INSTITUT FÜR INFORMATIK
Machine Learning

Universitätsstr. 1 D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Analyzing Brain Images with Deep Learning

ThoughtfulDev
(Name wurde geändert)

Bachelorarbeit

Beginn der Arbeit:	08. Mai 2018
Abgabe der Arbeit:	08. August 2018
Gutachter:	Prof. Dr. Stefan Harmeling
	Prof. Dr. Stefan Conrad

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 08. August 2018

_____

ThoughtfulDev

Zusammenfassung

In dieser Arbeit wird mit Hilfe von bildgebender Kernspintomografie (MRT) versucht, eine Vielzahl von Klassifikationen durchzuführen. Um diese Aufgabe erfolgreich zu meistern, werden die in Machine Learning immer mehr an Popularität gewinnenden, tiefen neuronalen Netze verwendet.
In unseren Experimenten werden verschiedene Regularisierungs- und Normalisierungsmethoden gebraucht und auch verglichen, um das bestmögliche Ergebnis zu erzielen. Außerdem werden potenzielle Probleme im Bezug auf den Datensatz aufgegriffen, erklärt und anschließend werden Möglichkeiten vorgestellt, um diese Probleme zu beheben. Auch wird versucht durch Konvolutionsfilter zu erkennen, wie genau das neuronale Netz zwischen Klassen unterscheidet.
Des Weiteren werden Gemeinsamkeiten verschiedener Klassenmerkmale analysiert und versucht Zusammenhänge zu erkennen.
Unsere Netzwerke ermöglichen am Ende die akkurate Bestimmung des Geschlechts einer Person anhand eines MRT Scans. Bei der Klassifizierung der Händigkeit übertrifft das neuronale Netz den Menschen bzw. Arzt im Bezug auf die Genauigkeit. Auch das Alter kann mit einer erstaunlichen Genauigkeit geschätzt werden.

Inhaltsverzeichnis

1 Einleitung

Machine Learning wird in der heutigen Zeit in sehr vielen Bereichen eingesetzt. Unter anderem findet Machine Learning Anwendung im Finanzbereich und sogar in der Medizin. Dort wurde Machine Learning bereits in Bereichen wie Krankheitserkennung und Medikamentensynthese[1] gebraucht.
In unseren nun folgenden Ausführungen werden wir uns auf die bildgebender Kernspintomografie fokussieren.
Moderne MRT Scanner sind in der Lage, hochauflösende Bilder von menschlichen Gehirnen zu erzeugen. Durch den rasanten Anstieg der Leistungsfähigkeit der heutigen Grafikkarten gewann ein spezieller Bereich des Machine Learning, das **Deep Learning**, an Popularität.
Die im Deep Learning verwendeten neuronalen Netze, versuchen anhand von vorherigen Beispielen eine Aussage über zukünftige Daten zu treffen. Dabei werden bestimmte Merkmale aus den Beispielen extrahiert. Ein neuronales Netz kann zum Beispiel angewendet werden, um geschriebene Zahlen zu erkennen. Es existieren auch Netzwerke, die nicht nur bestimmte Bilder klassifizieren können, sondern sind in der Lage auch eigene Bilder zu zeichnen. Dabei dienen ihnen zuvor gezeigte Bilder als Referenz.
In den folgenden Kapiteln dieser Arbeit werden wir neuronale Netze benutzen, um MRT Scans des menschlichen Gehirns zu analysieren. Wir werden uns hierbei auf die Vorhersage von Geschlecht, Alter und Händigkeit beschränken. Unser Ziel ist es neuronale Netzwerke zu erstellen, welche das Geschlecht, das Alter oder die Händigkeit einer Person nur anhand des MRT Scans ihres Gehirns vorhersagen können. Außerdem werden wir Gemeinsamkeiten zwischen Geschlecht, Alter und Händigkeit mittels **Lerntransfer** herausarbeiten.

[1] `https://www.techemergence.com/machine-learning-in-pharma-medicine/`

2 Stand der Forschung

Auf dem Gebiet der Gehirnanalyse mittels Machine Learning existieren unzählige Publikationen. Ein Unterschied zu unserer Untersuchung besteht darin, dass bei den bestehenden Arbeiten ausschließlich MRT Segmentierung durchgeführt wurde. So wurde durch Havaei et al.[1] eine Methode vorgestellt, um automatisch Tumore aus einem MRT Scan zu segmentieren. Hierbei wurden mit Hilfe von tiefen neuronalen Netzen und 2D Konvolutionen Glioblastome[2], Tumore, welche überall im Gehirn auftreten können und in der Größe stark variieren, mit erstaunlicher Präzision segmentiert. Auch andere Publikationen beschäftigen sich mit demselben Problem. S. Pereira[2] erzielte mit 2D Konvolutionen ähnliche Ergebnisse. Brosch et al.[3] hat mittels Deep Learning versucht, Multiple Sclerosis (MS) Lesionen in MRT Scans zu erkennen. Diese Methode schlägt alle bis zu dieser Zeit bekannten Verfahren der MS Segmentierung (EMS, LST-LPA, LST-LGA, Lesion-TOADS, und SLS). Um solch ein Resultat zu erzielen, wurde ein tiefer Encoder mit dreidimensionalen Konvolutionen verwendet. So lernt der Encoder durch 3D Konvolutionsfilter High Level Merkmale, während der Decoder daraus segmentierte MRT Scans erzeugt. Eine in diesem Jahr erschienene Publikation von Viktor Wegmayr et al.[4] beschäftigt sich mit der Klassifikation von Alzheimer. Es wurde ein MRT Voxel, entweder als „Healty (HC)“, „Mild Cognitive Impairment (MCI)“ oder als „Alzheimer's Disease (AD)“ eingestuft. Als neuronales Netz wurde eine Standardarchitektur mit 3D Konvolutionen benutzt. Dieses Netz wurde anschließend mit mehr als 20.000 Scans trainiert, um die Klassifikation vorzunehmen.

3 Medizinische Ergebnisse

Um die Ergebnisse unserer Arbeit bewerten zu können, wird ein Vergleich benötigt. Die MRT Scans können auch leicht von trainierten und erfahrenen Ärtzen klassifiziert werden. So lässt sich für einen Arzt das Geschlecht eines Menschen anhand der Kopfform bzw. der Gehirnrinde eines MRT Scans erkennen[5]. Das Alter kann ebenfalls relativ sicher mit einer Varianz von 5-10 Jahren geschätzt werden. Die Händigkeit nur anhand eines strukturellen T1-Scans abzuschätzen, gestaltet sich schwierig, wenn nicht sogar unmöglich. Es ist möglich, dass es subtile Unterschiede in der Dicke des motorischen Cortex gibt, welche einen Aufschluss über die Händigkeit der Person geben könnten.

4 Begriffserklärung

Neuronale Netze mit Konvolutionen wurden in der Vergangenheit oft verwendet, um bestimmte Merkmale aus Bildern zu extrahieren. Diese Architekturen werden heutzutage zunehmend komplexer, was die Anzahl an Verbindungen und Gewichten in dem Modell erhöht. Eine typische Netzwerkarchitektur besteht aus Konvolution, Pooling, Aktivierung und Klassifikation. Konvolutionsschichten extrahieren mittels sich verschiebenden Fenstern Merkmale aus dem Eingabebild. Die Poolingschicht komprimiert das von der

[2] https://en.wikipedia.org/wiki/Glioblastoma

Konvolution erhaltene Ergebnis, indem z.B. innerhalb einer bestimmten Nachbarschaft das Maximum gewählt wird. Um das Netz zu trainieren, wird versucht eine Kosten- oder Fehlerfunktion mittels des Gradientenabstieges zu minimieren. Im Folgenden werden zum besseren Verständnis nun häufig verwendete Begriffe kurz erklärt:

- **Genauigkeit**: Metrik für die Messung der Genauigkeit eines Modells. Beschreibt den Anteil an Datenpunkten, welche dem richtigen Label zugeordnet wurden. Wird üblicherweise auf den Testdaten ausgewertet, um das Modell zu evaluieren.
- **Aktivierungsfunktion**: Eine nichtlineare Funktion, welche typischerweise nach jeder Schicht eines neuronalen Netzes eingefügt wird.
- **Konvolution**: Wendet Filter auf Tensoren an. Wird oft für Bilder verwendet, um Merkmale herauszuarbeiten.
- **Epoche oder Durchlauf**: Ein Durchlauf mit allen Trainingsdaten.
- **Feature oder Merkmal**: Bestimmte Merkmale eines Datenpunktes, welche eventuell durch Konvolutionen herausgearbeitet wurden.
- **vollständig verbunden**: Verbindet alle vorherigen Knoten mit einer definierten Anzahl an Knoten.
- **Hyperparameter**: Frei konfigurierbare Parameter, welche typischerweise manuell eingestellt werden müssen. Diese Parameter werden normalerweise nicht durch Gradientenabstieg verändert.
- **Klassifikation**: Ein Verfahren, welches die Einordnung von Datenpunkten in bestimmte Klassen beschreibt.
- **Kostenfunktion**: Funktion, welche einen Skalar liefert. Dieser misst den Fehler des Netzes. Je geringer, desto besser.
- **Label**: Beschreibt die Klasse, in die der Datenpunkt eingeordnet wird (Geschlecht/Alter/Händigkeit).
- **Optimierer**: Es existieren viele verschiedene Optimierer. Alle beschreiben einen jeweils anderen Algorithmus, um neuronale Netze mittels Gradientenabstieg zu optimieren.

5 Datensatz

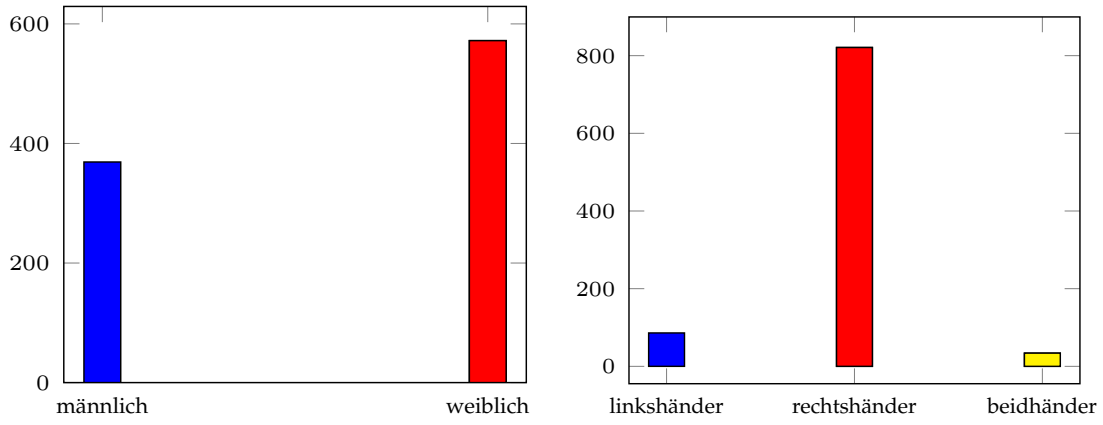
Eine Einrichtung, die sich unter anderem auf MRT Scans spezialisiert hat, ist das *Enhanced Nathan Kline Institute*. Dieses bietet das sogenannte Rockland Sample[3] an. Die Institution stellt Daten zu genetischen Informationen, physiologische und psychologische Bewertungen und Neuroimaging zur Verfügung. Die Neuroimaging Daten enthalten neben den strukturellen T1 MRT Scans eines Gehirns auch physiologische Daten, wie z.B.

[3] `http://fcon_1000.projects.nitrc.org/indi/enhanced/index.html`

Daten der Atmungsorgane. Des Weiteren sind die Daten anonymisiert. Es sind also nur wenige phänotypische Daten (Geschlecht, Alter und Händigkeit) für jede Person verfügbar. Diese phänotypischen Daten liegen in einer CSV Tabelle vor, wobei jede Person eine eindeutige Identifikationsnummer besitzt. Schaut man sich einen Scan mit dem Dateinamen *sub-A00008326_ses-DS2_T1w.nii.gz* an, so ist *A00008326* die Identifikationsnummer, nach der in der Tabelle gesucht werden muss.

5.1 Verteilung der Daten

Insgesamt enthält der Datensatz ungefähr 1200 Neuroimaging Scans, obwohl nur 941 Personen an der Studie teilgenommen haben. Somit existieren von einigen Personen zwei oder sogar mehr Scans. Der Datensatz fällt mit einer Gesamtgröße von neun Gigabyte relativ klein aus.



(a) Männlich/Weiblich Anzahl

(b) Händigkeit Anzahl

Abbildung 1: Verteilung von Geschlecht und Händigkeit

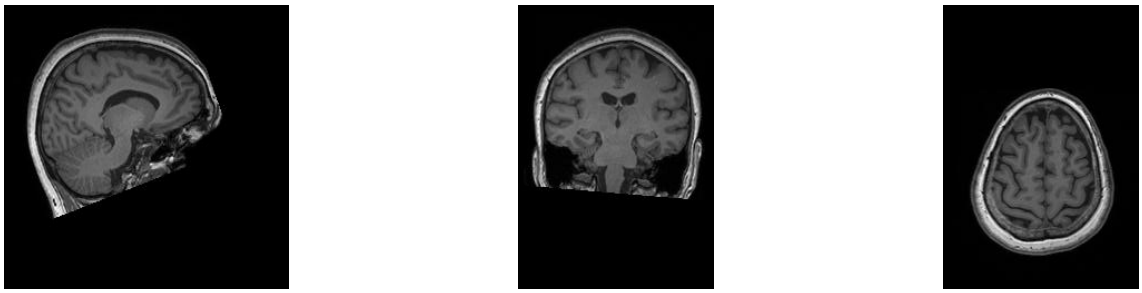
Abbildung 1 zeigt die Anzahl der Personen in dem Datensatz, welche männlich oder weiblich bzw. welche Links-, Rechts- oder Beidhänder sind. Es fällt auf, dass ungefähr 90% der Personen Rechtshänder sind. Auch wenn diese Information erst einmal unwichtig erscheint, spielt Datenimbalance bei dem Trainieren eines neuronalen Netzes eine sehr große Rolle. In diesem Fall würde das neuronale Netz schnell jede Person als Rechtshänder klassifizieren, da die meisten der Personen in der Studie Rechtshänder sind. Ein Verfahren, um das Problem des unausgeglichenen Datensatzes zu lösen, wird im späteren Verlauf der Arbeit vorgestellt.

5.2 Dateiformat

Die MRT Scans des *Enhanced Nathan Kline Institute* liegen in dem standardisierten *NIFTI*[4] Format vor. Jeder Scan hat drei Dimensionen, wobei die Dimensionen jeweils die X-, Y- und Z-Achse darstellen. Jedoch ist anzumerken, dass die Reihenfolge der Achsen nicht

[4]Neuroimaging Informatics Technology Initiative

immer identisch ist.
Dies könnte bei der Vorverarbeitung der Daten später eventuell zu Problemen führen. Neben den eigentlichen MRT Scans enthält jeder Scan auch Meta-Informationen darüber, inwiefern Punkte des MRT Scans auf eine Position des Scanners[5] abgebildet werden können[6].



(a) medial-lateral (b) rostral-caudal (c) dorsal-ventral

Abbildung 2: Alle Achsen eines MRT Scans

Die eigentlichen NIFTI Dateien lassen sich zum Beispiel mit der Python Bibliothek *nibabel*[7] einlesen. Wie in Abbildung 2 zu sehen ist, kann dann durch die Bibliothek jeder Frame des MRT Scans aus dem entstandenen Tensor extrahiert werden.

6 Erstellen eines neuronalen Netzes

6.1 Datenvorverarbeitung

Um ein neuronales Netz zu trainieren, müssen die Trainingsdaten standardisiert sein. Bewegt sich ein Patient während eines MRT Scans, so kann es vorkommen, dass die Bilder versetzt zueinander sind. In diesem Fall müssen die Bilder erst auf die gleiche Position verschoben werden (Registration). Auch ist es oft der Fall, dass die Intensität eines Scans sich innerhalb der verschiedenen Bilder ändert, also nicht konstant ist. Die N4ITK Bias Field Correction wird häufig benutzt, um dieses Problem zu lösen. Jedoch nehmen diese beiden Methoden der Normalisierung, gerade die Registration, bis zu eine Stunde pro Scan an Zeit in Anspruch. Aus diesem Grund wird auf diese beiden Methoden verzichtet. Wir stellen jedoch trotzdem sicher, dass die Bilder der MRT Scans alle die gleiche Auflösung besitzen. Die erste Möglichkeit dies zu erreichen, bestünde darin, die Größe der Bilder ohne Rücksicht auf die Seitenverhältnisse zu verändern. Dies hätte eventuell zur Folge, dass wichtige Informationen verloren gehen. Eine weitere Möglichkeit wäre es, Teile aus dem Bild herauszuschneiden, welche der gewünschten Größe entsprechen. Hierbei unterscheidet man zwischen *Randomcrop* und *Centercrop*, wobei Randomcrop eine zufällige Stelle aus dem Bild und Centercrop die Mitte des Bildes ausschneidet. Dadurch würden die Seitenverhältnisse beibehalten. Ein anderer Ansatz bestünde darin je-

[5] affine Transformationen
[6] `https://brainder.org/2012/09/23/the-nifti-file-format/`
[7] `http://nipy.org/nibabel/`

des Bild mit schwarzen Pixeln so zu erweitern, dass alle Bilder dieselbe Größe haben (*Padding*). Dieser Ansatz ist nicht zu empfehlen, da die MRT Scans, wie in Abbildung 2a zu sehen, schon anfangs verhältnismäßig viele schwarze Pixel enthalten.
Hier ist also *Random-* oder *Centercrop* die bessere Methode der Vorverarbeitung.

Wurden alle Scans auf die gleiche Größe ausgeschnitten, so ist außerdem sicherzustellen, dass keine Bilder einer Person sowohl in den Trainings- als auch in den Testdaten auftauchen. Dies würde ansonsten dazu führen, dass die Ergebnisse verfälscht werden. Auch werden die Bilder alle an den Wertebereich von 0 bis 1 angepasst, um erhebliche Varianzen in den Daten zu vermeiden.

```
from sklear.preprocessing import MinMaxScaler
img = MinMaxScaler().fit_transform(img)
```

Als Letztes ist darauf zu achten, den Mittelwert der Trainingsdaten auf Null zu setzen.

```
X -= np.mean(X, axis=0)
```

Dies sorgt dafür, dass das neuronale Netz schneller konvergiert, da es keine Rücksicht auf die Skalierung der Daten nehmen muss. Die Daten werden in Trainings- und Testdaten aufgeteilt (80% Trainingsdaten, 20% Testdaten).

Die Vorverarbeitung besteht aus folgenden Schritten:

1. Lade MRT Scan
2. Lade dazugehöriges Label (Geschlecht/Alter/Händigkeit)
3. Lade von jeder Achse N zufällige Bilder, optional mit Random- oder Centercrop
4. Passe den Wertebereich der Bilder an
5. Wiederhole dies, bis alle Scans abgearbeitet sind
6. Setze den Mittelwert der Daten auf null

6.2 Erstellen von Referenzergebnissen

Bevor nun mit dem Datensatz und verschiedenen neuronalen Netzen experimentiert wird, sollte ein Referenzergebnis vorhanden sein. Wir haben uns für zwei verschiedene Arten entschieden. Das erste Vergleichsergebnis ist das naive Raten. Bei einem Klassifizierungsproblem mit zwei Klassen sollte die Genauigkeit größer als 50% sein, da sonst kein Lernen stattfand. Ein anderer primitiver Algorithmus, welcher auch oft als Referenz dient, ist der K-nächste-Nachbar (KNN) Algorithmus. Ist das von uns benutzte neuronale Netzwerk nicht besser als der KNN Algorithmus, so sollten die Architektur und/oder die Hyperparameter optimiert werden. Tabelle 1 zeigt die Ergebnisse des K-nächste-Nachbar Algorithmus. Es wurde jeweils das K gewählt, welches die größte Genauigkeit erzielt. K durfte minimal den Wert drei und maximal den Wert 20 annehmen. Eine zu große Anzahl an Nachbarn würde eventuell wichtige Informationen in der Verteilung übersehen. Wird K jedoch zu klein gewählt, so entsteht das Risiko der Überanpassung.

Art	Label	Genauigkeit	Auswahl	K
Klassifikation	Geschlecht	68%	-	14
Klassifikation	Händigkeit	45,96%	Unterauswahl	10
Klassifikation	Händigkeit	57,61%	Überauswahl	5
Klassifikation	Händigkeit	87%	-	9
Regression	Alter ($M = 2.5$)	11%	-	5

Tabelle 1: Ergebnisse von KNN

6.3 Netzwerkarchitektur

Auch die Architektur des neuronalen Netzes ist neben der Hyperparameteroptimierung ein ausschlaggebender Punkt für die Leistung des Netzes. Bevor für ein Problem, wie z.B. die Bilderklassifikation direkt ein State-of-the-Art Netzwerk wie VGG[6] benutzt wird, sollte versucht werden, ein möglichst kleines und einfaches Netz zu finden, welches das Problem effizient lösen kann. Als Basis für die Analyse der MRT Scans wird zunächst ein Netz benutzt, welches Konvolutionen verwendet, da diese Technik bis jetzt gute Ergebnisse bei der Bilderklassifikation erzielt hat. Die Eingabeschicht hat nur einen Kanal, da wir Schwarzweißbilder mit einer Auflösung von 170x170 betrachten. Diese Auflösung wurde gewählt, damit mit jeder Achse des Voxels ein Centercrop durchgeführt werden kann, ohne außerhalb des Bildes zu operieren.

Die erste Schicht besteht aus drei hintereinandergeschalteten Konvolutionen mit einer Filtergröße von *4x4*, einer Filteranzahl von *32*, einer Schrittweite von *1* und dem validen Füllungsmodus. Es wurde gerade dieser Füllungsmodus gewählt, da im gesamten Netz nur eine Pooling Operation stattfindet und das Bild somit durch die Konvolutionen nicht die ursprüngliche Größe beibehalten muss. Die drei Konvolutionen ermöglichen es, die für Problemstellung wichtigen Informationen aus dem Bild zu extrahieren. Danach folgt eine Max-Pooling Schicht mit einer ungewöhnlich großen Filtergröße von *8x8*. Die Aufgabe der Max-Pooling Schicht ist es, die von den Konvolutionen extrahierten Informationen zu komprimieren, ohne die Anzahl an Parametern im Netzwerk zu erhöhen. Darauf folgen zwei vollständig verbundene Schichten mit jeweils 2048 Knoten. Jede dieser Schichten hat als Aktivierungfunktion ReLU (rectified linear unit) und als Gewichtsinitialisierung Glorot Uniform bzw. Xavier. ReLU hilft das Problem der verschwindenden Gradienten zu beheben. Die letzte Schicht hat für die Klassifikation von Geschlecht und Händigkeit zwei Knoten, jeweils einen Knoten für eine Klasse. Als Aktivierungsfunktion der letzten Schicht wird **softmax** verwendet, um eine Wahrscheinlichkeitsverteilung über die jeweiligen Klassen zu erhalten. **Kreuzentropie** ist die von uns gewählte Kostenfunktion, da diese in der Vergangenheit gute Ergebnisse bei Klassifikationsproblemen erzielt hat.

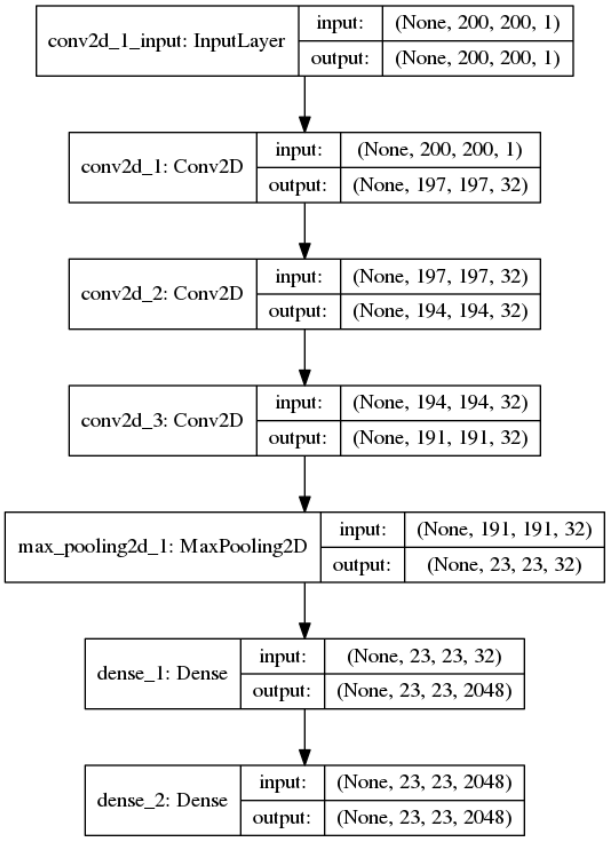


Abbildung 3: Graph des neuronalen Netzes

6.4 Hard- und Software

Um die NIFTI Dateien zu lesen und Scans zu extrahieren, wurde die Python Bibliothek *nibabel*[8] eingesetzt. Alle nachfolgenden Klassifikationen und Regressionen wurden mit dem Deep Learning Framework Tensorflow[9] und der Abstraktionsschicht Keras durchgeführt. Um das Training zu beschleunigen, wurde die Grafikkarte MSI GeForce GTX 770 verwendet. Es wurden gleichzeitig zehn Trainingsbeispiele durch das Netzwerk geschickt, damit der Arbeitsspeicher der Grafikkarte nicht überfüllt wird.

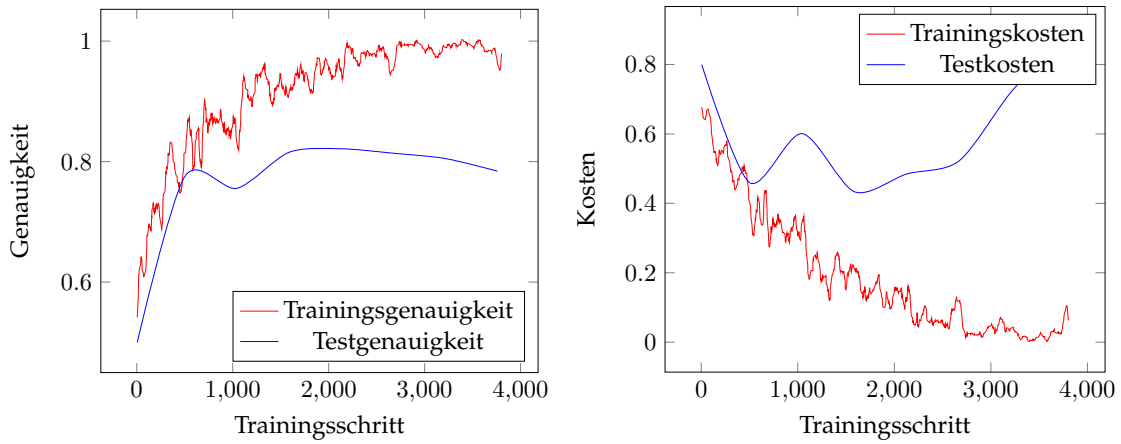
7 Geschlechtsklassifikation

7.1 Erste Ergebnisse

Um das Netz zu trainieren, wurden insgesamt 15 Bilder von jeder Person extrahiert. Für jede Person wurden also aus jeder Achse jeweils 5 Bilder zufällig ausgewählt. Anschließend wurde aus jedem dieser Bilder ein 170x170 großes Bild mit Hilfe von Centercrop

[8] `http://nipy.org/nibabel/`
[9] Tensorflow Version 1.5

ausgeschnitten. Insgesamt stehen somit $14.115 = 941 * 15$ Bilder zur Verfügung. Als Label wurde das jeweilige Geschlecht der Person in Form der 1-aus-n-Kodierung verwendet. Nachdem das Netzwerk aus Abbildung 3 für 20 Durchläufe mit einem Momentum basierten Optimierer[10] und einer Lernrate von $0,00005$ trainiert wurde, hatte es eine Genauigkeit von 80,76% und einen F1 Maß von 83,55%. Im Vergleich zu den Referenzergebnissen wurde hier eine deutliche Verbesserung erzielt.



(a) Trainingsgenauigkeit

(b) Trainingskosten

Abbildung 4: Erste Ergebnisse der Geschlechtsklassifikation

In Abbildung 4a lässt sich erkennen, dass das Netz, sowohl mit den Trainings- als auch mit den Testdaten, schon relativ früh konvergiert. Gleiches lässt sich in Abbildung 4b bei dem Fehler des Netzes bezüglich der Trainings und Testdaten beobachten. Die Trainingsgenauigkeit dieses Netzes ist mit fast 100% (d.h 1.0 in Abb 4a) sehr hoch. Die Differenz von Test- und Trainingsgenauigkeit beträgt etwa 20%, was auf Überanpassung hindeutet. Das bedeutet, dass das Netzwerk die Trainingsdaten auswendig lernt. Dem Modell fällt es dadurch sehr schwer durch das Gelernte zu generalisieren. Mit der von Tensorflow voreingestellten Lernrate von 10^{-3} konvergiert das Netz schon nach zwei Epochen und klassifiziert jede Person als weiblich. Dies zeigt sich an einer gleichbleibenden Genauigkeit von nur 60%. Dieser Prozentsatz entspricht der prozentualen Anzahl weiblicher Personen 1 in dem Datensatz.

7.2 Beheben von Überanpassung

Zu diesem Zeitpunkt existiert in dem vorgestellten Netz keine Art Schutz gegen Überanpassung. Dieses Phänomen ist ein großes Problem bei neuronalen Netzen. Aus diesem Grund wurde nun, nach den in Abbildung 3 gezeigten *dense_1* und *dense_2* Schichten, jeweils eine Dropoutschicht mit einer Ausfallwahrscheinlichkeit von 50% hinzugefügt. Dropout führt, wie der Name schon impliziert, dazu, dass Knoten in dem neuronalen Netz mit einer bestimmten Wahrscheinlichkeit ausgeschaltet, also auf null, gesetzt werden. Dies soll dazu führen, dass das Netz nicht einfach nur die Trainingsdaten auswendig

[10] Adam Optimizer [7]

lernt. Hier ist beim Testen des Netzes darauf zu achten, die Ausfallwahrscheinlichkeit auf 0% zu setzen, da die Ergebnisse bei gleichen Eingaben sonst stark variieren könnten. Die Zufälligkeit der Resultate würde durch das zufällige Ausschalten von Knoten entstehen.

Die Ergebnisse des Modells aus Abbildung 3 *(+ Dropout)* bleiben dabei fast identisch. Jedoch können wir uns nun sicherer sein, dass das Netzwerk unsere Trainingsdaten nicht nur auswendig gelernt hat und dass somit keine Überanpassung stattfindet. Eine andere Technik, welche statt Dropout verwendet werden kann, ist die Batchnormalisierung[8]. Jede Schicht eines neuronalen Netzwerkes erhält dessen Daten von der vorherigen Schicht. Demzufolge wäre es eine gute Idee diese Daten zu normalisieren. Eine Batchnormalisierung (BN) normalisiert also in jeder Schicht zuerst die Daten und sendet diese dann erst weiter zu der folgenden Schicht. Außerdem ermöglicht die BN das Nutzen von höheren Lernraten, was wiederrum zur Folge hat, dass die Netzwerke schneller trainiert werden können. Es sollte darauf geachtet werden, die Batchnormalisierung *vor* der Aktivierungsfunktion durchzuführen. Auch sollte ein Bias vermieden werden, da so Rechenzeit gespart wird. Des Weiteren besitzt die BN selbst Parameter (γ und β), um die sonst vom Bias durchgeführte Normalisierung rückgängig zu machen.

Als Parameter der Batchnormalisierung wurde ein Momentum von $0,99$ und $\epsilon = 0,001$ gewählt. Außerdem wurden β und der gleitende Mittelwert mit Nullen und γ mit Einsen initialisiert. Fügt man nun die BN bei jeder Konvolution und bei jeder vollständig verbundenen Schicht ein, so ändern sich die Ergebnisse nicht signifikant. Auch eine L2 Regularisierung, welche bei jeder Schicht dessen Parameter "bestraft", hat keinen großen Einfluss auf das Netz. Bei der L2 Regularisierung wurde $\lambda = 0,01$ gewählt. Tabelle 2 zeigt eine Übersicht der Ergebnisse.

Methode	Accuracy	Lernrate	Überanpassung
Dropout	83,26%	$5 * 10^{-5}$	$\times$
Batchnormalisierung	80%	10^{-4}	✓
L2 Reg. (Conv.)	80,44%	$5 * 10^{-5}$	✓
L2 Reg. (Conv.) & Dropout (Dense)	82,75%	$5 * 10^{-5}$	$\times$

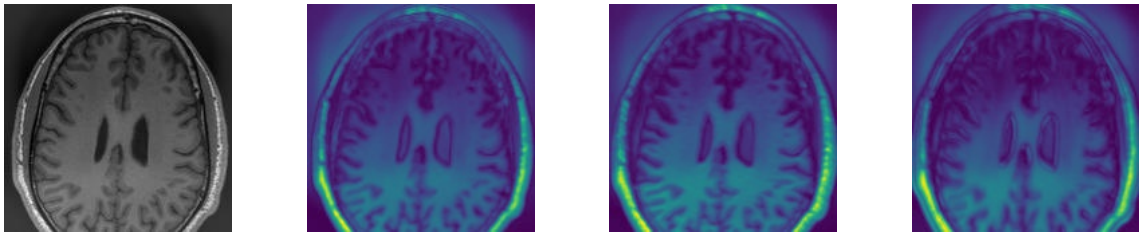
Tabelle 2: Ergebnisse der Geschlechtsklassifikation mit Dropout, BN & L2 Reg.

Wird die Batchnormalisierung eingesetzt, so beginnt das Netzwerk die Daten auswendig zu lernen, da die Trainingsgenauigkeit bei 98% liegt, während die Testgenauigkeit bei 80% bleibt. Jedoch konnte die Genauigkeit mit Dropout um etwa 3% gesteigert werden, während gleichzeitig die Stärke der Überanpassung verringert wurde, da die Trainingsgenauigkeit hier im Durchschnitt bei 93% liegt. Die L2 Regularisierung neigte mit einer Trainingsgenauigkeit von 99% und einer gleichbleibenden Testgenauigkeit von circa 80% auch zu Überanpassung. Es fällt auf, dass Dropout bei diesem Netzwerk hilft, Überanpassung zu vermeiden. Generell ist anzumerken, dass die in Tabelle 2 aufgeführten Methoden der Regularisierung und Normalisierung nur bei sehr tiefen neuronalen Netzen eine Auswirkung haben. In vielen Netzwerken, wie zum Beispiel Inception[9], wurde eine variable Lernrate benutzt, um das Netzwerk zu trainieren. Es wurde also ein Standard-Optimierer wie Stochastic Gradient Descent mit einer festen Startlernrate benutzt, welche dann nach einer bestimmten Anzahl von Durchläufen um einen Faktor

verringert wurde. Hier wird auf solch einen Ansatz verzichtet, da der von uns genutze Optimierer bereits intern die Lernrate verringert. Außerdem benötigt unser Modell schon zu Beginn eine sehr geringe Lernrate von $5 * 10^{-5}$, damit es nicht in einem lokalen Optimum feststeckt.
Eine Erhöhung von N auf beispielsweise $N = 10$ oder $N = len(AXIS)$, also die Auswahl aller Bilder, erhöht die Genauigkeit nicht.
$len(AXIS)$ bezeichnet hierbei die Auswahl aller Bilder aus einer Achse.

7.3 Konvolutionsfilter und Gradienten

Um die Geschlechtsklassifikation besser nachzuvollziehen, sind in Abbildung 5 jeweils die Konvolutionsfilter mit der höchsten Aktivierung dargestellt. Damit der Fokus der Bilder besser erkannt werden kann, wurde über die Konvolutionsergebnisse ein Wärmebild[11] gelegt. Dadurch können markante Werte schneller und leichter auf einen Blick erfasst werden. Das Wärmebild wurde mit Hilfe der Farbtabellen der Bibliothek *matplotlib*[12] erzeugt. Dabei werden die Graustufen durch Farben wie z.B. Dunkelblau (kalt) und Hellgrün (warm) ersetzt. Es lässt sich feststellen, dass die Filter der Konvolutionen keine großen sichtbaren Veränderungen vornehmen. Jedoch fällt auf, dass bei der tieferen Kon-

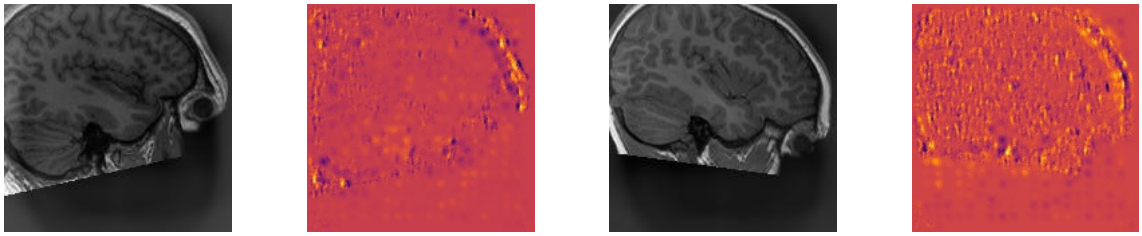


(a) Feldvergleich (b) Wärmebild Konv. 1 (c) Wärmebild Konv. 2 (d) Wärmebild Konv. 3

Abbildung 5: Konvolutionen mit der höchsten Aktivierung

volution (Abb. 5d) der Fokus anscheinend auf die Rinde des Kopfes gelegt wird, da dieser Bereich bei dem Wärmebild heller als der Rest des Gehirns aufleuchtet. Auch, wenn der Gradient der Kostenfunktion bezüglich X ausgerechnet wird, lässt sich erkennen, welche Gehirnareale für die Geschlechtsklassifikation am wichtigsten sind. In Abbildung 6 wird deutlich sichtbar, dass der Fokus bei Wärmebild 6b und 6d im Bereich des Frontallappens[13] des Gehirns liegt. Die Methode für die Erkennung des Geschlechts einer Person ist also identisch mit der Methode, die ein Arzt anwendet. Geschulte Ärzte können für die Geschlechtsklassifikation einer Person z.B. die Gehirnrinde verwenden[5].

[11] `https://de.wikipedia.org/wiki/Heatmap`
[12] `https://matplotlib.org/examples/color/colormaps_reference.html`
[13] http://arbeitsblaetter.stangl-taller.at/GEHIRN/GehirnFunktion.shtml



(a) Feldvergleich 1 (b) Wärmebild 1 (c) Feldvergleich 2 (d) Wärmebild 2

Abbildung 6: $\frac{\partial}{\partial X}K$ Wärmebild der Geschlechtsklassifikation mit $K =$ Kosten

8 Altersbestimmung

Das nächste Ziel ist es, das Alter einer Person anhand ihres MRT Scans abzuschätzen.

8.1 Regression statt Klassifikation

Die jüngste Person in unserem Datensatz ist 7 und die älteste Person ist 84 Jahre alt. Würde man hier also nun eine Klassifikation durchführen, bei der jedes Alter eine Klasse darstellt, würde man scheitern, da die Anzahl an Klassen einfach zu groß ist. Dies liegt daran, dass in unserem Datensatz nur 900 MRT Scans von jeweils einer Person vorhanden sind. Eine Lösung wäre es, die Personen in Altersgruppen einzuteilen. Zum Beispiel: 7-18 (Jugendlicher), 18-30 (junger Erwachsener), 30-65 (Erwachsener), 65-84 (Senior). Jedoch ermöglicht dies keine genaue Altersbestimmung. Der richtige Ansatz besteht darin, das Problem als ein Regressionsproblem zu betrachten. Das neuronale Netz benötigt nun als letzte Schicht nur noch einen Ausgangsknoten und **keine** Aktivierungsfunktion, da hier die ungefilterte Ausgabe bewertet wird. Außerdem ist darauf zu achten, die Kostenfunktion abzuändern, da hier, wie zuvor erwähnt, keine Klassifikation stattfindet. Die Kostenfunktion ist nun der **durchschnittlicher quadratischer Fehler** zum wahren Alter.

8.2 Netzwerkarchitektur

Da Regression eine Aufgabe ist, welche oft anspruchsvoller als eine einfache Klassifikation ist, ähnelt dieses Netzwerk sehr der VGG[6] Architektur. Das Netz ist in verschiedene Konvolutionsblöcke eingeteilt. Es existieren zwei verschiedene Blöcke, Typ 1 und Typ 2. Der erste Block (Typ 1) besteht aus zwei hintereinandergeschalteten Konvolutionen mit einer Filtergröße von drei. Als Füllmodus wurde „gleich“ ausgewählt. Das stellt sicher, dass das Konvolutionsresultat im Bezug auf die erste und zweite Dimension mit der Eingabe übereinstimmt. Beispiel: $170x170x1 \rightarrow Konvolution \rightarrow 170x170x32$. Es ist wichtig, dass die Konvolutionen in diesem Fall die Bildgröße beibehalten, da in dieser Architektur nur die Pooling Schichten für die Komprimierung von Informationen zuständig sind. Nach den Konvolutionsschichten folgt eine Max-Pooling Schicht (2x2), welche die von den Konvolutionen extrahierten Informationen zusammenfasst. Der zweite Block (Typ 2) ist bis auf einen Unterschied mit dem Block von Typ 1 identisch. Beim zweiten Block werden statt zwei nun vier Konvolutionen verwendet. Das Netzwerk besteht insgesamt

aus zwei Blöcken vom Typ 1 und einem Block vom Typ 2. Darauf folgen am Ende zwei vollständig verbundene Schichten mit einer Knotenanzahl von 512 und 256. Außerdem wird im gesamten Netz kein Bias verwendet, da nach jeder Konvolution vor der Aktivierung eine Batchnormalisierung stattfindet. Als Parameter der Batchnormalisierung wurde ein Momentum von $0,99$ und $\epsilon = 0,001$ gewählt. Außerdem wurden β und der gleitende Mittelwert mit Nullen und γ mit Einsen initialisiert. Der Block vom Typ 2 extrahiert nun also High-Level Merkmale, welche für die Bestimmung des Alters wichtig sind. Es wird der Momentum basierte Optimierer aus dem vorherigen Modell[14] verwendet, da dieser über eine fallende Lernrate verfügt. Die verwendete Lernrate ist $0,00005$.

8.3 Erste Ergebnisse

M sei die Varianz mit der ein geschätztes Alter noch als richtig gewertet wird. Die Genauigkeit beschreibt in diesem Fall die Anzahl der Personen, deren geschätztes Alter maximal um M von dem tatsächlichen Alter abweicht. y sei das echte Alter der Person und $\hat{y}$ die von dem Netzwerk getroffene Vorhersage.

$$\mathcal{L}_{y,\hat{y}} = \begin{cases} 1 & \text{wenn } \hat{y} \geq y - \mathcal{M} \text{ und } \hat{y} \leq y + \mathcal{M} \\ 0 & \text{sonst} \end{cases} \tag{1}$$

$\mathcal{L}$ berechnet, wie in Gleichung 1 gezeigt, für eine Vorhersage, ob diese in der Varianz $y \pm M$ liegt.

$$\mathcal{G} = \frac{\sum_{i=1}^{n} \mathcal{L}_{y_i,\hat{y_i}}}{n} \tag{2}$$

$\mathcal{G}$ (Genauigkeit) berechnet anschließend den Prozentsatz an Personen, deren Alter als richtig geschätzt wurde. Unsere Architektur erreichte nach 25 Epochen mit $M = 2,5$ eine Genauigkeit von 18%. Dieses Ergebnis hört sich im ersten Moment schlecht an. Fasst man dies als Klassifikationsproblem auf, dann könnte man die $78 = 84 - 7 + 1$ Klassen in Abschnitte mit Abstand M aufteilen. Daraus ergeben sich 15 Klassen. Die Wahrscheinlichkeit bei $M = 2,5$ durch Raten richtig zu liegen, liegt bei $1/15 = 0,0666 = 6,66\%$. Somit ist unser Modell dreimal genauer als das bloße Raten.

Modell	Genauigkeit	Durchschnittlicher Quadratischer Fehler	R^2 Wert	Lernrate
BN	18,13%	236,45	0,53	$5 * 10^{-5}$
Dropout	21,21%	156,49	0,69	$5 * 10^{-5}$

Tabelle 3: Ergebnisse für die Altersbestimmung mit Dropout & BN

Um herauszufinden, ob andere Verfahren, die Überanpassung vermeiden, zu einem besseren Ergebnis führen, wird die Batchnormalisierung entfernt. Danach wird in jeder Schicht ein Bias und eine Dropout Schicht mit einer Droprate von 50% nach jeder vollständig verbundenen Schicht eingefügt. Mit dieser Änderung ist die Genauigkeit mit $M = 2,5$ um ca. 3% auf 21,21% gestiegen. Tabelle 3 fasst unsere Resultate zusammen.

[14] Adam Optimizer [7]

8.4 Durchschnittlicher Quadratischer Fehler mit Varianz

Die in Tabelle 3 gezeigten Ergebnisse sind zwar nicht schlechter als die Ergebnisse des K-Nächsten-Nachbarn Algorithmus 1, aber eventuell lässt sich durch Anpassung der Kostenfunktion eine Verbesserung erzielen. Die jetzige Kostenfunktion (*durchschnittlicher quadratischer Fehler*) ergibt nur für die Vorhersagen null, die exakt mit dem echten Label übereinstimmen.

$$DQF = \frac{1}{n}\sum_{i=1}^{n}(y\_true_i - y\_pred_i)^2 \tag{3}$$

Wir messen unsere Genauigkeit jedoch, wie in Gleichung 1 und 2 dargestellt, mit einer Varianz M. Unter Umständen kann es für das neuronale Netz schwierig sein, eine Funktion zu finden, die unsere Trainingspunkte genau beschreibt. Wir schlagen nun eine Änderung der DQF Funktion vor, welche es uns ermöglicht, eine bestimmte Varianz M festzulegen. Die Funktion soll auch dann null ergeben, wenn sich Vorhersage und echtes Alter maximal um M unterscheiden. Dadurch wird das Netzwerk nicht mehr gezwungen, alle Trainingspunkte genau zu approximieren. Stattdessen wird ein gewisser Spielraum M festgelegt.
Eine Implementierung dieser Kostenfunktion könnte folgendermaßen aussehen:

```
# Tensorflow und Keras Framework Code
# K denotiert das Keras Backend, in unserem Fall Tensorflow.
def loss_helper(y, y_real):
    '''
    Hilfsfunktion welche fuer Vorhersage und echtes Alter
    die Differenz berechnet.
    Ist diese kleiner oder gleich M so gilt nun Vorhersage = echtes Alter
    '''
    MARGIN = float(TOLERANCE / 2) # M
    diff = K.abs(y - y_real) # Differenz von Label und Vorhersage
    MARGIN = tf.convert_to_tensor(MARGIN, dtype=tf.float32)
    MARGIN = tf.reshape(MARGIN, shape=[])
    diff = tf.reshape(diff, shape=[])
    gap = K.less_equal(diff, MARGIN) # Differenz <= M ?
    return tf.cond(
        gap,
        true_fn=lambda: (y_real, y_real), # Falls ja setze Vorhersage = echtes
                                            Alter
        false_fn=lambda: (y, y) # sonst nichts
    )

def get_margin_mse():
    def margin_mse(y_true, y_pred):
        y_true = tf.reshape(y_true, shape=(-1, 1))
        (y_pred_new, _) = K.map_fn(lambda x: loss_helper(x[0], x[1]) , (y_pred,
                                                y_true))
        # Am Ende wird auf dem neuen Vektor normale DQF ausgefuehrt.
        return K.mean(K.square(y_pred_new - y_true), axis=-1)
    return margin_mse
```

Um diese neue Kostenfunktion zu realisieren, wird für jeden Eintrag in unserer Vorhersage die Differenz zwischen dieser und dem echten Wert genommen. Ist der absolute Betrag der Differenz nun kleiner als die von uns festgelegte Varianz, so gilt er als richtige

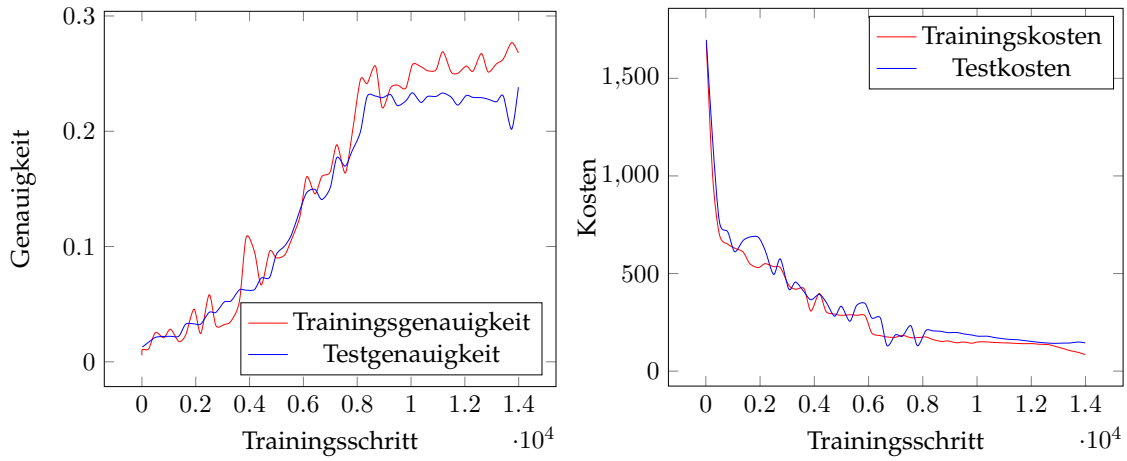
Vorhersage. Bei Verwendung unserer neuen Kostenfunktion stellt sich heraus, dass unser neuronales Netz nach der gleichen Anzahl an Trainingsschritten keine sichtbaren Verbesserungen zeigt, sondern sich sogar verschlechtert. Die Genauigkeit nach 25 Durchläufen beträgt ca. 16,5%. Obwohl die Kostenfunktion in der Theorie besser zu unseren Anforderungen passt, verbessert sich das Netzwerk nicht. Oft müssen bei Änderung der Kostenfunktion die Hyperparameter des Netzes neu eingestellt werden. Es wird also nun eine Lernrate von $0,001$ verwendet. Diese wird jedoch alle 15 Epochen mittels *exponentiellem Verfall*[15], wie in Gleichung 4 dargestellt, verringert.

$$\text{lernrate} = \text{lernrate} * \text{fallrate}^{\frac{\text{durchlauf}}{\text{n\_durchlauf}}} \tag{4}$$

Wir haben $\text{fallrate} = 0,16$ und $\text{n\_durchlauf} = 15$ gewählt. Der Parameter $durchlauf$ ist eine Variable mit Initialwert null, welche nach jedem Durchlauf um eins erhöht wird.

M	Genauigkeit	Durchschnittlicher Quadratischer Fehler	R^2 Wert
2,5	23,82%	144,44	0,66
4	33,41%	142,69	0,6632
5	40,44%	140,97	0,6696

Tabelle 4: Ergebnisse für die Altersbestimmung mittels DQF mit Varianz



(a) Trainingsgenauigkeit

(b) Trainingskosten

Abbildung 7: Altersregression Genauigkeit und Kosten für DQF mit Varianz

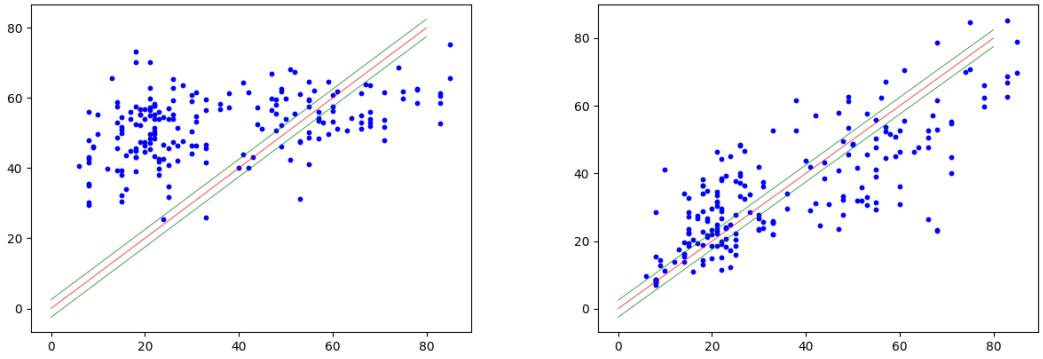
Das ermöglicht ein zu Anfang schnelles Lernen und stellt trotzdem sicher, dass das Netzwerk nicht in einem lokalen Optimum steckenbleibt. Auch wird zu einem anderen Optimierer mit adaptiver Lernrate[16] gewechselt. Nach 25 Epochen lässt sich erkennen, dass die neue Kostenfunktion unser Ergebnis erheblich verbessert. Die Genauigkeit stieg um fast 7% im Vergleich zu dem vorherigen Netzwerk.

[15] https://en.wikipedia.org/wiki/Exponential_decay

[16] RMSprop http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

An Tabelle 4 lässt sich beobachten, dass sich die Genauigkeit, wenn die Varianz auf 10 Jahre, also $M = 5$, angehoben wird, fast proportional zu M verhält. Hierbei ist anzumerken, dass das Netzwerk mit $M = 2,5$ trainiert wurde. Zur Evaluierung des Netzwerkes wurde M auch auf $M = 4$ und $M = 5$ verändert.

8.5 Analyse der Vorhersagen



(a) Durchlauf 1 (b) Durchlauf 25

Abbildung 8: Testdaten Altersvorhersage

Abbildung 8 zeigt die Vorhersagen von 200 Testpunkten. Die X-Achse beschreibt das echte Alter und die Y-Achse das vorhergesagte Alter der Person. Die rote Linie ist die Idealgerade. Sie bietet eine Orientierung für die idealen Vorhersagen ($Y_{echt} = Y_{vorhersage}$). Die beiden grünen Linien stellen jeweils die Varianz $+M$ und $-M$ dar. In der ersten Epoche (8a) sind die Punkte noch sehr weit von der Idealgerade und der Varianz M entfernt. In Durchlauf 25 (8b) hingegen sind die Punkte deutlich geordneter. Wird Abbildung 8b genauer betrachtet, so lassen sich ein paar Rückschlüsse auf die Schwächen des Netzes schließen. Als erstes fällt auf, dass Personen bis etwa 30 Jahre öfter als ältere Personen richtig eingestuft werden. Die jungen Leute, welche falsch eingestuft werden, werden oft als viel zu alt eingeschätzt. So wird z.B. eine Person, die 20 Jahre alt ist, auf 40 Jahre geschätzt. Das genaue Gegenteil tritt bei Personen auf, die älter als 40 Jahre, sind. Hier wird z.B. eine 60-jährige Person auf 40 Jahre geschätzt. Dies lässt sich daran erkennen, dass die entsprechenden Punkte unterhalb der negativen Grenze von M liegen.

9 Klassifizierung der Händigkeit

Das nächste zu lösende Problem besteht darin, die Händigkeit einer Person anhand ihres MRT Scans zu erkennen. Wir klassifizieren hier nur zwischen Links- und Rechtshänder. Statistisch gesehen sind etwa 10% aller Personen Linkshänder[17]. Wie in Abbildung

[17] `http://www.rightleftrightwrong.com/statistics.html`

1 schon dargestellt ist, beträgt auch der Prozentsatz an Linkshändern in unserem Datensatz ungefähr 10%. Auch wenn neuronale Netze oft höchstkomplexe Probleme mittels geschickter Matritzenmultiplikation lösen können, sind die Qualität und Quantität der Daten trotzdem ein wichtiger Punkt. Wie schon erwähnt, ist unser Datensatz im Bezug auf die Händigkeit enorm unbalanciert[18]. Ein neuronales Netz, welches diese unbalancierten Trainingsdaten präsentiert bekommt, würde relativ schnell mit einer Testgenauigkeit von 90% konvergieren. Das Problem an dieser sehr hohen, fast schon unglaubwürdig hohen Genauigkeit ist, dass hier jede Person als Rechtshänder klassifiziert wird. Dies lässt sich daran erkennen, dass diese 90% dem Anteil an Rechtshändern in unserem Datensatz entsprechen.

9.1 Problem des unbalancierten Datensatzes

9.1.1 Unter-/ Überauswahl

In der Datenanalyse wird bei solchen Problemen oft die **Unter-/ Überauswahl** verwendet. Generell passen Unter- und Überauswahl die Daten so an, dass der Anteil jeder Klasse im Datensatz äquivalent ist. Die Unterauswahl betrachtet dabei die Minderheitsklasse. Es werden dann so viele Elemente aus den anderen Klassen entfernt, dass die Anzahl der Elemente in den Klassen der Anzahl der Elemente in der Minderheitsklasse entspricht. Die Überauswahl hingegen erhöht die Anzahl der Elemente in der Minderheitsklasse, solange bis diese an die anderen Klassen angeglichen ist. Hierbei könnte man z.B. einfach Elemente in der Minderheitsklasse verdoppeln. Dies ist jedoch ein naiver Ansatz. Normalerweise werden bei der Überauswahl mittels der KNN Methode neue Daten synthetisiert. Es existieren viele Methoden, um Daten für die Überauswahl zu erzeugen. Eine beliebte Methode ist die Synthetic Minority Over-sampling Technik[19]. SMOTE benutzt die KNN Methode, um neue Daten herzustellen. Nehmen wir an es wird ein Datenpunkt aus der Minderheitsklasse ausgewählt, hier c genannt. Der Algorithmus betrachtet nun im Bezug auf die Merkmale die K nächsten Nachbarn des Punktes. Danach wird zufällig einer der K Nachbarn ausgewählt. n sei dieser Nachbar. Nun wird mit folgender Formel ein neuer Datenpunkt synthetisiert:

$$c + (c - n) * \text{rand}(0,1) \tag{5}$$

Dieser Algorithmus dupliziert also nicht nur vorhandene Datenpunkte, sondern erzeugt ähnliche neue Beispiele. Bei der Unterauswahl reicht es einfach nur überschüssige Datenpunkte aus allen Klassen außer der Minderheitsklasse zu entfernen. Jedoch tritt bei der Unterauswahl ein extremer Datenverlust auf. In unserem Fall würden, bei einer angestrebten Verteilung von 50/50, also nur noch $l * 2$ Datenpunkte existieren, wobei l die Anzahl der Linkshänder denotiert.

[18] Verhältnis 1:9

[19] Synthetic Minority Over-sampling Technik = SMOTE

9.1.2 Anpassen der Kostenfunktion

Das Problem eines unbalancierten Datensatzes lässt sich eventuell aber auch nur durch das Anpassen der Kostenfunktion lösen. Betrachtet wird die Kreuzentropie Kostenfunktion.

$$\text{kosten} = -\sum_{x} y \log \hat{y} \tag{6}$$

Hierbei entspricht x den Daten, y den echten Labels und $\hat{y}$ den Vorhersagen des Netzwerkes. Bei der in Gleichung 6 gezeigten Funktion wird jede Klasse identisch gewichtet. Bei einem unbalancierten Datensatz wäre es jedoch sinnvoll, den Fehler zu erhöhen, wenn eine falsche Vorhersage bei der Minderheitsklasse auftritt. Dazu werden die echten Labels zuvor mit einem Gewichtsvektor w multipliziert.

```
#Tensorflow Codebeispiel
#Gewichtvektor w
class_weights = tf.constant([[0.9, 0.1]])
# Gewicht mit echten Labels multiplizieren
weights = tf.reduce_sum(class_weights * _y, axis=1)
# Normale Kreuzentropiefunktion
unweighted_losses = tf.nn.softmax_cross_entropy_with_logits_v2(labels=_y,
                                        logits=brain_cnn)
# Kreuzentropiefunktion gewichten
weighted_losses = unweighted_losses * weights
loss = tf.reduce_mean(weighted_losses)
```

In diesem Fall wird ein Fehler bei einem Datenpunkt, bei dem das echte Label ein Linkshänder ist, neunmal höher gewertet, als wenn das Netzwerk bei einem Rechtshänder einen Fehler macht. An dem Gewichtsvektor lässt sich erkennen, dass oft die inverse Anzahl der Datenpunkte der jeweiligen Klasse verwendet wird. Auch w ist wieder ein Hyperparameter, welcher manuell auf die optimalen Werte angepasst werden muss. Alternativ lässt sich der Parameter w auch durch *sklearn.utils.class_weight.compute_class_weight* mit dem Parameter class_weight=„balanced“ ausrechnen.

9.2 Die richtige Evaluierungsmetrik

Neben der Ausbalancierung der Daten oder dem Anpassen der Kostenfunktion spielt auch die Art der Ergebnisevaluierung eine große Rolle. Die Standardmetrik Genauigkeit spiegelt im Falle eines unbalancierten Datensatzes nicht unbedingt immer das gewünschte Ergebnis wider. Würden unsere Ergebnisse im Bezug auf die Händigkeitsklassifizierung nur mit der Genauigkeitsmetrik bewertet werden, so entstünde keine Klarheit über die wahre Genauigkeit des Modells. Dies liegt daran, dass genau dieser entsprechende Prozentsatz der Personen auch Rechtshänder ist und somit, wie bereits vorher erwähnt, jede Person als ein solcher klassifiziert wird. Eine Lösung für dieses Problem ist die Einführung neuer Metriken, welche ein besseres Verständnis für die Ergebnisse eines Modells liefern. Diese Metriken sind folgende: **Relevanz**, **Sensitivität** und **F1 Maß**[20]. Um diese Metriken zu verstehen, muss erst einmal das Konzept von **richtig negativ**[21],

[20] https://en.wikipedia.org/wiki/Precision_and_recall
[21] richtig negativ = rn

falsch negativ[22], **richtig positiv**[23] und **falsch positiv**[24] aufgezeigt werden. Nehmen wir an, unser Klassifizierungsalgorithmus unterscheidet zwischen Rechtshändern (positiv) und Nicht-Rechtshändern (negativ):

- **Richtig negativ** bezeichnet die Datenpunkte, welche korrekterweise der negativen Klasse, also Linkshändern, zugeordnet wurden. Ein MRT Scan von einem Linkshänder wurde also korrekt als Nicht-Rechtshänder klassifiziert.
- **Falsch negativ** bezeichnet die Datenpunkte, welche fälschlicherweise der negativen Klasse, also Linkshändern, zugeordnet wurden. Ein MRT Scan von einem Rechtshänder wurde also als Nicht-Rechtshänder klassifiziert.
- **Richtig positiv** bezeichnet die Datenpunkte, welche korrekterweise der positiven Klasse, also Rechtshändern, zugeordnet wurden. Ein MRT Scan von einem Rechtshänder wurde also korrekt als Rechtshänder klassifiziert.
- **Falsch positiv** bezeichnet die Datenpunkte, welche fälschlicherweise der positiven Klasse, also Rechtshändern, zugeordnet wurden. Ein MRT Scan von einem Linkshänder wurde also als Rechtshänder klassifiziert.

Relevanz und Sensitivität werden nun wie folgt berechnet:

$$\text{relevanz} = \frac{rp}{rp + fp} \tag{7}$$

$$\text{sensitivitaet} = \frac{rp}{rp + fn} \tag{8}$$

Um Relevanz und Sensitivität intuitiv zu erläutern, werden nun folgende Extremfälle betrachtet:

- Große Sensitivität, kleine Relevanz: Für den Algorithmus sind viele Personen Rechtshänder, aber nur wenige davon sind wirklich Rechtshänder
- Kleine Sensitivität, große Relevanz: Der Algorithmus schätzt wenige Personen als Rechtshänder ein, aber fast alle davon sind auch wirklich Rechtshänder

Damit nicht immer separat auf Relevanz und Sensitivität geachtet werden muss, vereint das **F1 Maß** beide Werte in eine Metrik. Im Idealfall ist also eine hohe Relevanz und Sensitivität anzustreben, deren Varianz in einem akzeptablen Rahmen liegt.

9.3 Datenvorverarbeitung

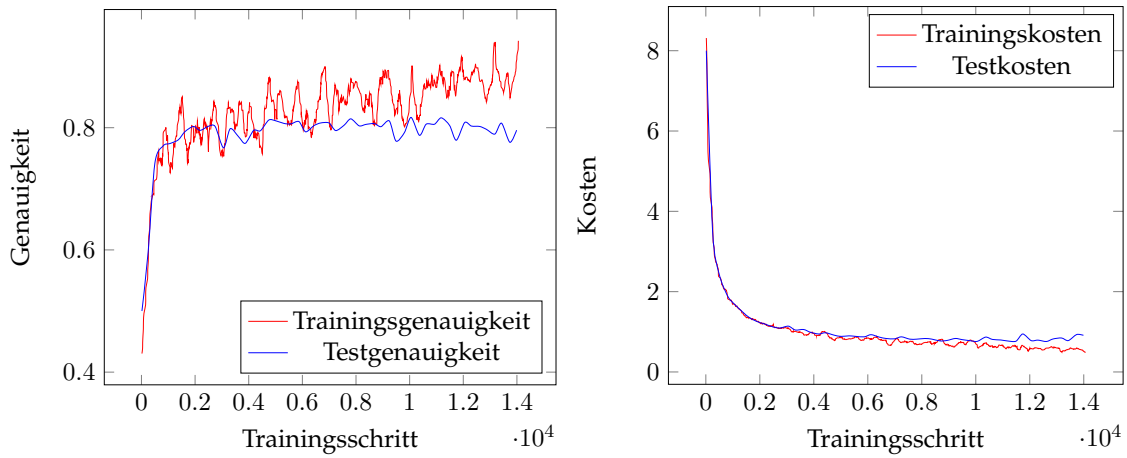
Um nun jegliche Irreführung des Netzwerkes auszuschließen, wurden die NIFTI Daten mithilfe von FSL so bearbeitet, dass alle Achsen und alle Orientierungen identisch sind, da die Orientierung der Person im Scanner für die Händigkeitsklassifizierung eine große

[22] falsch negativ = fn
[23] richtig positiv = rp
[24] falsch positiv = fp

Rolle spielt. Das liegt daran, dass im Gehirn die entgegengesetzte Gehirnhälfte für die Händigkeit ausschlaggebend ist. So ist bei einem Linkshänder die rechte Gehirnhälfte für die Klassifikation interessant. Deshalb ist hier darauf zu achten, dass die Orientierung bei allen MRT Scans übereinstimmt. Dies wurde mit der Funktion *fslreorient2std* aus dem FSL[25] Paket realisiert.

9.4 Ergebnisse

Zu Anfang wurde mit der Netzwerkarchitektur aus Kapitel 8 (ohne Batchnormalisierung) und einer Auswahlmethode (Unter-/ Überauswahl) versucht, das Netzwerk mit balancierten Daten jeweils für 25 Epochen zu trainieren. Bei $N = 5$, also 5 Bildern pro Achse, stehen bei Verwendung der Unterauswahl nur 1950 Trainings- und 570 Testdaten zur Verfügung. Wie sich herausstellt, hat dieses Netzwerk mit einer Trainingsgenauigkeit von fast 100% die Trainingsdaten nur auswendig gelernt und kann somit nicht generalisieren. Es fand also eine Überanpassung statt. Das resultiert in einer Testgenauigkeit von 48-50%. Als Fazit lässt sich daraus schließen, dass durch die Unterauswahl die Anzahl an Datenpunkten viel zu klein ist, als dass das Netz daraus etwas lernen kann.



(a) Trainingsgenauigkeit

(b) Trainingskosten

Abbildung 9: Händigkeit Überauswahl mit L2 Regularisierung

Wird der zuvor beschriebe SMOTE Algorithmus zum synthetischen Erzeugen neuer Linkshänder, also eine Überauswahl, verwendet, so ändern sich die Ergebnisse dramatisch. Da die Trainings- und Testdaten durch die Überauswahl gleich verteilt sind, kann die Genauigkeitsmetrik auch als Evaluationsergebnis herangezogen werden. Im Gegensatz zum vorherigen Ergebnis erreichte das Netzwerk mit einem Datensatz, der durch Überauswahl entstanden ist, eine Genauigkeit von 70%. Jedoch leidet dieses Netzwerk an Überanpassung. Um dieses Problem zu beheben, wird nun die L2 Regularisierung mit $\lambda = 0,01$ in das Netzwerk integriert. Die Genauigkeit verbessert sich um etwa 10% und die Trainingsgenauigkeit ist, wie sich in Abbildung 9 erkennen lässt, mit dieser fast identisch, was auf eine gute Generalisierungsfähigkeit schließen lässt.

[25] `https://www.ndcn.ox.ac.uk/divisions/fmrib/fmrib-analysis-group`

Werden nun die Originaldaten ohne Auswahl benutzt und die Kostenfunktion mit Gewichten angepasst, so muss das Netzwerk mittels der *Relevanz*, *Sensitivität* und dem *F1 Maß* evaluiert werden. Bei diesem Modell ist anzumerken, dass auf L2 Regularisierung verzichtet und als Gewichtsvektor folgender verwendet wurde[26]:

$$w = \begin{bmatrix} 5,1641791 \\ 0,5536 \end{bmatrix} \tag{9}$$

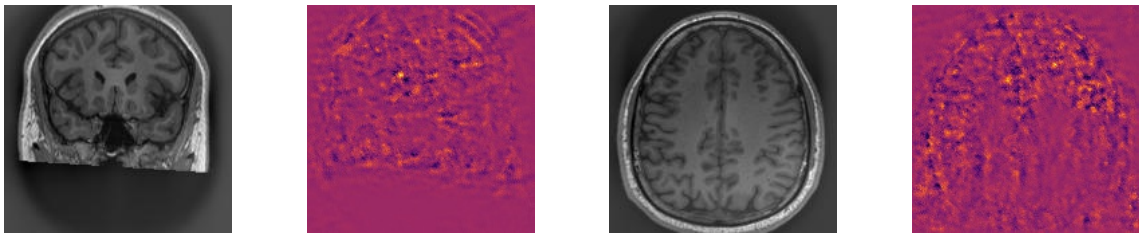
Tabelle 5 fasst unsere Ergebnisse zusammen. Anhand dieser Tabelle lässt sich erkennen,

Methode	Genauigkeit	Sensitivität	Relevanz	F1 Maß
Unterauswahl	51,05%	56,14%	50,95%	53,42%
Überauswahl	80%	94,01%	73,44%	82,46%
gewichtete Kreuzentropie	82,34%	93,88%	86,85%	90,23%

Tabelle 5: Ergebnisse der Händigkeitsklassifikation mit Unter-/ Überauswahl & gewichteter Kreuzentropie

dass die Genauigkeitsergebnisse von Überauswahl und gewichteter Kreuzentropie sehr ähnlich sind. Jedoch sollte hier nicht nur auf die Genauigkeitsmetrik geachtet werden. Im Gegensatz zu dem Überauswahlsergebnis, liegt die Relevanz bei gewichteter Kreuzentropie bei fast identischer Sensitivität deutlich höher. Es werden also viele Personen als Rechtshänder klassifiziert. Von diesen Personen sind auch viele tatsächlich Rechtshänder. Es ist aber trotzdem eindeutig schwierig festzustellen, welches der beiden Netzwerke nun besser ist. Aus diesem Grund ist es sicherer das Netzwerk, was auf Überauswahlsdaten trainiert wurde, zu benutzen, da die Daten balanciert sind und somit eine Bewertung nach den herkömmlichen Methoden wie Trainings-/ Testgenauigkeit möglich ist.

9.5 Klassifikationsmerkmale



(a) Feldvergleich 1 (b) Wärmebild 1 (c) Feldvergleich 2 (d) Wärmebild 2

Abbildung 10: $\frac{\partial}{\partial X} K$ Wärmebild der Händigkeitsklassifikation mit $K =$ Kosten

Anhand von Abbildung 10 lässt sich erkennen, dass hier, im Gegensatz zu der Geschlechtsklassifikation, die Hirnrinde nicht betrachtet wird. Je nach Gehirnschnitt wird

[26] Dieser Vektor wurde durch sklearn.utils.class_weight.compute_class_weight ausgerechnet

eine andere Region für die Klassifikation verwendet. So wird bei der rostral-caudal Ansicht (Abb. 10b) ein eher kleiner Teil aus der Mitte des Gehirns benötigt. Bei der dorsal-ventral Ansicht (Abb. 10d) werden hingegen, Teile des vorderen rechten Kortexes verwendet.

10 Lerntransfer

Lerntransfer ist eine oft benutze Technik im Deep Learning. Dabei wird in der Hoffnung, bessere Ergebnisse zu erzielen, auf die Arbeit von anderen neuronalen Netzen aufgebaut.

10.1 Was ist Lerntransfer?

Lerntransfer eine Technik, welche ein Modell, das schon für ein bestimmtes Problem trainiert wurde, für eine neue Aufgabe recycelt und das schon Gelernte verwertet. Lerntransfer ist im Deep Learning Bereich populär, da massive Ressourcen gebraucht werden, um ein tiefes neuronales Netz zu trainieren. Dazu zählen natürlich auch die Quantität und Qualität der Trainingsdaten ebenso wie die Hardware. Größere Netze benötigen eine bessere Hardware. Lerntransfer funktioniert jedoch nur, wenn das benutzte Modell sehr generalisiert ist, also keine Über- oder Unteranpassung stattfindet.

10.2 Wie wird Lerntransfer benutzt?

Die meisten SOTA[27] Modelle wie VGG[6] oder ResNet[10] wurden auf den ImageNet[11] Klassen trainiert. ImageNet enthält ca. 14 Millionen Bilder mit insgesamt 1000 Klassen. Solche Modelle können also mit einer sehr hohen Genauigkeit, Objekte auf Bildern bestimmen. Diese Netze verwenden unter anderem auch Konvolutionen, um bestimmte Merkmale aus Bildern herauszuarbeiten und daraufhin entsprechende Objekte zu klassifizieren. Bei dem Lerntransfer werden genau diese Konvolutionen benutzt, um Merkmale von Bildern zu extrahieren. Viele Modelle besitzen am Ende zwei oder mehr vollständig verbundene Schichten und eine Klassifikationsschicht. In dem Fall von VGG hat diese Schicht 1000 Knoten. Wird Lerntransfer benutzt, so werden diese drei letzten Schichten abgetrennt und durch neue vollständig verbundene Schichten und eine neue Klassifikationsschicht ersetzt. Das hat zur Folge, dass durch die Konvolutionen schon sehr gute Informationen aus den Bildern herausgearbeitet werden und somit nur die letzten drei Schichten trainiert werden müssen. Es sollte darauf geachtet werden, die Konvolutionen nur als Extrahierer zu nutzen, diese also nicht weiter zu trainieren. Jedoch ist es auch möglich, die Konvolutionen weiter zu trainieren, also zu verfeinern. Diese Methode nennt sich „Feinabstimmung“.

	Similar dataset	Different dataset
Small dataset	Transfer learning: highest level features + classifier	Transfer learning: lower level features + classifier
Large dataset	Fine-tune*	Fine-tune*

Abbildung 11: Aus Standford Class CS231n[28]

10.3 Lerntransfer oder Feinabstimmung?

Wie in Abbildung 11 zu sehen ist, wird bei einem großen Datensatz immer empfohlen, das gesamte Netzwerk weiter oder von Anfang an zu trainieren. Ist der Datensatz relativ klein, so kommt es auf den Inhalt des Datensatzes und auf die Klassen des vortrainierten Netzwerkes an. Netzwerk A sei z.B. schon auf verschiedene Hunderassen trainiert. Möchte man nun weitere Hunderassen hinzufügen, reicht es aus, *Higher Level Merkmale*, also Daten aus späten, überlicherweise der letzten, Konvolutionen zu extrahieren, da der Datensatz sehr ähnlich ist. Möchte man das Netzwerk A nun auf verschiedenen Arten von Pflanzen trainieren, empfiehlt es sich, Merkmale aus früheren Konvolutionen zu extrahieren. Das liegt daran, dass diese Merkmale sehr abstrakt sind und somit nicht nur auf Hunderassen angewendet werden können. Ein Vorteil von Lerntransfer ohne Feinabstimmung ist, dass das Netz schneller konvergiert und die benötigte Trainingszeit enorm sinkt. Die benötigten Merkmale sind schon vorhanden und müssen somit nicht gelernt werden.

10.4 Imagenet

Netzwerke wie VGG16/19[6] oder ResNet[10] wurden auf dem ImageNet[11] Datensatz trainiert. Dieser Datensatz enthält über 14 Millionen handannotierte Bilder, auf denen vermerkt ist, welches Objekt sich darauf befindet. 10% der Bilder enthalten auch Informationen darüber, wo genau sich das Objekt auf dem Bild befindet[29]. Jede Kategorie, von denen insgesamt 20.000 existieren, enthält mehrere 100 Bilder, auf denen das entsprechende Objekt in verschiedenen Szenarien abgebildet ist.

[27] State-of-the-Art
[28] `https://cs231n.github.io/`
[29] Auch Bounding Box genannt

10.5 SOTA und MRT Scans

Werden nun die Daten aus dem Imagenet mit unserem Datensatz, bestehend aus MRT Scans von Gehirnen, verglichen, so wird klar, dass sich bei diesen Daten keinerlei Gemeinsamkeiten finden lassen. Aus diesem Grund sind wir skeptisch, ob Lerntransfer auf Netzwerken, welche auf Imagenet trainiert wurden, einen positiven Effekt hat.

10.5.1 Datenvorverarbeitung

Unsere MRT Scans sind Schwarzweißbilder, haben also nur einen Farbkanal. VGG und andere Netze wurden jedoch mit Farbbildern trainiert, benötigen also drei Farbkanäle. Eine einfache Möglichkeit dieses Problem zu umgehen, besteht darin, unsere Grauschicht einfach dreimal hintereinander zu schieben, um somit drei Farbwerte zu erhalten.

```
w, h = img.shape
ret = np.empty((w, h, 3), dtype=np.float32)
ret[:, :, 2] =  ret[:, :, 1] =  ret[:, :, 0] =  img
```

10.5.2 VGG16 und VGG19

Das VGG16 und dessen größere Variante VGG19 sind beide sehr große Netzwerke[30], welche beide auf dem Imagenet Datensatz eine geringe Fehlerrate von nur $7,5\%$ aufweisen. Um nun beispielsweise eine Geschlechtsklassifikation mittels Lerntransfer auf VGG16 auszuführen, muss zuerst entschieden werden, welche Art von Merkmalen extrahiert werden soll. Laut Abbildung 11 sollten in unserem Fall *Highest Level Merkmale*, also Merkmale aus frühen Konvolutionen, extrahiert werden. Wir haben uns dazu entschieden, die Merkmale aus der ersten Max-Pooling Schicht zu extrahieren. Bei einer Eingangsgröße von *170x170x3* werden nun Merkmale der Größe *85x85x64* erzeugt, unabhängig davon, ob VGG16 oder das größere VGG19 verwendet wird. Anschließend werden diese mittles *Global Average Pooling* auf einen Vektor der Länge *64* reduziert. Danach folgen zwei vollständig verbundene Schichten und eine Klassifikations- oder Regressionsschicht.

Netzwerk	Label	Genauigkeit
VGG16	Geschlecht	70,29%
VGG16	Händigkeit	76,87%
VGG16	Alter	6%
VGG19	Geschlecht	70,33%
VGG19	Händigkeit	75,83%
VGG19	Alter	6%

Tabelle 6: Ergebnisse von Lerntransfer mit VGG16/19

Es sollte darauf hingewiesen werden, dass alle Schichten vor der Max-Pooling Schicht eingefroren wurden. Sie wurden also nicht weiter trainiert. Nur die vollständig verbun-

[30] VGG16 hat 138.357.544 Parameter `https://stackoverflow.com/a/28242883`

denen Schichten werden trainiert. Als Optimierer wurde Stochastic Gradient Descent mit einem Momentum von $0,9$, einer Fallrate von $0,0005$ und einer Lernrate von $0,001$ verwendet. Die Kostenfunktion ist die Kreuzentropie. Außerdem wurde die Lernrate mittels exponentiellem Verfall und dem Faktor $0,2$ alle 20 Epochen verringert. Für die Regression des Alters wurde $M = 2,5$ und ein Optimierer mit adaptiver Lernrate[31] gewählt. Die Lernrate betrug $0,001$. Außerdem wurde für die Altersregression auch der in Kapitel 8.4 angesprochene *Durchschnittliche Quadratische Fehler mit Varianz* als Kostenfunktion verwendet. Allgemein fällt in Tabelle 6 auf, dass sich die Ergebnisse der beiden neuronalen Netze, trotz der höheren Komplexität von VGG19, nur um einen kleinen Prozentsatz unterscheiden. Die extrahierten Merkmale im Bezug auf Geschlecht und Händigkeit haben trotz des sehr unterschiedlichen Datensatzes immer noch genug Informationen enthalten, um damit zwischen männlich und weiblich oder zwischen Links- und Rechtshänder zu unterscheiden. Die extrahierten Merkmale für das Alter waren von keinem großen Nutzen. Netzwerke wie ResNet[10] oder sogar Inception-ResNet-V2[12] erzielten im Vergleich zu dem VGG Netzwerk keine Verbesserungen. Aus diesem Grund sind diese hier auch nicht aufgeführt.

10.5.3 SOTA Fazit

Vergleicht man diese Ergebnisse mit unseren bisherigen Resultaten, so ist das Fazit zu ziehen, dass Lerntransfer mit Netzwerken, welche auf Imagenet trainiert wurden, keinen sichtbaren Vorteil bringt, da die Domänen zu verschieden sind.

10.6 Lerntransfer innerhalb einer Domäne

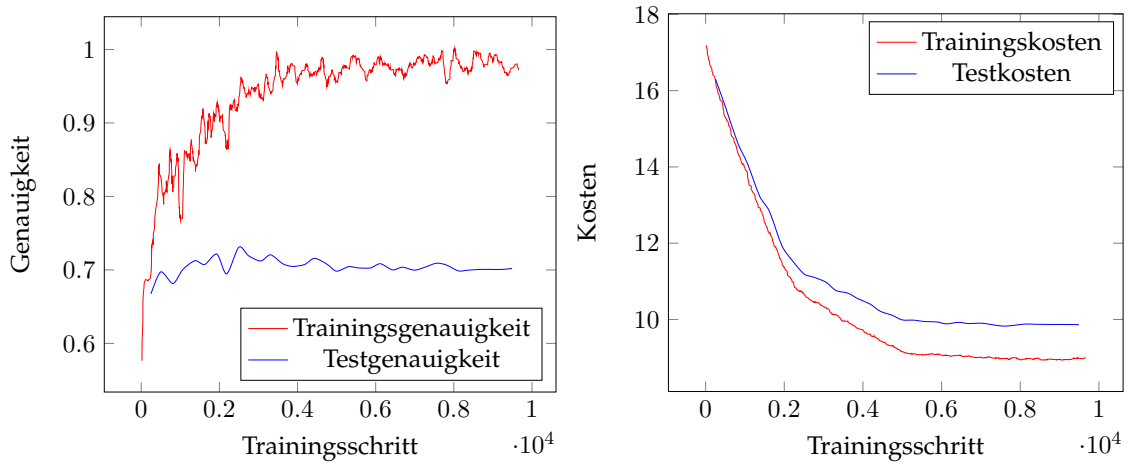
Da die neuronalen Netze, welche von uns von Grund auf trainiert wurden, schon gute Ergebnisse liefern, wird nun versucht, innerhalb einer Domäne mit diesen Netzen Lerntransfer durchzuführen. Mit diesen Experimenten lässt sich dann auch die Frage beantworten, inwiefern z.B. Geschlecht und Händigkeit zusammenhängen.

10.6.1 Händigkeitsklassifikation mittels Geschlechtsmerkmalen

In dem ersten Versuch wird nun das Netzwerk, welches für das Problem der Geschlechtsklassifikation trainiert wurde, benutzt, um die Händigkeit einer Person vorherzusagen. Zur Erinnerung: Die Netzwerkarchitektur ist in Abbildung 3 dargestellt. Zunächst wird versucht, die Händigkeit ohne weiteres Training auf dem Netzwerk vorherzusagen. Dabei erzielt das Netz eine Genauigkeit von 53%. Um die Genauigkeit nun weiter zu steigern oder sogar eine bessere Klassifikation als das Netzwerk aus Kapitel 7 zu erzielen, wird dieses nun weiter trainiert. Hierbei werden die High-Level Merkmale aus der letzen Konvolution, inklusive Max-Pooling, extrahiert. Es werden alle Schichten vor dieser Max-Pooling Schicht eingefroren, also nicht mehr trainiert. Die darauf folgenden vollständig verbundenen Schichten und die Klassifikationsschicht werden als Einzige weiter

[31] RMSprop `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

trainiert. Hierbei ist zu beachten, dass diese Schichten die vorherigen Gewichte beibehalten, aber trotzdem weiter trainiert werden.
Als Optimierer wurde Stochastic Gradient Descent mit einer Lernrate von $0,001$ und einem Momentum von $0,9$ verwendet. Die Lernrate wurde mittels exponentiellem Verfall und einer Fallrate von $0,3$ alle 10 Durchläufe verringert. Die Kostenfunktion ist, wie zu erwarten, die Kreuzentropie Funktion. Das Netzwerk wurde für insgesamt 25 Epochen trainiert. Das daraus resultierende Ergebnis ist mit einer Genauigkeit von *70%* um etwa 10% schlechter als unser bisheriges Resultat. Daraus lässt sich die Schlussfolgerung ziehen, dass zumindest ein paar Merkmale, welche für die Geschlechtsklassifikation zuständig sind, auch für die Klassifikation der Händigkeit eine Rolle spielen.



(a) Trainingsgenauigkeit

(b) Trainingskosten

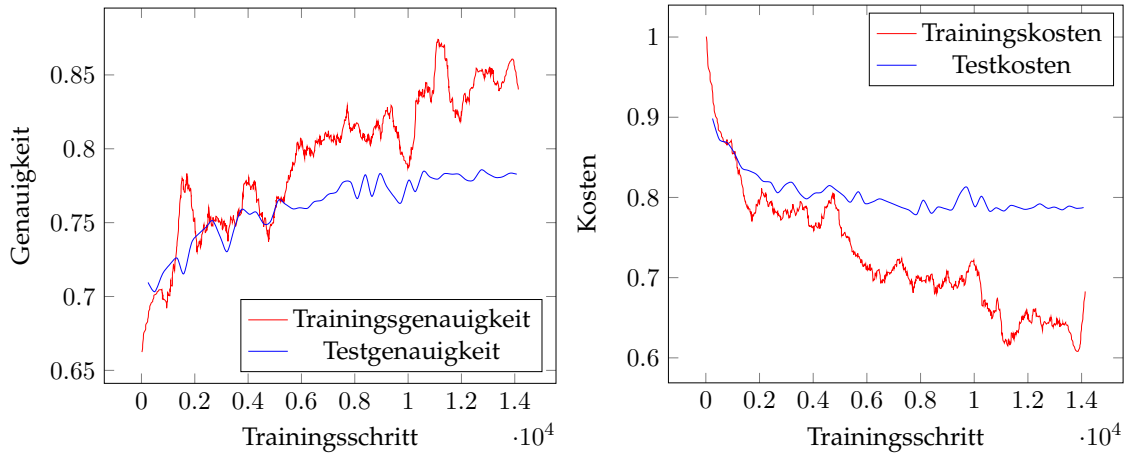
Abbildung 12: Lerntransfer: Händigkeitsklassifikation mittels Geschlechtsmerkmalen Ergebnisse

Jedoch wird durch Abbildung 12a klar, dass das Netzwerk trotz der Maßnahmen gegen Überanpassung zu Überanpassung neigt. Das lässt sich an der großen Differenz von Trainings- und Testgenauigkeit erkennen. Auch ist in Abbildung 12b zu beobachten, dass die Werte der Kostenfunktion, auch wenn diese fallen, sehr hoch sind.

10.6.2 Geschlechtsklassifikation mittels Händigkeitsmerkmalen

In dem zweiten Versuch wird das Netzwerk, welches zuvor für die Händigkeitsklassifikation trainiert wurde, verwendet, um das Geschlecht einer Person zu bestimmen. Hier wird die Netzwerkarchitektur aus Kapitel 8 verwendet, welche auf den mittels SMOTE Überauswahl erzeugten Daten, trainiert wurde. Wird das Netzwerk nicht weiter trainiert, so erzielen wir eine Genauigkeit von 55%. Wie bei dem ersten Versuch werden hier auch die High-Level Merkmale aus dem Netzwerk extrahiert und nur die letzten vollständig verbundenen Schichten (inklusive Klassifikationsschichten) weiter trainiert. Hierbei wurde wie in Kapitel 9.4 erwähnt L2 Regularisierung verwendet, um Überan-

passung zu vermeiden. Als Optimierer wurde ein Momentum basierter Optimierer[32] mit einer Lernrate von $0,0001$ eingesetzt. Außerdem wurde die Lernrate nach jeweils 10 Epochen mittels exponentiellem Verfall und einer Fallrate von $0,3$ verringert, damit das Netz nicht in einem lokalen Optimum stecken bleibt. Auch dieses Netzwerk wurde für insgesamt 25 Epochen trainiert. Erstaunlicherweise erzielt das neuronale Netz durch diese Methode eine Genauigkeit von 78%. Daraus lässt sich schlussfolgern, dass Merkmale, welche für die Händigkeitsklassifikation zuständig sind, auch einen sehr großen Teil der Geschlechtsklassifikation beeinflussen.



(a) Trainingsgenauigkeit (b) Trainingskosten

Abbildung 13: Lerntransfer: Geschlechtsklassifikation mittels Händigkeitsmerkmalen Ergebnisse

Im Gegensatz zu Kapitel 10.6.1 findet hier nur wenig Überanpassung statt. Jedoch lässt sich in Abbildung 13 beobachten, dass das neuronale Netz sehr instabil ist. Die Werte der Graphen weisen innerhalb kürzester Zeit große Varianzen auf.

10.6.3 Andere Features

Neben Geschlecht und Händigkeit steht uns noch das Alter als Label zur Verfügung. Einen Lerntransfer zwischen Alter und z.B. Geschlecht durchzuführen, würde aus unserer Sicht keine positiven Resultate erzielen. Der entscheidende Unterschied zwischen Geschlecht/ Händigkeit und Alter ist, dass hier zwei verschiedene Arten von Ergebnissen erwartet werden. Bei der Geschlechts- und Händigkeits*klassifikation* wird, wie der Name impliziert, eine Einordnung in Klassen als Resultat erwartet. Bei der Alters*regression* hingegen wird ein Skalar, welcher das Alter abschätzen soll, erwartet. Dies resultiert logischerweise in einer in Teilen unterschiedlichen Netzwerkarchitektur. Die Ergebnisse sind trotz dieser nur kleinen Veränderung der Architektur nicht besser als der KNN Algorithmus1. Wir verwenden das Geschlechts- oder Händigkeitsnetzwerk aus Kapitel 7 und 9 und ersetzen die letzte Klassifikationsschicht durch eine Regressionsschicht. Der Optimierer ist weiterhin Momentum basiert. Die Lernrate beträgt $0,0001$. Das Netzwerk

[32] Adam Optimizer [7]

wurde für insgesamt 25 Epochen trainiert. Es erreicht mit $M = 2,5$ eine Genauigkeit von 11%. Jegliches Anpassen der Hyperparameter resultiert in schlechteren oder identischen Ergebnissen. Als Fazit lässt sich daraus schließen, dass entweder Klassifikation und Regression zu unterschiedlich sind oder Geschlechts- und Händigkeitsmerkmale keine Informationen für das Alter liefern.

11 Fazit und Ausblick

Aufgabe	Genauigkeit
Geschlechtsklassifikation	83,26%
Händigkeitsklassifikation (Überabtastung)	80%
Altersregression ($M = 2,5$)	23,82%

Tabelle 7: Zusammenfassung der Ergebnisse

In Tabelle 7 sind unsere besten Endresultate zusammengefasst. Im Bereich der Geschlechtsklassifikation lässt sich zusammenfassen, dass unser neuronales Netz an die Genauigkeit eines geschulten und erfahrenen Arztes heranreicht. Mensch und Maschine achten bei Bestimmung des Geschlechts unter anderem auf die Schädelform. In diesem Bereich unterscheiden sich ein Arzt und das Netz also sehr wenig voneinander.
Die Bestimmung des Alters hingegen liegt selbst mit $M = 5$ deutlich unter der Leistung eines Arztes. Jedoch konnten Muster in der falschen Erkennung des Alters festgestellt werden. Eventuell lassen sich diese Probleme durch weiteres Anpassen der Kostenfunktion beheben.
In der Händigkeitsklassifikation wurde der größte Durchbruch erzielt. Für einen Arzt ist es sehr schwierig, anhand von strukturellen T1-Scans die Händigkeit einer Person zu erkennen. Erstaunlicherweise stellt dies für unser vorgestelltes Netzwerk, sofern vorher entsprechende Maßnahme getroffen wurden, kein Problem dar.
Es wurde auch festgestellt, dass das in Deep Learning oft benutzte Verfahren des Lerntransfers hier keine großen Vorteile erzielt. Ein großes Problem sind hierbei die speziellen Anforderungen der Domäne. Ein von Grund auf trainiertes Netzwerk liefert oft dieselben oder sogar bessere Resultate.
Eine neue mögliche Fragestellung, welche sich hieraus ergibt, ist, ob es möglich wäre, ein Netzwerk zu trainieren, welches alle drei Labels erkennt. Es würde dann nur ein Netz benötigt, um Alter, Geschlecht und Händigkeit einer Person mit hoher Genauigkeit festzustellen. Auch könnte, wie in anderen Arbeiten, welche in Kapitel 2 erwähnt wurden, versucht werden, in einem neuronalen Netz dreidimensionale statt den jetzigen zweidimensionalen Konvolutionen zu verwenden. Dies würde die Möglichkeit eröffnen, einen ganzen MRT Scan als dreidimensionales Array für das Training zu verwenden. Ein Nachteil dieser Methode ist, dass ihre Hardwareanforderungen sehr hoch sind. Aus diesem Grund wurde dieses Verfahren von uns nicht weiter verfolgt.

Literatur

[1] Havaei M. et al. Brain tumor segmentation with deep neural networks. 2016.

[2] S. Pereira et al. Brain tumor segmentation using convolutional neural networks in mri images. 2016.

[3] T. Brosch et al. Deep 3d convolutional encoder networks with shortcuts for multiscale feature integration applied to multiple sclerosis lesion segmentation. 2016.

[4] Viktor Wegmayr et al. Classification of brain mri with big data and deep 3d convolutional neural networks. 2018.

[5] Williams Brenda A. and Rogers Tracy L. Evaluating the accuracy and precision of cranial morphological traits for sex determination. *Journal of Forensic Sciences*, 51(4):729–735.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[9] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[11] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[12] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

Abbildungsverzeichnis

Tabellenverzeichnis