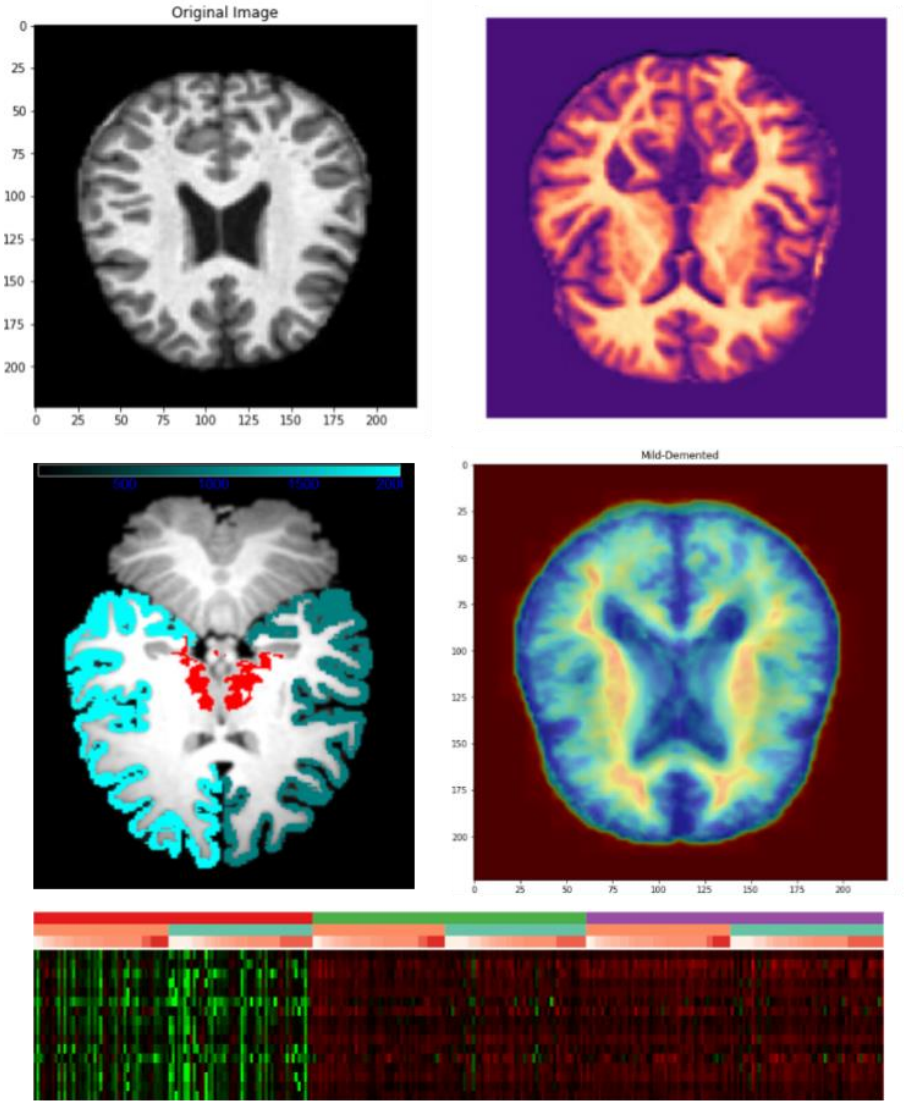
Alzheimer-Erkennung durch Künstliche Intelligenz



Teilnehmer: Niklas Bennewiz, (17 Jahre)
Erarbeitungsort: Romain-Rolland-Gymnasium Berlin
Projektbetreuerin: Frau Dr. Köhler-Krützfeldt
Fachgebiet: Informatik/Mathematik
Wettbewerbssparte: Jugend forscht
Bundesland: Berlin
Wettbewerbsjahr: 2023

Inhaltsverzeichnis

Seite

1. Einleitung

Alzheimer ist eine progressive, neurodegenerative Erkrankung, die dazu führt, dass die Fähigkeit des Gehirns, Informationen zu speichern und abzurufen, abnimmt. Die Krankheit betrifft hauptsächlich die Bereiche des Gehirns, die für Gedächtnis, Denken und sprachliche Fähigkeiten verantwortlich sind. Die Symptome von Alzheimer können sich langsam entwickeln und beginnen oft mit vergesslichen Episoden, die im Laufe der Zeit zu schweren Gedächtnisverlust und Beeinträchtigungen der kognitiven Funktionen führen. Es gibt keine Heilung für Alzheimer und die Ursachen sind noch nicht vollständig verstanden. Es gibt jedoch Behandlungen, die helfen können, die Symptome zu verlangsamen und die Lebensqualität der betroffenen Personen und ihrer Familien zu verbessern. Alzheimer zählt zu den primären Demenzen, bei denen das demenzielle Verhalten direkt auf Gehirnveränderungen zurückzuführen ist.

In den Gehirnen von Menschen mit Alzheimer treten charakteristische Veränderungen auf. Eine dieser Veränderungen ist die Ansammlung von Beta-Amyloid-Plaques. Dies sind Ablagerungen von A-beta-Peptiden, die sich außerhalb der Nervenzellen bilden und deren normale Funktion beeinträchtigen und die Kommunikation zwischen ihnen unterbrechen. Außerdem treten in den Tau-Proteinen, die normalerweise ein wichtiger Bestandteil des Zellgerüsts im Gehirn sind und eine wichtige Rolle bei der Unterstützung der Struktur und Funktion von Nervenzellen spielen, Veränderungen auf bzw. sie sind falsch gefaltet. Diese Veränderungen im Tau-Protein führen zu Ansammlungen von abnormen Proteinen, die als Neurofibrillen bekannt sind, welche die Funktion von Nervenzellen beeinträchtigen und schließlich zu deren Tod führen. Diese Ansammlung von Beta-Amyloid-Plaques und Tau-Proteinen im Gehirn von Alzheimer-Patienten führen durch einen Abbau von Nervenzellen und Synapsen zum Verlust der kognitiven Funktionen, wie Gedächtnisverlust, Schwierigkeiten bei der Sprache und Problemen bei der Orientierung.

Die frühzeitige Erkennung von Alzheimer ist sehr wichtig, um so in einem frühen Stadium schon anzufangen gegen die Neurodegeneration vorzugehen. Neben der frühzeitigen Erkennung von Alzheimer, forschen auch viele Institute und Organisationen an der Ursache und einem Heilmittel für Alzheimer. Einige dieser Organisationen sind die Alzheimer's Disease Neuroimaging Initiative (ADNI) [1] und das Allen Institute for Brain Science [2].

Ich interessiere mich sehr für Maschine Learning und noch genauer für Deep Learning, habe auch schon viele Praktika gemacht, wie z.B. beim Deutschen Forschungsinstitut für Künstliche Intelligenz (DFKI). Zurzeit arbeite ich neben der Schule bei der Firma SVA, wo ich mich mit der Architektur von Deep Learning Modellen beschäftige. Da das Maschine Learning und somit auch das Deep Learning sich aus den Beobachtungen des Gehirns, bzw. einfacher Neuronen entwickelt hat, interessiere ich mich auch sehr für das Gehirn und möchte noch mehr darüber lernen. Um so mithilfe von Deep Learning mehr über das Gehirn herauszufinden und durch vom Gehirn gegebene Praktiken neue Deep Learning Modelle zu entwickeln.

Für Jugend Forscht möchte ich Deep Learning Modelle zur Feststellung von Alzheimer erkrankten Personen bauen und durch diese Rückschlüsse auf besondere Auffälligkeiten in Gehirnregionen feststellen. Dazu stelle ich in meiner Forschungsarbeit mehrere Modelle zur Klassifizierung von Alzheimer erkrankten Personen anhand deren MRT-Scans (Magnet-Resonanz-Tomographie) vor. Diese Modelle sollen es einerseits Ärzten erleichtern Menschen mit Alzheimer zu erkennen, andererseits soll durch die Explainability (Erklärbarkeit) der Modelle genau gezeigt werden, in welchen Regionen des Gehirns und in welcher Phase der Alzheimer-Erkrankung diese besonders auffällig ist. Die Ergebnisse und Modelle sollen frei zugänglich in einer interaktiven App dargestellt werden. So können vielleicht auch neue Informationen über die Ursache bzw. das Krankheitsmuster von Alzheimer gewonnen werden oder andere Entdeckungen bestätigt werden. Durch die App können meine Ergebnisse und Modelle auch von Ärzten zur Diagnostik und Forschung verwendet werden.

2. Vorgehensweise

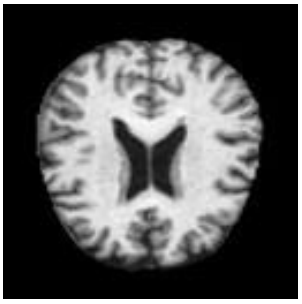
Um dies zu erreichen, werden MRT-Scans und Genexpressions-Beziehungen von gesunden und erkrankten Menschen benutzt. Die MRT-Scans sind in vier Klassen aufgeteilt, die von Nicht-Erkrankt bis zu Mittel-Erkrankt reichen. Um die beste Genauigkeit zu erreichen, trainiere, hyper-optimiere und vergleiche ich mehrere Modelle miteinander. Zugleich werde ich auch die Erklärbarkeit von all diesen Modellen darstellen, da es unterschiedliche Methoden und somit auch unterschiedliche Ergebnisse für die jeweiligen Modelle gibt. Alle meine Programme und Modelle befinden sich in einem GitHub-Repository [14].

3. MRT-Scan-Klassifizierung

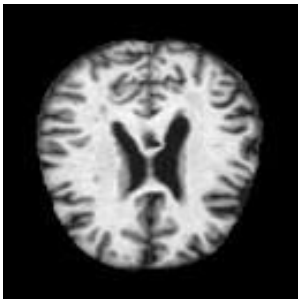
3.1. Datensatz

Der Datensatz besteht insgesamt aus 6.400 Bildern und lässt sich in vier Klassen einteilen: nicht dement (3.200 Bilder), sehr leicht dement (2.240 Bilder), leicht dement (896 Bilder) und mittel-schwer dement (64 Bilder). Wie an der Bezeichnung der Bilder schon zu erkennen ist, hat man nicht die gesamte Gehirn-Matrix einer Magnet-Resonanz-Tomographie, sondern nur einen Scan, bzw. ein Bild. Die Scans sind nach der Reisberg- oder Global-Deterioration-Skala eingeteilt und werden im GitHub-Repository nochmal detaillierter erklärt. Im Folgenden sind Beispiele für jede Klasse gegeben:

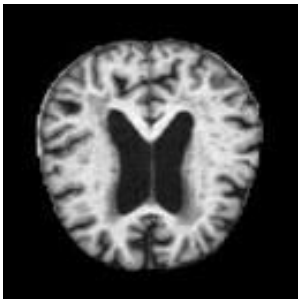
Nicht-Dement



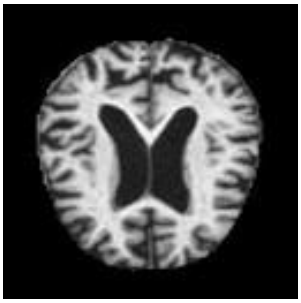
Sehr-Leicht-Dement



Leicht-Dement



Mittel-Schwer-Dement



Die Bilder sind alle schon vorverarbeitet und in unterschiedlichen Graustufen. Die Daten sind von Kaggle [3], wo auch die ursprünglichen Quellen angegeben sind. Von diesen sind alle international anerkannte Organisationen, die zum Thema Alzheimer forschen. Leider war es nicht möglich mehr Daten zu bekommen, da man auf den Webseiten der Organisationen, um sich anzumelden, ein Institut angeben muss. Ich stehe jedoch zurzeit schon im Kontakt mit Professoren des Instituts Leiden, in der Niederlande, und erhoffe mir bald noch mehr und genauere Scans zu bekommen.

3.2. Preprocessing

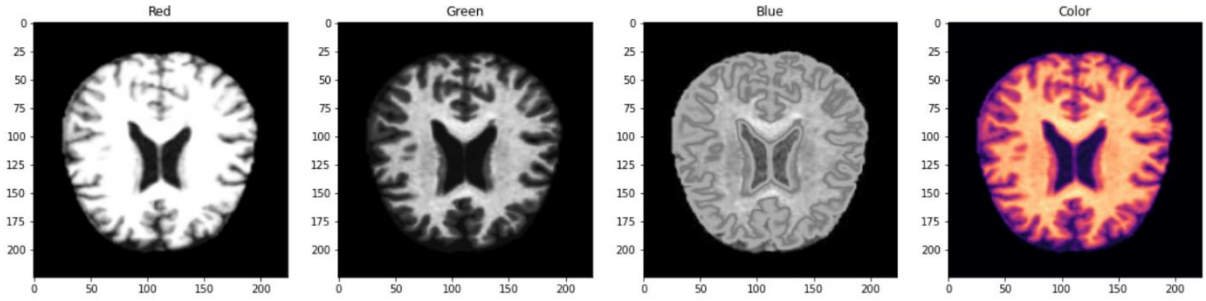
Die MRT-Scans sind schon vorverarbeitet, es wurde also eine Ungleichmäßigkeitskorrektur, Gradverzerrung und T1-Gewichtung vorgenommen. Da die Daten schon vorverarbeitet sind, haben sie kaum Noise und mussten nicht zentriert werden. Es wird jedoch noch als Teil der Vorverarbeitung, als

ersten Schritt, Pseudo-Color-Enhancement verwendet. Dies wird getan, um bestimmte Merkmale in einem Bild bzw. Scan hervorzuheben, wie graue und weiße Substanz und Knochen-Masse. Bei an Alzheimer erkrankten Patienten kann es aber auch dazu verwendet werden, um die Sichtbarkeit von anderen Merkmalen, wie z.B. toten Nervenzellen und Synapsen hervorzuheben. Die Farben, die verwendet werden stellen keine natürlichen Farben des Bildes dar, sondern werden auf der Grundlage bestimmter Schwellenwerte und Farbpaletten zugewiesen. Es werden also die verschiedenen Graustufen eines Bildes mithilfe einer linearen oder nicht-linearen Funktion in ein RGB-Farbbild bzw. drei weitere Graubilder transformiert. Die 3 Graubilder stellen die Farbintensität Rot, Grün und Blau dar und ergeben übereinandergelegt ein Farbbild. Die Farbverläufe der Algorithmen, die ausgesucht wurden, sehen so aus:

Magma-Funktion

Jet-Funktion

Beispiel eines Bildes mit der Magma-Funktion:

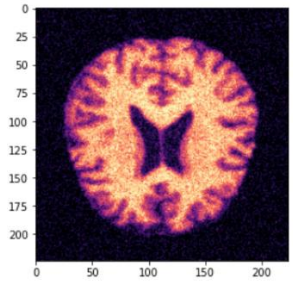


Was man sich durch das Anwenden der Funktionen erhofft, ist die bestimmten Gehirn-Substanzen besser darzustellen. So können Betrachter und Modell bei den Graubildern ganz links, die Umrandungen und Windungen des ganzen Gehirns gut sehen. In der Mitte sieht man die Auswüchse der weißen Masse sehr gut und rechts sieht man die Abtrennung zwischen grauer und weißer Masse, wobei die graue Masse hervorgehoben ist. Zudem werden hier (rechtes Bild) in der Mitte die Ventrikel sehr gut dargestellt. So hat das Modell statt einem Bild, drei unterschiedliche Bilder, um unterschiedliche Merkmale hervorheben, zur Verfügung, was es besser Alzheimer erkennen lassen sollte.
Das Problem, was sich generell bei der Datenlage ergibt, ist, dass die Klassen ungleich groß sind. Eine ungleichmäßige Verteilung der Klassen in einem Multi-Klassen-Klassifizierungsproblem kann dazu führen, dass das Modell die häufigeren Klassen besser erkennt als die selteneren Klassen. Es gibt mehrere Möglichkeiten, dieses Problem anzugehen:

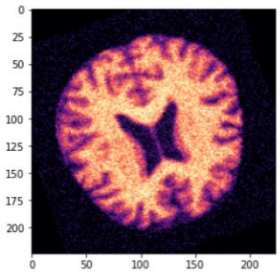
1. Oversampling (Überabtastung) der seltenen Klassen: Hierbei werden zusätzliche Kopien der Beispiele der seltenen Klassen zum Trainingsset hinzugefügt, um die Anzahl der Beispiele dieser Klassen zu erhöhen.
2. Undersampling (Unterabtastung) der häufigeren Klassen: Hierbei werden Beispiele der häufigeren Klassen aus dem Trainingsset entfernt, um die Anzahl der Beispiele dieser Klassen zu verringern.
3. Synthetische Datenerstellung: Hierbei werden neue Beispiele der seltenen Klassen durch die Verwendung von generativen Modellen erstellt.
4. Klassen-Gewichtung: Hierbei werden die Verlustfunktionen im Training so modifiziert, dass sie die Beispiele der seltenen Klassen stärker gewichten.
5. F1-Score oder ähnliche Metriken anstelle von Genauigkeit/Accuracy als Optimierungsziel: Hierbei werden Metriken verwendet, die die Leistung des Modells für jede Klasse einzeln betrachten anstatt nur die Gesamtleistung.

Da dies noch zur Vorverarbeitung gehört wird sich erstmal nur auf die ersten 3 Punkte bezogen. Undersampling (Punkt 2) ist eine schlechte Idee, da die kleinste Klasse (non-demented) nur über 64 Bilder verfügt. Würde von allen Klassen nur 64 Bilder genommen werden, so würden zu viele Bilder nicht benutzt werden, die das Modell aber zum Lernen braucht. Oversampling (Punkt 1) ist eine gute Idee, da so noch mehr Bilder für kleinere Klassen erzeugt werden. Die Bilder aber einfach zu verdoppeln, würde zu einem schlechteren Lernen bzw. einer schlechteren Generalisierung des Modells führen. Deswegen wird Oversampling mit Hilfe von Synthetischen Datenerstellung (Punkt 3) vorgeschlagen, da sich so die Daten ähneln würden, aber nicht genau gleich seien. Es werden hierzu Data-Augmentation-Techniken vorgestellt, bei denen das Bild rotiert, invertiert und/oder durch Noise verändert wird. Diese Bilder sehen dann wie folgt aus:

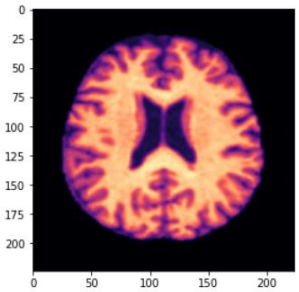
Gaussian Noise



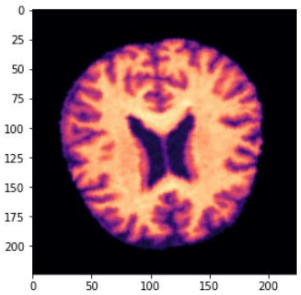
Gaussian Noise und Rotiert



Vertikaler Flip



Rotiert



Dies hat dem Modell leider nichts gebracht und seine Genauigkeit viel schlechter gemacht. Zudem wurde die Größe von allen Bildern noch auf 224x224 Pixel geändert. So haben besonders die CNNs mehr Pixel, um lokale Auffälligkeiten zu finden. Danach wurden aus den vorverarbeiteten Daten die Datensätze für das Trainieren, Validieren und Testen erstellt. Wobei darauf geachtet wurde, dass, je nach Größe des Datensatzes relativ gleich viele Bilder jeder Klasse enthalten sind:

Trainings-Datensatz: 4896 Bilder

Validation-Datensatz: 864 Bilder

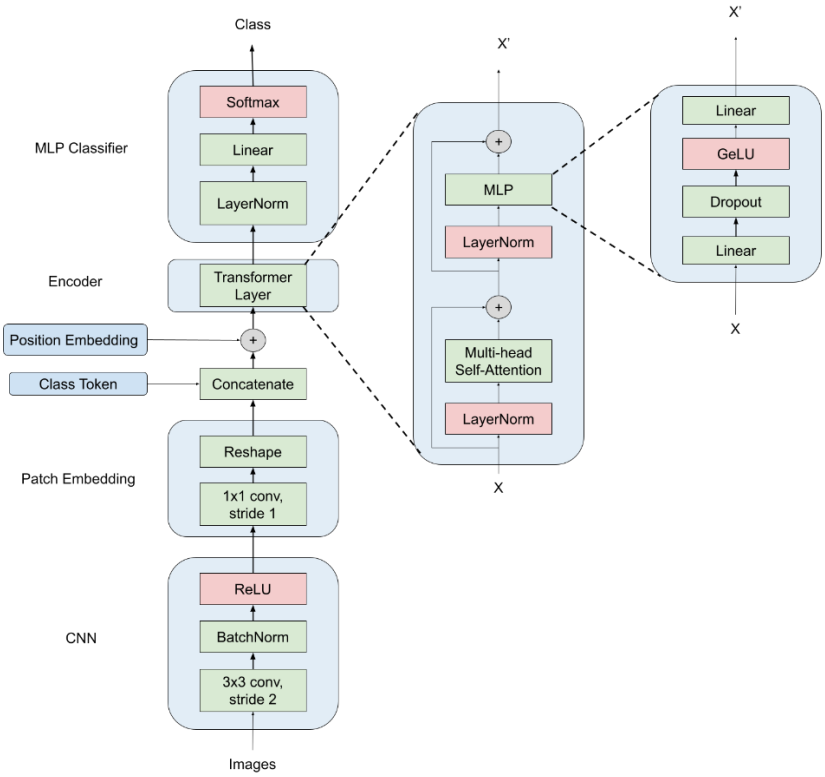
Test-Datensatz: 640 Bilder

Diese wurden dann noch in einen Dataloader von PyTorch [3] gepackt, was wichtig ist, damit das Modell diese auch später verwenden kann bzw. der Trainer von PyTorch-Ligthning [4] keine Probleme bekommt. Leider müssen die Batches dabei eine bestimmte Größe haben, sonst kriegen einige der Modelle, die verwendet werden, Probleme. Somit können ein paar der Bilder nicht verwendet werden.

3.3. CNN-Transformer-Modell

Ein CNN-Transformer (CNN-T) ist eine Kombination von zwei Arten von neuronalen Netzwerken: Convolutional-Neural-Networks (CNNs) und Transformer Networks. Ein CNN, bestehend aus mehreren Convolutional Layern, lernt lokale Muster und kann so vielversprechende semantische Informationen entdecken. Zudem ist es auch viel effizienter als andere Netzwerke, was besonders eine Rolle bei der Hyperparameter-Optimierung spielt. CNNs sind gut darin, lokale Merkmale in einem Bild

zu extrahieren und zu verarbeiten, da sie Convolutional Layer verwenden, die sich auf kleine Bereiche des Eingabebildes konzentrieren. Diese lokalen Merkmale können dann verwendet werden, um die Klasse eines Bildes vorherzusagen. Ein Nachteil von CNNs ist jedoch, dass sie Schwierigkeiten haben, globale Beziehungen zwischen Merkmalen in einem Bild zu berücksichtigen. Dies kann dazu führen, dass das Netzwerk Schwierigkeiten bei der Interpretation von Bildern hat, die komplexe Strukturen oder Beziehungen aufweisen. Transformer-Modelle hingegen ignorieren lokale Informationen (am Anfang und auf kleiner Ebene) und konzentrieren sich auf die Modellierung von globalen Informationen. Dies erreichen sie durch den Self-Attention-Mechanismen (Selbstaufmerksamkeits-Mechanismen). Ein CNN-Transformer kombiniert die Vorteile von CNNs und Transformer Networks, indem es die Convolutional Layer verwendet, um lokale Merkmale zu extrahieren und die Self-Attention-Mechanismen verwendet, um globale Beziehungen zwischen diesen Merkmalen zu berücksichtigen. Dies ermöglicht es dem Netzwerk, sowohl lokale als auch globale Beziehungen in einem Bild zu verstehen und zu interpretieren, was es besser darin macht, komplexe Aufgaben wie Bildklassifizierung und Bildbeschreibungen zu lösen. Die Konzentration auf kleine Bereiche ist wichtig, um sich auf bestimmte Gehirnregionen zu konzentrieren und Auffälligkeiten zu finden. Durch die globale Konzentration werden diese lokalen Auffälligkeiten miteinander verglichen, um so dann festzustellen, zu welcher Klasse der Patient einzuordnen ist. Am Ende des Transformers befindet sich noch ein Multi-Layer-Perceptron (MLP), welches die endgültige Klassifizierung vornimmt. Das fertige Modell sieht so aus:



Transformer

Zusammengefasst ist die Transformer-Architektur eine neuronale Netzwerkarchitektur, die Selbstaufmerksamkeitsmechanismen und Multi-Head-Attention verwendet, um die Bedeutung verschiedener Teile der Eingabe zu bewerten, was parallele Berechnungen ermöglicht und die Fähigkeit des Modells, komplexe Beziehungen zwischen Eingabeelementen zu erfassen, verbessert, wodurch es besser als frühere Architekturen in Natural-Language-Processing-Aufgaben ist. Ein Vision-Transformer (Vit) hingegen ist eine spezielle Art von Transformer-Architektur, die auf visuelle Daten ausgelegt ist und auf der Verwendung von Self-Attention Mechanismen basiert, um auf die Daten zu achten. Ein Vision-Transformer wird auch in diesem Modell verwendet, da er extra auf die Bildverarbeitung bzw. Klassifizierung spezialisiert ist.

3.4. Training

Für die ersten Trainings wurden folgende Parameter verwendet:

Batch-size:	100
Learning-Rate:	0.001
Encoder-Layer:	2
Head-Dim:	56
Mhsa-n-dim:	8
Multilayer-perceptron-dim:	64
Dropout-Rate:	0.0

Mit diesen Parametern und einer Ungleichheit in den Daten hat das Modell auf dem Test-Datensatz eine Accuracy/Genauigkeit von 96.45% erzielt.

Wie oben schon erwähnt, ist ein Problem, dass es überhaupt sehr wenige Daten gibt und die Klassen ungleich verteilt sind. Im Folgenden werden Lösungsansätze und deren Anwendung besprochen.

1. Um mit mehr Daten als vorhanden zu trainieren, kann man einen Trick anwenden, indem man den Validation-Datensatz weglässt und stattdessen diese Daten in den Trainings-Datensatz gibt. Um nun trotzdem noch eine Validation-Accuracy zu bekommen wird im Trainingsschritt n mit der n-ten Batch trainiert und mit der (n+1)-ten Batch validiert. Dies wird wiederholt, bis es keine Batches im Dataloader mehr gibt bzw. bis die Epoche zu Ende ist.

2. Klassen-Gewichtung: Hierbei werden die Verlustfunktionen im Training so modifiziert, dass sie die Beispiele der seltenen Klassen stärker gewichten.

3. F1-Score oder ähnliche Metriken anstelle von Genauigkeit/Accuracy als Optimierungsziel: Hierbei werden Metriken verwendet, die die Leistung des Modells für jede Klasse einzeln betrachten anstatt nur die Gesamtleistung.

3.5. Hyperparameter-Optimierung

Hyperparameter-Optimierung bezieht sich auf den Prozess, den geeigneten Wert von Hyperparametern für ein bestimmtes Modell oder eine bestimmte Aufgabe zu finden. Hyperparameter sind Parameter, die nicht während des Trainings gelernt werden, sondern vorher festgelegt werden müssen. Beispiele für Hyperparameter sind die Lernrate, die Anzahl der Schichten in einem neuronalen Netzwerk, die Anzahl der Neuronen pro Schicht usw. Die Wahl der richtigen Hyperparameter kann entscheidend für die Leistung eines Modells sein. Eine geeignete Hyperparameter-Optimierung kann dazu beitragen, dass das Modell besser auf die Daten generalisiert und bessere Ergebnisse erzielt. Durch eine Hyperparameter-Optimierung können also die Learning-Rate, Batch-Size, Epochen, Image-Size, pseudo-color-Funktion und die Parameter des Modells trainiert werden. Ein CNN hat kaum Parameter, da es schon bewiesene Voreinstellungen gibt, die die beste Leistung erzielen. Ein Transformer hingegen hat sehr viele Parameter, die alle angepasst werden müssen. Für die Hyperparameter-Optimierung wird Tune [5] von Ray zusammen mit PyTorch verwendet. Dazu wurde noch der ASHA-Scheduler von Tune verwendet, der als Parameter bekommt, welche Modelle als gut erachtet werden. Es soll die Accuracy des Validation-Datensatzes maximiert werden. Der ASHA-Scheduler bestimmt außerdem wie viele Epochen ein Modell noch weiter trainieren darf, wenn sich seine Validation-Accuracy nicht mehr erhöht. Dies wurde hier auf 5 Epochen festgelegt. Als maximale Epochen, die pro Modell trainiert werden dürfen, wurde 100 und als Anzahl von Versuchen 100 angegeben. Pro Versuch/Modell können

16 CPUs und 0.25 GPUs verwendet werden. Es gibt insgesamt 256 CPUs und 4 GPUs mit jeweils 40 GB Ram auf einer DGX A100 als Accelerator. Ein Durchlauf sieht dann z.B. so aus:

```
Using AsyncHyperBand: num_stopped=9
Bracket: Iter 80.000: None | Iter 40.000: None | Iter 20.000: None | Iter 10.000: 0.8150000013411045 | Iter 5.000: 0.5643749963492155
Resources requested: 256.0/256 CPUs, 4.0/4 GPUs, 0.0/907.7 GiB heap, 0.0/9.31 GiB objects (0.0/1.0 accelerator_type:A100)
Result logdir: /root/ray_results/train_specific_model_2023-01-04_17-14-45
Number of trials: 26/100 (1 PENDING, 16 RUNNING, 9 TERMINATED)
```

Trial name	status	loc	dropout_p	head_dim	lr	mhsa_n_dim	multilayer_perceptron_dim	loss	accuracy	training_iteration
train_specific_model_49457_00002	RUNNING	172.17.0.2:1980855	0.0208464	70	0.000739831	18	82	0.00944436	0.97625	8
train_specific_model_49457_00003	RUNNING	172.17.0.2:1980857	0.0472688	23	0.00322	9	62	0.105674	0.69	9
train_specific_model_49457_00006	RUNNING	172.17.0.2:1980866	0.00103407	16	0.000184449	20	85	0.0302192	0.94	10
train_specific_model_49457_00007	RUNNING	172.17.0.2:1980869	0.142084	25	0.00278219	19	39	0.0509626	0.86375	10
train_specific_model_49457_00011	RUNNING	172.17.0.2:1981479	0.061681	38	0.000353224	1	90	0.0302066	0.925	16
train_specific_model_49457_00012	RUNNING	172.17.0.2:1981770	0.00121482	5	0.0230545	4	68	0.118218	0.6425	16
train_specific_model_49457_00013	RUNNING	172.17.0.2:1982009	0.0133643	5	0.000804071	9	100	0.0164648	0.9575	10
train_specific_model_49457_00016	RUNNING	172.17.0.2:1985339	0.00355287	21	0.000152735	9	88	0.11712	0.65125	2
train_specific_model_49457_00017	RUNNING	172.17.0.2:1985623	0.00136186	60	0.0110667	3	37	0.152359	0.5175	3
train_specific_model_49457_00018	RUNNING	172.17.0.2:1985926	0.182301	76	0.00343426	1	71	0.129857	0.56375	2
train_specific_model_49457_00025	PENDING		0.00403002	18	0.00463495	23	41			
train_specific_model_49457_00000	TERMINATED	172.17.0.2:1980560	0.00166037	10	0.02898	7	74	0.152439	0.50125	5
train_specific_model_49457_00001	TERMINATED	172.17.0.2:1980853	0.0109121	61	0.0351925	14	63	0.152186	0.5125	5
train_specific_model_49457_00004	TERMINATED	172.17.0.2:1980860	0.0260585	37	0.00243806	23	4	0.0838985	0.76625	10
train_specific_model_49457_00005	TERMINATED	172.17.0.2:1980863	0.194711	35	0.0928532	9	53	0.244375	0.51125	5
train_specific_model_49457_00008	TERMINATED	172.17.0.2:1980999	0.0194683	51	0.0155323	4	19	0.153912	0.495	5
train_specific_model_49457_00009	TERMINATED	172.17.0.2:1981026	0.100997	81	0.00848695	11	16	0.126982	0.59375	10
train_specific_model_49457_00010	TERMINATED	172.17.0.2:1981259	0.0239895	71	0.00289325	4	29	0.144358	0.50875	5
train_specific_model_49457_00014	TERMINATED	172.17.0.2:1982160	0.00360729	19	0.00870417	16	62	0.131489	0.58625	10
train_specific_model_49457_00015	TERMINATED	172.17.0.2:1982291	0.033152	34	0.0353153	6	78	0.151875	0.50125	5

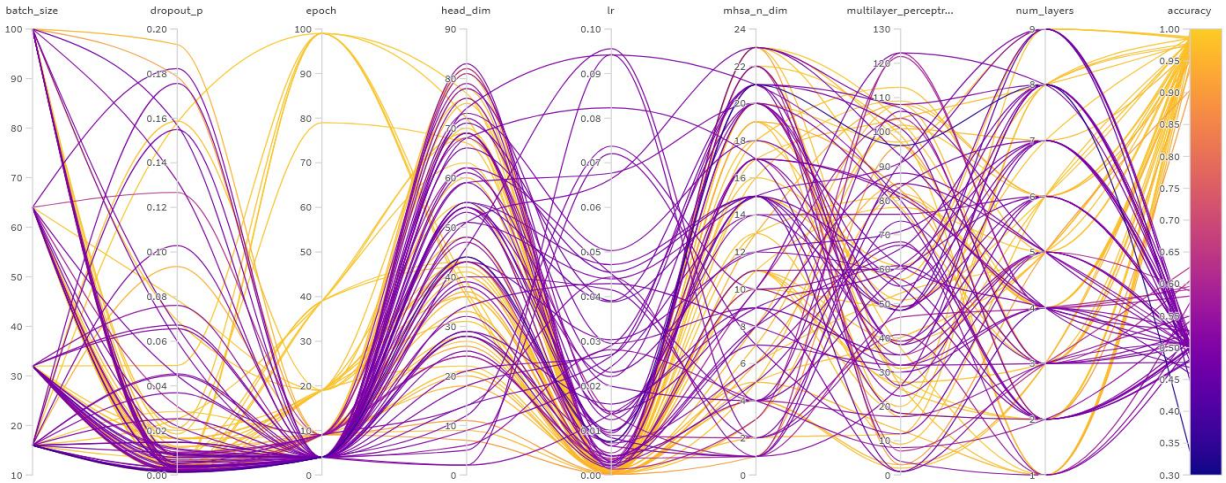
```
... 6 more trials not shown (6 RUNNING)
```

Da dies schwer zu erkennen ist, kann sich die Hyperparameter-Optimierung auch nochmal im Github-Repository angeschaut werden.

Aus der Hyperparameter-Optimierung ergaben sich die Parameter für das beste Modell mit einer Test-Accuracy von ca. 99, 54%:

Batch-size:	16
Learning-Rate:	0.0004526
Encoder-Layer:	2
Head-Dim:	39
Mhsa-n-dim:	20
Multilayer-perceptron-dim:	28
Dropout-Rate:	0.158

Um die Hyperparameter-Optimierung noch besser auszuwerten und zu sehen, was dem Modell sehr viel gebracht hat bzw. wenig, benutze ich Weights and Biases (W&B oder wandb) [6], welches es mir ermöglicht die Parameter-Beziehungen grafisch darzustellen:



Die Auswertung der Hyperparameter-Optimierung basiert auf dem selbst erstellten, interaktiven W&B-Report [15].

3.6. Vergleich

Da dieses Modell nun abgeschlossen ist, muss festgestellt werden, wie gut es im Vergleich zu anderen State-of-the-Art-(SotA)-Modellen performt. Als SotA-Modelle habe ich mir die Architekturen des ResNet [7] und die des EfficientNetV2 [8] rausgesucht. Beide werden im Folgenden erläutert:

ResNet

Man hat bei CNNs entdeckt, dass je mehr Layer es hat, desto besser performt es auch. Dies liegt daran, dass das Modell durch eine höhere Anzahl an convolutional Layern eine bessere Möglichkeit hat sich dem ihm zur Verfügung stehenden Raum anzupassen. Jedoch stagniert die Performance nach einer gewissen Anzahl an Layern bzw. das Modell verliert seine Fähigkeit zu generalisieren. Durch die Residual-Connections, die in einem Residual-Network (ResNet) verwendet werden, wird das Vanishing-Gradient-Problem gelöst. Das Problem beschreibt, dass die Gradienten, die am Output durch die Loss-Function berechnet werden, wenn sie an den ersten Eingabe-Layern ankommen, durch die Kettenregel fast Null geworden sind. Somit lernen die ersten Layer des Modells nicht/kaum. Bei einem ResNet hingegen können die Gradients durch die Residual-Layer fließen, um so auch die ersten Layer zu erreichen. Ein Standard-ResNet hat am Anfang einen Konvolution- und Pooling-Step, danach werden 4 Layer mit gleicher Architektur mit Residual-Connections durchlaufen. Jeder Layer besteht aus einer festgelegten Anzahl an Blöcken. Jeder Block besteht aus zwei Konvolutionen, die jeweils von einer Batch-Normalization und einer ReLU-Funktion gefolgt sind. Danach wird der Input dieses Blocks zum Output der letzten ReLU-Funktion gegeben, was die Residual-Connection darstellt. Auf den Output der Residual-Connection folgt noch eine ReLU-Funktion. Am Anfang jedes Layers gibt es einen Block, der den Input verkleinert. Dabei wird bei der ersten Konvolution dieses Blocks die kernel-size auf 1(statt auf 3) und der Konvolution-Schritt auf 2 (statt auf 1) gesetzt. Am Ende des gesamten Modells wird der Output geflattet und geht durch einen AdaptiveAveragePooling-Layer und einen Dense-Layer, der die Klassifizierung vornimmt. Diese Erklärung eines ResNets trifft auf ResNet18 und ResNet34 zu. Die Modelle ResNet50, ResNet100 und ResNet152 haben einen veränderten Aufbau. Sie haben auch 4 Layer, der einzige Unterschied liegt darin, dass sie keinen Block, sondern einen Bottleneck benutzen. Ein Bottleneck benutzt 3 Konvolutions, gefolgt von BatchNorm und ReLU und hat dann erst seine Residual-Connection, so wird auch ein größeres Modell weniger anfällig für Dimension-Explosionen oder dass sich die Genauigkeit verschlechtert. Die Parameter der jeweiligen ResNets sind hier dargestellt:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				

EfficientNet

EfficientNet verbessert die Architektur des ResNet durch den Einsatz von drei Techniken:

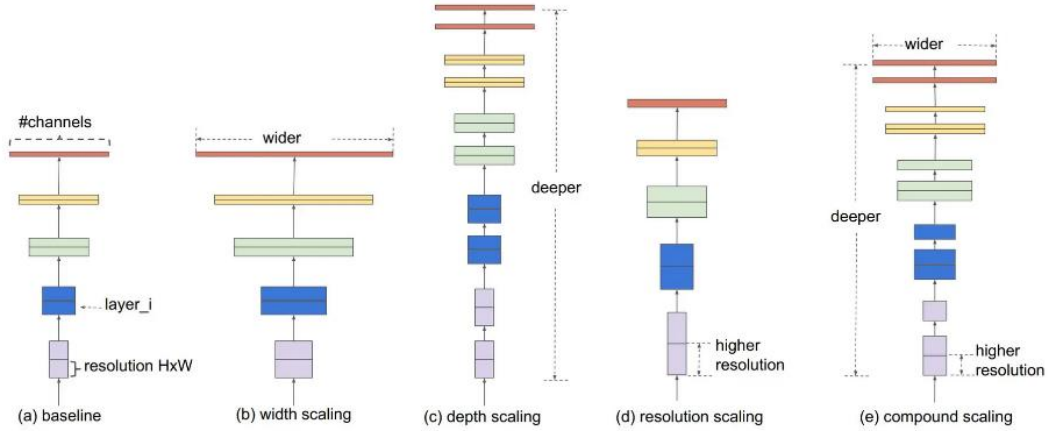
1. Skalierung von Netzwerkbreite, -tiefe und Auflösung: EfficientNet verwendet eine Methode zur Skalierung von Netzwerkbreite, Tiefe und Auflösung, die die Leistung des Netzwerks

automatisch an die verfügbaren Ressourcen anpasst. Dadurch kann EfficientNet sowohl auf Geräten mit geringer Rechenleistung als auch auf Hochleistungsrechnern verwendet werden.

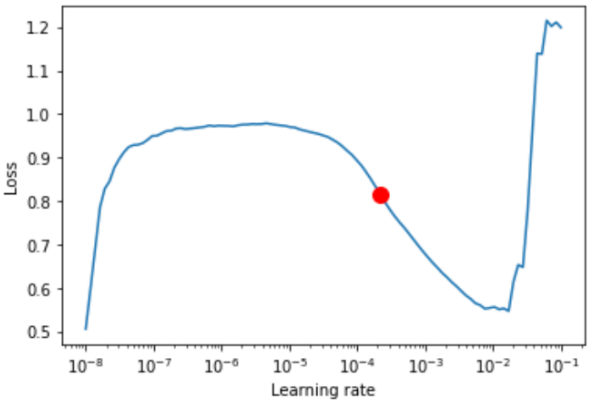
2. Effiziente Architektur: EfficientNet verwendet eine effiziente Architektur, die die Anzahl der benötigten Neuronen und die Anzahl der Berechnungen reduziert. Dies ermöglicht es EfficientNet, besser zu skalieren und zu generalisieren als herkömmliche Netzwerke.
3. AutoML: EfficientNet nutzt AutoML, um die besten Architektur- und Hyperparameterkombinationen automatisch zu finden. Dies ermöglicht es EfficientNet, die bestmögliche Leistung bei minimalem Aufwand zu erzielen.

Durch diese Techniken erreicht EfficientNet eine höhere Leistung als das ResNet bei vergleichbaren oder sogar geringeren Anforderungen an Ressourcen und Rechenleistung. Es hat in vielen Benchmarks und Wettbewerben die höchste Genauigkeit unter allen vergleichbaren Architekturen erreicht.

Hier sind die unterschiedlichen Skalierungs-Methoden dargestellt:



Da beide Modelle eine feste Architektur haben, müssen deren Parameter nicht mehr optimiert werden. Um die optimale Learning-Rate herauszufinden, benutze ich den Learning-Rate-Finder von PyTorch-Lightning. Ein Learning-Rate-Finder funktioniert, indem er automatisch verschiedene Lernraten testet, während er das Netzwerk trainiert. Er speichert dann den Verlust, den das Netzwerk während des Trainings für jede Lernrate erreicht. Der optimale Wert der Lernrate ist in der Regel der Wert, bei dem der Verlust am schnellsten fällt, bevor er wieder anfängt zu steigen. Dieser Punkt wird als "steilster Abfall"(Wendepunkt) bezeichnet.



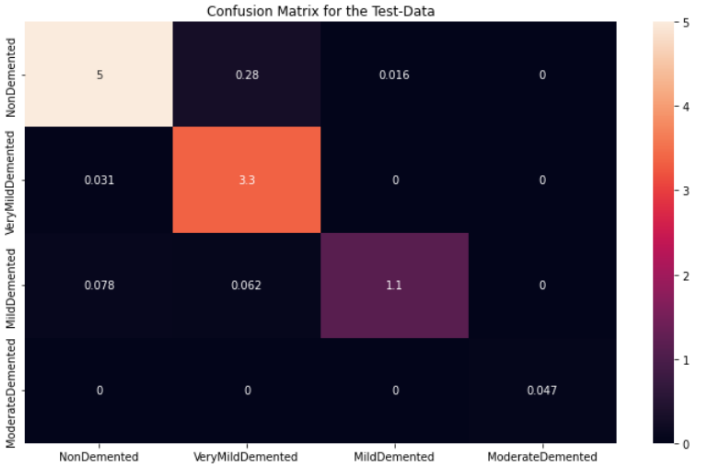
Nun kann man alle Modelle in einer Tabelle vergleichen:

Modelle:	CNN_T	ResNet18	ResNet34	ResNet50	ResNet100	ResNet152	EfficientNet-Small	EfficientNet-Medium	EfficientNet-Large	EfficientNet-ExtraLarge
Accuracy:	99,25%	77,97%	98,75%	98,91%	97,43%	93,75%	97,81%	96,15%	97,83%	98,92%

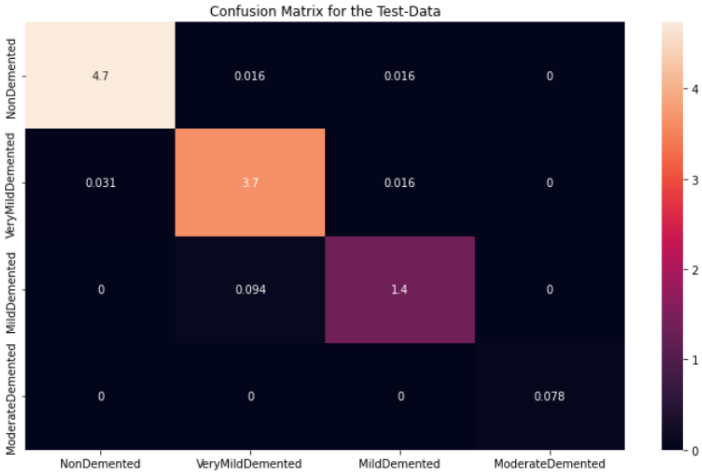
Da die Klassen ungleich verteilt sind, reicht es nicht nur auf die Accuracy zu achten, sondern es muss auch auf Precision und Recall geachtet werden. Precision und Recall sind Metriken, die verwendet werden, um die Leistung eines Klassifizierungsmodells zu messen. Precision gibt an, wie viele der von dem Modell als positiv klassifizierten Beispiele tatsächlich positiv sind, während Recall angibt, wie

viele der tatsächlich positiven Beispiele von dem Modell als positiv klassifiziert werden. In einem multi-class Klassifizierungsproblem, gibt es mehrere Klassen und ein Beispiel kann nur einer Klasse zugeordnet werden. Um die Leistung des Modells für jede Klasse einzeln zu messen, kann man eine Confusion-Matrix verwenden. Eine Confusion-Matrix ist eine Tabelle, die zeigt, wie viele Beispiele von jeder Klasse von dem Modell richtig und falsch klassifiziert wurden. Aus der Confusion-Matrix kann man dann den Recall und Precision für jede Klasse einzeln berechnen. Es ist jedoch nicht möglich, einen einzigen Wert für Precision und Recall für das gesamte multi-class Klassifizierungsproblem zu berechnen, da jede Klasse möglicherweise unterschiedliche Leistungen aufweist. Daher ist es wichtig, die Ergebnisse für jede Klasse einzeln zu betrachten, um ein vollständigeres Verständnis der Leistung des Modells zu erhalten. Dazu wird von jeder Architektur das beste Modell benutzen, also der CNN_T, ResNet50 und EfficientNet-Large. Accuracy, Prediction und Recall für jede Klasse einzeln werden in der Auswertung nochmal näher betrachtet.

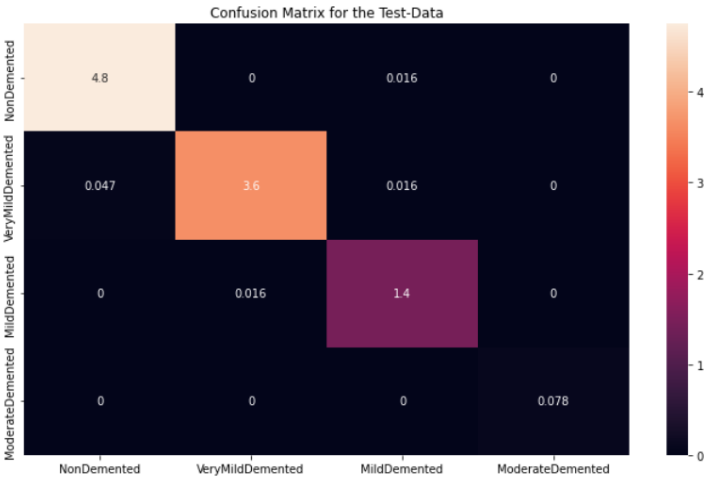
EfficientNet-Large



ResNet50



CNN-Transformer



Die Diagonale in der Mitte der Matrizen sind Scans, die als richtig positiv (TP) erkannt wurden. Bei allen fällt auf, dass die Klasse ModerateDemented, also die Patienten, die am schwersten an Alzheimer erkrankt sind, von allen Modellen 100% richtig klassifiziert wurde. Das EfficientNet-Large hatte vor allem Probleme damit Scans der Klassen NonDemented und VeryMildDemented zu unterscheiden. Beide oben genannten Klassen werden auch manchmal mit der Klasse MildDemented verwechselt. Bei dem ResNet50 ist es ähnlich, nur dass durch die höhere Genauigkeit dieses Modells insgesamt weniger Fehler passieren. Zudem ist die Verwechslung zwischen den Klassen NonDemented und VeryMildDemented nicht so hoch. Der CNN-Transformer macht die gleichen Fehler wie das ResNet50, nur da er eine noch höhere Genauigkeit hat, noch weniger. Im Vergleich zum ResNet50 macht er nicht so viele Fehler bei der Unterscheidung der Klassen VeryMildDemented und MildDemented. Es ist schwieriger für die Modelle die Klassen NonDemented, VeryMildDemented und MildDemented auseinanderzuhalten, da diese zueinander geringere Unterschiede aufweisen.

4. Auswertung
4.1. Auswertung Modelle

Der CNN-T hat von allen Modellen am besten performt, was man damit erklären kann, dass das CNN erst lokale Auffälligkeiten herausfiltert und präzise Informationen sammelt. Diese lokalen Auffälligkeiten werden dann an den Transformer gegeben, der diese verarbeitet und einen globalen Bezug herstellt. Außerdem hat der CNN-T aufgrund der Aufgabenstellung einen Vorteil im Vergleich zu den anderen Modellen. Die Schwierigkeit der Aufgabe bezieht sich nämlich darauf, kleine Auffälligkeiten in der Gehirnmasse festzustellen und damit zu vergleichen, in welchen Hirnregionen diese auftreten. Die Hirnregionen sind in allen MRT-Scans am gleichen Ort, da alle Bilder auf die gleiche Größe skaliert und zentriert wurden. Somit muss das Modell nicht lernen, die Hirnregionen für jeden Scan erst einmal neu zu entdecken, sondern kann sich gleich auf das Suchen nach Auffälligkeiten konzentrieren. Zudem haben die anderen Modelle es schwerer, einen globalen Zusammenhang zwischen diesen sehr kleinen Details herzustellen, da sie keine globale Komponente, wie einen Transformer haben. Das CNN-T leidet, wie die anderen Modelle auch, nicht unter dem Vanishing-Gradient-Problem, da der Transformer und das CNN auch Skip-Connections haben und somit die Gradienten auch zu den ersten Layern gelangen können.

Betrachtet man die unterschiedlichen Größen der ResNets (mit Bottleneck, also 50, 100, 152) und EfficientNetV2s, so fällt auf, dass je größer die Modelle werden, also je mehr Ressourcen sie zur Verfügung haben, desto schlechter performen sie. Dies ist auch damit zu erklären, dass diese Modelle darauf ausgelegt sind, erst einmal die Klasse in dem Bild zu finden und sie dann zu klassifizieren, was sie bei dieser Aufgabe aber gar nicht müssen. Außerdem bestehen sie auch nur aus CNNs, die für diese Aufgabe aber gar nicht so hilfreich sind. Gibt man nun noch mehr CNNs hinzu, so vergrößern sich zwar die Ressourcen des Modells, diese werden aber gar nicht gebraucht, da sie trotzdem nur lokale Auffälligkeiten finden. Was passiert ist also, dass das Modell sehr viele Layer gleichzeitig trainieren muss, welche die Performance aber nicht verbessern. Deswegen performt der CNN-T auch so gut, er ist nicht zu groß und muss nicht unnötige Parameter lernen.

Jedoch performen alle Modelle sehr gut, was auch dafür spricht, dass die Aufgabe an sich erst einmal einfach ist, man kann aber nur eine sehr gute Performance erreicht, wenn man ganz genau auf die Auffälligkeiten aufpasst.

EfficientNet-Large

Classification Report for the the Test-Data

	precision	recall	f1-score	support
NonDemented	0.98	0.94	0.96	339
VeryMildDemented	0.91	0.99	0.95	216
MildDemented	0.99	0.89	0.94	82
ModerateDemented	1.00	1.00	1.00	3
accuracy			0.95	640
macro avg	0.97	0.96	0.96	640
weighted avg	0.96	0.95	0.95	640

ResNet50

Classification Report for the the Test-Data

	precision	recall	f1-score	support
NonDemented	0.99	0.99	0.99	305
VeryMildDemented	0.97	0.99	0.98	237
MildDemented	0.98	0.94	0.96	93
ModerateDemented	1.00	1.00	1.00	5
accuracy			0.98	640
macro avg	0.99	0.98	0.98	640
weighted avg	0.98	0.98	0.98	640

CNN-Transformer

Classification Report for the the Test-Data

	precision	recall	f1-score	support
NonDemented	0.99	1.00	0.99	305
VeryMildDemented	1.00	0.98	0.99	237
MildDemented	0.98	0.99	0.98	93
ModerateDemented	1.00	1.00	1.00	5
accuracy			0.99	640
macro avg	0.99	0.99	0.99	640
weighted avg	0.99	0.99	0.99	640

Bei dem EfficientNet-Large ist der Recall bei den Klassen NonDemented und MildDemented am niedrigsten. Es wurden also viele positive Scans dieser Klassen von dem Modell falsch klassifiziert. Außerdem ist die Precision bei der Klasse VeryMildDemented am niedrigsten, was besagt, dass relativ wenige der als positiv klassifizierten Scans wirklich der Klasse VeryMildDemented

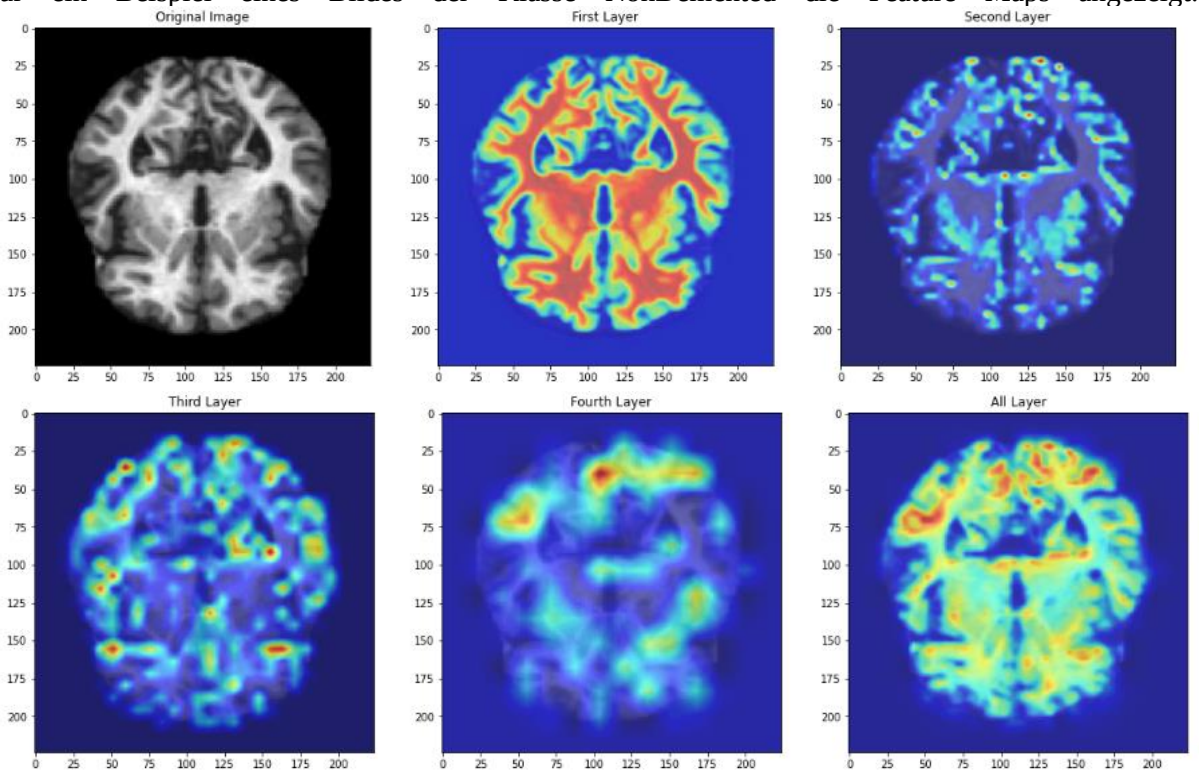
zuzuordnen waren. Das ResNet50 hat bei allen Klassen circa die gleichen Precision- und Recall-Werte, nur der Recall für die Klasse MildDemented ist sehr niedrig. Viele der Scans dieser Klasse wurden also falsch als andere Klassen gelabelt. Der CNN-Transformer hat auch sehr gute Precision- und Recall-Werte für alle Klassen. Der Recall der Klasse VeryMildDemented ist jedoch am niedrigsten, was dafür spricht, dass viele dieser Klassen als NonDemented und vor allem VeryMildDemented klassifiziert wurden. Dies erklärt auch die Precision für die Klasse MildDemented.

4.2. Explainability

Die Explainability (Erklärbarkeit) bezieht sich auf die Fähigkeit, die Entscheidungen und Prozesse eines künstlichen Intelligenz-Systems (KI) zu verstehen und erklären zu können. Es geht darum, die Gründe für die Entscheidungen eines KI-Systems zu verstehen und zu erklären, wie es zu seinen Entscheidungen kommt. Explainability ist wichtig, um Vertrauen in das KI-System aufzubauen und seine Entscheidungen zu überprüfen und zu validieren.

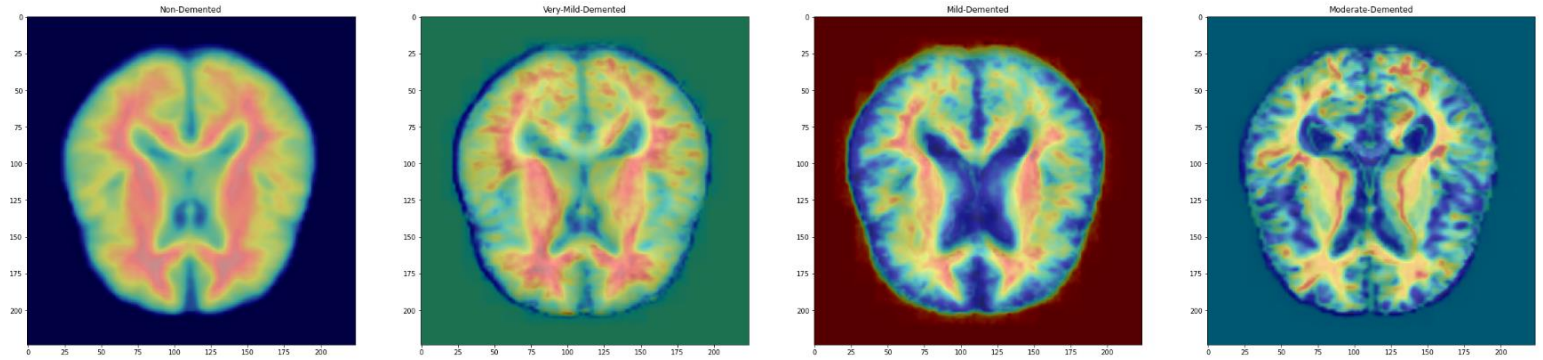
Es gibt verschiedene Methoden, um die Explainability von KI-Systemen zu erreichen, wie zum Beispiel die Verwendung von Erklärbarkeits-Tools, die die Entscheidungen des KI-Systems visualisieren wie z.B. Grad-Cam [9], oder die Verwendung von Methoden wie LIME (Local Interpretable Model-agnostic Explanations) [10] und SHAP (SHapley Additive exPlanations) [11] die die Beiträge der einzelnen Merkmale zur Entscheidung des KI-Systems erklären. Im Folgenden wird Grad-Cam als Auswertungsmethode erklärt und verwendet.

Die Grundidee hinter Grad-CAM (Gradient-weighted Class Activation Mapping) besteht darin, den Gradienten der Ausgabe des Modells in Bezug auf die Feature-Maps der letzten konvolutionalen Schicht zu berechnen. Dieser Gradient repräsentiert, wie sehr jede Feature-Map zur endgültigen Ausgabe beiträgt. Anschließend wird eine gewichtete Summe der Feature-Maps berechnet, wobei das Gewicht für jede Feature-Map der entsprechende Gradientenwert ist. Dadurch entsteht eine Heatmap, die die Bereiche des Eingabebildes hervorhebt, die am wichtigsten für die Vorhersage des Modells sind. Hierzu wird sich nur der CNN-Transformer angeguckt, da dieser die beste Performance hatte. Es werden hier für ein Beispiel eines Bildes der Klasse NonDemented die Feature Maps angezeigt:



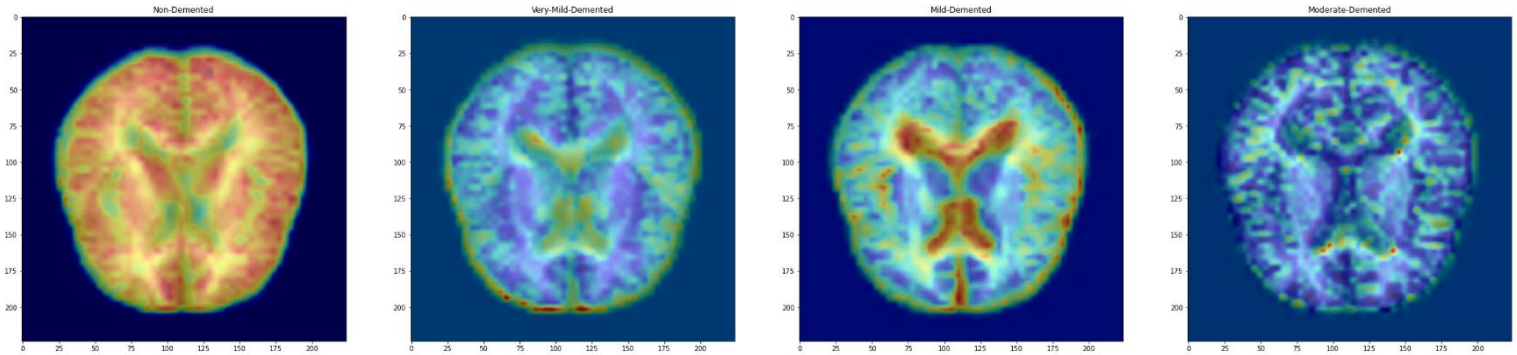
Was man bei diesem Beispiel sieht, ist dass der erste Layer auf den Bereich der weißen Substanz achtet, welcher bei diesem nicht an Alzheimer erkrankten Patienten vollständig aussieht, da keine Gehirnmasse fehlt. Worauf Layer zwei bis vier ihre Aufmerksamkeit legen, lässt sich hier schwer feststellen. Legt man alle Layer übereinander, so fällt jedoch auf, dass das Modell vor allem einen Bereich auffällig findet, wo die Okzipitallappen und Temporallappen liegen. Dieser Bereich ist also unter anderem für die Verarbeitung visueller Informationen zuständig. Da die Auswertung so sehr anstrengend wäre und von Patient zu Patient sich unterscheiden würde, werden für jede Klasse und jeden Layer im Folgenden die Scans übereinander gelegt, um so besser zu erkennen, worauf sich jeder Layer spezialisiert hat. Dies ist möglich, da die Bilder durch die Vorverarbeitung zentriert sind und die Kopf- bzw. Gehirnformen der Menschen ähnlich sind.

Convolutional Layer 1

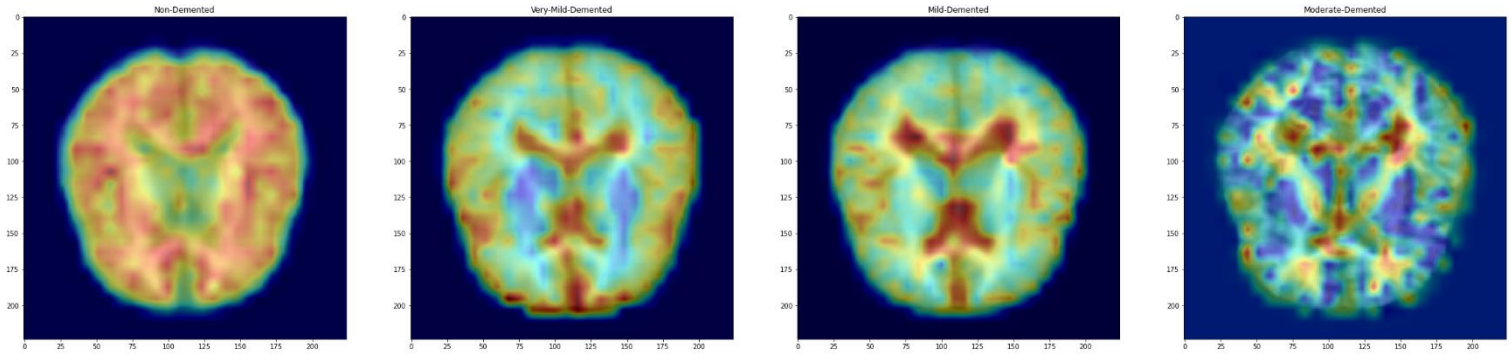


Der erste Layer bildet die weiße Substanz, hier in rot zu sehen, nach. Dabei achtet er darauf, wie stark diese Strukturen ausgeprägt sind. Bei der Klasse Very-Mild-Demented kann man schon sehen, dass die rote Farbe nicht mehr so stark ist. Daraus könnte man schlussfolgern, dass das Zellsterben von Nervenzellen und Synapsen schon eingesetzt hat, aber noch genügend Nervenzellen nicht betroffen sind. Bei der Klasse MildDemented nimmt die rote Farbe nicht nur weiter ab, sondern die Regionen, die bei nicht dementen Patienten rot markiert waren, verringern sich auch. Diese Regionen sind hier vor allem die Okzipitallappen und die Temporallappen. Funktionen der Temporallappen sind die Geräuschwahrnehmung, Sprachverständnis, das visuelle und faktische Gedächtnis, sowie die emotionale Verarbeitung. Da all diese Funktionen auch bei an Alzheimer erkrankten Personen eingeschränkt sind, wie z.B. der Gedächtnisverlust und das Verschlechtern der sprachlichen Fähigkeiten, lässt sich hier ein Zusammenhang herstellen zwischen dem Modell und der Alzheimer-Diagnostik. Bei der Klasse ModerateDemented sieht man eine Abnahme der Nervenzellen im gesamten Gehirn, vor allem aber in äußeren Regionen. Im Unterschied zu der Klasse MildDemented, hat auch die Farbe in den mittleren Regionen abgenommen. Dort liegt der Thalamus, welcher sensorische und motorische Funktionen übernimmt. Daher könnte man also deuten, warum bei einer Alzheimer-Erkrankung zuerst ein Gedächtnisverlust vorkommt und erst im weiteren Verlauf motorische und sensorische Funktionen eingeschränkt werden, wie z.B. am Zittern von Händen zu sehen ist.

Convolutional Layer 2

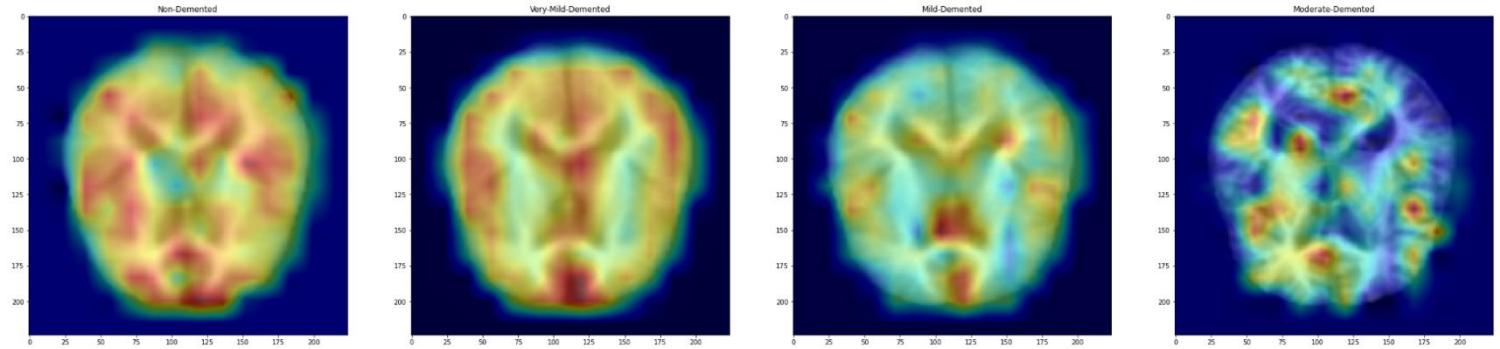


Convolutional Layer 3

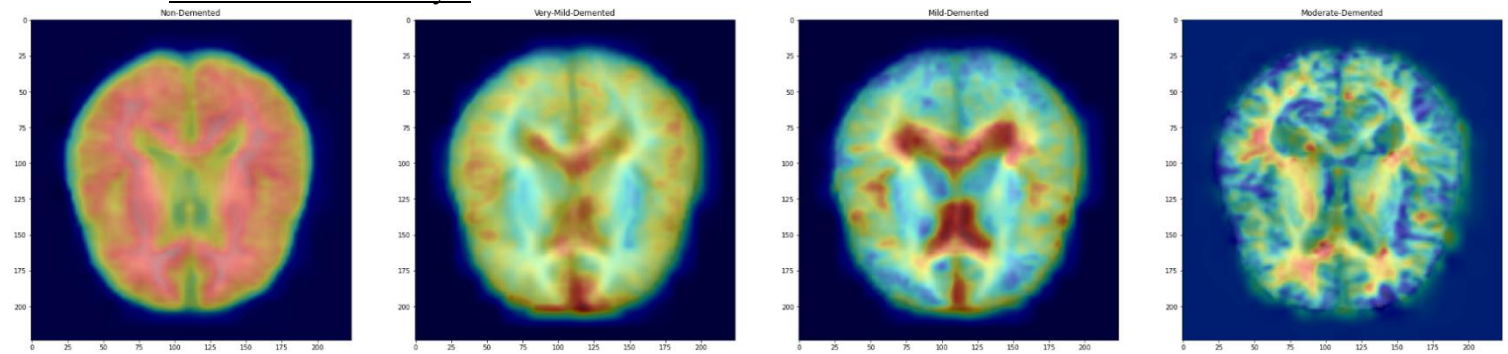


Bei der Klasse NonDemented wird in den Layern zwei bis vier, die gesamte Struktur des Gehirns abgedeckt. Der Interpretationsansatz wäre hier, dass alle Regionen deswegen gesund sind. Layer zwei und drei beschäftigen sich vor allem mit den Regionen rund um das Ventrikel-System, welches das Gehirn mit Nahrung versorgt und Stoffwechsel-Abbauprodukte abtransportiert. Dies könnte für eine Belastung dieses Systems sprechen, da die Beta-Amyloid-Plaques nicht mehr abtransportiert werden können und so die Funktion des Systems gestört ist.

Convolutional Layer 4



Alle Convolutional Layer

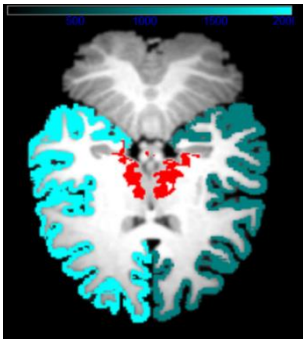


Legt man alle Layer übereinander, so ist vor allem bei der Klasse VeryMildDemented zu sehen, dass die Region, wo die Frontallappen, genauer der orbitofrontale Cortex liegen. Der orbitofrontale Cortex spielt eine entscheidende Rolle bei der Entscheidungsfindung und der Überwachung sozialer Interaktionen. Die Abnahme dieser Funktionen ist auch ein typisch für Patienten, die an Alzheimer erkrankt sind.

Die App, welche sich auch im GitHub-Repository befindet, kann Gehirn-Atlasse anzeigen. Mithilfe des "Structural Similarity Index" (SSIM) kann ein Scan eines Alzheimer-Patienten in die richtige Axial-ebene des Gehirn-Atlas eingeordnet werden, um so ein besseres dreidimensionales Verständnis zu bekommen. Zudem kann nun der Scan auch für eine bessere Auswertung/Diagnostik segmentiert bzw. gelabelt werden. Außerdem ist der CNN_T in die App integriert und es können somit die Feature-Maps bzw. Auffälligkeiten des Scans auch dreidimensional und zweidimensional angezeigt werden. Eine genauere Beschreibung, wie die App funktioniert, befindet sich auch nochmal im GitHub-Repository.

5. Ergebnisdiskussion

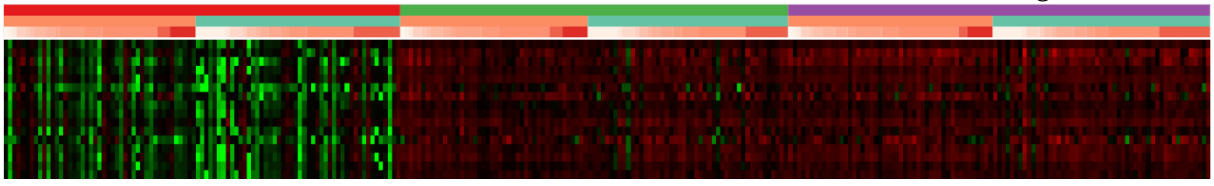
Da ich leider noch keinem Institut angehöre, ist die Datenlage relativ klein. Zudem hat man auch keine kompletten MRTs, sondern nur die Scans als Ausschnitte. Außerdem gibt es keine weiteren Informationen, außer die Klasse zu den Scans, was die Auswertung dieser erschwert. Damit sind Informationen gemeint, wie z.B. die Position der Scans in der Axialebene. Dies würde die Segmentierung erleichtern, konnte jedoch dank der Verwendung von Algorithmen gut übergangen werden. Dass die vorgestellten Augmentation-Methoden keine Verbesserung erzielen liegt daran, dass die Modelle für diese Aufgabe der Klassifizierung sehr genau aufpassen müssen, wo und ob sie Auffälligkeiten finden. Anscheinend hindern die Augmentation-Methoden sie eher daran, da so die Gehirnregionen von Scan zu Scan ihre Position ändern. Trotzdem hat das Trainieren der Modelle sehr gut und ohne Schwierigkeiten funktioniert. Die Hyperparameter-Optimierung war schwierig zu implementieren, da es am Anfang ein Problem mit der Speichergröße der Daten gab, was jedoch gelöst werden konnte. Die Erklärbarkeit der Modelle zeigt eindeutig die Aufmerksamkeit der Modelle an, jedoch kann man diese verschieden interpretieren. Ich denke die Interpretation ist hier aber sehr gut gelungen. Mit weiteren Segmentierungsmethoden, wie z.B. Deep-Learning-basierten Methoden, wurde auch experimentiert. Eine schlechte Datenlage hat dies jedoch erschwert. Die App funktioniert sehr flüssig und bietet eine übersichtliche Methode zur Diagnostik. Das automatische Zuweisen eines Scans mit Grad-Cam-Map zu einer Position in einem Atlas, welche danach automatisch gelabelt wurde, funktioniert sehr gut. Statt dem SSIM könnte man noch bessere Methoden zum Vergleichen von Ebenen und Scan verwenden, wie den PSNR-HVS-M [12]. Es ist also möglich einen Scan in die App zu geben, sodass dieser automatisch klassifiziert und gelabelt wird und zudem kann man sich noch die Feature-Map des Modells für diesen Scan anzeigen lassen. Es wurde zum Verstehen der Scans der MRIcon Brain Viewer verwendet. Ein Beispiel einer, noch selbst zugewiesenen Position auf einen Scan, der dann gelabelt wird, lässt sich hier sehen.



6. Fazit und Ausblick

Es wurden also Modelle entwickelt und miteinander verglichen, die eine sehr genaue Klassifizierung vornehmen können. Des Weiteren kann die Aufmerksamkeit dieser Modelle angezeigt und interpretiert werden und ist durch eine App frei und leicht zugänglich. Daraus erhoffe ich mir, Ärzten die Erkennung von Alzheimer-Patienten zu erleichtern. Da das Modell auch seine Aufmerksamkeit anzeigt, kriegen die Ärzte so nicht nur eine klassifizierte Klasse angezeigt, sondern sie können auch nachvollziehen, warum das Modell zu diesem Entschluss gekommen ist und so können sie selbst ein fundiertes Urteil bilden.

Ich würde gerne noch ein Modell zur Klassifizierung von Genexpressionsbeziehungen entwickeln. Dazu werde ich eine selbstgebautes rekurrentes Netzwerk nehmen. Dieses besteht aus einer Legendre-MemoryUnit (SotA-Modell) [13] bei der die Memory States hierarchisch miteinander verbunden sind. Ich habe dieses Modell genommen/gebaut, da eine Legendre-MemoryUnit sehr schnell und effizient ist. Zudem kann die Sequence-Length dieses Modells sehr hoch sein und das Modell trotzdem noch genau. Durch die hierarchische Struktur werden nicht alle Memory States bei jedem Time Step geupdatet, und so können wichtige hierarchische Informationen hergestellt und länger behalten werden. Dies ist optimal für die langen Genexpressions-Beziehungen. Beim Preprocessing wird dabei der K-Means-Algorithmus zur Gruppierung der Features genutzt. Eine Genexpressions-Beziehung kann so dargestellt werden. Zudem stehe ich in Kontakt mit Professoren des Institut Leidens und des HPIs, um so meine Datenlage im Bereich der MRT-Scans zu verbessern und noch mehr Kenntnisse über die Forschung zu erhalten.



7. Literaturverzeichnis

[1] https://adni.loni.usc.edu/

[2] https://portal.brain-map.org/

[3] https://pytorch.org/

[4] https://www.pytorchlightning.ai/

[5] https://docs.ray.io/en/latest/tune/getting-started.html

[6] https://wandb.ai/site

[7] https://arxiv.org/pdf/1512.03385.pdf

[8] https://arxiv.org/pdf/2104.00298.pdf

[9] https://arxiv.org/pdf/1610.02391.pdf

[10] https://christophm.github.io/interpretable-ml-book/lime.html

[11] https://shap.readthedocs.io/en/latest/

[12] https://www.ponomarenko.info/psnrhvsm.htm

[13] https://proceedings.neurips.cc/paper/2019/file/952285b9b7e7a1be5aa7849f32ffff05-Paper.pdf

[14] GitHub-Repository: https://github.com/Niklas1225/JuFo2023

[15] W&B-Report: https://api.wandb.ai/links/nbennewiz/j8svnd3w

8. Unterstützer

Frau Dr. Angela Köhler- Krützfeldt, Unterstützung bei Korrektur der schriftlichen Arbeit