

**Technische Hochschule Mittelhessen  
Campus Friedberg**

Fachbereich 12 – Maschinenbau, Mechatronik, Materialtechnologie

Hausarbeit  
zur Vorlesung  
**Neuronale Netze**

Thema  
**Klassifizierung von Bildern  
Hunde und Katzen**

**vorgelegt von:**

Alois Rickert: 5334085

Daniel Stiehl: 5284630

Collins Kuetché: 997414

Vigny Sibanou: 5304293

**Prüfer:** Prof. Dr.-Ing. Dorra Baccar

**Abgabedatum:** 27.01.2021

## Inhaltsverzeichnis

I.	Abbildungsverzeichnis.....	II
II.	Tabellenverzeichnis.....	II
1	Einleitung.....	1
2	Aufgabenstellung.....	1
3	Grundlagen.....	2
4	Aufbau der Netze .....	3
4.1	From Scratch .....	3
4.2	Transferlearning PyTorch .....	4
4.2.1	Alex-Net Architektur .....	4
4.2.2	VGG Architektur: .....	5
4.2.3	ResNet .....	7
4.3	Transferlearning fast.ai .....	7
5	Ergebnisse .....	8
5.1	From Scratch .....	9
5.2	Transferlearning PyTorch .....	10
5.3	Transferlearning fast.ai .....	18
6	Bewertung des Datensatzes.....	19
7	Fazit .....	21
III.	Literaturverzeichnis .....	III

## I. Abbildungsverzeichnis

Abbildung 1: From Scratch entwickelte Netz-Architecture .....	3
Abbildung 2: Verfeinerung von Alex-Net Architektur .....	5
Abbildung 3: VGG-Net Architecture .....	6
Abbildung 4: ResNet Block .....	7
Abbildung 5: Res-Net-Blockarchitekturen mit unterschiedlichen Verwendungen von Aktivierungen .....	7
Abbildung 6: Erhöhung der Epochs .....	12
Abbildung 7: Hund oder Katze? .....	15
Abbildung 8: Transferlearning in fast.ai: Confusion Matrix .....	19
Abbildung 9: Fehlerhafter Datensatz: Icons, Schriftzüge, Logos .....	20
Abbildung 10: Schlechter Trainingsdatensatz: Hund und Katze zusammen.....	20
Abbildung 11: Fehlerhafter Datensatz: Tier zu klein abgebildet .....	20
Abbildung 12: Problematik: Tier hinter Gitter / Zaun .....	21

## II. Tabellenverzeichnis

Tabelle 1: VGG-Netz- Konfigurationen.....	6
Tabelle 2: Lernergebnisse mit zwei Output-Perzeptrons im letzten Layer. (Aktivierungsfunktion: Softmax, Loss-Funktion: Cross-Entropy).....	9
Tabelle 3: Lernergebnisse mit einem Output-Perzeptron im letzten Layer. (Aktivierungsfunktion: Sigmoid, Loss-Funktion: Binary-Cross-Entropy).....	10
Tabelle 4: Auswirkungen von Feature Extract.....	11
Tabelle 5: Änderung der Batchsize.....	12
Tabelle 6: Adam lr 0.001.....	13
Tabelle 7: Adam lr 0.01.....	14
Tabelle 8: Adam lr 0.1.....	14
Tabelle 9: VGG19.....	16
Tabelle 10: Bestes Ergebnis.....	17
Tabelle 11: Confusion Matrix für zwei Durchgänge.....	17
Tabelle 12: Transferlearning in fast.ai: Parameter und Ergebnisse.....	18
Tabelle 13: Darstellung der besten Ergebnisse.....	22

## **1 Einleitung**

Onlinedienste werden oft durch einen Test geschützt, der für einen Menschen leicht zu lösen ist, für ein Computerprogramm jedoch eine Schwierigkeit darstellt. Dies können beispielsweise verzerrte Texte oder eine Auswahl von Bildern sein. Hierdurch soll festgestellt werden, ob es sich bei dem Anwender wirklich um einen Menschen handelt [1].

Diese Methode wird als „CAPTCHA“ (Completely Automated Public Turing Test To Tell Computers and Humans Apart) oder „HIP“ (Human Interactive Proof) bezeichnet und wurde erstmals im Jahr 2000 an der Carnegie Mellon University in den USA eingeführt [1] [2].

„Asirra“ (Animal Species Image Recognition for Restricting Access) ist beispielsweise eine solche Methode. Dabei wird der Anwender aufgefordert Bilder von Hunden und Katzen auszuwählen. Die Besonderheit von „Asirra“ besteht in der Kooperation mit der Webseite Petfinder.com, der größten Plattform für Haustierversmittlungen. Das Portal konnte Microsoft Research Millionen von Hunden- bzw. Katzenbildern zu Verfügung stellen. Auf der Webseite Kaggle.com steht ein Teil diese Bilder als gelabelter Datensatz öffentlich zu Verfügung [1]. „Asirra“ gilt allerdings nicht mehr als sonderlich sicher. Heutige Algorithmen können problemlos eine Treffersicherheit von über 80% leisten [1].

Von der Webseite Kaggle.com wurde ein Wettbewerb gestartet, um herauszufinden, wie gut moderne Systeme hinsichtlich dieser Aufgabe trainiert werden können. Die zu gewinnenden Preise waren eher von symbolischer Art. So war der erste Preis eine Spende an eine Tierschutzorganisation und der zweite Preis ein Kuscheltier [1].

## **2 Aufgabenstellung**

Aufgabe dieser Hausarbeit ist es Deep-Learning Netze der Form Convolutional Neural Network (CNN) aufzubauen und auf für bereits beschriebene Problematik zu trainieren. Der Begriff CNN wird im folgenden Kapitel erläutert.

Es sollen CNNs „from Scratch“, also von Beginn an aufgebaut werden und Netze über Transferlearning angepasst werden.

Für Netze „from scratch“ sollen entsprechende Netzarchitekturen entworfen und implementiert werden, sowie mehrere Optimierungs- und Regulierungsmethoden angewandt werden. Das Ergebnis soll mit der Metrik „accuracy“ und mit einer Confusion-Matrix bewertet werden.

Beim Transfer Learning sollen vortrainierte Netze (PyTorch und fast.ai) eingesetzt und angepasst werden. Das Ergebnis von dieser Methode soll, nach den gleichen Kriterien, bewertet werden.

### **3 Grundlagen**

Eine Bilddatei besteht aus einer Matrix, deren Dimensionen der Auflösung des Bildes entspricht. Die Intensität eines Pixels wird mit einem Wert zwischen Null und 255 beschrieben. Da Farbbilder aus drei Kanälen bestehen, werden diese durch eine Überlagerung von drei Matrizen erzeugt. [3]

Um topografische Eigenschaften aus einem Bild zu erkennen, werden häufig „Convolutional Neural Networks“, kurz CNN, eingesetzt. Der Unterschied zu einem „Multi-Layer-Perceptrons“, kurz MLP, liegt darin, dass der Eingang nicht aus einem Vektor, sondern aus einer Matrix besteht. Zwar kann ein Bild auch als ein Vektor dargestellt werden (Flattening), allerdings ist das Neuronale Netz dann nicht in der Lage die Position des Objekts im Bild zu erkennen. Ein gleiches Objekt in einer anderen Positionierung würde durch einen grundlegend anderen Input dargestellt werden. [4]

Der Begriff „Convolution“ lässt sich mit „Faltung“ übersetzen. In diesem Zusammenhang ist damit gemeint, dass auf Bilder ein sogenannter Filter angewendet wird.

Ein Filter besteht ebenfalls aus einer Matrix, von deutlich geringeren Dimensionen. Um einen Filter auf ein Bild anzuwenden, wird die Matrix, die das Bild beschreibt mit der Filter-Matrix multipliziert. Hierdurch entsteht eine Matrix mit geringeren Dimensionen als das Ursprungsbild. Das Ergebnis wird unter anderem auch als Feature-Map bezeichnet. Auf die Feature-maps selbst kann im Weiteren wieder ein Filter angewendet werden. Durch einen Pooling Layer können anschließend die relevanten Signale aus einer Ausgabe gefiltert werden.

Die Ergebnisse jedes Layers werden meist mit einer ReLU Funktion aktiviert. Hierdurch werden alle negativen Werte auf Null gesetzt. Die übrigen Werte bleiben unverändert.

Bei einer Klassifizierung wird eine Softmax-Aktivierung angewandt. Die Output-Neuronen stellen dadurch die Wahrscheinlichkeit für die jeweilige Klasse dar. Die Summe der Ergebnisse aller Output-Neuronen und somit aller Klassen beläuft sich auf Eins.

## 4 Aufbau der Netze

### 4.1 From Scratch

Mit dem Begriff „From Scratch“ ist ein Netz, bzw. ein Code gemeint, das/der von Null an aufgebaut wird. Die Gewichte, werden meist zufällig zugeteilt. In der Abbildung 1 ist die entwickelte Netzstruktur dargestellt. Es besteht hier aus fünf Convolution-Layern und vier Fully-Connected-Layern.

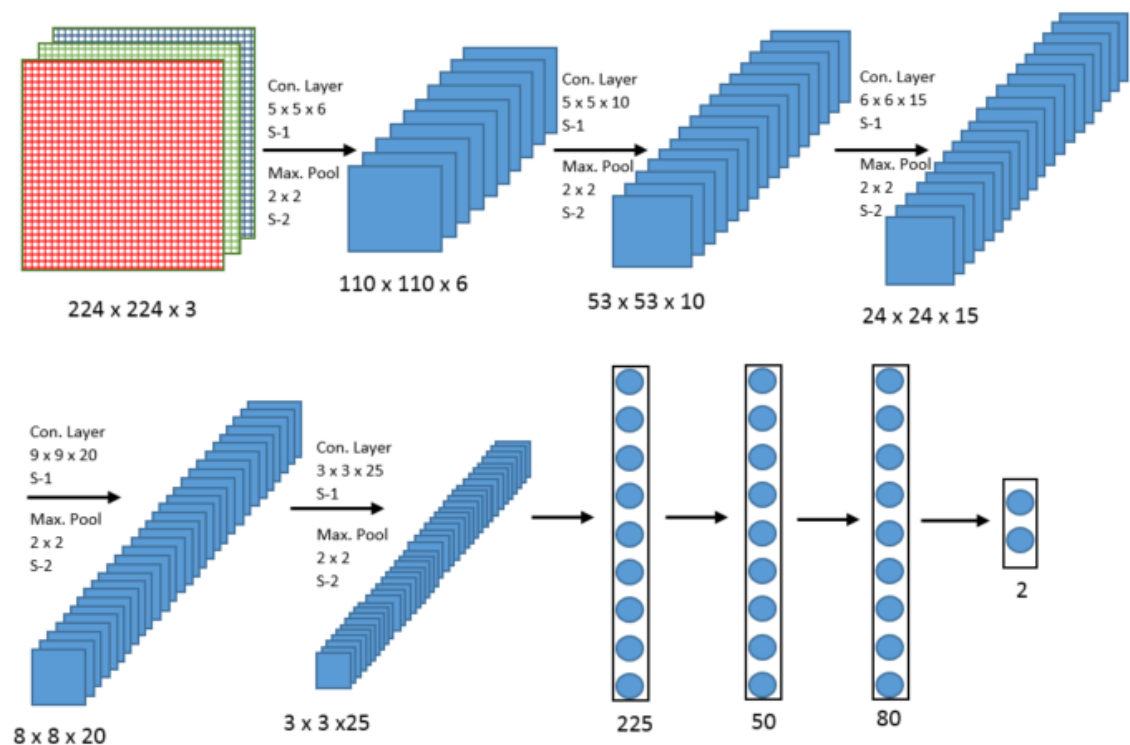


Abbildung 1: From Scratch entwickelte Netz-Architecture

Da das Netz mit zwei Outputs im letzten Layer definiert ist, wird hier die Aktivierungsfunktion Softmax verwendet und somit wird die CE-Fehlerfunktion (Cross Entropy) genutzt.

Allerdings besteht auch die Möglichkeit das Netz mit einem Output, statt zwei zu entwickeln. In dem Fall wird eine BCE-Fehlerfunktion (Binary Cross Entropy) und die Sigmoid Aktivierungsfunktion verwendet oder auf die Aktivierungsfunktion verzichtet und die BCEWithLogits-Fehlerfunktion herangezogen, da diese Fehler nichts anders als eine Kombination von Sigmoid-Funktion und BCE-Fehlerfunktion darstellt.

## 4.2 Transferlearning PyTorch

PyTorch wurde von einem Facebook-Forschungsteam für künstliche Intelligenz initiiert [5] und ist eine, auf maschinelles Lernen ausgerichtete, Open-Source-Programmbibliothek für Python. So stellt PyTorch verschiedene Modelle für unterschiedliche Anwendungsfälle, unter anderem für visuelle Verarbeitung, bereit. Für das Klassifizieren stehen folgende Modellarchitekturen über PyTorch zu Verfügung [5]:

- |              |              |               |
|--------------|--------------|---------------|
| • AlexNet    | • DenseNet   | • MobileNet   |
| • VGG        | • Inception  | • ResNeXt     |
| • ResNet     | • GoogLeNet  | • Wide ResNet |
| • SqueezeNet | • ShuffleNet | • MNASNet     |

Diese Modelle sind für die Aufgabe der Bildsegmentierung ausgelegte Modellarchitekturen und wurden bereits vortrainiert. Bei der Anpassung an ein Problem bzw. an einen Datensatz kann zum einen nur der Output-Layer angepasst und nachtrainiert werden. Hierbei bleiben die übrigen Layer mit den zugehörigen Gewichten konstant. Zum anderen können mehrere, bzw. alle Layer auf Grundlage des vortrainierten Standards nachtrainiert werden, um bessere Ergebnisse zu erhalten [3].

### 4.2.1 Alex-Net Architektur

Das Alex-Net besteht aus 8 Schichten. Als Eingabe wird ein 224 x 224 großer Ausschnitt eines Bildes (mit 3 Farbebenen) präsentiert. Dieser wird mit 96 verschiedenen Filtern der ersten Schicht (rot) gefaltet, jeder mit einer Größe von 7 x 7, unter Verwendung einer Schrittweite von zwei in x und y. Die resultierenden Merkmalskarten werden dann:

- Durch eine entzerrte lineare Funktion geleitet.

- Gepoolt (max. innerhalb von 3x3 Regionen, unter Verwendung von Strieder 2) und
- Kontrastnormalisiert über die Merkmalskarten hinweg, um 96 verschiedene Merkmalskarten mit 55 Elementen zu erhalten.

Ähnliche Operationen werden in den Schichten 2, 3, 4 und 5 wiederholt. Die letzten beiden Schichten sind voll verschaltet und nehmen die Merkmale aus der obersten Faltungsschicht als Eingabe in Vektoren von 9216 Dimensionen ( $6 \times 6 \times 256$ ). Die letzte Schicht ist eine C-fache Softmax-Funktion, wobei C die Anzahl der Klassen ist (In Abbildung- 2). Alle Filter und Merkmalskarten haben eine quadratische Form [6].

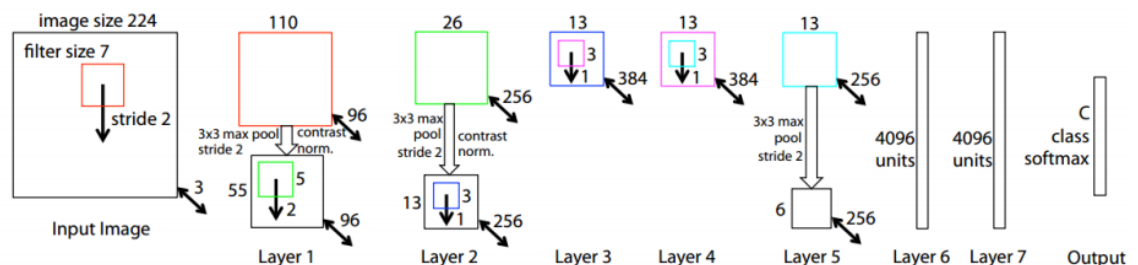


Abbildung 2: Verfeinerung von Alex-Net Architektur

#### 4.2.2 VGG Architektur:

Der Zweitplatzierte der ILSVRC (ImageNet Large Scale Visual Recognition Challenge) 2014 war das Netzwerk von Karen Simonyan und Andrew Zisserman, das als VGGNet bekannt wurde. Ihr Hauptbeitrag bestand darin, zu zeigen, dass die Tiefe des Netzwerks eine kritische Komponente für eine gute Leistung ist. Ihr endgültiges bestes Netz enthält 16 CONV/FC-Schichten und besteht aus einer homogenen Architektur, die von Anfang bis Ende nur 3x3 Faltungen und 2x2 Pooling durchführt. Ein Nachteil des VGG-Nets ist, dass es in der Auswertung teuer ist und viel mehr Speicher und Parameter (140M) benötigt. Die meisten dieser Parameter befinden sich in der ersten vollverknüpften Schicht. Inzwischen hat sich herausgestellt, dass diese FC-Schichten ohne Leistungseinbußen entfernt werden können, wodurch die Anzahl der erforderlichen Parameter deutlich reduziert wird [7].





In Tabelle- 1 ist die Architektur der ConvNet-Konfigurationen von VGG dargestellt. Die Tiefe der Konfigurationen nimmt von links (A) nach rechts (E) zu, wenn weitere Schichten hinzugefügt werden (die hinzugefügten Schichten sind fett dargestellt). Die Parameter der Faltungsschichten werden als „Faltungsfeldgröße - Anzahl der Kanäle“ bezeichnet.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256	conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512	conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512	conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Network	A, A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

### 4.2.3 ResNet

Sehr tiefe Lernnetzwerke sind aufgrund der Probleme mit dem verschwindenden Gradienten notorisch schwer zu trainieren. Je tiefer die Netze werden, desto schneller verschlechtert sich die Leistung. ResNet wurde entwickelt, um dieses Problem zu lösen. Die Kernidee von ResNet ist die „Identitätskurzschlussverbindung“, die eine oder mehrere Schichten überspringt, Wie in Abbildung- 5 gezeigt:

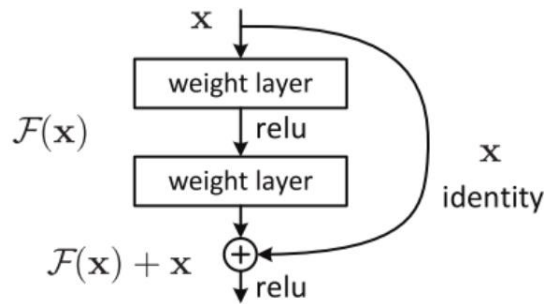


Abbildung 4: ResNet Block

Die Autoren behaupten, dass das Stapeln von Schichten die Leistung des Netzwerks nicht beeinträchtigen sollte, da viele der Schichten am Ende nur eine Identitätszuordnung durchführen und somit nichts tun. Dies deutet darauf hin, dass Restverbindungen es den Schichten leichter machen, die gewünschte zugrundeliegende Abbildung zu erfüllen. Abbildung- 6 veranschaulicht die verschiedenen Varianten, wie Restverbindungen in einem Block angeordnet werden können [8].

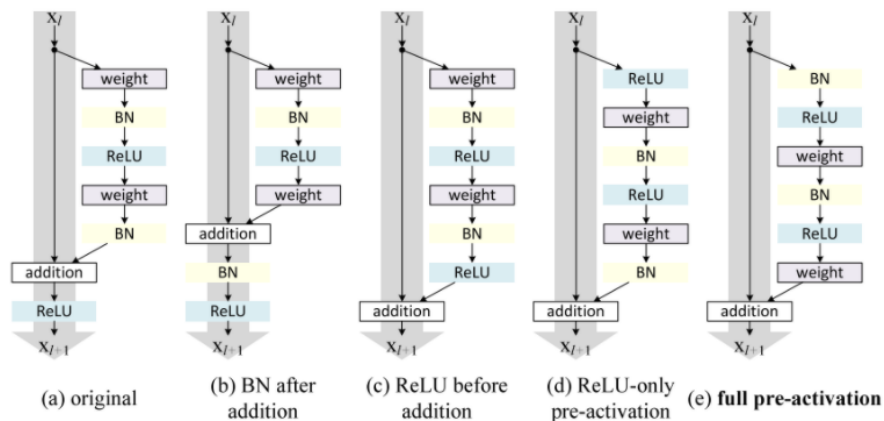


Abbildung 5: Res-Net-Blockarchitekturen mit unterschiedlichen Verwendungen von Aktivierungen

### 4.3 Transferlearning fast.ai

Die non-profit Organisation fast.ai hat sich zum Ziel gemacht die Anwendung von Deep Learning zu vereinfachen und bietet kostenlose Programmierkurse, sowie nutzerfreundliche Softwarebibliotheken an. [9] Durch die Bibliothek ist es in der

Praxis möglich schnell und einfach gute Ergebnisse für standardisierte Probleme zu erzielen. Der Training-Prozess läuft äquivalent zum Transferlearning in yTorch ab. D. h. es können verschiedene vortrainierte Netzarchitekturen geladen werden. Anschließend können das gesamte Netz, nur der letzte Layer oder mehrere Layer trainiert werden [9].

Auf das Problem zur Klassifizierung von Hunde- und Katzenbildern lässt sich fast.ai einfach anwenden. Es wird, wie in dem Vorlesungspraktikum, eine vortrainierte ResNet Netzstruktur mit 34 bzw. 50 Layern verwendet und das Label direkt über eine fast.ai-Funktion, bzw. über die Pythonstandardbibliothek Regex, aus dem Dateinamen generiert. Der Dateipfad muss auf das Verzeichnis in Google Drive verwiesen werden. Über eine Funktion in fast.ai können die Trainingsdaten direkt geladen werden. Gleichzeitig wird der Datensatz in Trainings- und Validierungsdaten aufgeteilt und in Batches geordnet, sowie das Bild transformiert und skaliert. Es muss nur die Batchsize und die Bildgröße definiert werden, die restlichen Parameter können optional angepasst werden, ansonsten können die Default-Werte von fast.ai genutzt werden. Der letzte Layer eines vortrainierten Netzes lässt sich, wie bei PyTorch, über eine Funktion in das Collab Notebook laden und durch eine weitere Funktion ausführen, hierzu muss lediglich die Anzahl der Epochs festgelegt werden. Durch die Funktion wird beim Trainieren zu jeder Epoch ein Ladebalken mit Prozent angezeigt. Bei einem Unfreeze muss zu der Anzahl der Epoch noch die Lernrate definiert werden. Hierzu gibt es eine fast.ai Funktion, die einen Plot des Losses in Abhängigkeit der Lernrate ausgibt.

Das Ergebnis lässt sich zum einen direkt in der Ausgabe der Funktion zum Ausführen des Trainings bzw. der Validierung entnehmen. Für jede Epoch wird das Trainings- und Validationloss und die Fehlerrate ausgegeben. Zusätzlich wird noch die Trainingszeit jeder Epoch angezeigt. Des Weiteren kann das Ergebnis durch weitere in fast.ai enthaltene Funktionen bewertet werden. Beispielsweise kann eine Auswahl der Daten mit den höchsten Losses ausgegeben oder eine Confusion-Matrix generiert werden.

## **5 Ergebnisse**

Für die Implementierung der Neuronalen Netze wird ausgehend von Codebeispielen aus dem Dog vs. Cats Wettbewerb [1] und den Vorlesungsunterlagen [3],

bzw. den entsprechenden Beispielen aus der Übung, zu jedem Beispiel ein Basiscode geschrieben, der in verschiedenen Varianten optimiert wird. Es werden Tabellen zum Vergleich der Parameter und Ergebnisse dargestellt.

## 5.1 From Scratch

In folgender Tabelle sind Lernergebnisse mit unterschiedlichen gesammelt.

*Tabelle 2: Lernergebnisse mit zwei Output-Perzeptrons im letzten Layer. (Aktivierungsfunktion: Softmax, Loss-Funktion: Cross-Entropy)*

Nb. Epoch	20	20	20	20	20	20	20
Batch Size	32	32	32	32	32	32	16
Optimizer	SGD	SGD	RMB-prop	Adam	SGD	SGD	SGD
Lernrate	0,01	0,01	0,01	0,01	0,01	0,1	0,01
Regulazierung	-	-	-	-	Drop-out(0,5)	-	-
Train Acc.	0,7504	0,4923	0,4997	0,4997	0,8663	0,5003	0,8462
Validation Acc.	0,8080	0,5012	0,5012	0,5012	0,8564	0,4988	0,8480
	M. Pool + Relu	A. Pool + Tanh	M. Pool + Relu	M. Pool + Relu	M. Pool + Relu	M. Pool + Relu	M. Pool + Relu

Wie es in der Tabelle zu sehen ist, wird für einen Trainingsdurchlauf ein Average-Pooling und Tangens-Hyperbolicus als Aktivierungsfunktion statt Max-Pooling und ReLu(Rectified Linear Unit) verwendet. Die beste Accuracy wird mit folgenden Parametern erzielt:

- Optimizer: SGD mit momentum 0,9
- Lernrate 0,01
- Batch-Size: 32
- Regulazierung: Dropout(0,5)

Wie schon in Kapitel 4.1 erwähnt, ist es auch möglich, das Netz mit einem Output zu entwickeln. Für diesen Fall sind die Ergebnisse in der Tabelle 2 aufgeführt.

*Tabelle 3: Lernergebnisse mit einem Output-Perzeptron im letzten Layer. (Aktivierungsfunktion: Sigmoid, Loss-Funktion: Binary-Cross-Entropy)*

Nb. Epoch	20	20	20	20
Batch Size	32	32	32	32
Optimizer	SGD	Adam	SGD	SGD
Lernrate	0,01	0,01	0,01	0,001
Regulasie- rung	-	-	-	-
Train Acc.	0,5011	0,4989	0,5009	0,4994
Validation Acc.	0,4956	0,5044	0,4962	0,5026
	M. Pool + Relu	M. Pool + Relu	M. Pool + Relu	M. Pool + Relu

## 5.2 Transferlearning PyTorch

Der Basiscode für die PyTorch-Variante wurde ursprünglich von einem professionellen Programmierer auf Kaggle hochgeladen. Das Feintuning und die Anpassung an unsere Umgebung erweist sich jedoch als aufwendig.

In diesem Kapitel sollen vor allem verschiedene Ansätze zur Optimierung gezeigt werden, bzw. auch Probleme, die auftreten können.

Da bereits erste Probleme mit dem Hochladen der Datensätze auftreten, werden bei diesem Code die Bilder direkt aus einem ZIP-Archiv entpackt. Dies erfordert zwar bei jedem Durchlauf Zeit zum Entpacken, aber der Datensatz ist im ZIP-Format innerhalb von wenigen Minuten hochgeladen und lässt sich dadurch flexibler handhaben.

Die beiden Klassen, Hund und Katze, werden bei diesem Code durch den Dateinamen generiert, der mit Funktionalitäten der GLOB-Bibliothek zerlegt wird.

Es wird das VGG-16 pretrained importiert, wobei verglichen wird wie es sich auswirkt, nur die letzte neuronale Schicht oder alle Schichten zu trainieren:

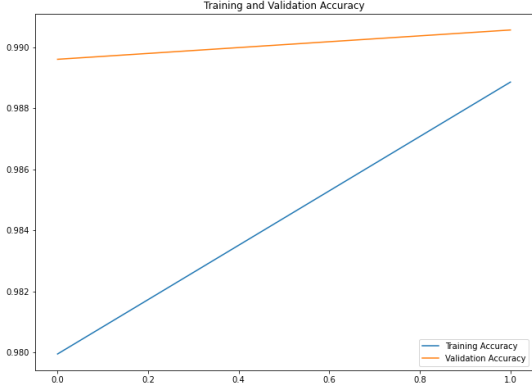
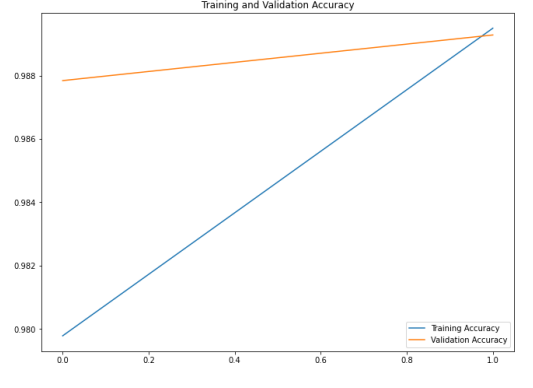
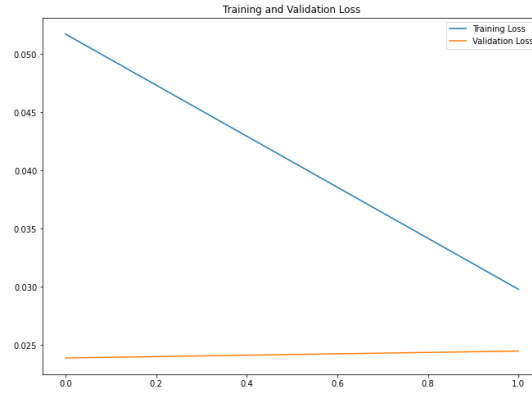
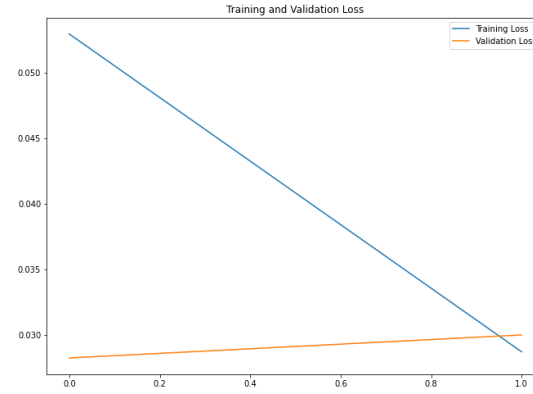
Nur letztes Layer trainiert	Alle Layer trainiert
batch_size=32 num_epochs=2	batch_size=32 num_epochs=2
Training complete in 20m 33s Best val Acc: 0.990560	Training complete in 20m 8s Best val Acc: 0.989280
	
	

Tabelle 4: Auswirkungen von Feature Extract

Bei in etwa gleicher Zeitdauer (Differenzen sind auch durch die zugeteilte GPU bedingt) erkennt man eine in etwa gleiche Genauigkeit, aber dafür in den Grafiken eine deutlich bessere Accuracy und einen sehr guten Verlauf der Lossfunktion. Es empfiehlt sich also für das Feintuning des Netzes, nachdem alle Parameter soweit eingestellt sind, sämtliche Schichten zu trainieren.

Die folgenden Werte erweisen sich als die Besten: input\_size = 224, mean = [0.485, 0.456, 0.406] und std = [0.229, 0.224, 0.225]. Diese Werte entsprechen auch den Expertenmeinungen der aktuellen Forschung. [3]

Im nächsten Versuch wird die Anzahl der Epochs erhöht, um die Accuracy weiter zu steigern. Dabei zeigt sich bei drei Epochs eine gleichbleibende Best Value Accuracy, aber eine deutliche Verschlechterung der Grafiken:

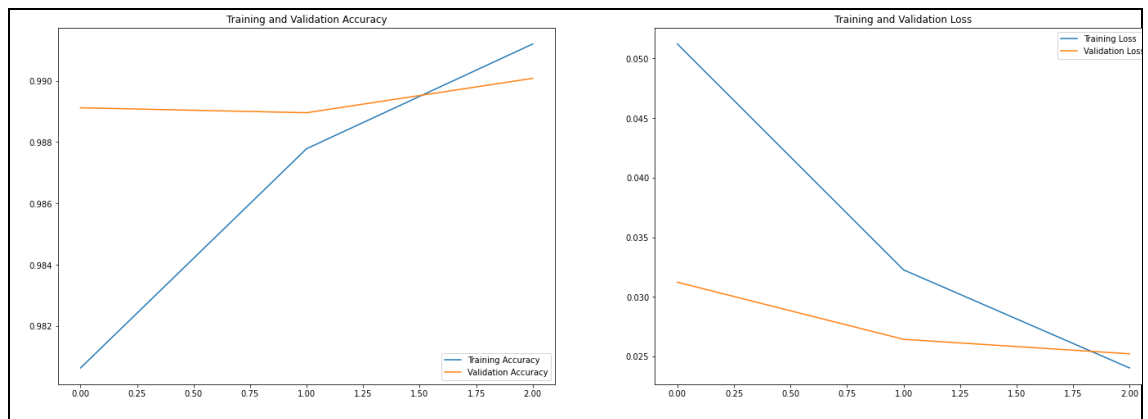
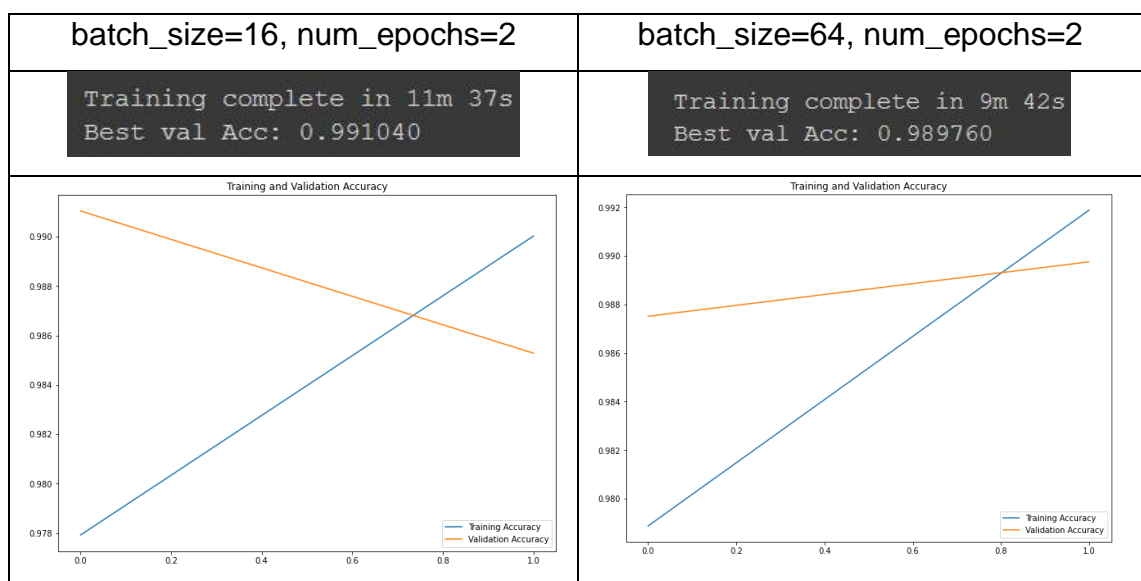


Abbildung 6: Erhöhung der Epochs

Bei einer größeren Anzahl als zwei Epochs ergibt sich ein beginnendes Overfitting der Funktionen. Weitere Experimente mit höheren Epochzahlen werden daher nicht durchgeführt.

Als nächstes wird versucht die Batchsize zu ändern. Die bisher genutzte Batchsize von 32 wird versuchsweise auf 16, 64 und andere Werte gesetzt. Da dies nicht zu einer Verbesserung der Accuracy führt, sind hier nur die Ergebnisse mit 16 und 64 dargestellt:



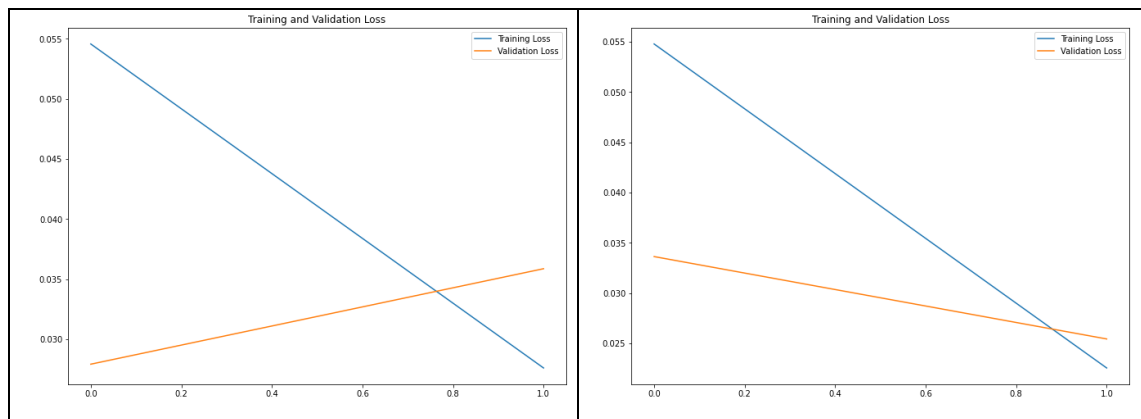


Tabelle 5: Änderung der Batchsize

Das Overfitting wird schlechter, aber die Laufzeit verbessert sich. Im Sinne eines möglichst guten Netzes kann aber eine längere Laufzeit in Kauf genommen werden.

Es wird nun versucht den Optimizer anstatt mit SGD, wie bisher, mit Adam laufen zu lassen, um möglicherweise Verbesserungen zu erzielen. Dabei wird auch die Lernrate variiert. Änderungen der Lernrate und des Momentums bei SGD erweisen sich als nicht erfolgreich.

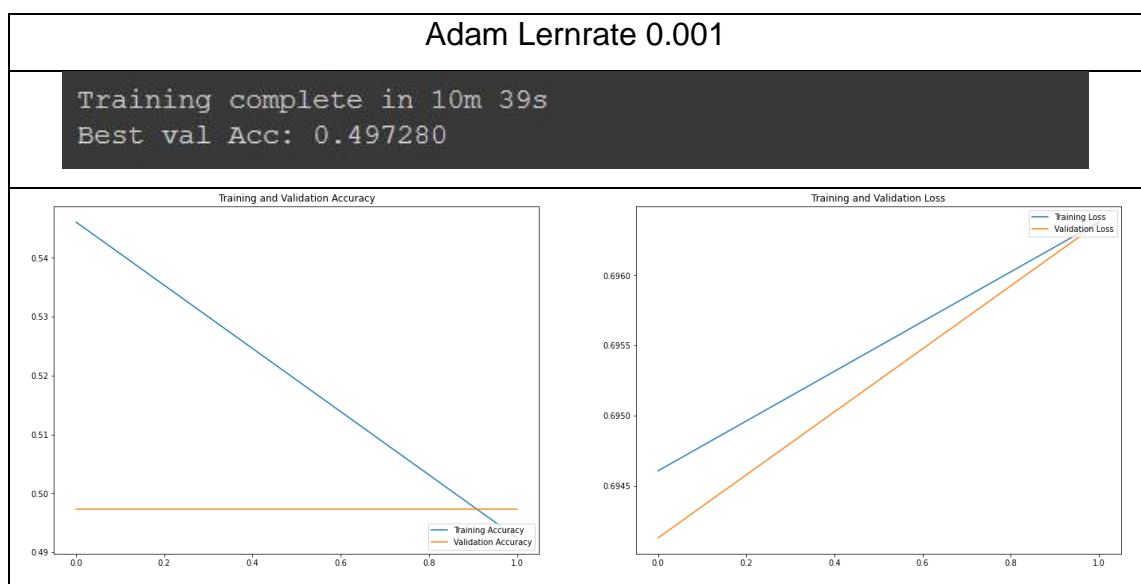


Tabelle 6: Adam lr 0.001



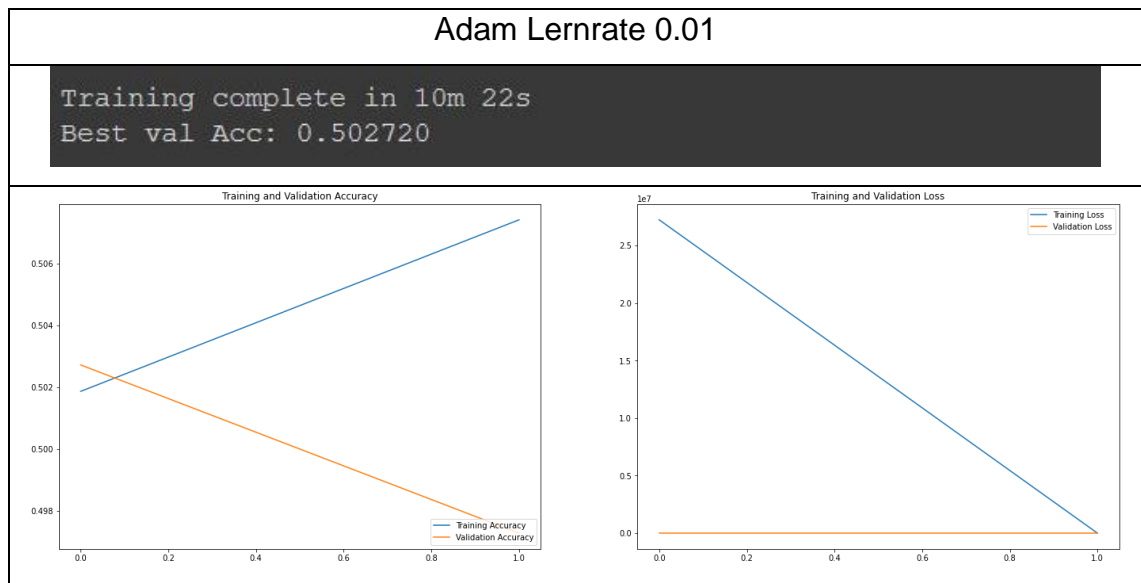


Tabelle 7: Adam lr 0.1

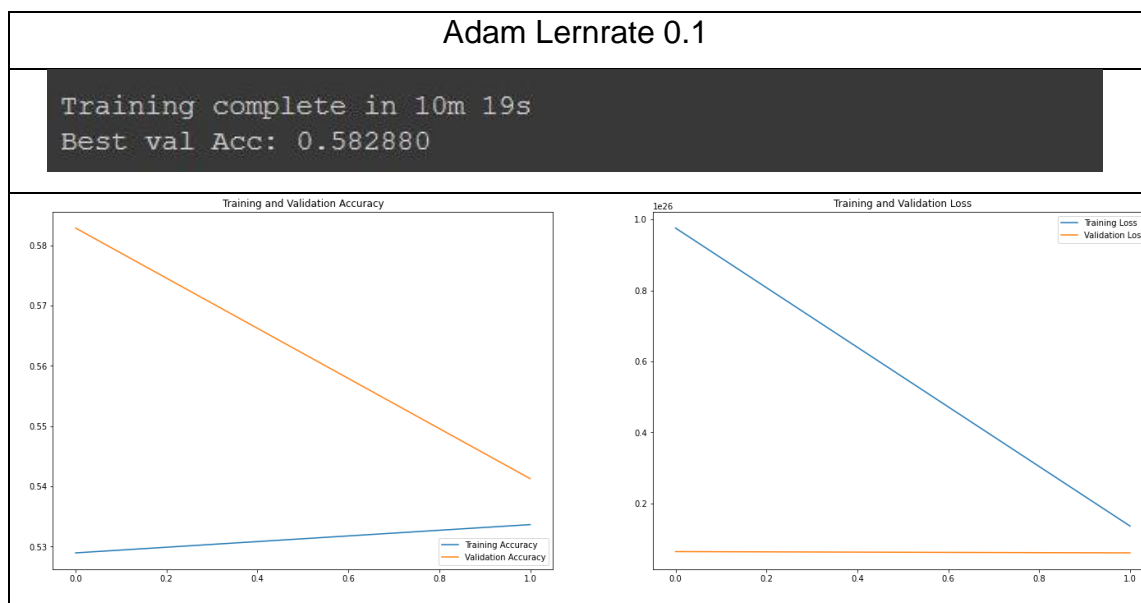


Tabelle 8: Adam lr 0.1

Wie man sieht sind die Ergebnisse bei zwei Epochs sehr schlecht. Teilweise so schlecht, dass der Algorithmus nur noch zufällig zwischen Hund und Katze unterscheiden kann:

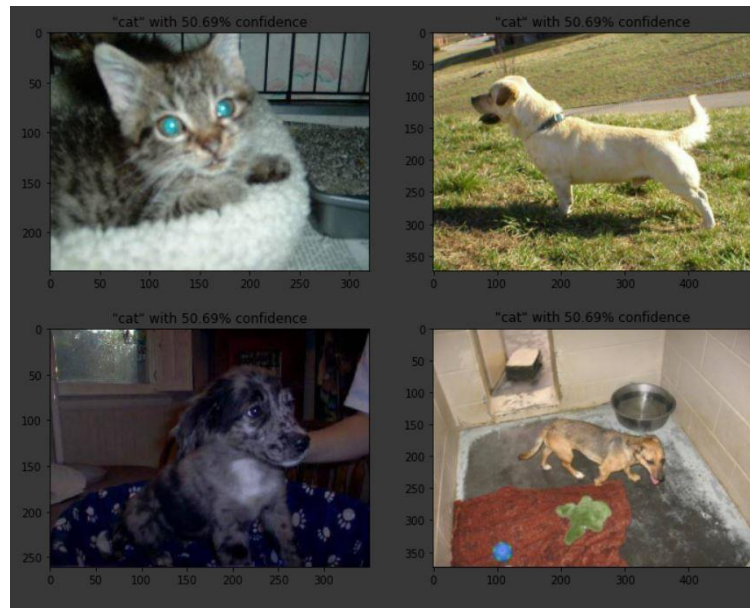
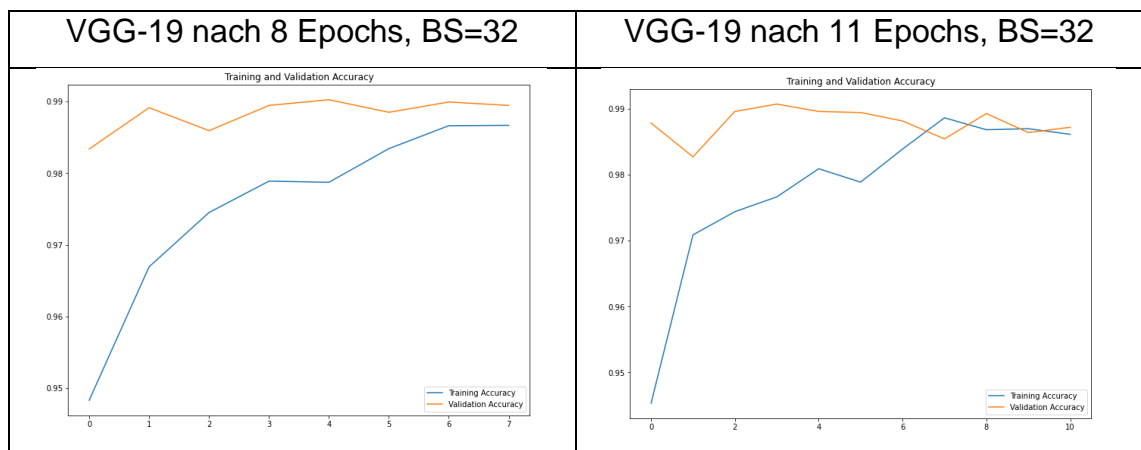


Abbildung 7: Hund oder Katze?

Bei einer höheren Zahl Epochs wird das Ergebnis langsam besser, aber dadurch erhöht sich die Trainingszeit merklich. Der Ansatz mit Adam ist nicht zielführend.

Es wird also weiterhin SGD mit einer Lernrate von 0.001 und einem Momentum von 0.9 verwendet. Als letzte Variation soll noch das VGG-19-Netz geladen werden, dass über drei Layer mehr wie das VGG-16-Netz verfügt. Es werden 8 Epochs und 11 Epochs trainiert:



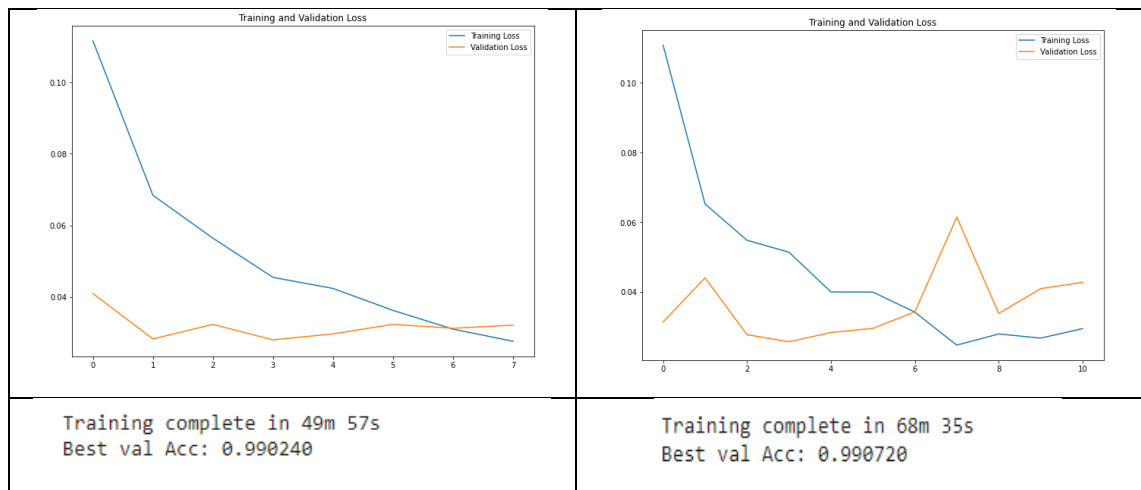
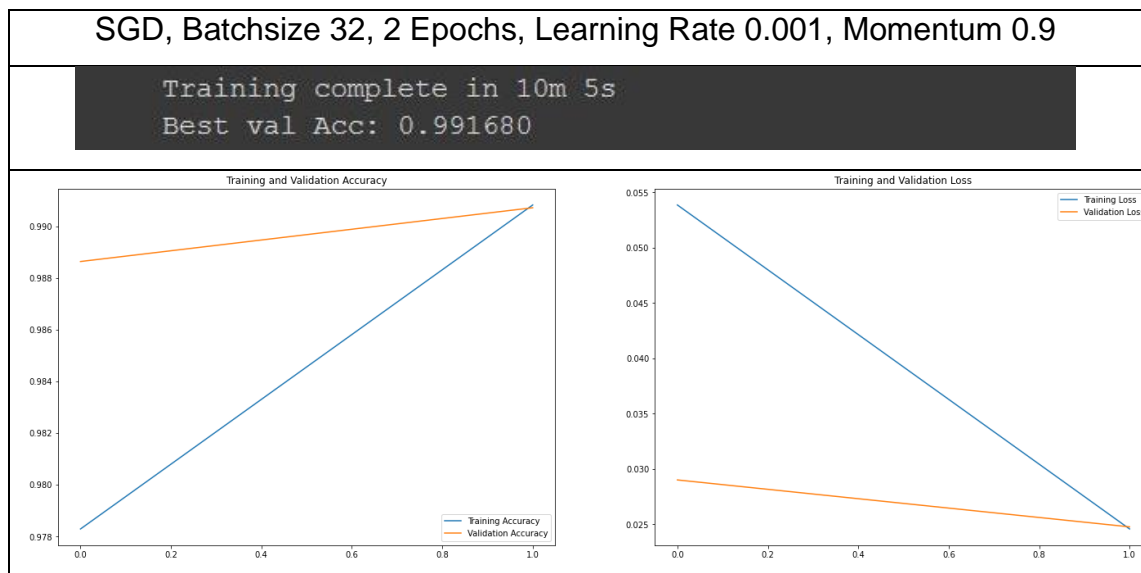


Tabelle 9: VGG19

Leider bringt VGG-19 keine deutlich besseren Ergebnisse, sondern erhöht lediglich die Trainingszeit. Es wird also weiterhin VGG-16 verwendet.

Es gibt allerdings noch einen Hinweis zur Codekorrektur des Dataloaders von Frau Prof. Baccar. Die Transformationen für das Imagecropping werden neu geordnet und ergänzt. Es geht dabei darum, die Eingabebilder für eine größere Datenmenge zuzuschneiden und in der Größe zu variieren.

Die abschließend beste Konfiguration mit den Anpassungen im Dataloader liefert dann folgende Ergebnisse:



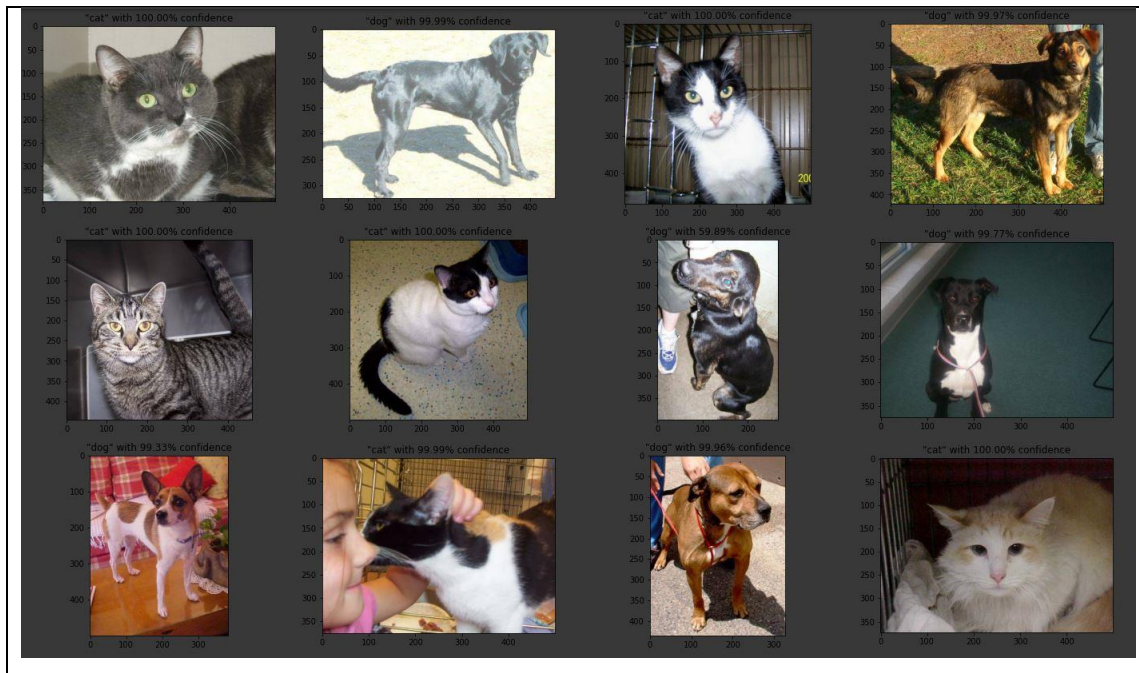


Tabelle 10: Bestes Ergebnis

Die erstaunliche Accuracy von über 99% bei nur zwei Epochs ist unerreichbar. Im folgenden Kapitel mit fastai werden 8 Epochs und damit deutlich mehr Zeit benötigt. Der effektive Aufbau des Pytorchnetzes mit Early Stopping nach zwei Epochs, sehr kleiner Lernrate und SGD als Optimizer ist unser Favorit.

Die zugehörige Confusion Matrix ist hier für zwei verschiedene Durchgänge aufgeführt:

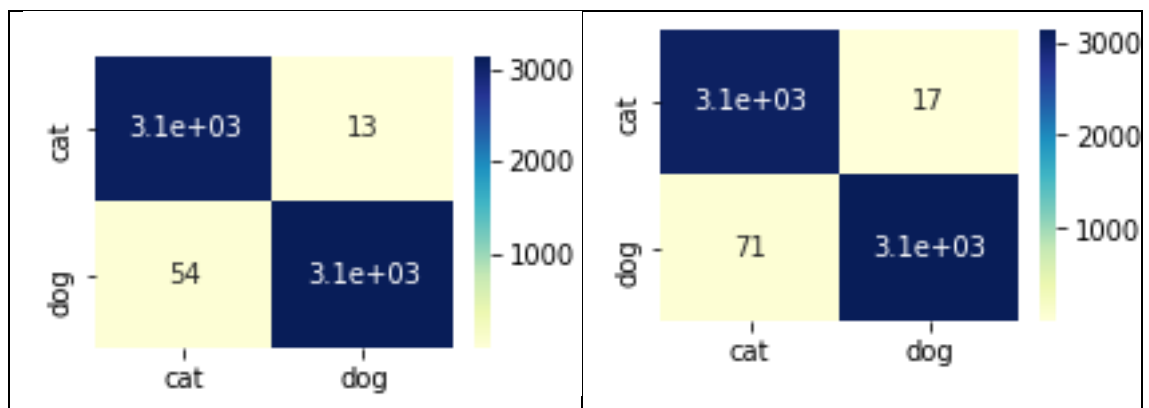


Tabelle 11: Confusion Matrix für zwei Durchgänge

Es wird auch versucht das trainierte Modell abzuspeichern und erneut für das Training zu verwenden. Die Accuracy, Graphen und Matizen verbessern sich jedoch nicht, sondern verschlechtern sich wieder.

### 5.3 Transferlearning fast.ai

Beim Transferlearning mit ResNet über fast.ai werden, ähnlich wie bei Pytorch, gute bis sehr gute Ergebnisse erwartet. Im Folgenden werden vier Trainings durchgeführt: Zu einer 34- und zu einer 50-Layer Netzstruktur wird je ein Training nur im letzten Layer und mit dem gesamten Netz durchgeführt. Bei jedem Training wurde mit einer Batch-Size von 32 gearbeitet und es wurden bei allen Varianten als Vergleich 8 Epochen durchlaufen, weil sonst die Laufzeit zu sehr ansteigt.

ResNet				
<b>Anzahl Layer insgesamt</b>	34	34	50	50
<b>Anzahl Trainierter Layer</b>	1 (letzter)	Alle	1 (letzter)	Alle
<b>Batch Size</b>	32	32	32	32
<b>Lernrate</b>	n.A.	$10^{-6} - 10^{-4}$	n.A.	$10^{-6} - 10^{-4}$
<b>Epochs</b>	8	8	8	8
<b>Zeit pro Epoch (Min.)</b>	Ø 4:44,1	Ø 4:53,4	Ø 7:10,9	Ø 8:03,3
<b>Bestes Trainingsloss</b>	0,028 bei Epoch 7	0,007 Bei Epoch 8	0,011 bei Epoch 8	0,017 bei Epoch 7
<b>Bestes Validationloss</b>	0,031 bei Epoch 8	0,0334 bei Epoch 1	0,021 bei Epoch 1	0,017 Bei Epoch 8
<b>Beste Accuracy</b>	0,9914	0,9914	0,9942	0,9936

Tabelle 12: Transferlearning in fast.ai: Parameter und Ergebnisse

Bei dem Training des gesamten Netzes wurde eine Lernrate festgelegt. Diese wurde durch eine in fast.ai integrierte Funktion ermittelt. Die beste Accuracy wird durch eine Netzstruktur mit 50 Layern und Training des letzten Layers erzielt. Es ist denkbar, dass durch eine optimierte Lernrate und eine längere Trainingszeit

bei dem Training des Gesamten 50-schichtigen Netzes noch bessere Ergebnisse zu erreicht werden.

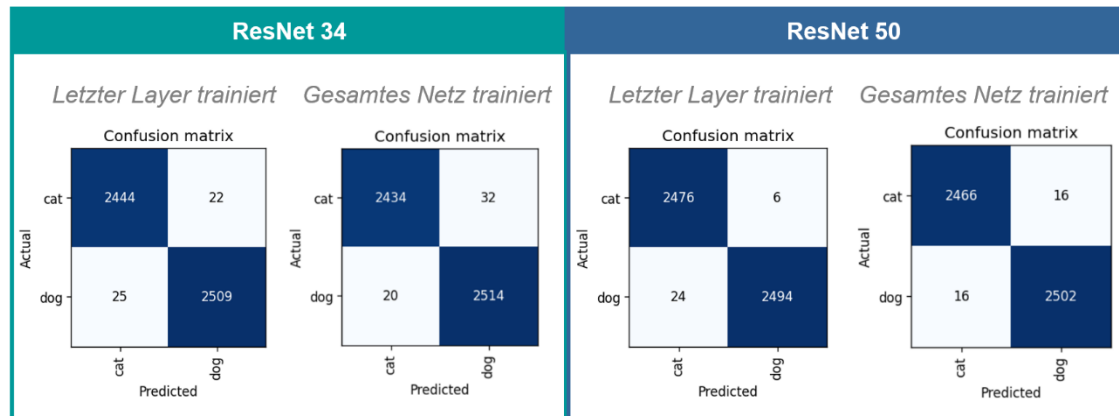


Abbildung 8: Transferlearning in fast.ai: Confusion Matrix

Anhand der abgebildeten Confusion Matrizen ist erkennbar, wie sich die richtigen und falschen Zuordnungen nach den beiden Klassen verteilen. Bei dem nur im letzten Layer trainierten ResNet50 Modell wird zwar die beste Accuracy erzielt, allerdings ist vollständig trainierten ResNet50 Modell die Verteilung ausgewogener.

## 6 Bewertung des Datensatzes

Bei der Auswertung von falsch vorhergesagten Klassifizierungen, ist uns aufgefallen, dass einige Daten aus dem in Abschnitt 1 beschriebenen Datensatz, fehlerhaft sind. Es wurden vermutlich Bilder aus Inseraten von Vermittlungsportalen ohne manuelle Überprüfung übernommen und automatisch über die inserierte Kategorie gelabelt.

Webcrawler, bzw. Bots, suchen automatisiert nach den Stichworten „cat“ und „dog“, um den Datensatz zu erstellen [3].

War kein Foto des Tieres verfügbar, hat der Inserent beispielsweise ein Icon, einen Schriftzug oder ein Logo der Organisation hochgeladen:





Abbildung 9: Fehlerhafter Datensatz: Icons, Schriftzüge, Logos

Auf einigen Fotos sind Hund und Katze abgebildet. Der Algorithmus sollte solche Daten nur zum Testen, aber nicht zum Lernen oder Validieren bekommen. Tests werden so anspruchsvoller.



Abbildung 10: Schlechter Trainingsdatensatz: Hund und Katze zusammen

Bei anderen Fotos ist das Tier nur sehr klein abgebildet, dadurch ist es für das Netz schwer bis unmöglich das Muster des Tieres darin zu erkennen:



Abbildung 11: Fehlerhafter Datensatz: Tier zu klein abgebildet

Teilweise sind auch gut erkennbare Fotos falsch gelabelt. Dies lässt sich möglicherweise dadurch erklären, dass der Inserent versehentlich die falsche Kategorie ausgewählt hat. Zusätzlich ist uns aufgefallen, dass ein Tier hinter einem

Gitter bzw. einem Zaun oftmals falsch zugeteilt wird, da das Netz vermutlich in erster Linie das Muster des Gitters erkennt.



*Abbildung 12: Problematik: Tier hinter Gitter / Zaun*

Die fehlerhaften Daten erschweren es das Netz zu trainieren und machen es unmöglich eine Accuracy von 100% zu erreichen.

## **7 Fazit**

Wie bereits in Kapitel 5.2 erwähnt, liefert der Pytorchcode in 2 Epochs das Beste Ergebnis, allerdings lässt sich aufgrund der Differenzen durch andere GPU-Zuteilung kein „Sieger“ ermitteln. Der Fastai-code liefert nach 8 Epochs eine minimal bessere Accuracy.

Allgemein lässt sich sagen, dass Transferlearning bessere Ergebnisse liefert, als from scratch. Dies deckt sich mit den Informationen aus der Vorlesung. [3] Große vortrainierte Netze, die auf Serverfarmen initialisiert wurden liefern aufgrund der besseren Ressourcen bessere Ergebnisse.



Abschließend sollen hier die besten Werte der jeweiligen Codes dargestellt werden:

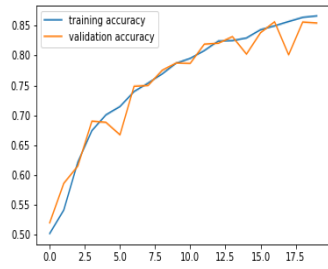
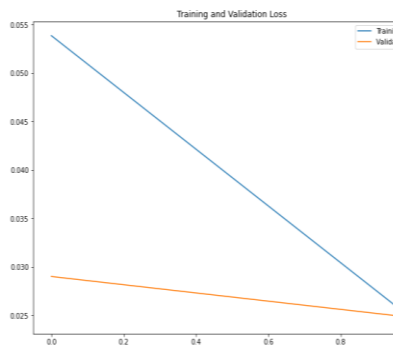
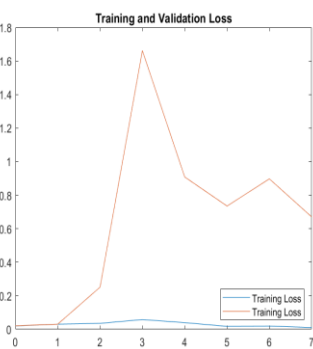
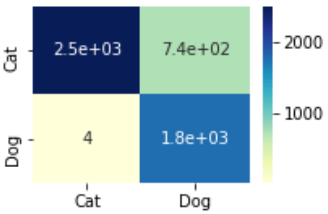
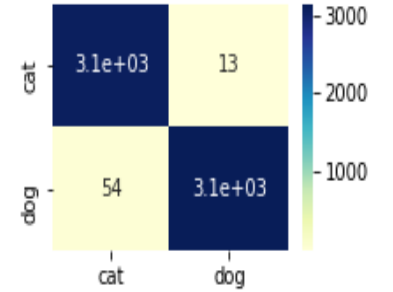
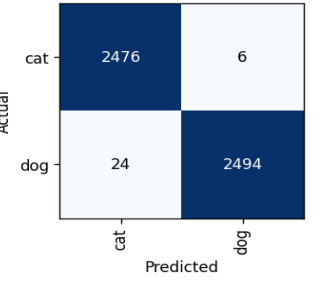
From scratch	Pytorch	Fastai												
SGD, BS:32, 20 Ep., Lr: 0,01, Mom: 0.9, dropout(0,5)	SGD, BS: 32, 2 Ep., Lr: 0.001, Mom.: 0.9	BS: 32, 2 Ep.												
2	<pre>Training complete in 10 Best val Acc: 0.991680</pre>	<table><tr><th>epoch</th><th>train_loss</th><th>valid_loss</th><th>error_rate</th></tr><tr><td>0</td><td>0.020148</td><td>0.021012</td><td>0.005800</td></tr><tr><td>1</td><td>0.030615</td><td>0.030796</td><td>0.006200</td></tr></table>	epoch	train_loss	valid_loss	error_rate	0	0.020148	0.021012	0.005800	1	0.030615	0.030796	0.006200
epoch	train_loss	valid_loss	error_rate											
0	0.020148	0.021012	0.005800											
1	0.030615	0.030796	0.006200											
														
														

Tabelle 13: Darstellung der besten Ergebnisse

### III. Literaturverzeichnis

- [1] Kaggle, „Dogs vs. Cats - Create an algorithm to distinguish dogs from cats,“ 2013. [Online]. Available: <https://www.kaggle.com/c/dogs-vs-cats>. [Zugriff am 15 01 2021].
- [2] Carnegie Mellon University, „CAPTCHA: Telling Humans and Computers Apart Automatically,“ 2000-2010. [Online]. Available: <http://www.captcha.net/>. [Zugriff am 15 01 2021].
- [3] P. D.-I. D. Baccar, „Vorlesung Neuronale Netze: Convolutional Neural Networks (CNN),“ Technische Hochschule Mittelhessen, Friedberg , 2020.
- [4] R. Becker, „CONVOLUTIONAL NEURAL NETWORKS – AUFBAU, FUNKTION UND ANWENDUNGSGEBIETE,“ JAAI, 06 02 2019. [Online]. Available: <https://jaai.de/convolutional-neural-networks-cnn-aufbau-funktion-und-anwendungsgebiete-1691/>. [Zugriff am 15 01 2021].
- [5] PyTorch, „torchvision.models — PyTorch 1.7.0 documentation,“ 2020. [Online]. Available: <https://pytorch.org/docs/stable/torchvision/models.html#classification>. [Zugriff am 15 01 2021].
- [6] M. Z. a. R. Fergus, „ „Visualizing and Understanding Convolutional Networks“, (Best Paper Award winner),“ [https://slazebni.cs.illinois.edu/spring17/lec01\\_cnn\\_architectures.pdf](https://slazebni.cs.illinois.edu/spring17/lec01_cnn_architectures.pdf), ECCV 2014 .
- [7] K. S. & A. Zisserman, „ „VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION“,“ karen, az}@robots.ox.ac.uk, Published as a conference paper at ICLR, Visual Geometry Group, Department of Engineering Science, University of Oxford, 2001.
- [8] K. K. & K. Dewang, „ „Lecture 7: CNN Architectures and Applications“,“ [https://www.seas.upenn.edu/~cis522/lecture\\_notes/lec7.pdf](https://www.seas.upenn.edu/~cis522/lecture_notes/lec7.pdf), CIS 522 - University Of Pennsylvania, Spring 2020 .

[9] J. Howard und R. Thomas, „About fast.ai,“ fast.ai, 2020. [Online]. Available: <https://www.fast.ai/about/>. [Zugriff am 16 01 2021].