci`-Befehl (s.o.) aufruft, dann sollte diesmal nicht *nouveau* sondern *nvidia* angezeigt werden.

```
01:00.0 VGA compatible controller [0300]: NVIDIA Corporation Device [10de:1b06] (rev a1)
      Subsystem: Gigabyte Technology Co., Ltd Device [1458:3752]
      Kernel driver in use: nvidia
```

CuDNN installieren Das Akronym CuDNN steht für *CUDA Deep Neural Network Library*. Es ist eine Software-Bibliothek von Nvidia, die auf CUDA (s.o.) aufsetzt und mit der tiefe Künstliche Neuronale Netzwerke (KNN) auf dem Grafikprozessor trainiert werden können. D.h. hierdurch wird das *Deep Learning* softwaretechnisch also erst ermöglicht. Damit man CuDNN installieren kann, benötigt man zunächst einen Mitgliedslogin für Entwickler bei Nvidia. Auf der Webseite https://developer.nvidia.com/rdp/form/cudnn-download-survey kann man sich kostenlos hierfür registrieren [Nvi18]. Danach ist es möglich, die zum CUDA Toolkit passende Version von CuDNN herunterzuladen (vgl. Abb. 5.2). In unserem Fall ist dies: `cuDNN v7.0.5 (Dec 11, 2017), for CUDA 9.1`.

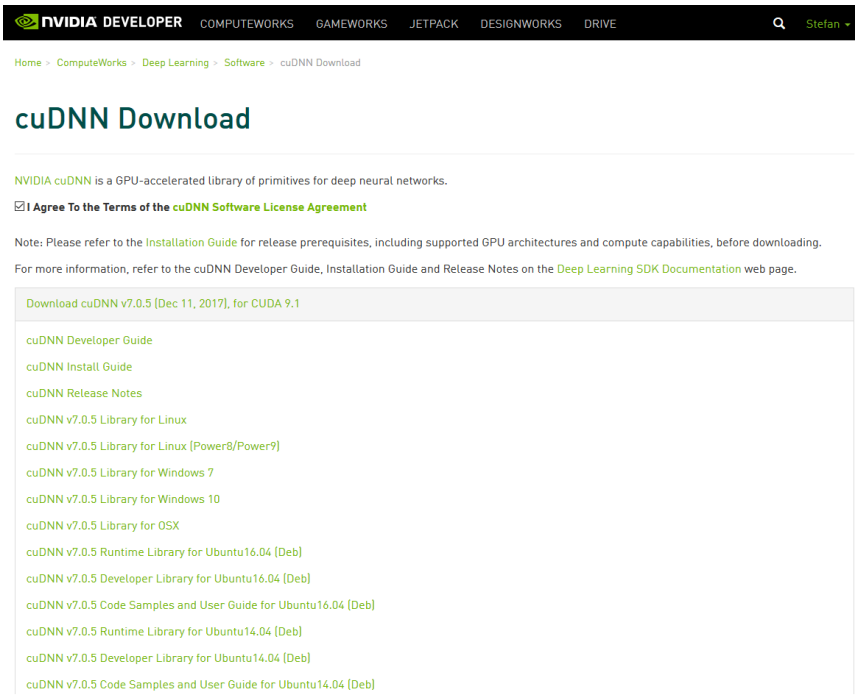

Abbildung 5.2: cuDNN-Installationsdateien

Für Ubuntu 16.04. gibt es drei Installationsdateien:
- cuDNN v7.0.5 Runtime Library for Ubuntu16.04 (Deb)
- cuDNN v7.0.5 Developer Library for Ubuntu16.04 (Deb)
- cuDNN v7.0.5 Code Samples and User Guide for Ubuntu16.04 (Deb)

Diese drei Dateien sollten nach dem Herunterladen ins Verzeichnis `/install` verschoben werden. Sie haben die Endung `deb`, denn zum Installieren (Option `-i`) wird der *Debian Package Manager* benutzt [Ubu18y].

```
# Ins Installationsverzeichnis wechseln
cd /install
# Debian Package Manager zur Installation benutzen
sudo dpkg -i libcudnn7_7.0.5.15-1+cuda9.1_amd64.deb
sudo dpkg -i libcudnn7-dev_7.0.5.15-1+cuda9.1_amd64.deb
sudo dpkg -i libcudnn7-doc_7.0.5.15-1+cuda9.1_amd64.deb
```

Listing 5.19: cuDNN installieren

Zum Testen, ob die Installation von cuDNN erfolgreich war, kann man die folgenden Befehle nutzen, um ein Beispiel zu kompilieren [Ubu18y] und dann auszuführen. Wenn am Ende die Meldung «`Test passed!`» erscheint, war die Installation erfolgreich.

```
# Beispiele rekursiv (-r) ins Homeverzeichnis kopieren (wegen Schreibberechtigung)
cp -r /usr/src/cudnn_samples_v7/ /home/stefan/
# Zum MNIST-CuDNN-Beispiel wechseln
cd /home/stefan/cudnn_samples_v7/mnistCUDNN
# MNIST-CuDNN-Beispiel kompilieren (vorher aufräumen)
make clean && make
# MNIST-CuDNN-Beispiel ausführen
./mnistCUDNN
```

Listing 5.20: cuDNN testen

5.3 Anaconda

Anaconda ist eine Distribution für die Programmiersprachen Python [Ubu18c]. Anaconda installiert neben der Programmiersprache u.a. Pakete, die zur Verarbeitung und Analyse von großen Datenmengen eingesetzt werden können. Außerdem ist mit dem Jupyter Notebook ein Webbasiertes Frontend enthalten, mit dem sich per Browser Python-Skripte erstellen und ausführen lassen. Anaconda ist im Bereich *Data Science* sehr verbreitet. Python wird aktuell in zwei Versionslinien entwickelt: Python 2 und Python 3. Für den DL-Rechner wird mit Python 3 und in der derzeit aktuellen Version 3.6 gearbeitet.

Anaconda installieren Die aktuellste Version von Anaconda lässt sich mit `wget`-Befehl aus dem Internet herunterladen [Ana18]. Die Datei ist ein ausführbares Skript, welches dann nur noch in einer eigener Shell gestartet werden muss.

```
# Ins Installationsverzeichnis wechseln
cd /install
# Aktuelle Version von Anaconda herunterladen
wget https://repo.continuum.io/archive/Anaconda3-5.0.1-Linux-x86_64.sh
# Installationsskript in eigener Shell starten (sh = shell)
sudo sh Anaconda3-5.0.1-Linux-x86_64.sh
```

Listing 5.21: Anaconda installieren

Während der Installation wird man gefragt, unter welchem Pfad Anaconda installiert werden soll. Hier kann man das Verzeichnis `/install` angeben. Nach der Installation sollte man die Umgebungsvariable `PATH` anpassen (s.o.). Hierzu kann man wieder den Editor `vi` benutzen.

```
# Bash-Konfiguration editieren
vi ~/.bashrc
```

Listing 5.22: Konfigurationsdatei .bashrc editieren

Am Ende der Datei sollten die folgende Zeile geändert werden:

```
export PATH=/usr/local/cuda/bin:/install/anaconda3/bin:$PATH
```

Listing 5.23: Konfigurationsdatei .bashrc

Mit Hilfe des `source`-Kommandos werden die Änderungen wieder wirksam.

```
# .bashrc-Datei neu einlesen und ausführen
source ~/.bashrc
```

Listing 5.24: Änderungen umsetzen

Anaconda enthält u.a. einen eigenen Paketmanager namens `conda`. Mit diesem Befehl lassen sich Pakete hinzufügen, aktualisieren oder auch wieder entfernen. Mit den folgenden Befehlen wird Anaconda aktualisiert und eine *Root*-Umgebung eingerichtet.

```
# Anaconda aktualisieren
conda upgrade -y --all
# Anaconda-Root-Umgebung aktivieren
source activate root
# Anaconda-Umgebung(en) anzeigen
conda info --env
```

Listing 5.25: Anaconda aktualisieren

Jupyter Notebook einrichten Das Jupyter Notebook (s.o.) kann folgendermaßen gestartet werden.

```
# Ins Home-Verzeichnis wechseln
cd
# Jupyter-Notebook-Konfigurationsdatei erstellen
jupyter notebook --generate-config
# Jupyter-Notebook (auf Port 8888) starten
jupyter notebook --port=8888 --NotebookApp.token=''
```

Listing 5.26: Jupyter Notebook einrichten und starten

Zum Testen kann man einen Webbrowser (z.B. Firefox) benutzen und die URL http://localhost:8888 eingeben. Damit das Jupyter Notebook automatisch beim Hochfahren des Rechners gestartet wird, muss ein Eintrag in der `crontab` vorgenommen werden. In dieser Tabelle stehen alle Dienste, die automatisch an bestimmten Tagen zu bestimmten Zeiten oder bei bestimmten Aktionen vom sogenannten Cron-Daemon gestartet werden sollen [Ubu18f]. Zuvor sollte aber erst ein Verzeichnis erstellt werden, in dem die Dateien des Jupyter Notebooks gespeichert werden. Mit den folgenden Befehlen wird dies konfiguriert.

```
# Verzeichnis für die Deep Learning Notebooks erstellen
mkdir /data/DL
# Berechtigungen setzen
chmod -R 777 /data/DL
# Herausfinden, unter welchem Pfad das Jupyter Notebook installiert ist
which jupyter
# crontab ansehen (l = list)
crontab -l
# crontab editieren (e = edit)
crontab -e
```

Listing 5.27: Jupyter Notebook als Dienst

Am Ende der Datei sollte die folgende Zeile hinzugefügt werden.

```
@reboot /install/anaconda3/jupyter notebook --no-browser --port=8888 --
    NotebookApp.token='' --notebook-dir='/data/DL' &
```

Listing 5.28: crontab jupyter

Bei jedem Start des Rechners wird das Jupyter Notebook nun automatisch mitgestartet. Dies kann wieder getestet werden, indem nach einem Neustart die URL http://localhost:8888 im Webbrowser aufgerufen wird.

SSH verwenden Damit man das Jupyter Notebook auch von einem anderen Rechner aus nutzen kann, muss zum DL-Rechner ein SSH-Tunnel aufgebaut werden. Hierzu ist zunächst ein SSH-Server als Dienst zu installieren. Es wird eine *Open Source* Version verwendet, das OpenSSH [Ubu18m].

```
# OpenSSH-Server installieren
sudo apt-get install openssh-server
# Aktuellen Status des SSH-Servers aufrufen
sudo service ssh status
```

Listing 5.29: SSH-Server konfigurieren

Vom anderen Rechner aus muss man nun den SSH-Tunnel einrichten. Hierzu kann das kostenlose Programm `PuTTY` benutzt werden [Ubu18l]. Unter der URL https://www.putty.org/ findet man weitere Informationen und einen Link zum Herunterladen [PuT18]. Nachdem `PuTTY` gestartet wurde, lässt sich eine neue Session anlegen. Hierzu muss man die IP-Adresse des DL-Rechners kennen (hier: 192.168.178.44) und eintragen (vgl. Abb. 5.3). SSH benutzt den Port 22.

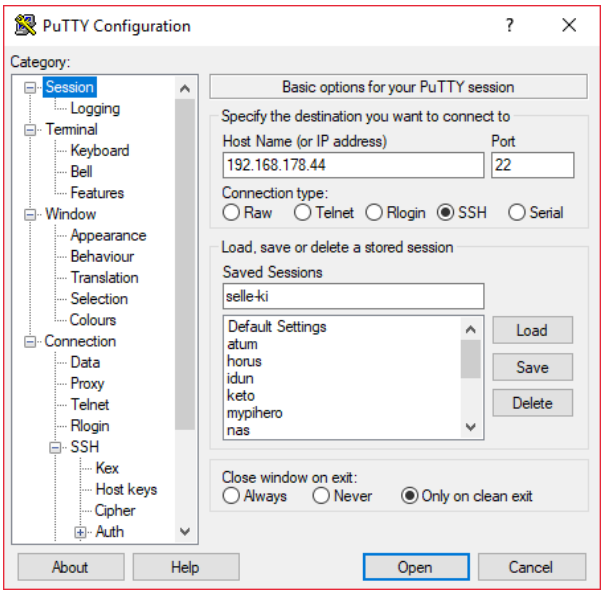

Abbildung 5.3: PuTTY: Session

Im Register `Connection > SSH > Tunnels` kann man dann einen neuen SSH-Tunnel einrichten. Der `Source port` ist 8888 und die `Destination` ist `localhost:8888` (vgl. Abb. 5.4).

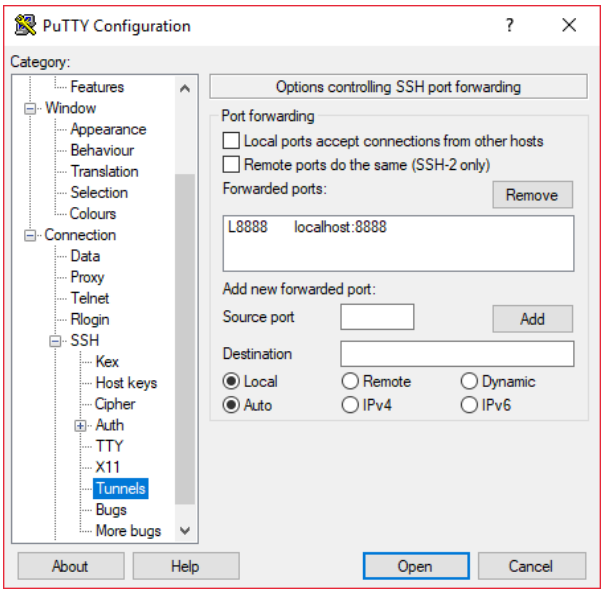

Abbildung 5.4: PuTTY: SSH-Tunnel

Wenn nun die Schaltfläche `Open` geklickt wird, öffnet sich ein Terminal-Fenster. Dort kann man sich mit seiner Kennung am DL-Rechner anmelden (vgl. Abb. 5.5). Gleichzeitig wird dabei der eingerichtete SSH-Tunnel aufgebaut.

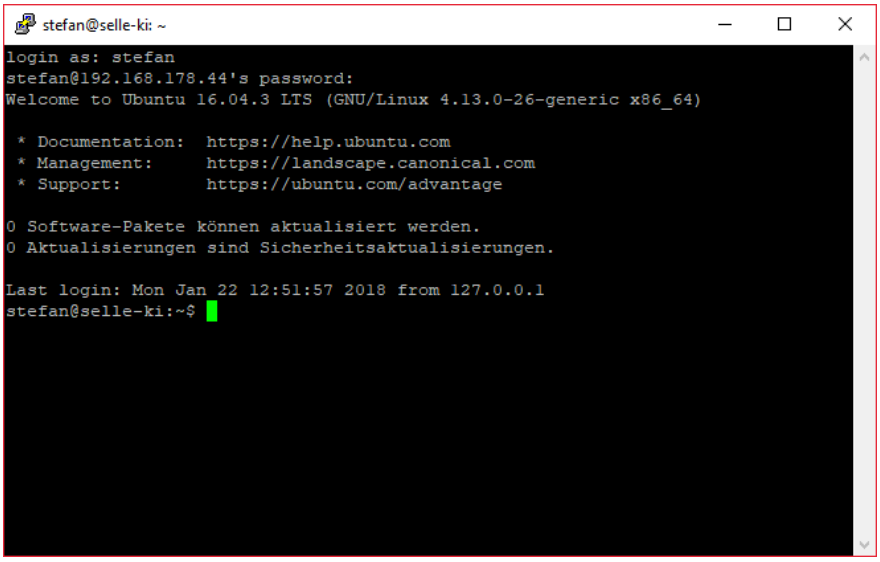

Abbildung 5.5: Terminal

Jetzt ist es möglich, dass Jupyter Notebook auch von diesem Rechner aus zu benutzen. Hierzu muss nur ein Webbrowser gestartet und die URL http://localhost:8888 eingegeben werden (vgl. Abb. 5.6).

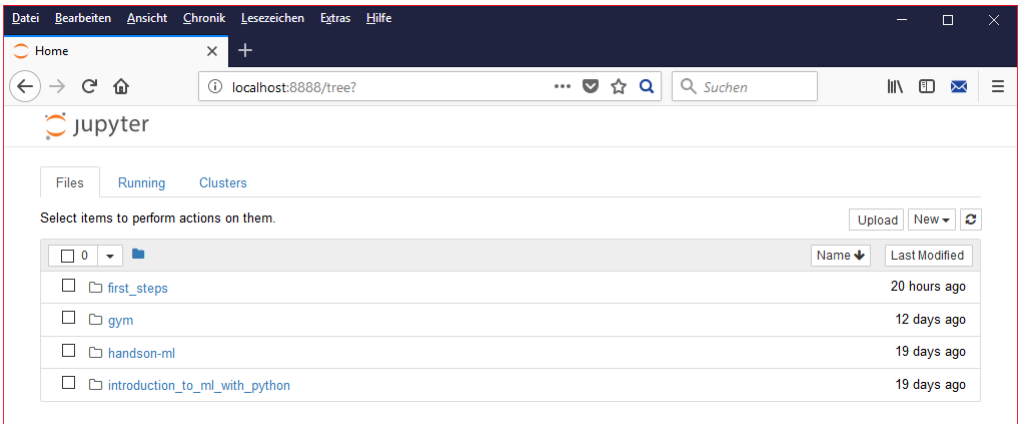

Abbildung 5.6: Jupyter Notebook: Startseite

Im Jupyter Notebook können dann Python-Skripte ausgeführt werden (vgl. Abb. 5.7).

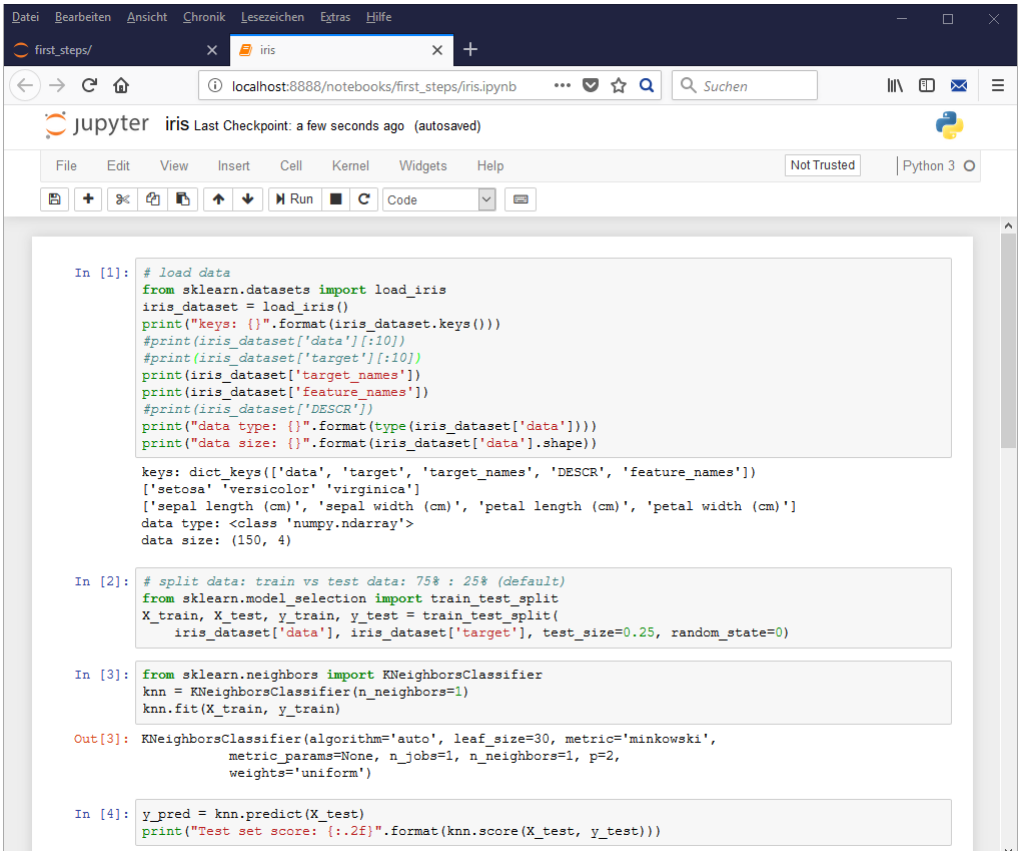

Abbildung 5.7: Jupyter Notebook: Python-Skript

6 Anwendungssoftware

Im letzten Kapitel wurde gezeigt, wie sich Treiber zur Grafikkarte von Nvidia installieren lassen. Außerdem wurden CUDA 9.1 und CuDNN 7.0.5 installiert, zwei Werkzeuge von Nvidia, mit denen u.a. auch ein tiefes Künstliches Neuronales Netzwerk (KNN) auf Grafikprozessoren trainiert werden kann. CuDNN ist in C/C++ programmiert und zu dieser Programmiersprache existiert auch eine entsprechende *low level* Schnittstelle (*Application Programming Interface (API)*). Darauf bauen einige Softwarelösungen im Bereich *Deep Learning* auf, die das Arbeiten mit KNN hinsichtlich Modellbildung und Training wesentlich vereinfachen und außerdem *high level* Schnittstellen zu anderen Programmiersprachen wie bspw. Python zur Verfügung stellen. Insbesondere im Bereich *Data Science* ist Python sehr verbreitet. Aus diesem Grund wurde bereits die Python-Distribution Anaconda installiert. DL-Softwarelösungen, die CuDNN unterstützen, sind: Caffe, Caffe2, Microsoft Cognitive Toolkit (CNTK), TensorFlow, Chainer, Theano, PyTorch usw. In dieser Arbeit wird exemplarisch die Bibliothek TensorFlow als DL-Softwarelösung verwendet. In den folgenden Abschnitten wird ausführlich dargestellt, wie TensorFlow installiert werden kann. Anhand einer Beispielanwendung wird schließlich ein Leistungsvergleich zu Berechnungen auf der CPU und der GPU des DL-Rechners durchgeführt.

6.1 Vorbereitung

Einige der genannten DL-Softwarepakete lassen sich sehr einfach mittels `pip`-Befehl installieren [Ubu18ap]. Es ist ein Paketverwaltungsprogramm für Python-Pakete aus dem sogenannten *Python Package Index* [Pyt18].

```
# pip installieren
sudo apt install python3-pip
```

Listing 6.1: pip installieren

Eine andere Möglichkeit der Installation besteht darin, dass die Quellcodes – die verwendete Software ist ja *Open Source* – auf dem DL-Rechner direkt kompiliert und daraus dann die ausführbaren Programme gebaut werden. Hierzu werden somit ein Compiler und ein Programm zum Bauen (engl. *build*) als Werkzeuge benötigt. Als Compiler können Programme aus der *Gnu Compiler Collection (GCC)* benutzt werden [Ubu18h]. Zu ihnen gehören bspw. die Programme `gcc` und `g++`, die sich mit den `build-essential` von Ubuntu 16.04 LTS installieren lassen. Es sind Compiler zu der Programmiersprache C/C++, in der sehr häufig Software entwickelt ist, bspw. auch die Bibliothek TensorFlow.

Als Programm zum Bauen und Testen der Software wird oft *Bazel* verwendet, welches u.a. Java, Python und C/C++ unterstützt [Baz18]. Es lässt sich sehr einfach mit dem *Advanced Packaging Tool* installieren. Hierzu muss man jedoch erst die Paketlisten editieren. Mit dem Kommando `echo` wird normalerweise ein Text auf der Konsole ausgegeben [Ubu18aa]. Mit `tee` wird eine Eingabe sowohl in eine Datei umgeleitet als auch auf der Konsole ausgegeben [Ubu18p]. Eine *Pipeline* aus `echo` und `tee` kann somit benutzt werden, um auf sehr elegante Weise die Paketliste zu aktualisieren (siehe nachfolgendes Listing).

```
# Compiler und andere Werkzeuge installieren
sudo apt-get install build-essential
# Java Development Kit installieren (wird von Bazel benötigt)
sudo apt-get install openjdk-8-jdk
# Bazel-URL zu den Paketlisten hinzufügen
echo "deb [arch=amd64] http://storage.googleapis.com/bazel-apt stable
    jdk1.8" | sudo tee /etc/apt/sources.list.d/bazel.list
# Installationsschlüssel herunterladen und hinzufügen
wget https://bazel.build/bazel-release.pub.gpg | sudo apt-key add -
# Paketlisten aktualisieren
sudo apt-get update
# Bazel installieren
sudo apt-get install bazel
```

Listing 6.2: Bazel installieren

6.2 TensorFlow

TensorFlow ist eine *Open Source* Bibliothek für numerisches Rechnen, die von Mitarbeitern des Google Brain Teams entwickelt wurde [Ten18]. Mathematisch betrachtet sind Tensoren mehrdimensionale Arrays. In einer Dimension spricht man von Vektoren, in zwei Dimensionen von Matrizen. Mathematische Operationen von diesen Tensoren werden mit der TensorFlow-Bibliothek graphenbasiert formuliert. Jeder Knoten repräsentiert dabei eine mathematische Operation und jede Kante ein Tensor. Der Fluss (engl. *Flow*) durch den Graphen spiegelt dann die Berechnung wieder. Diese lassen sich auf Prozessoren (CPUs) oder Grafikprozessoren (GPUs) ausführen. Normalerweise lässt sich TensorFlow sehr einfach per PIP installieren. Es muss dabei nur angegeben werden, welche Variante installiert werden soll. Die Standard-Variante ist für Berechnungen auf CPUs gedacht. Im Rahmen dieser Arbeit ist die Installation der zweiten Variante, also die für GPUs, interessanter.

```
# TensorFlow installieren (v 1.4.1)
pip install tensorflow-gpu
```

Listing 6.3: TensorFlow-Installation – Alternative 1

Aktuell ist die Release-Version 1.4.1 die Standard-Installation von TensorFlow im *Python Package Index*. Diese ist aber nur kompatibel mit CUDA 8.0 und CuDNN 6.0. Installiert sind aber die neusten Versionen CUDA 9.1.85 und CuDNN 7.0.5 (vgl. Kap. 5). Mittels PIP lassen sich auch neuere Versionen installieren.

```
# TensorFlow deinstallieren
pip uninstall tensorflow-gpu
# TensorFlow installieren (v 1.5.0rc1)
pip install tensorflow-gpu==1.5.0rc1
```

Listing 6.4: TensorFlow-Installation – Alternative 2

Die aktuellste Version im *Python Package Index* ist der erste *Release Candidate* zur Version 1.5.0. Es handelt sich also noch nicht um das eigentliche Release von 1.5.0, sondern um eine Vorgängerversion, die aber schon weit fortgeschritten ist. Auch diese Version ist leider nicht mit der bereits installierten Nvidia-Software kompatibel. Sie unterstützt zwar CUDA 9.0 und CuDNN 7.0, aber eben nicht die installierte Version CUDA 9.1. Die geplante Version TensorFlow 1.6.0 wird CUDA 9.1 unterstützen. Diese erscheint aber voraussichtlich erst Anfang März 2018 [Mar18]. Aus diesem Grund wird nun eine dritte

Möglichkeit beschrieben, um TensorFlow zu installieren – und zwar auf Basis der vorhandenen Quelltexte [Man17]. Hierzu wird das Werkzeug *Bazel* (s.o.) zum Bauen verwendet. Im ersten Schritt wird die aktuelle Version von TensorFlow von der Plattform GitHub heruntergeladen und für das Bauen vorbereitet.

```
# Ins Installationsverzeichnis wechseln
cd /install
# Tensorflow herunterladen
git clone https://github.com/tensorflow/tensorflow.git
# Ins Verzeichnis tensorflow wechseln
cd tensorflow
# Konfigurationsskript ausführen
./configure
```

Listing 6.5: TensorFlow-Installation – Alternative 3: Schritt 1

Das Konfigurationsskript benötigt einige Eingaben. Die meisten kann man bestätigen, weil die Standard-Vorschläge sinnvoll sind. Aufpassen muss man direkt beim Pfad zu Python, beim CUDA-Support und den Versionen von CUDA und CuDNN.

```
Please specify the location of python.
        [Default is /usr/bin/python]: /install/anaconda3/bin/python

Do you wish to build TensorFlow with jemalloc as malloc support?
        [Y/n]: Y

Do you wish to build TensorFlow with Google Cloud Platform support?
        [Y/n]: Y

Do you wish to build TensorFlow with Hadoop File System support?
        [Y/n]: Y

Do you wish to build TensorFlow with Amazon S3 File System support?
        [Y/n]: Y

Do you wish to build TensorFlow with XLA JIT support?
        [y/N]: N

Do you wish to build TensorFlow with GDR support?
        [y/N]: N

Do you wish to build TensorFlow with VERBS support?
        [y/N]:N

Do you wish to build TensorFlow with OpenCL support?
        [y/N]:N

Do you wish to build TensorFlow with CUDA support?
        [y/N]: Y

Please specify the CUDA SDK version you want to use, e.g. 7.0.
        [Leave empty to default to CUDA 8.0]: 9.1

Please specify the location where CUDA 9.1 toolkit is installed.
Refer to README.md for more details.
        [Default is /usr/local/cuda]: /usr/local/cuda

Please specify the cuDNN version you want to use.
        [Leave empty to default to cuDNN 6.0]: 7.0.5

Please specify the location where cuDNN 7.0.5 library is installed.
Refer to README.md for more details.
```

```
        [Default is /usr/local/cuda]: /usr/lib/x86_64-linux-gnu

Please note that each additional compute capability significantly increases your
build time and binary size.
        [Default is: 6.1] 6.1

Do you want to use clang as CUDA compiler?
        [y/N]: N

Please specify which gcc should be used by nvcc as the host compiler.
        [Default is /usr/bin/gcc]: /usr/bin/gcc

Do you wish to build TensorFlow with MPI support?
        [y/N]: N

Please specify optimization flags to use during compilation when bazel option
"--config=opt" is specified
        [Default is -march=native]: -march=native
```

Nun wird mit *Bazel* ein Softwarepaket für den *Python Package Index* selbst gebaut. Dies kann einige Zeit in Anspruch nehmen. Auf dem DL-Rechner hat dieser Schritt mehr als eine Stunde gedauert. Das Ergebnis ist eine ausführbare Datei *build_pip_package*, mit der dann eine sogenannte *Wheel*-Datei (Endung `whl`) erzeugt wird, die PIP zur Installation benötigt.

```
# PIP-Paket selber bauen
bazel build --config=opt --config=cuda --
    incompatible_load_argument_is_label=false //tensorflow/tools/
    pip_package:build_pip_package
# Wheel-Datei erzeugen
bazel-bin/tensorflow/tools/pip_package/build_pip_package tensorflow_pkg
```

Listing 6.6: TensorFlow-Installation – Alternative 3: Schritt 2

Abschließend wird dann wieder `pip` zur Installation verwendet.

```
# Ins Verzeichnis tensorflow_pkg wechseln
cd tensorflow_pkg
# TensorFlow installieren
pip install tensorflow*
```

Listing 6.7: TensorFlow-Installation – Alternative 3: Schritt 3

Zum Testen von TensorFlow kann man ein mitinstalliertes Beispiel ausführen.

```
# Ins entsprechende Verzeichnis wechseln
cd /install/tensorflow/tensorflow/examples/tutorials/mnist
# Beispiel-Python-Skript aufrufen
python fully_connected_feed.py
```

Listing 6.8: TensorFlow testen

6.3 Performance

In diesem Abschnitt soll mit Hilfe eines TensorFlow-Beispiels ein Vergleich zwischen den Rechenleistungen von GPU und CPU durchgeführt werden [Hal16]. Hierzu wird das Python-Skript, welches im nachfolgenden Listing dargestellt ist, im Jupyter Notebook geladen und ausgeführt. Erklärungen zum Skript sind direkt als *inline* Kommentare angegeben. Im Kern des Programms werden immer größer werdende Matrizen miteinander

multipliziert. Jedes dieser Multiplikationen wird sowohl von der GPU als auch von der CPU ausgeführt und die hierfür benötigte Zeitdauer gemessen. Als Ergebnisse erhält man Ausgaben auf der Konsole mit den Berechnungszeiten (s.u.) sowie eine grafische Darstellung (vgl. Abb. 6.1).

```
# Grafische Ausgabe direkt im Jupyter Notebook
%matplotlib inline
# Bibliothek für die grafische Ausgabe importieren
import matplotlib.pyplot as plt
# TensorFlow-Bibliothek importieren
import tensorflow as tf
# Bibliothek zur Zeitmessung importieren
import time

# Funktion get_times() definieren
# Parameter: maximum_time (dient zum Abbruch der Berechnungen)
def get_times(maximum_time):

  # Array zum Speichern der gemessenen Zeiten
  # TensorFlow Notation:
  # /gpu:0 für den Grafikprozessor
  # /cpu:0 für den Hauptprozessor
  device_times = { "/gpu:0":[], "/cpu:0":[] }

  # Matrix-Dimension: Start 100, Ende 2000, Schrittweite 100
  matrix_sizes = range(100,2000,100)

  # Schleife für die Matrix-Dimension
  for size in matrix_sizes:

    # Schleife über die Prozessoren (GPU, CPU)
    for device_name in device_times.keys():

      # Leere Matrix shape mit Dimension size x size erstellen
      mat = (size,size)
      # Rechnen mit halber Genauigkeit (16 Bit Gleitkomma-Arithmetik)
      dt = tf.float16

      # Block für die Definition der Matrix-Multiplikation
      # Je nach device_name wird entweder /gpu:0 oder /cpu:0 (s.o.) benutzt
      with tf.device(device_name):
        # Matrizen r1 und r2 werden gleichmäßig mit 16-Bit-Zufallszahlen
        # im Intervall [0, 1] gefüllt
        r1 = tf.random_uniform(shape=mat, minval=0, maxval=1, dtype=dt)
        r2 = tf.random_uniform(shape=mat, minval=0, maxval=1, dtype=dt)
        # Definition der Operation als Matrix-Multiplikation: r1 x r2
        dot_operation = tf.matmul(r2, r1)

      # Block für die Zeitmessung und Ausführung der Matrix-Multiplikation
      with tf.Session(config=tf.ConfigProto(log_device_placement=True))
          as session:
          # Startzeit wird gemessen
          start_time = time.time()
          # Matrix-Multiplikation wird ausgeführt
          result = session.run(dot_operation)
          # Endzeit wird gemessen und Differenz zur Startzeit (s.o.) berechnet
          time_taken = time.time() - start_time
          # Zeitdauer wird auf der Konsole ausgegeben
          print(device_name + ", Matrix Dimension: {}, Zeit[s]: {}".
              format(size, time_taken))
```

```
            # Zeitdauer wird im Array gespeichert
            device_times[device_name].append(time_taken)

        # Abbruchbedingung testen
        if time_taken > maximum_time:
          # Array mit Zeitdauer und Dimensionen zurückliefern
          return device_times, matrix_sizes

# Funktion get_times() mit Parameter 5 aufrufen
# D.h. max. Rechenzeit pro Matrix-Multiplikation: 5 Sekunden
device_times, matrix_sizes = get_times(5)
# Aus den Array-Rückgaben die jeweiligen Zeitdauern extrahieren
gpu_times = device_times["/gpu:0"]
cpu_times = device_times["/cpu:0"]

# Grafische Ausgabe (4 x 3 Zoll) erzeugen und als PNG-Datei speichern
plt.figure(figsize=(4, 3), dpi=300)
plt.title('Performance-Vergleich GPU vs. CPU', fontsize=12, color='gray')
plt.ylabel('Zeit [s]')
plt.xlabel('Matrix Dimension')
plt.plot(matrix_sizes[:len(gpu_times)], gpu_times, 'o-', label='GPU')
plt.plot(matrix_sizes[:len(cpu_times)], cpu_times, 'o-', label='CPU')
plt.legend(loc='upper left', frameon=True)
plt.savefig('/home/stefan/performance.png')
plt.show()
```

Listing 6.9: Performance-Vergleich – in Anlehnung an [Hal16]

Konsolenausgabe:

```
/gpu:0, Matrix Dimension: 100, Zeit[s]: 0.14745688438415527
/cpu:0, Matrix Dimension: 100, Zeit[s]: 0.17074894905090332
/gpu:0, Matrix Dimension: 200, Zeit[s]: 0.16395139694213867
/cpu:0, Matrix Dimension: 200, Zeit[s]: 0.1664729118347168
/gpu:0, Matrix Dimension: 300, Zeit[s]: 0.15210866928100586
/cpu:0, Matrix Dimension: 300, Zeit[s]: 0.20986437797546387
/gpu:0, Matrix Dimension: 400, Zeit[s]: 0.15052461624145508
/cpu:0, Matrix Dimension: 400, Zeit[s]: 0.279346227645874
/gpu:0, Matrix Dimension: 500, Zeit[s]: 0.1477947235107422
/cpu:0, Matrix Dimension: 500, Zeit[s]: 0.396808385848999
/gpu:0, Matrix Dimension: 600, Zeit[s]: 0.15151309967041016
/cpu:0, Matrix Dimension: 600, Zeit[s]: 0.5872442722320557
/gpu:0, Matrix Dimension: 700, Zeit[s]: 0.1598372459411621
/cpu:0, Matrix Dimension: 700, Zeit[s]: 0.8289539813995361
/gpu:0, Matrix Dimension: 800, Zeit[s]: 0.1526784896850586
/cpu:0, Matrix Dimension: 800, Zeit[s]: 1.1491296291351318
/gpu:0, Matrix Dimension: 900, Zeit[s]: 0.1611344814300537
/cpu:0, Matrix Dimension: 900, Zeit[s]: 1.546900749206543
/gpu:0, Matrix Dimension: 1000, Zeit[s]: 0.1639571189880371
/cpu:0, Matrix Dimension: 1000, Zeit[s]: 2.0673458576202393
/gpu:0, Matrix Dimension: 1100, Zeit[s]: 0.1545107364654541
/cpu:0, Matrix Dimension: 1100, Zeit[s]: 2.673412799835205
/gpu:0, Matrix Dimension: 1200, Zeit[s]: 0.1533374786376953
/cpu:0, Matrix Dimension: 1200, Zeit[s]: 3.415879726409912
/gpu:0, Matrix Dimension: 1300, Zeit[s]: 0.15220165252685547
/cpu:0, Matrix Dimension: 1300, Zeit[s]: 4.344688177108765
/gpu:0, Matrix Dimension: 1400, Zeit[s]: 0.15960025787353516
/cpu:0, Matrix Dimension: 1400, Zeit[s]: 5.332015037536621
```

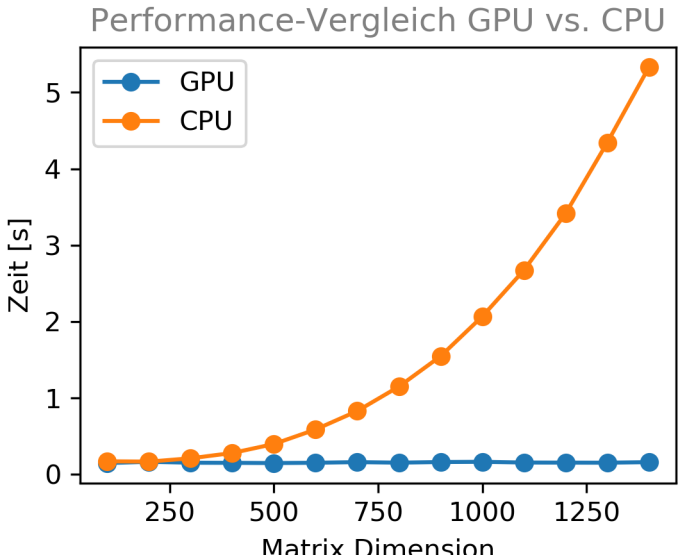

Abbildung 6.1: Performance-Vergleich zwischen GPU und CPU

Aufgrund der Konsolenausgabe bzw. der Abb. 6.1 erkennt man sehr deutlich den Geschwindigkeitsvorteil von Berechnungen, die vom Grafikprozessor ausgeführt werden gegenüber denen, die der Hauptprozessor übernimmt. Bspw. benötigt die CPU bei einer Multiplikation von Matrizen mit der Dimension 1.400 bereits mehr als 5 Sekunden, während die GPU dieselbe Berechnung in nur 0,16 Sekunden erledigt. Wenn ein Künstliches Neuronales Netzwerk (KNN) trainiert wird, müssen viele solche Operationen durchgeführt werden. Je größer bzw. tiefer das KNN ist, desto größer sind auch die Dimensionen der Matrizen, die miteinander multipliziert werden müssen. Deshalb ist es sinnvoll, im Bereich *Deep Learning* eine leistungsstarke GPU zu verwenden. Das Beispielprogramm wurde nun noch etwas modifiziert, um nur noch die Leistung der GPU zu messen. Bei diesem *Performance*-Test ging es darum, festzustellen wie groß die Dimension der Matrizen bei deren Multiplikation werden kann, wenn maximal ca. 5 Sekunden Rechenzeit zur Verfügung stehen. Es folgen wieder die Konsolenausgabe und eine grafische Darstellung (vgl. Abb. 6.2).

Konsolenausgabe:

```
/gpu:0, Matrix Dimension: 1, Zeit[s]: 0.0843205451965332
/gpu:0, Matrix Dimension: 1001, Zeit[s]: 0.07046842575073242
/gpu:0, Matrix Dimension: 2001, Zeit[s]: 0.06271672248840332
/gpu:0, Matrix Dimension: 3001, Zeit[s]: 0.09510970115661621
/gpu:0, Matrix Dimension: 4001, Zeit[s]: 0.09822797775268555
/gpu:0, Matrix Dimension: 5001, Zeit[s]: 0.09345674514770508
/gpu:0, Matrix Dimension: 6001, Zeit[s]: 0.13262033462524414
/gpu:0, Matrix Dimension: 7001, Zeit[s]: 0.16773366928100586
/gpu:0, Matrix Dimension: 8001, Zeit[s]: 0.1967921257019043
/gpu:0, Matrix Dimension: 9001, Zeit[s]: 0.29787778854370117
/gpu:0, Matrix Dimension: 10001, Zeit[s]: 0.33289098739624023
/gpu:0, Matrix Dimension: 11001, Zeit[s]: 0.462526798248291
/gpu:0, Matrix Dimension: 12001, Zeit[s]: 0.5865075588226318
/gpu:0, Matrix Dimension: 13001, Zeit[s]: 0.7488923072814941
/gpu:0, Matrix Dimension: 14001, Zeit[s]: 0.9392485618591309
/gpu:0, Matrix Dimension: 15001, Zeit[s]: 1.1026854515075684
/gpu:0, Matrix Dimension: 16001, Zeit[s]: 1.368166208267212
/gpu:0, Matrix Dimension: 17001, Zeit[s]: 1.5845584869384766
/gpu:0, Matrix Dimension: 18001, Zeit[s]: 1.8853495121002197
```

```
/gpu:0, Matrix Dimension: 19001, Zeit[s]: 2.163663148880005
/gpu:0, Matrix Dimension: 20001, Zeit[s]: 2.6049575805664062
/gpu:0, Matrix Dimension: 21001, Zeit[s]: 2.9827589988708496
/gpu:0, Matrix Dimension: 22001, Zeit[s]: 3.2453114986419678
/gpu:0, Matrix Dimension: 23001, Zeit[s]: 3.7004647254943848
/gpu:0, Matrix Dimension: 24001, Zeit[s]: 4.380356073379517
/gpu:0, Matrix Dimension: 25001, Zeit[s]: 4.696216821670532
/gpu:0, Matrix Dimension: 26001, Zeit[s]: 5.324223756790161
```

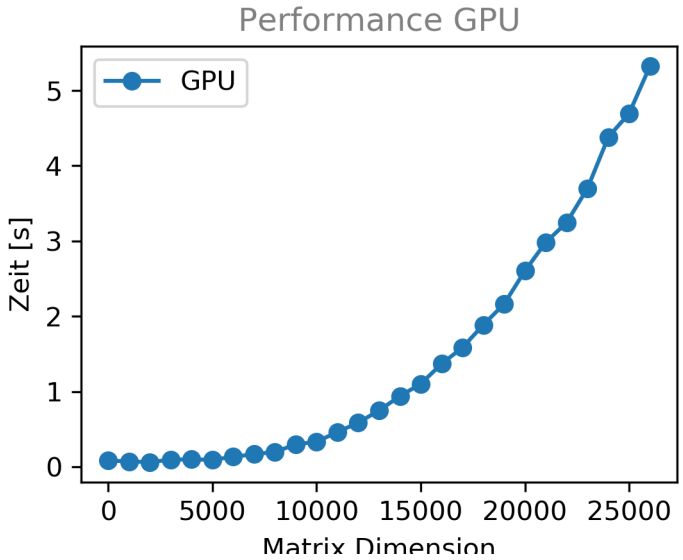

Abbildung 6.2: Performance der GPU

Während die CPU in 5,3 Sekunden Matrizen mit der Dimension 1.400 multipliziert hat, schafft die GPU bereits die Dimension 26.000! Beim Ausführen von diesen Python-Skripten kann es vorkommen, dass *Out-of-Memory*-Fehler angezeigt werden und die Berechnungen unerwartet abgebrochen werden. Mit Hilfe des Befehls `nvidia-smi` kann man sich über den aktuellen Status zu der GPU informieren.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 387.34                 Driver Version: 387.34                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 108...  Off  | 00000000:01:00.0 Off |                  N/A |
|  0%   42C    P8    20W / 250W |  10671MiB / 11172MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1076      G   /usr/lib/xorg/Xorg                           106MiB |
|    0      1861      G   compiz                                        82MiB |
|    0      6410      G   /usr/lib/firefox/firefox                       5MiB |
|    0      6736      C   /install/anaconda3/bin/python              10465MiB |
+-----------------------------------------------------------------------------+
```

In diesem Beispiel sieht man, dass fast der gesamte Speicher der Grafikkarte für den Prozess `/install/anaconda3/bin/python` mit der Prozess-ID 6736 verwendet wird. Falls

ein Python-Skript einmal hängen bleibt oder Speicherfehler auftreten, dann kann dieser Prozess mittels `kill`-Kommando beendet werden [Ubu18ag]. Dabei wird der vom Prozess allozierte Speicher auch wieder frei gegeben. Mit dem Befehl `ps`, engl. für *process*, kann man sich einen Überblick über laufende Prozesse verschaffen [Ubu18aq].

```
# Alle Prozesse anzeigen
sudo ps aux
# Prozess hart (Option -9) beenden
sudo kill -9 <PID>
```

Listing 6.10: Prozess beenden

7 Zusammenfassung und Ausblick

Es vergeht kaum eine Woche, in der nicht über eine neue KI-Anwendung in den Medien berichtet wird. *Deep Learning (DL)* ist dabei die am meisten beachtete Technik des maschinellen Lernens. Bereits im März 2016 schlug die Maschine *AlphaGo* von Google DeepMind den südkoreanische Profispieler Lee Sedol beim Brettspiel Go mit 1:4 Spielen unter Turnierbedingungen. Die Künstliche Intelligenz von *AlphaGo* basiert u.a. auf tiefe Künstliche Neuronale Netzwerke (KNN), die mit Hilfe von Lernalgorithmen des überwachten Lernens (engl. *Supervised Learning*) trainiert werden. Aufgrund der Größe bzw. Tiefe dieser KNN spricht man auch von *Deep Learning (DL)*. Die enorme Rechenleistung von *AlphaGo* stammt von 1.202 Prozessoren (engl. *Central Processing Unit (CPU)*) und 176 Grafikprozessoren (engl. *Graphics Processing Unit (GPU)*). Grafikkarten werden gerne für solche aufwändigen Berechnungen genutzt, weil sie über sehr viele Streamprozessoren verfügen. Beim Trainieren von KNN müssen nämlich viele Matrix-Multiplikationen durchgeführt werden, eine Aufgabe die sich gut parallelisieren und effizient auf GPUs ausführen lässt.

Das Ziel dieser Arbeit ist die Dokumentation der Zusammenstellung, Installation und Konfiguration eines Personal Computers (PC), der für *Deep Learning* Projekte in der Forschung und Lehre an der Hochschule für Wirtschaft und Technik des Saarlandes (htw saar) eingesetzt werden kann. Es werden also zwei Ergebnisse angestrebt: (1) eine nützliche Anleitung, mit der sich ein nackter PC in nur einem Versuch in einen fertig konfigurierten DL-Rechner verwandeln lässt und (2) diesen voll funktionsfähigen Rechner. Diese Anleitung kann als eine Art Kochrezept dienen. In ihr steckt bereits viel *Know-how*. Praktische Erfahrungen, die im Rahmen dieses Projekts gemacht wurden, darunter auch Irrwege und Fehlversuche (*Lessons Learned*), sind in diese Dokumentation eingeflossen.

Im Mittelpunkt des DL-Rechners steht die Grafikkarte. Es kommt ein High-End-Modell mit der GeForce GTX 1080 Ti GPU von Nvidia zum Einsatz. Sie verfügt über 3584 Streamprozessoren, taktet mit 1594 MHz und kann auf 11 GB eigenen Speicher zugreifen. Die anderen Hardware-Komponenten wurden passend zu dieser Grafikkarte ausgewählt (vgl. Kap. 2). Der komplette Rechner hat 2.663,41 Euro gekostet (Stand 04.10.2017). In Kap. 3 wurde zunächst beschrieben, wie sich die Firmware des *Basic Input Output System (BIOS)* des Mainboards von MSI aktualisieren lässt. Anschließend wurden spezielle Einstellungen bezüglich der Grafikkarten und Festplatten im BIOS vorgenommen, damit die Installation des Betriebssystems gelingt und der Rechner-Betrieb möglichst effizient abläuft. Die verwendete Software ist *Open Source* oder *Freeware*. Es fallen also keine Lizenzkosten für deren Nutzung an. In Kap. 4 wurde gezeigt, wie man das Linux-Betriebssystem Ubuntu 16.04 LTS auf dem Rechner installiert. LTS steht für *Long Time Support*. Die Version 16.04 wird bis April 2021 von Ubuntu unterstützt, d.h. bis dahin gibt es regelmäßig Aktualisierungen, die einen stabilen und sicheren Betrieb gewährleisten. In Kap. 5 wurde beschrieben, wie das Betriebssystem konfiguriert wird. Es wurden einige nützliche Linux-Befehle kurz erklärt und das System so eingerichtet, dass es von Anwendern effizient genutzt werden kann. Etwas schwierig gestaltete sich dabei allerdings die Installation funktionsfähiger Treiber für die Grafikkarte. Die Standard-Treiber mussten deaktiviert werden, um proprietäre Treiber des Herstellers Nvidia installieren zu können. Es gab

auch Abhängigkeiten zum Werkzeug CUDA 9.1 von Nvidia, mit dem sich parallele Rechnungen auf der GPU ausführen lassen. In deren Standard-Installation werden immer die nicht-funktionierenden Nvidia-Grafiktreiber mitinstalliert. Somit wurde eine alternative Installationsart verwendet, in der man die Installation dieser Treiber unterdrücken kann. Außerdem wurde auch das Werkzeug CuDNN 7.0.5 installiert. Dies ist eine Software-Bibliothek von Nvidia, die auf CUDA aufsetzt und mit der sich tiefe KNN trainieren lassen. CuDNN steht für *CUDA Deep Neural Network Library*. Diese Software ist in C/C++ programmiert. Mittlerweile gibt es aber auch viele DL-Softwarelösungen, die CuDNN unterstützen und weitere *high level* Schnittstellen zu anderen Programmiersprachen anbieten. Die DL-Bibliothek *TensorFlow* von Google bietet bspw. eine API zur Programmiersprache Python an. Python ist deshalb so interessant, weil diese Programmiersprache im Bereich *Data Science* sehr verbreitet ist. Aus diesem Grund wurde zunächst Anaconda 5.0.1 für Python 3.6 installiert. Anaconda ist eine Python-Distribution, in der u.a. auch Pakete mitgeliefert werden, die zur Verarbeitung und Analyse von großen Datenmengen eingesetzt werden können. Außerdem ist das Jupyter Notebook enthalten, ein Webbasiertes Frontend, mit dem sich Python-Skripte einfach per Browser erstellen und ausführen lassen. Das Jupyter Notebook wurde so eingerichtet, dass es sich per SSH-Tunnel von einem anderen Rechner aus bedienen lässt. Hierzu wurde auf dem DL-Rechner ein SSH-Dienst eingerichtet und auf dem Client das Programm PuTTY verwendet. In Kap. 6 wurde dann das DL-Framework TensorFlow 1.5.0 installiert. Normalerweise ist dies ein sehr einfacher Prozess. Aufgrund von Inkompatibilitäten der aktuell ausgelieferten TensorFlow-Version zu CUDA 9.1 und CuDNN 7.0.5 mussten jedoch die offenen Quellcodes erst kompiliert und das Programm danach mit Bazel gebaut werden. Nach der erfolgreichen Installation wurde dann abschließend ein Python-Skript als Anwendungsbeispiel ausgeführt, um die Leistung des Systems zu messen und zu dokumentieren. Hierzu wurden mittels der TensorFlow-Bibliothek immer größer werdende Matrizen miteinander multipliziert. Die Rechnungen wurden sowohl von der GPU als auch der CPU ausgeführt und die dafür benötigten Zeitdauern gemessen. Die Multiplikation zweier 1.400 x 1.400 Matrizen hat auf der CPU bereits 5,3 Sekunden gedauert. Dabei handelt es sich immerhin um den Intel Core i7-7700K Prozessor mit 4 Kernen und 4,2 GHz Taktfrequenz. Für die gleiche Berechnung benötigte die GPU nur 0,16 Sekunden, das ist die 33-fache Geschwindigkeit. In 5,3 Sekunden schaffte die GPU eine Matrix-Multiplikation der Dimension von 26.000.

Das Ziel dieser Arbeit wurde erreicht. Ein fertig konfigurierter Rechner steht nun zur weiteren Verwendung zur Verfügung. Außerdem kann diese Arbeit auch als Blaupause in Form einer Anleitung und Dokumentation dienen, um sehr schnell weitere Rechner zu konfigurieren. Das könnte zukünftig nämlich auch im Bereich der Lehre interessant werden. Themen wie Digitalisierung, Cloud Computing, Mobile Applikationen, Big Data und Künstliche Intelligenz durchdringen unser Leben und halten auch Einzug in das Curriculum der htw saar. So ist zu prüfen, in wie weit die Technik *Deep Learning* in den folgenden Master-Modulen gelehrt werden kann:

- Angewandte Methoden der Informationsbeschaffung (MMF-130)
- Angewandte Informatik (DFMMS-222)
- Big Data Analysis (MAMS-120)
- Data Science (MASCM-141)

Daneben könnte dieses Thema auch im Rahmen von Projektarbeiten (Modul BBWL-622) oder Abschlussarbeiten durch Studierende bearbeitet werden.

Eine andere Verwendungsmöglichkeit ist der Einsatz in der angewandten Forschung. Natürlich kann man diesen DL-Rechner nicht mit *AlphaGo* von Google DeepMind vergleichen. Trotzdem ist die GPU leistungsstark genug, sodass einige Analysen damit durchgeführt werden können. Zwei Ideen werden im Folgenden kurz skizziert.

Personalmanagement: Recruiting Die Personalbeschaffung (engl. *Recruiting*) ist ein wichtiger Prozess im Personalmanagement. Vereinfacht gesagt soll durch die Personalbeschaffung sichergestellt werden, dass die richtige Anzahl der richtigen Menschen mit der richtigen Qualifikation zur richtigen Zeit am richtigen Ort des Unternehmens eingesetzt werden kann. Hierzu müssen also u.a. die Anforderungen an eine Mitarbeiterstelle mit den Qualifikationen der internen oder externen Bewerber abgeglichen werden. In Zeiten des Fachkräftemangels erfährt das *Recruiting* einen hohen Stellenwert. Mit Hilfe einer mobilen App kann den Bewerbern die Stellenangebote der Unternehmen präsentiert werden. Diese Stellenangebote sind bereits im Internet auf sehr vielen Jobportalen vorhanden. Der Bewerber kann nun ähnlich wie bei der mobilen Dating-App *Tinder* [Tin18][Wik18c] sehr schnell durch ein Wischen nach rechts oder links entscheiden, ob das präsentierte Stellenangebot für ihn interessant ist oder nicht. Die gesammelten Daten lassen sich dann mit Hilfe von Algorithmen des Maschinenlernens analysieren, um zukünftig bessere Matching-Vorschläge zu generieren. Dabei können auch KNN und *Deep Learning* eingesetzt werden. Eine Zusammenarbeit innerhalb der Fakultät für Wirtschaftswissenschaften der htw saar ist diesbezüglich mit Kollegen aus dem Cluster Personalmanagement möglich, bspw. mit Prof. Dr. Wolfgang Appel.

Rechnungs- und Prüfungswesen: Wirtschaftsprüfung Eine der Hauptaufgaben von Wirtschaftsprüfern ist die Jahresabschlussprüfung von Unternehmen. Die Prüfung umfasst auch die Buchführung, es ist jedoch keine Volluntersuchung, sondern nur eine Stichprobenprüfung. Anhand der ausgewählten Stichproben muss der Wirtschaftsprüfer erkennen können, ob es Unrichtigkeiten und Verstöße bei der Rechnungslegung gibt, die sich auf die Vermögens-, Finanz- und Ertragslagebeurteilung des Unternehmens wesentlich auswirken. Viele Buchungen werden mittlerweile von ERP- oder anderen Computersystemen erfasst. Außerdem werden auch unstrukturierte Daten wie Verträge, Sitzungsprotokolle, Geschäftsberichte, Rechnungen, Lieferscheine usw. analysiert und geprüft. Insgesamt ist das Datenvolumen enorm und die Komplexität sehr hoch – es ist ein typisches *Big Data* Problem. Mit Hilfe von *Deep Learning* könnten bspw. intelligente Textanalysen (*Natural Language Processing (NLP)*) dieser unstrukturierten Dokumente durchgeführt oder auch die Massendaten auf Anomalien untersucht werden. In der Fakultät für Wirtschaftswissenschaften der htw saar könnte eine mögliche Kooperation mit Kollegen aus dem Cluster Rechnungs- und Prüfungswesen angestrebt werden. Bspw. verfügt Prof. Dr. Jochen Pilhofer über das entsprechende anwendungsorientierte Fachwissen in diesem Bereich.

Quellenverzeichnis

[Alt18a] Alternate. *Alpenföhn Brocken Eco, CPU-Kühler*. 2018. URL: https://www.alternate.de/Alpenf%C3%B6hn/Brocken-Eco-CPU-K%C3%BChler/html/product/1142560 (besucht am 11. 01. 2018).

[Alt18b] Alternate. *Corsair Carbide 270R Window-Tower-Gehäuse*. 2018. URL: https://www.alternate.de/Corsair/Carbide-270R-Window-Tower-Geh%C3%A4use/html/product/1312339 (besucht am 11. 01. 2018).

[Alt18c] Alternate. *Corsair DIMM 32 GB DDR4-2400 Kit, Arbeitsspeicher*. 2018. URL: https://www.alternate.de/Corsair/DIMM-32-GB-DDR4-2400-Kit-Arbeitsspeicher/html/product/1262872 (besucht am 11. 01. 2018).

[Alt18d] Alternate. *Corsair RM1000X 1000W, PC-Netzteil*. 2018. URL: https://www.alternate.de/Corsair/RM1000X-1000W-PC-Netzteil/html/product/1227636 (besucht am 11. 01. 2018).

[Alt18e] Alternate. *GIGABYTE AORUS GeForce GTX 1080 Ti, Grafikkarte*. 2018. URL: https://www.alternate.de/GIGABYTE/AORUS-GeForce-GTX-1080-Ti-Grafikkarte/html/product/1344809 (besucht am 11. 01. 2018).

[Alt18f] Alternate. *Intel Core i7-7700K, Prozessor*. 2018. URL: https://www.alternate.de/Intel/Core-i7-7700K-Prozessor/html/product/1305350 (besucht am 11. 01. 2018).

[Alt18g] Alternate. *MSI Z270 GAMING PRO CARBON, Mainboard*. 2018. URL: https://www.alternate.de/MSI/Z270-GAMING-PRO-CARBON-Mainboard/html/product/1312234 (besucht am 11. 01. 2018).

[Alt18h] Alternate. *Samsung 960 EVO 250 GB, Solid State Drive*. 2018. URL: https://www.alternate.de/Samsung/960-EVO-250-GB-Solid-State-Drive/html/product/1305238 (besucht am 11. 01. 2018).

[Alt18i] Alternate. *WD WD40EFRX 4 TB, Festplatte*. 2018. URL: https://www.alternate.de/WD/WD40EFRX-4-TB-Festplatte/html/product/1098412 (besucht am 11. 01. 2018).

[Ana18] Anaconda. *Download Anaconda Distribution*. 2018. URL: https://www.anaconda.com/download (besucht am 12. 01. 2018).

[Baz18] Bazel. *Build and test software of any size, quickly and reliably*. 2018. URL: https://bazel.build (besucht am 22. 01. 2018).

[Dee18] DeepMind. *The story of AlphaGo so far*. 2018. URL: https://deepmind.com/research/alphago (besucht am 17. 01. 2018).

[GBC16] I. Goodfellow, Y. Bengio und A. Courville. *Deep Learning*. 1. Aufl. Cambridge, Massachusettts, USA: The MIT Press, 2016.

[Gei18a] Geizhals. *Alpenföhn Brocken Eco, CPU-Kühler*. 2018. URL: https://gzhls.at/i/16/23/1081623-n1.jpg (besucht am 11. 01. 2018).

[Gei18b] Geizhals. *Corsair Carbide 270R Window-Tower-Gehäuse*. 2018. URL: https://gzhls.at/i/14/50/1541450-n0.jpg (besucht am 11. 01. 2018).

[Gei18c] Geizhals. *Corsair DIMM 32 GB DDR4-2400 Kit, Arbeitsspeicher*. 2018. URL: https://gzhls.at/i/56/10/1305610-n0.jpg (besucht am 11. 01. 2018).

[Gei18d] Geizhals. *Corsair RM1000X 1000W, PC-Netzteil*. 2018. URL: https://gzhls.at/i/10/32/1331032-n0.jpg (besucht am 11. 01. 2018).

[Gei18e] Geizhals. *GIGABYTE AORUS GeForce GTX 1080 Ti, Grafikkarte*. 2018. URL: https://gzhls.at/i/21/48/1602148-n0.jpg (besucht am 11. 01. 2018).

[Gei18f] Geizhals. *Intel Core i7-7700K, Prozessor*. 2018. URL: https://gzhls.at/i/10/97/1551097-n7.jpg (besucht am 11. 01. 2018).

[Gei18g] Geizhals. *MSI Z270 GAMING PRO CARBON, Mainboard*. 2018. URL: https://gzhls.at/i/74/71/1557471-n3.jpg (besucht am 11. 01. 2018).

[Gei18h] Geizhals. *Samsung 960 EVO 250 GB, Solid State Drive*. 2018. URL: https://gzhls.at/i/11/96/1511196-n2.jpg (besucht am 11. 01. 2018).

[Gei18i] Geizhals. *WD WD40EFRX 4 TB, Festplatte*. 2018. URL: https://gzhls.at/i/20/27/992027-n0.jpg (besucht am 11. 01. 2018).

[Hal16] E. Hallström. *Introduction to TensorFlow - CPU vs GPU*. 11. Nov. 2016. URL: https://medium.com/@erikhallstrm/hello-world-tensorflow-649b15aed18c (besucht am 25. 01. 2018).

[Iva17a] S. Ivanov. *Picking a GPU for Deep Learning – Buyer's guide at the end of 2017*. 22. Nov. 2017. URL: https://blog.slavv.com/picking-a-gpu-for-deep-learning-3d4795c273b9 (besucht am 17. 01. 2018).

[Iva17b] S. Ivanov. *The $1700 great Deep Learning box: Assembly, setup and benchmarks*. 29. Mai 2017. URL: https://blog.slavv.com/the-1700-great-deep-learning-box-assembly-setup-and-benchmarks-148c5ebe6415 (besucht am 17. 01. 2018).

[Lau18] T. Lauzière. *LinuxLive USB Creator 2.9.4*. 2018. URL: https://www.heise.de/download/product/linuxlive-usb-creator-90060 (besucht am 12. 01. 2018).

[MSI18a] MSI. *How to flash the BIOS*. 2018. URL: https://de.msi.com/files/pdf/How_to_flash_the_BIOS.doc (besucht am 12. 01. 2018).

[MSI18b] MSI. *Support für Z270 GAMING PRO CARBON*. 2018. URL: https://de.msi.com/Motherboard/support/Z270-GAMING-PRO-CARBON (besucht am 12. 01. 2018).

[Man17] A. Mandal. *How to install Tensorflow GPU with CUDA Toolkit 9.1 and cuDNN 7.0.5 for Python 3 on Ubuntu 16.04-64bit*. 29. Dez. 2017. URL: http://www.python36.com/install-tensorflow141-gpu (besucht am 22. 01. 2018).

[Mar18] S. Maruthachalam. *CUDA 9.1 and TensorFlow*. 2018. URL: https://github.com/tensorflow/tensorflow/issues/15656 (besucht am 22. 01. 2018).

[Nvi17] Nvidia. *CUDA Installation Guide for Linux*. 19. Dez. 2017. URL: http://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html (besucht am 16. 01. 2018).

[Nvi18] Nvidia. *CuDNN Download Survey*. 2018. URL: https://developer.nvidia.com/rdp/form/cudnn-download-survey (besucht am 22. 01. 2018).

[PuT18] PuTTY. *Download PuTTY*. 2018. URL: https://www.putty.org (besucht am 12. 01. 2018).

[Pyt18] Python. *PyPI*. 2018. URL: https://pypi.python.org/pypi (besucht am 22. 01. 2018).

[Sha50] C. E. Shannon. „Programming a Computer for Playing Chess“. In: *Philosophical Magazine* 41.314 (1950), S. 256–275.

[Sil+16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel und D. Hassabis. „Mastering the game of Go with deep neural networks and tree search“. In: *Nature* 529 (2016), S. 256–275. URL: https://storage.googleapis.com/deepmind-media/alphago/AlphaGoNaturePaper.pdf.

[Ten18] TensorFlow. *An open-source software library for Machine Intelligence*. 2018. URL: https://www.tensorflow.org (besucht am 22. 01. 2018).

[Tin18] Tinder. *Swipen. Matchen. Chatten.* 2018. URL: https://tinder.com (besucht am 30. 01. 2018).

[Ubu18a] Ubuntu. *16.04.3 LTS (Xenial Xerus)*. 2018. URL: http://releases.ubuntu.com/16.04 (besucht am 12. 01. 2018).

[Ubu18b] Ubuntuusers. *APT*. 2018. URL: https://wiki.ubuntuusers.de/APT (besucht am 22. 01. 2018).

[Ubu18c] Ubuntuusers. *Anaconda*. 2018. URL: https://wiki.ubuntuusers.de/Anaconda (besucht am 22. 01. 2018).

[Ubu18d] Ubuntuusers. *Bash*. 2018. URL: https://wiki.ubuntuusers.de/Bash (besucht am 22. 01. 2018).

[Ubu18e] Ubuntuusers. *Befehlsübersicht*. 2018. URL: https://wiki.ubuntuusers.de/Shell/Befehls%C3%BCbersicht (besucht am 22. 01. 2018).

[Ubu18f] Ubuntuusers. *Cron*. 2018. URL: https://wiki.ubuntuusers.de/Cron (besucht am 22. 01. 2018).

[Ubu18g] Ubuntuusers. *Formatieren*. 2018. URL: https://wiki.ubuntuusers.de/Formatieren (besucht am 22. 01. 2018).

[Ubu18h] Ubuntuusers. *GCC*. 2018. URL: https://wiki.ubuntuusers.de/GCC (besucht am 22. 01. 2018).

[Ubu18i] Ubuntuusers. *Gerätemanager*. 2018. URL: https://wiki.ubuntuusers.de/Ger%C3%A4temanager (besucht am 22. 01. 2018).

[Ubu18j] Ubuntuusers. *Git*. 2018. URL: https://wiki.ubuntuusers.de/Git (besucht am 22. 01. 2018).

[Ubu18k] Ubuntuusers. *PPA*. 2018. URL: https://wiki.ubuntuusers.de/Paketquellen_freischalten/PPA (besucht am 22. 01. 2018).

[Ubu18l] Ubuntuusers. *PuTTY*. 2018. URL: https://wiki.ubuntuusers.de/PuTTY (besucht am 22. 01. 2018).

[Ubu18m] Ubuntuusers. *SSH*. 2018. URL: https://wiki.ubuntuusers.de/SSH (besucht am 22. 01. 2018).

[Ubu18n] Ubuntuusers. *UUID*. 2018. URL: https://wiki.ubuntuusers.de/UUID (besucht am 22. 01. 2018).

[Ubu18o] Ubuntuusers. *Umgebungsvariable*. 2018. URL: https://wiki.ubuntuusers.de/Umgebungsvariable (besucht am 22. 01. 2018).

[Ubu18p] Ubuntuusers. *Umleitungen*. 2018. URL: https://wiki.ubuntuusers.de/Shell/Umleitungen (besucht am 22. 01. 2018).

[Ubu18q] Ubuntuusers. *VIM*. 2018. URL: https://wiki.ubuntuusers.de/VIM (besucht am 22. 01. 2018).

[Ubu18r] Ubuntuusers. *apt-get*. 2018. URL: https://wiki.ubuntuusers.de/apt/apt-get (besucht am 22. 01. 2018).

[Ubu18s] Ubuntuusers. *blkid*. 2018. URL: https://wiki.ubuntuusers.de/blkid (besucht am 22. 01. 2018).

[Ubu18t] Ubuntuusers. *cd*. 2018. URL: https://wiki.ubuntuusers.de/cd (besucht am 22. 01. 2018).

[Ubu18u] Ubuntuusers. *chmod*. 2018. URL: https://wiki.ubuntuusers.de/chmod (besucht am 22. 01. 2018).

[Ubu18v] Ubuntuusers. *chown*. 2018. URL: https://wiki.ubuntuusers.de/chown (besucht am 22. 01. 2018).

[Ubu18w] Ubuntuusers. *cp*. 2018. URL: https://wiki.ubuntuusers.de/cp (besucht am 22. 01. 2018).

[Ubu18x] Ubuntuusers. *df*. 2018. URL: https://wiki.ubuntuusers.de/df (besucht am 22. 01. 2018).

[Ubu18y] Ubuntuusers. *dpkg*. 2018. URL: https://wiki.ubuntuusers.de/dpkg (besucht am 22. 01. 2018).

[Ubu18z] Ubuntuusers. *du*. 2018. URL: https://wiki.ubuntuusers.de/du (besucht am 22. 01. 2018).

[Ubu18aa] Ubuntuusers. *echo*. 2018. URL: https://wiki.ubuntuusers.de/echo (besucht am 22. 01. 2018).

[Ubu18ab] Ubuntuusers. *ext*. 2018. URL: https://wiki.ubuntuusers.de/ext (besucht am 22. 01. 2018).

[Ubu18ac] Ubuntuusers. *fdisk*. 2018. URL: https://wiki.ubuntuusers.de/fdisk (besucht am 22. 01. 2018).

[Ubu18ad] Ubuntuusers. *find*. 2018. URL: https://wiki.ubuntuusers.de/find (besucht am 22. 01. 2018).

[Ubu18ae] Ubuntuusers. *fstab*. 2018. URL: https://wiki.ubuntuusers.de/fstab (besucht am 22. 01. 2018).

[Ubu18af] Ubuntuusers. *grep*. 2018. URL: https://wiki.ubuntuusers.de/grep (besucht am 22. 01. 2018).

[Ubu18ag] Ubuntuusers. *kill*. 2018. URL: https://wiki.ubuntuusers.de/kill (besucht am 22. 01. 2018).

[Ubu18ah] Ubuntuusers. *less*. 2018. URL: https://wiki.ubuntuusers.de/less (besucht am 22. 01. 2018).

[Ubu18ai] Ubuntuusers. *ln*. 2018. URL: https://wiki.ubuntuusers.de/ln (besucht am 22. 01. 2018).

[Ubu18aj] Ubuntuusers. *ls*. 2018. URL: https://wiki.ubuntuusers.de/ls (besucht am 22. 01. 2018).

[Ubu18ak] Ubuntuusers. *mkdir*. 2018. URL: https://wiki.ubuntuusers.de/mkdir (besucht am 22. 01. 2018).

[Ubu18al] Ubuntuusers. *more*. 2018. URL: https://wiki.ubuntuusers.de/more (besucht am 22. 01. 2018).

[Ubu18am] Ubuntuusers. *mount*. 2018. URL: https://wiki.ubuntuusers.de/mount (besucht am 22. 01. 2018).

[Ubu18an] Ubuntuusers. *mv*. 2018. URL: https://wiki.ubuntuusers.de/mv (besucht am 22. 01. 2018).

[Ubu18ao] Ubuntuusers. *opt*. 2018. URL: https://wiki.ubuntuusers.de/opt (besucht am 22. 01. 2018).

[Ubu18ap] Ubuntuusers. *pip*. 2018. URL: https://wiki.ubuntuusers.de/pip (besucht am 22. 01. 2018).

[Ubu18aq] Ubuntuusers. *ps*. 2018. URL: https://wiki.ubuntuusers.de/ps (besucht am 22. 01. 2018).

[Ubu18ar] Ubuntuusers. *rm*. 2018. URL: https://wiki.ubuntuusers.de/rm (besucht am 22. 01. 2018).

[Ubu18as] Ubuntuusers. *sudo*. 2018. URL: https://wiki.ubuntuusers.de/sudo (besucht am 22. 01. 2018).

[Ubu18at] Ubuntuusers. *wget*. 2018. URL: https://wiki.ubuntuusers.de/wget (besucht am 22. 01. 2018).

[Ubu18au] Ubuntuusers. *which*. 2018. URL: https://wiki.ubuntuusers.de/which (besucht am 22. 01. 2018).

[Wik18a] Wikipedia. *AlphaGo gegen Lee Sedol*. 2018. URL: https://de.wikipedia.org/wiki/AlphaGo_gegen_Lee_Sedol (besucht am 17. 01. 2018).

[Wik18b] Wikipedia. *Deep Blue*. 2018. URL: https://de.wikipedia.org/wiki/Deep_Blue (besucht am 17. 01. 2018).

[Wik18c] Wikipedia. *Tinder*. 2018. URL: https://de.wikipedia.org/wiki/Tinder (besucht am 30. 01. 2018).

Kolophon

Dieses Dokument wurde mit der LaTeX-Vorlage für Abschlussarbeiten an der htw saar der Fakultät für Wirtschaftswissenschaften im Bereich Wirtschaftsinformatik erstellt (Version 1.0). Die Vorlage wurde von Stefan Selle erstellt und basiert weitestgehend auf der entsprechenden Vorlage der Fakultät für Ingenieurswissenschaften, die von Yves Hary, André Miede, Thomas Kretschmer, Helmut G. Folz und Martina Lehser entwickelt wurde.