FREIE UNIVERSITÄT BERLIN

BACHELORARBEIT

# "Explainable Artificial Intelligence" für Stammzellmodelle des Leigh-Syndroms

*Autor:*
Maximilian OTTO

*Betreuer:*
Dr. Jakob J. METZGER

*Diese Thesis dient dem Erlangen des*
*Bachlor of Science (Bioinformatik)*

.

*am*

Fachbereich Mathematik und Informatik
Bioinformatik

27. September 2022

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Kapitel 1

# Einführung

## 1.1 Einleitung

In der biologischen Forschung gibt es zunehmend Anwendungsgebiete und Verwendungen künstlicher neuronaler Netze. Diese angewandten Algorithmen, deren allgemeine Architektur für eine Vielzahl von Fragestellungen verwendet werden kann, erzielen nach einer Anpassung auf teils sehr spezifische Probleme und Datensätze oftmals sehr gute Ergebnisse. Beispielsweise können diese neuronalen Netzwerke [**Reference1**] nach entsprechender Trainingszeit sogar auf hoch spezifischen Datensätzen noch immer überraschend gute Prädiktionen für Klassifizierungsprobleme liefern und die ihnen übergebenen Daten korrekt einordnen, oder neue Bilder generieren, welche dem vorgegebenen Datensatz möglichst ähnlich sind um Ungleichgewichte in Vergleichsgruppen für weitere statistische Anwendungen zu beheben.
Die dafür verwendeten Algorithmen basieren jedoch oftmals auf der Grundlage von sehr großen Datensätzen. Da die Datenerhebung im biologischen oder medizinischen Kontext oftmals langwierig oder teuer ist, gilt es, die Stabilität und Genauigkeit dieser Algorithmen weiter zu optimieren, sodass selbst auf deutlich kleineren Datensätzen die Anwendbarkeit dieser Methoden gewährleistet bleibt.
Wenn es jedoch darum geht, innerhalb eines Klassifizierungsproblems herauszufinden, weshalb ein solcher Algorithmus in der Lage ist, solch gute Vorhersagen zu treffen, bzw. woran sie die Unterschiede und Merkmale innerhalb eines Datensatzes identifiziert, erscheint die Erklärung und Interpretation oftmals unintuitiv und komplex. Um das Verständnis für solche Klassifikationen einfacher zu gestalten, arbeiten viele Forscher*innen daran, immer neue Algorithmen mit möglichst genauen und zugleich einfach zu interpretierenden Ergebnissen und Visualisierungen zu etablieren. Daher gibt es mittlerweile eine Bandbreite an Algorithmen und Verfahren, welche sich dem Gebiet der "Explainable Artificial Intelligence" (XAI) zuordnen.
Neben vielen anderen Anwendungsmöglichkeiten, bietet sich eine interdisziplinäre Kooperation auf biologischen und medizinischen Daten an, um bspw. anhand eines Bildes der Retina eines Menschen diabetes-assoziierte Krankheiten zu erkennen [18], oder die Areale eines Röntgenbildes zu lokalisieren, in denen Kariesläsionen vorhanden sind [2]. Diese folgende Arbeit befasst sich mit der Klassifizierung von Bildern, in unserem Falle Mikrospkopaufnahmen von Zellkulturen, und bietet eine Möglichkeit, Bilder zu klassifizieren und zugleich den Unterschied zwischen einem Eingabebild zu seiner komplementären Klasse mittels der Generierung von *Counterfactuals* [23] zu visualisieren.

## 1.2 Projekt und Ziele

Die AG Metzger am Max-Delbrück-Center steht in Kooperation mit der AG Prigione des Uniklinikum Düsseldorf, welche die Leitung des Projektes "CureMILS" koordiniert. Dabei geht es um die Untersuchung von maternal vererbten, mitochondrialen Krankheiten, insbesondere des Leigh-Syndroms. Sie haben eine Reihe von *in vitro* Modellen aus Stammzellen gezüchtet, um zu untersuchen, wie genau sich diese Krankheit verhält und ob es einen Wirkstoff gibt, der die Symptome des Leigh-Syndroms reduzieren, und bspw. das mitochondriale Membranpotential oder andere Faktoren wie die Morphologie der Zellkulturen ausreichend beeinflussen kann. Dafür sollen im späteren Verlauf des Projekts mittels Hochdurchsatz-Screening mehrere tausend Medikamente an den Zellmodellen getestet und deren Wirkung analysiert werden. Für die Unterstützung der Analyse sollen auch neuronale Netze daran anknüpfend zum Einsatz kommen. Dadurch können die Zellkulturen als "gesund" oder "erkrankt" klassifiziert werden. Zellmodelle, die an dem Leigh-Syndrom erkrankt sind und aufgrund eines Wirkstoffes nicht weiter als solche erkannt werden, könnten somit Auskunft über die Wirksamkeit des hinzugegebenen Stoffes liefern.
Neben der Differenzierbarkeit der Experimente ist ein allgemeines Verständnis für die Unterschiede der erhobenen Daten wichtig, um weitere Nachforschungen zielgerichtet durchführen zu können. Ziel dieser Arbeit ist es, herauszufinden, ob die Unterschiede, die von einem neuronalen Netzwerk für die Entscheidung bei der Klassifizierung von Zellmodellen verwendet werden, durch die Anwendung des selben Netzwerkes aufzeigbar und erklärbar sind. Dafür kommen Architekturen mit generierenden, sowie klassifizierenden Eigenschaften zum Einsatz, um die Daten so zu modulieren, dass die Unterschiede zwischen den Klassen visuell darstellbar sind. Die daraus resultierenden Bilder sollen die minimalen Änderungen beinhalten, welche nötig wären, um das Bild einer anderen Klasse zuzuweisen. Damit sollen unter anderem Biologen dabei unterstützt werden, bspw. Phänotypen oder Auswirkungen von Medikamenten mittels Techniken des maschinellen Lernens in der Analyse der Zellmodelle nicht nur statistisch, sondern auch mittels visueller Repräsentation identifizieren zu können. Ebenso können zeitgleich sich irregulär verhaltende Zelllinien oder -modelle durch eine Analyse der Repräsentation innerhalb des neuronalen Netzwerks ausfindig gemacht werden, um die Versuchsreihe zu optimieren.

## 1.3 Verwandte Arbeiten

Für die Visualisierung der wichtigen Areale eines Bildes gibt es viele Methoden. Eine dieser Attributionsmethoden, neben der weit verbreiteten okklusionsbasierten Attributionsmethode, die ebenfalls vielversprechende Visualisierungen produziert, ist *Information Bottlenecks for Attribution* [19]. Diese könnte ebenfalls angewandt werden, um Unterscheide zwischen Klassen von Zellkulturen zu extrahieren. Die Wirkungsweise von Medikamenten durch die Klassifizierung von *Micropattern*, auf denen diese angewandt wurden, zu bestimmen, wurde bereits erfolgreich in [15] durchgeführt. Somit konnte zwischen gesunden und erkrankten Organoiden von reproduzierbarer Form, den *Micropattern*, unterschieden werden. Die Wirkungsweise eines Medikaments konnte durch die Klassifizierung der Organoide effizient ermittelt werden. Die im Zuge dieser Arbeit implementierten Quellcodes basieren auf invertierbaren, residualen Netzwerken [3] für gute Klassifizierungsergebnisse, und wenden auf diese Netzwerkarchitektur eine Methode zur effizienten Bestimmung von *Counterfactuals* [10] an.

# Kapitel 2

# Theoretischer Hintergrund

## 2.1 Convolutional Neural Networks

Klassischerweise besteht ein künstliches neuronales Netzwerk aus Neuronen, die in beliebig vielen Ebenen mit beliebiger Anzahl vorliegen. Die Verschaltungen zwischen den Neuronen und die Neuronen selbst bestehen aus einer vereinfachten mathematischen Repräsentation der Funktionsweise von biologischen Neuronen. Die Eingabewerte jedes Neurons einer Schicht werden durch die Verknüpfungen zu den Neuronen der vorherigen Schicht bestimmt. Jede Verknüpfung besitzt eine individuelle Gewichtung, welche die Ausgabe des vorangegangen Neurons modifiziert. Durch die Summe aller modifizierten Eingaben wird die Aktivierung eines Neurons der nächsten Ebene festgelegt. Somit lässt sich eine Ebene als als eine transformierende Funktion ihrer Eingabe verstehen.
Um Merkmale aus einem Eingabebild, welches in unserem Fall dreidimensional vorliegt, zu extrahieren, und diese anschließend klassifizieren zu können, wird oftmals ein mehrschichtiges, neuronales Netz verwendet, auch genannt *Convolutional Neural Network* (CNN). Die Neuronen innerhalb dieses Netzwerks sind für diesen Zweck entsprechend angeordnet und ihre Aktivität wird dabei durch eine diskrete Faltungsoperation auf der Eingabe $x(t)$, zusammen mit einem darauf angewandten Kernel, der die Gewichte $w(a)$ enthält, bestimmt. Der Kernel lässt sich als Matrix darstellen, welche über die Eingabe gelegt wird und mithilfe der in ihr enthaltenen Gewichte die Merkmale der Eingabe extrahiert. Da der Kernel über das gesamte Bild wandert, führen die Gewichte zu einer ortsunabhängigen Betrachtung und verarbeiten lediglich die lokale Umgebung, welche durch die Größe des Kernels bestimmt wird. Eine diskrete Faltungsoperation lässt sich nach [7] also als

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(t-a)w(a) \tag{2.1}$$

definieren. Für zweidimensionale Daten kann dies zu

$$S(i,j) = (K * I)(i,j) = \sum_{m}\sum_{n} I(i-m, j-n)K(m,n) \tag{2.2}$$

erweitert werden, wobei $I$ eine zweidimensionale Eingabe ist, und $K$ ein zweidimensionaler Kernel.

Eine Konkatenation mehrerer Faltungsoperationen für die jeweils einzelnen Farbkanäle der Eingabematrix kann somit als dreidimensionaler Filter angesehen werden. Eine gewöhnliche Faltungsebene besteht aus der Anwendung mehrerer solcher Filter. Da ein CNN aus mehreren, hintereinander ausgeführten Faltungsebenen

besteht, kann das Eingabebild sukzessiv durch diese Ebenen, welche in den Aktivierungen von den Neuronen resultieren, hindurchgereicht werden. Die Gewichte innerhalb der Kernel werden im Laufe des Trainings eines solchen Netzwerks individuell erlernt und adjustiert, wodurch genauere Merkmale herausgearbeitet werden können. Der Trainingsprozess wird im nächsten Abschnitt erläutert.

### 2.1.1 Residuale Netzwerke - Klassifizierung von Bildern

Um mit Hilfe eines CNNs einen Datensatz zu klassifizieren, werden mehrere Faltungsebenen zusammen mit der darauf folgenden nichtlinearen Aktivierungsfunktion und einer komprimierenden Funktion zur Informationsverdichtung in einem Block zusammengefasst. Angefangen bei einer Eingabeschicht, werden die Blöcke konsekutiv angewandt, bevor die extrahierten Merkmale an eine vollständig verknüpfte Schicht übergeben werden, welche für die schlussendliche Klassifizierung zuständig ist. Die Knoten der letzten Schicht bilden die Zielklassen ab. Dabei steht der Knoten mit der höchsten Aktivierung für vorhergesagte Klasse der jeweiligen Eingabe. Um zu vermeiden, dass die einzelnen Transformationen der Ebenen als Komposition sich wiederum als eine lineare Transformation zusammenfassen lassen, wird Nichtlinearität durch eine Aktivierungsfunktion nach jeder Faltungsebene eingeführt, welche auf alle Ausgabewerte einer Ebene angewandt wird. Beispielsweise begrenzt die *Rectified Linear Unit* die Aktivierung auf positive Werte, und wird als

$$ReLU(x) = x^+ = \max(0, x) \tag{2.3}$$

definiert, wobei $x$ der Ausgabewert eines Neurons ist.
Komprimierende Funktionen, welche überflüssige Informationen verwerfen, sind beispielsweise *Max-Pooling* oder *AdaptiveAveragePooling*.

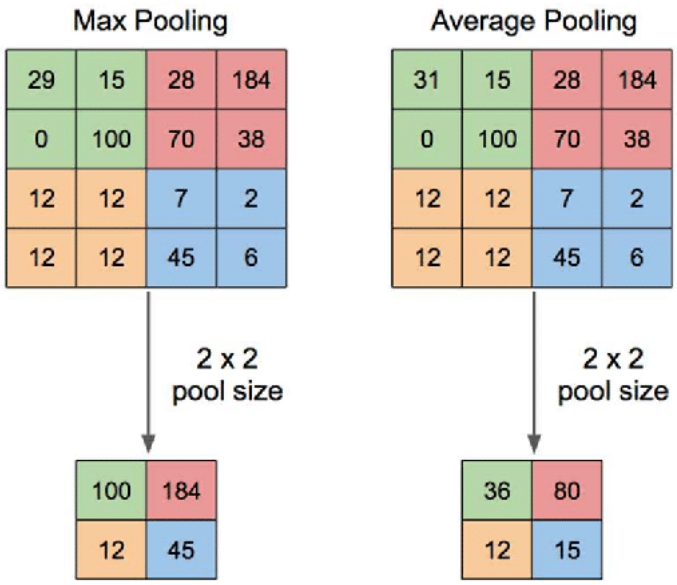


ABBILDUNG 2.1: Illustration von *Max-Pooling* und *Average Pooling* Auf einer $4 * 4$-Eingabe wird ein *Pooling Filter* der Größe $2 * 2$ angewandt. Beim *Max-Pooling* wird der maximale Wert, beim *Average Pooling* der Durchschnitt jeder Teilfläche behalten. [25]

In dem folgenden verwendeten Modell *ResNeXt* [24], welches als Grundlage für die Evaluierung der Klassifizierung verwendet wird, wird erstere Methode zu Beginn

angewandt, und *Adaptive Average Pooling* nach der letzten Faltungsebene. Durch diese Dimensionsreduktion wird aus der resultierenden Ebene, in unserem Falle mit $2048 * 7 * 7$ Neuronen, der Durchschnitt jedes der 2048 Felder mit der Größe $7 * 7$ verwendet, um eine vollständig verknüpfte Schicht aus 2048 Neuronen mit den Zielneuronen zu verbinden, welche die einzelnen Klassen bei einer Klassifizierung repräsentieren.
In den dazwischen liegenden Faltungsebenen wird bei der Anwendung des Kernels, eine vorgegebenen Schrittweite angewandt, welche die Größe der Ausgabe einer Ebene reduziert. Dieser sorgt dafür, dass beim Verschieben des Kernels Pixel übersprungen werden können und die einzelnen Ebenen immer kleiner werden.

Das Netzwerk verwendet ebenso eine residuale Verbindung, welche die Ausgabe eines Blocks mit der Ausgabe des vorherigen Blocks verrechnet, bevor die Aktivierungsfunktion (*ReLU* [16]) angewandt wird. Somit ist ist eine die Ausgabe des Blocks $i$

$$B_i = F_i(x) + x, \tag{2.4}$$

wobei $F$ die Transformationen des Blocks, und $x$ die Eingabe des Blocks beschreibt. Dadurch kann das Problem der verschwindenden Gradienten umgangen werden, und folglich stabile Netzwerke mit tieferer Schichtung kreiert werden.
Ebenso wird in der Architektur von neuronalen Netzwerken der Ansatz der Aufteilung der Pfade innerhalb eines Blocks verwendet, welcher bereits in einer der Architektur *Inception* [21] eingeführt wurde. Dies soll einer besseren räumlichen Abstraktion dienen und führt die Kardinalität $C$ ein, welche beschreibt, in wie viele unabhängige Pfaden mit gleichen Hyperparametern (Filter-Größen, Anzahl der Neuronen, etc.) die Variablen eines Blocks unterteilt werden soll. Die am Ende des Blocks aggregierten Aktivierungen werden dann mit der residualen Verbindung gemeinsam über die Aktivierungsfunktion zur Ausgabe des Blocks verrechnet.

$$B_i = \sum_{j=1}^{C} T_j(x) + x, \tag{2.5}$$

wobei $T_j$ die Transformationen eines einzelnen Pfades darstellt. [24]

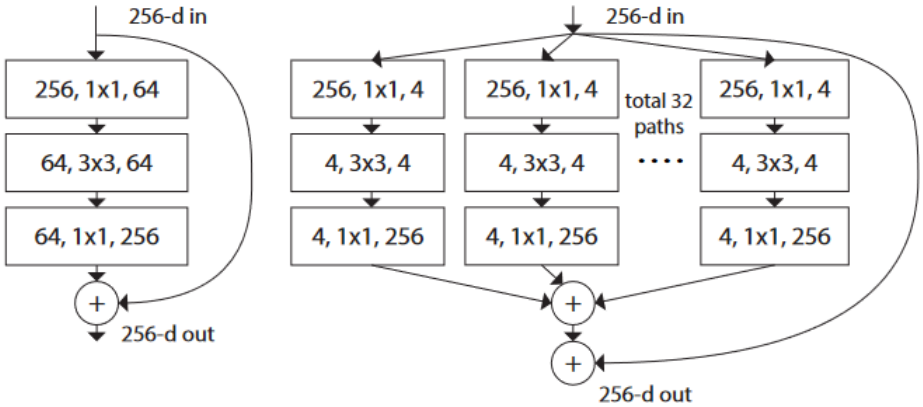


ABBILDUNG 2.2: Links: Ein Block eines *ResNets* [9]. Rechts: Ein Block eines *ResNeXts* mit Kardinalität = 32, mit ungefähr der gleichen Komplexität. Eine Ebene ist als (# Eingabekanäle, Filtergröße, # Ausgabekanäle) notiert. [24]

Beim beaufsichtigten Training für die korrekte Zuweisung von Bildern durch ein solches Netzwerk wird ein zuvor korrekt annotierter Datensatz verwendet. Ein Teil des Datensatzes wird zurückgehalten, um anschliessend für die Evaluierung als Testdatensatz verwendet werden zu können. Im Regelfall bilden 20% den Testdatensatz. Das Netzwerk bekommt zunächst den Trainingsdatensatz mitsamt der Information der zu jedem Bild gehörenden Klasse als Eingabe und bildet diese auf die Ausgabeschicht des Netzwerks ab. Die Diskrepanz zwischen den vorhergesagten und tatsächlichen Klassen jedes Bildes soll dabei minimiert werden, indem die Gewichte der trainierbaren Parameter des Netzwerks mittels des Gradientenverfahrens adjustiert werden. Die Gewichte der Verknüpfungen können im ersten Durchlauf willkürlich initialisiert sein, oder von einem zuvor trainierten Netzwerk ähnlicher Architektur stammen. Letztere Methode nennt sich *Transfer Learning* und bedient sich der Annahme, dass Netzwerke mit vielen Schichten in den ersten Ebenen die Konzepte von Flächen und Formen als Merkmale erlernen und herausfiltern, und detaillierte Strukturen und räumliche Verhältnisse dieser in den später folgenden Ebenen extrahiert werden. Somit kann ein Netzwerk, welches auf einem anderen Datensatz verwendet wurde, auf einen neuen und gegebenenfalls deutlich kleineren Datensatz spezialisiert werden.
Für das Training eines Netzwerkes mittels des Gradientenverfahrens wird die Abweichung der Ausgabe zu der originalen Annotation (Loss) mittels einer *Loss Function* bestimmt. Im Kontext der binären Klassifikation kann für die Evaluierung der Genauigkeit neben der *Cross Entropy* die *Binary Cross Entropy* für $N$ Eingaben verwendet werden. Letztere ist wie folgt definiert:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i * log(p(y_i)) + (1 - y_i) * log(1 - p(y_i), \tag{2.6}$$

wobei $p(y)$ die vorhergesagte Wahrscheinlichkeit der Zugehörigkeit der Klasse $y$ (kodiert mittels *one-hot encoding* als 0 oder 1) ist. Die Ausgabewerte der letzten Ebene, auch genannt *Logits*, werden für die Evaluierung zunächst mittels einer sigmoidalen Aktivierungsfunktion

$$Sigmoid(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

in numerische Werte zwischen 0 und 1 abgebildet, um die Wahrscheinlichkeiten der Vorhersagen für die beiden Klassen zu erhalten.
Ziel des Trainings durch die *Backpropagation* mittels des Gradientenverfahrens ist die Minimierung der Abweichung der Ausgaben. Dafür wird in jedem Schritt nach einem Teil des Datensatzes (festgelegt durch die *Batch Size*) für jedes Gewicht eine partielle Ableitung der verketteten Anwendung der *Loss Function* bestimmt. Die Gewichte können in die Richtung der ihr zugehörigen Gradienten angepasst werden. Das Ausmaß der Anpassung pro Schritt kann durch einen *Optimizer* und die Lernrate modifiziert werden. Meist wird der *ADAM-Optimizer* [12] verwendet, welcher die vorherigen Anpassungen miteinbezieht. Wenn nach allen Teildatensätzen des Trainingsdatensatzes die Gewichte angepasst wurden, gilt eine gesamte Trainingsepoche als komplettiert.
Die Anzahl der korrekt klassifizierten Eingaben im Verhältnis zu der Gesamtzahl aller eingaben beschreibt die Genauigkeit in Prozent. Um eine repräsentative Genauigkeit trotz ungleichgroßer Klassen zu erhalten, wird der *F1-Score* berechnet. Dieser berechnet die Genauigkeit für jede Klasse individuell, bevor in Bezug auf die Anzahl der Klassen die Gesamtgenauigkeit ausgegeben wird.

### 2.1.2 Encoder-Decoder-Architektur

Ein CNN kann ebenfalls für die Generierung von Daten verwendet werden. Ein *Autoencoder* ist ein Netzwerk, welches darauf trainiert ist, seine Eingabe so genau wie möglich zu rekonstruieren [8]. Dafür wird im Regelfall ein CNN trainiert, welches ein Eingabebild bezüglich seiner Merkmale hin abbildet, doch anstelle der darauf folgenden klassifizierenden Ebenen mehrere hochskalierende Ebenen verwendet. Das daraus resultierende Ergebnis hat somit das Format der Eingabe. Wie in den zuvor beschrieben Netzwerken, werden bei dieser Variante Aktivierungsfunktionen, und gegebenenfalls residuale oder normalisierende Ebenen verwendet, um eine bessere Generalisierung der Merkmale des Datensatzes zu ermöglichen. Eine Eigenschaft eines Autoencoders ist, dass die Eingabe mittels des Erlernens von Merkmalen durch den *CNN-Encoder* in den sog. *Latent Space* beliebiger Größe abgebildet werden kann. Der *Decoder* wiederum rekonstruiert daraus dann das Bild. Die hochskalierenden Faltungsebenen des *Decoders* bedienen sich ebenfalls mehrere Kernel, um die fehlenden Pixel aufzufüllen. Die Gewichte der Kernel müssen folglich im Zuge des Trainings erlernt werden, um möglichst genaue Resultate zu erzielen. Somit wird ein reguläres Hochskalieren vermieden.
Beim unüberwachten Training eines Autoencoders kann kein *Cross-Entropy Loss* verwendet werden, da der Autoencoder keine Klassifizierung vornimmt. Stattdessen wird der *Mean-Squared Error*

$$MSE = \frac{\sum_{i=1}^{N}(X_i - Y_i^{pred})^2}{N} \tag{2.8}$$

zwischen allen originalen Bildern $X$ und ihren Rekonstruktionen $Y^{pred}$ berechnet, welcher die durch das Netzwerk entstehende Abweichung beschreibt. Im Laufe des Trainings wird diese ebenfalls durch die Anpassung der Gewichte minimiert.
Hat man einen Datensatz, der aus verschiedenen Klassen besteht, liegt die Annahme nahe, dass die Abbildungen der jeweiligen Klassen auf den *Latent Space* $Z$ sich nach adäquater Trainingszeit mit dem Gradientenverfahren selbstständig in Cluster anordnen können, sofern die Daten aus unterschiedlichen Klassen sich in ihren Merkmalen unterscheiden. Auf eine auf den *Latent Space* abgebildete Eingabe kann zugegriffen und demnach eine Modifikation vorgenommen werden, sodass der *Decoder* ein neues Bild anhand der veränderten Merkmale generiert.

### 2.1.3 Invertierbare Klassifizierer

Ein Netzwerk, welches neben einer klassifizierenden Eigenschaft auch die Möglichkeit zur Generierung durch eine Inverse des Netzwerks bietet, wird *Flow-based generative model* genannt. Diese Netzwerke modellieren für einen gegebenen Datensatz Wahrscheinlichkeitsverteilungen, die auf komplexere Zielverteilungen abgebildet werden können. Weil die Abbildungen invertierbar sind, kann eine Probe aus einer Zielverteilung invertiert werden, was die Generierung neuer Daten durch ein solches Netzwerk ermöglicht. Insgesamt stellen Netzwerke dieser Art eine bijektive Funktion dar, weshalb für jede Eingabe eine eindeutige Abbildung auf den *Latent Space* existiert. Viele dieser Netzwerke werden auf die Maximierung der Log-Likelihood mittels der Substitutionsmethode hin trainiert, um eine Eingabe möglichst genau in eine Zielverteilung abzubilden [22].

Das *Invertible Residual Network* [3] besitzt die Eigenschaft eines invertierbaren Netzwerkes durch die Anwendung eines approximativen Verfahrens. Die Invertierbarkeit ist vor allem durch den Fixpunktsatz von Banach gegeben, da für die kontraktiven Faltungsebenen durch eine Normalisierung ein Wert der Lipschitzkonstante < 1 erzwungen wird. Somit kann die Invertierung der Ausgabe einer Ebene durch eine Approximation mittels einer endlichen Potenzmethode direkt auf der Faltungsebene angewandt werden.
Die allgemeine Architektur des Netzwerkes basiert auf der Idee der residualen Blöcke des *ResNets*. Sie ist jedoch dahingehend angepasst, dass ein *Scale-block* aus mehreren *i-ResNet-Blöcken* besteht. Jeder *i-ResNet-Block*, ähnlich der Funktionsweise der residualen Blöcke des *ResNeXt*, besteht aus drei Faltungsebenen, deren Ausgaben jeweils von der Aktivierungsfunktion *ELU* [4] auf das Intervall [-1, 1] abgebildet werden. Nach jedem *Scale-block* wird eine invertierbare Operation für das Herunterskalieren der räumlichen Dimension verwendet [3]. Das *i-ResNet* kann aufgrund dieser Struktur die Merkmale eines Datensatzes auf einen *Latent Space* abbilden, welcher an eine klassifizierende Ebene übergeben wird. Diese Arbeit bedient sich der Ähnlichkeit zu herkömmlichen, residualen Netzwerken, in Bezug auf die hohe Genauigkeit der Klassifizierungen. Die bijektive Eigenschaft wird anschließend verwendet, um Abbildungen auf den *Latent Space* des Netzwerkes zu modifizieren und durch die Invertierung des Netzwerkes ein neues Bild zu generieren.

## 2.2 Erklärbare künstliche Intelligenz (XAI)

Während immer mehre verschiedene Architekturen und Algorithmen für diverse Problemstellungen entwickelt und optimiert werden, befasst sich ein Zweig der Forschungs- und Anwendungsgebiete für künstlichen Intelligenz mit dem Verständnis für die daraus entstehenden Resultate. Geht es beispielsweise um eine Klassifizierung von komplexen Daten, eine Entscheidung für oder gegen ein Manöver eines autonomen Fahrzeugs, oder die Vorhersage von Kursentwicklungen im Finanzsektor, ist das Verständnis des Zustandekommens der Resultate oftmals unerlässlich.
Da die Klassifizierung von Bilddaten mittels größerer Architekturen mit Millionen von Parametern nur schwer nachvollziehbar ist, und die Netzwerke wie eine "black box"[14] erscheinen lassen, gibt es eine Reihe Methoden, die versuchen, visuelle Erklärungen für die Eingaben zu ermöglichen.

### 2.2.1 Attribution

Eine Möglichkeit, die für die Klassifizierung wichtigen Merkmale eines Bilder herauszuarbeiten, wäre eine Zuschreibung der räumlich wichtigen Areale einer Eingabe. Dafür kommt meistens ein leicht zu implementierender Ansatz der Attribution zum Einsatz [20], für den die Gradienten des Netzwerks analysiert werden. Bei der in diesem Projekt initial verwendeten Methode handelt es sich um eine okklusionsbasierte Zuschreibung der wichtigen Bildareale. Dafür wird eine Sensitivitätsanalyse der Ausgabe eines klassifizierenden Netzwerks durchgeführt, indem einzelne Bereiche eines Eingabebildes okkludiert werden, und somit festgestellt werden kann, welche der okkludierten Areale für die Klassifizierung wichitg sind [26]. Die okkludierte Fläche ist meist ein quadratisches Fenster einer zuvor gewählten Größe und wird über einen Teil des Eingabebildes gelegt. Damit ein Fenster mit einer festgelegten Schrittweite über das Bild geschoben werden kann, wird die Klassifizierung

pro Eingabebild mehrfach vorgenommen. Die Unterschiede in den Klassifizierungsergebnissen jedes Schrittes bieten Aufschluss darüber, ob ein abgedecktes Areal die korrekte Klassifizierung positiv oder negativ beeinflusst. Diese Beeinflussung der Areale kann anschliessend als *Heatmap* dargestellt werden. Das Verfahren kann auf jedem Bild eines Datensatzes angewandt werden, um jeweils individuelle, visuelle Informationen darüber liefern, welche Bereiche für die jeweilige Klassifizierung wichtig sind. Dies kann unter anderem beim weiteren Herausarbeiten von Unterschieden zwischen den Klassen, oder der Erklärung von falschen Klassifizierungen mehrere Objekte hilfreich sein.

### 2.2.2 Counterfactuals

Eine andere Methode, einen Einblick in die ausschlaggebenden Faktoren für die Vorhersagen eines klassifizierenden Netzwerks zu erlangen, bedient sich dem Konzept der *Counterfactuals*. Dabei wird für die Aussage einer getroffenen Entscheidung, hier die Klassifizierung eines Bildes, eine weitere Ausgabe generiert, die besagt, wie das eingegebene Bild hätte verändert sein müssen, um ein anderes erwünschtes Ergebnis zu erzielen [23]. Ziel ist es, eine minimale Änderung einer Eingabe zu finden, die zu einer Änderung der Entscheidung führen würde. Ebenso kann mit der Methode ein Bild generiert werden, welches die Eingabe so moduliert, dass es mit einer hohen Wahrscheinlichkeit einer anderen Klasse zugeordnet wird, wobei jedoch lediglich die für die Klassifizierung wichtigen Merkmale verändert wurden. Dies eignet sich, um die Unterscheide zwischen den Klassen direkt zu visualisieren, basiert jedoch oftmals auf einem Optimierungsproblem.
Anstatt ein Bild mehrfach zu modifizieren und Auszuwerten, bis das gewünschte Klassifizierungsergebnis vorliegt, ist mit der Methode der *Efficient Counterfactuals based on Invertible Neural Networks* [10] lediglich ein Schritt der Vorverarbeitung des Traingsdatensatzes notwending, um anschliessend mit nur einem Durchlauf für die Klassifizierung, und einer anschliessenden Invertierung des dabei entstanden *Latent Space*, ein *Counterfactual* zu generiern.
Für die Vorverarbeitung wird der gesamte Traingsdatensatz klassifiziert und die durchschnittlichen Merkmale, also der durchschnittliche *Latent Space* $\mu_C$, pro Klasse extrahiert. Anschliessend kann für jede Eingabe $x$ des Testdatensatzes ein latenter Vektor $z = f(x)$ mit dem Netzwerk bestimmt werden. Dieser kann nach der Manipulation mit $\alpha * \Delta$ von der vorhergesagten Klasse $p$ in die Richtung der Zielklasse $q$ verschoben werden. Mit der anschließenden Invertierung des Netzwerks $f$ kann ein *Counterfactual* $\hat{x}^{(q)}$ nach [10] wie folgt generiert werden:

$$\hat{x}^{(q)} = f^{-1}\left(f(x) + \alpha_0 \Delta_{p,q}\right). \tag{2.9}$$

$\Delta = \bar{\mu_q} - \bar{\mu_p}$ ist die Differenz der durchschnittlichen *Latent Spaces* für die vorhergesagten Klassen $p$ und $q$. $\alpha_0$ ist dabei wie folgt definiert:

$$\alpha_0 = -\frac{w^\top z + b}{w^\top \Delta_{p,q}}. \tag{2.10}$$

$w$ hingegen ist der Abstand der durchschnittlichen *Latent Spaces* für die tatsächliche Zugehörigkeit der Klassen $p$ und $q$. Mit $b = -\left(\frac{\mu_p + \mu_q}{2}\right)^\top * w$, kann so für jede Eingabe ein *Counterfactual* generiert werden, dessen Prädiktion und latente Abbildung auf der Entscheidungsebene zwischen der Eingabe und der erwünschten Klasse $q$ liegt. Das *Counterfactual*, dessen latentes Abbild mit mit hoher Wahrscheinlichkeit

innerhalb des Clusters der Abbildungen der Zielklasse liegt und folglich auch als diese evaluiert würde, kann mit $\alpha_1 = \frac{4}{5} * \frac{\alpha_0}{2}$ heuristisch generiert werden [10].

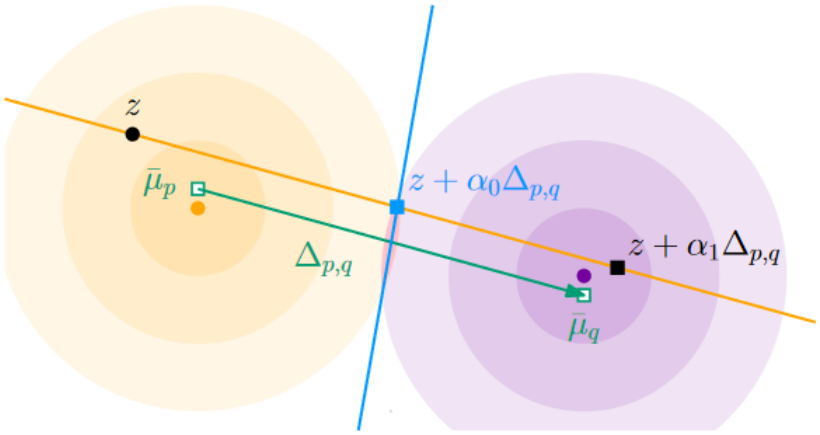


ABBILDUNG 2.3: *Latent Space*-Korrektur nach ECINN [10]. Ein latenter Vektor $z$ wird in Richtung $\Delta_{p,q}$ zum Cluster der Zielklasse $q$ (lila) moduliert. Auf der Entscheidungsgrenze (blau) liegt das Counterfactual.

Da diese Methode auf dem *Information Bottleneck Invertible Neural Network* [1] angewandt wird, welches ebenfalls einen normalisierenden Fluss und somit einen generativen Klassifizierer darstellt, sollte es möglich sein, die effiziente Berechnung der *Counterfactuals* auf ein anderes invertierbares Netzwerk anzuwenden.

# Kapitel 3

# Methoden

## 3.1 Datensatz

Bei den Trainings- und Testdatensätzen handelt es sich um Mikroskopaufnahmen von neuronal differenzierten Zellkulturen aus induzierten pluripotenten Stammzellen (IPSCs). Diese stammen von Probanden mit dem Leigh-Syndrom und einer quantitativ äquivalenten Kontrollgruppe ab. Um im weiteren Verlauf des Projektes "CureMILS" einen Medikamententest durchzuführen, wurden ebenfalls Zellkulturen herangezüchtet, welche mit *FCCP* behandelt wurden. Dabei handelt es sich um einen Wirkstoff, der die oxidative Phosphorylierung in den Mitochondrien entkoppelt und Protonen über die Membranen transportiert. Dies führt zu einer Änderung des mitochondrialen Membranpotentials.
Die Nuklei und Mitochondrien der Zellkulutren wurden für die Mikroskopie mit zwei unterschiedlich fluoreszierenden Markern hervorgehoben. Für die Einfärbung der Nuklei kam *DAPI* zum Einsatz, ein gängiges mittel für die Hervorhebung von Zellkernen. Für die Einfärbung der Mitochondrien wurde der Fluoreszenzfarbstoff *TMRM* eingesetzt. Auf Grund des Konzentrationsgradienten in den Mitochondrien reichert sich dieser in Abhängigkeit des mitochondrialen Membranpotentials an. Somit sollte ein Unterschied zwischen der Vergleichsklasse "erkrankt" (DS) und der mit dem Kontrollwirkstoff *FCCP* behandelten Klasse (TT) oder der gesunden Klasse (WT) erkennbar sein. Es ist zu erwarten, dass die Klassifizierungsergebnisse für den Vergleich der erkrankten mit der behandelten Gruppe besser ausfallen, da *FCCP* zu einer veränderten Membranaktivität und somit auch zu einer veränderten Fluoreszenz von *TMRM* führt. [11]
Das Mikroskop, welches die Bilder automatisiert für beide Marker generierte, fertige in einem Raster mehrere Bilder pro Zellkultur an. Somit liegen die Strukturen von Interesse, welche es zu erkennen gilt, nicht im Fokus oder der Mitte des Bildes, wie es beispielsweise bei *Micropattern* der Fall wäre, sondern sind über das gesamte Bild verteilt.
Der Fokus dieser Arbeit liegt darin, die Anwendbarkeit der Konzepte zu testen, während diese auf den größeren Datensatz (Klassen: DS und TT), welcher ein klares Merkmale aufgrund der Änderung der Intensität des Markers *TMRM* beinhalten sollte, angewandt werden.

### 3.1.1 Datenvorbereitung

Die Daten liegen als grauskalierte Bilder mit einer maximal verfügbaren Bittiefe von 16bit in einer Auflösung von $2160x2160$ Pixeln vor. Da die meisten CNNs für alltägliche Bilder mit drei Farbkanälen und einer Bittiefe von 8bit konzipiert sind, wurden die beiden Marker in einem Prozess der Vorverarbeitung als individuelle Farbkanäle

in einem Bild zusammengefügt. Um den dritten Farbkanal nicht leer lassen zu müssen, und alle Parameter eines Netzwerks zu verwenden, besteht dieser aus einer Kombination der ersten beiden. Dieses kombinierte Bild wurde auf $224x224$ Pixel herunterskaliert, da dies die selbe Auflösung ist, die für ein auf *ImageNet* [5] vortrainiertes *ResNet* und *ResNeXt* verwendet wurde.
Der Datensatz besteht aus 880 Bildern der Klasse DS und 920 Bildern der Klasse TT. Diese wurden in Traings- und Testdatensätze mit dem Verhältnis 70 zu 30 randomisiert erstellt. Aufgrund der geringen Anzahl an Bildern pro Klasse, wird eine Reihe an Augmentationen auf die Daten angewandt, um ein *Overtraining*, eine Spezialisierung auf den Trainingsdatensatz, zu vermeiden. Dafür wurde jedes Bild im Trainingsprozess mit einer Wahrscheinlichkeit von jeweils $p = 0.5$ über die horizontale oder vertikale Achse gespiegelt. Ebenso wurde eine Rotation des Bildes um einen zufälligen Wert aus dem Intervall $[-10, 10]$ Grad vorgenommen. Die Bilder wurden jeweils auf den Bereich $[-1, 1]$ mit Hilfe des Durchschnitts und der Standardabweichung aller Pixel des Trainingsdatensatzes pro Farbkanal normalisiert. Die Normalisierung kann nach der Erstellung von Rekonstruktionen oder *Counterfactuals* mittels eines Parameters rückgängig gemacht werden. Die resultierenden Bilder sind jedoch sehr dunkel, daher wird die normalisierte Variante in den Visualisierungen verwendet.

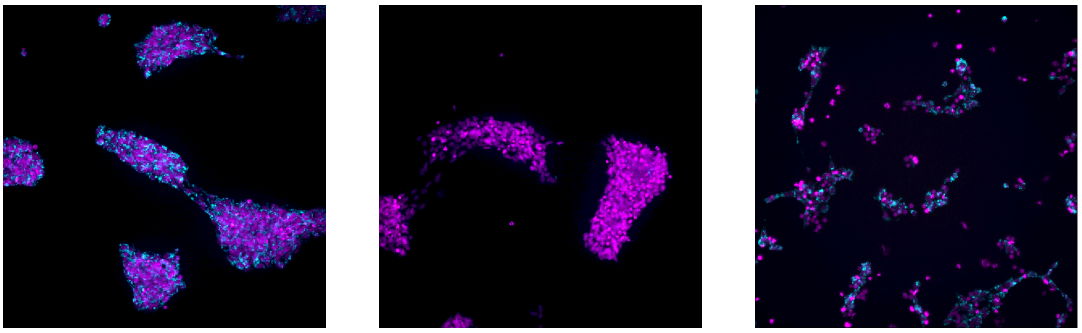

ABBILDUNG 3.1: Beispielbilder aus den drei Klassen (DS, TT und WT) des Datensatzes nach der Vorverarbeitung. Links: DS, Mitte: TT, Rechts: WT. Für eine bessere Darstellung sind diese auf eine Kantenlänge von 400 Pixeln skaliert worden und die Intensitäten mit ImageJ [17] hervorgehoben.

## 3.2 Modelle

Es wurden drei verschiedene Architekturen angewandt, auf deren Anwendung in den folgenden Sektionen eingegangen wird.
Die Netzwerke wurden in der Programmiersprache Python mittels der Bibliothek *PyTorch* [16] implementiert. Alle Skripte, bis auf die Initialisierung des *i-ResNets* wurden lokal in einem *Jupyter Notebook* auf einer NVIDIA GTX 1080-GPU unter Verwendung eines eines festgelegten Startwerts für den Zufahlszahlengenerator ausgeführt.

### 3.2.1 Klassifizierendes Vergleichsmodell

Für den initialen Test, ob die Klassen der Datensätze mittels neuronaler Netze unterscheidbar sind, wurde ein auf *ImageNet* vortrainiertes *ResNeXt* mit insgesamt 50 Ebenen verwendet (*ResNeXt50*), um möglichst gute Ergebnisse auf Grundlage des *Transfer Learnings* zu erhalten. Zusätzlich wurde die klassifizierende Ebene gegen

folgende Sequenz ausgetauscht:

| Ebenenname | # Ausgaben der Ebene |
|---|---|
| Linear: | 256 |
| Dropout (p=0.4): | 256 |
| Linear: | 100 |
| ReLU: | 100 |
| Linear: | 2 |

TABELLE 3.1: Klassifizierende Ebenen des *ResNeXt*

Ebenso wurden folgende Hyperparameter für das Training auf dem augmentierten Traingsdatensatz verwendet:

| Hyperparameter | |
|---|---|
| Epochen: | 10 |
| Lernrate: | 0.001 |
| Batch Size: | 15 |
| Aktivierungsfunktion: | ReLU |
| Loss-Funktion: | Cross-Entropy |
| Lernraten-Aktualisierung: | Exponentiell, Schrittweite=5, Gamma=0.1 |
| Optimierer: | ADAM |

TABELLE 3.2: Hyperparameter des *ResNeXt*

Unabhängig der Anzahl der Trainingsepochen wurde immer nur das Netzwerk mit der höchsten erzielten Genauigkeit der Prädiktionen des Testdatensatzes gespeichert.

### 3.2.2 Attribution

Um zu überprüfen, ob es möglich ist, anhand des Trainierten Netzwerkes wichtige Merkmale der Klassen ausfinding zu machen, wurde die Methode der okklusionsbasierten Zuschreibung auf dem Testdatensatz angewandt. Die Implementation wurde durch die Bibliothek *Captum* [13] ermöglicht, welche auf *PyTorch* aufbaut.
Das okkludierende Fenster wurde mit einer Größe von 15x15 Pixeln auf allen Farbkanälen angewandt. Die Schrittweite beim Verschieben des Fensters betrug sechs Pixel in horizontaler, sowie in vertikaler Richtung. Bei jedem Schritt wurde das Bild ausgewertet, um den Einfluss des jeweiligen Areals auf die Klassenzugehörigkeit zurückzuverfolgen. Zwei der daraus resultierenden *Heatmaps* sind in Kapitel 4 eingefügt.

### 3.2.3 Autoencoder

Um eine deutlichere Visualisierung der Änderung der Merkmale zwischen zwei Klassen generieren zu können, wurde ein Autoencoder implementiert. Der *Convolutional Autoencoder* besteht aus insgesamt 12 Blöcken. In der Mitte dieser Encoder-Decoder-Architektur liegt der *Latent Space*. Der Encoder setzt sich aus sechs Blöcken zusammen. Diese bestehen jeweils aus einer Faltungsebene mit Kernelgröße=3, einer Funktion zur Normalisierung der Neuronenaktivität (*BatchNorm2d*), einer darauf folgenden Aktivierungsfunktion (*ReLU*) und schlussendlich einer Max-Pooling-Ebene mit Kernelgröße=2 und Schrittweite=2. Der Letzte Block kommt ohne *BatchNorm2d* und *Max-Pooling* aus. Um den 98-dimensionalen *Latent Space* leichter extrahieren und darstellen zu können, wird dieser in eine eindimensionale Liste überführt.
Der Decoder besteht ebenfalls aus 6 Blöcken und stellt als erstes die Dimensionalität des *Latent Space* wieder her. Für das Hochskalieren werden transponierende Faltungsebenen mit Schrittweite=2 und Kernelgröße=2 verwendet. Zusätzlich kommen auch in diesen Blöcken die gleichen Normalisierungs- und Aktivierungsfunktionen zum Einsatz, die bereits im *Encoder* verwendet wurden. Der erste Block des *Decoders* kommt ohne Normalisierung aus und hat die Kernelgröße=3 und Schrittweite=1.
Des weiteren wurden folgende Hyperparameter für das Training verwendet:

| Hyperparameter | |
|---|---|
| Epochen: | 1000 |
| Lernrate: | 0.001 |
| Batch Size: | 20 |
| Aktivierungsfunktion: | ReLU |
| Loss-Funktion: | MSELoss |
| Optimierer: | ADAM (weight-decay=$e^{-5}$) |

TABELLE 3.3: Hyperparameter für den *Autoencoder*

Anschließend wurde der gesamte *Latent Space* des Testdatensatzes als Liste an Vektoren extrahiert. Diese wurde mittels einer Hauptkomponentenanalyse visualisiert und anhand der Annotationen der Bilder konnte eine mögliche Bildung von Clustern überprüft werden. Eine simple, schrittweise Interpolation zwischen den latenten Vektoren ($z$) zweier Bilder aus unterschiedlichen Klassen $p$ und $q$, sowie für die Interpolation eines Bildes der klasse $p$ in die Richtung des durchschnittlichen latenten Vektors einer anderen Klasse $q$ wurde mit folgende Formel ermöglicht:

$$\hat{X}^{(q_i)} = F_{Decoder}(z_p * \frac{1-i}{n} + z_q * \frac{i}{n}), \tag{3.1}$$

wobei der $i$-te von $n$ Schritten berechnet wird und die Anwendung von $F_{Decoder}$ ein neues Bild $\hat{X}^{(q_i)}$ generiert.

### 3.2.4 Invertible Residual Network

Um eine hohe Genauigkeiten bei der Klassifizierung zu erzielen und mit dem selben Netzwerk *Counterfactuals* generieren zu können, wurde das invertierbare, residuale Netzwerk von Behrman et al. [3] als zugrunde liegende Implementation gewählt. Das Netzwerk wird dabei in erster Linie als klassifizierendes Netzwerk trainiert, da der Fokus auf einer kompetetiven Genauigkeit der Vorhersagen liegt. Der Aufbau ist in Bezug auf die Anzahl der residualen Blöcke und Parameter mit dem *ResNeXt50*

vergleichbar und übergibt ebenfalls einen 2048-dimensionalen *Latent Space* an die klassifizierende Ebene. Diese wurde hier ebenfalls gegen die gleiche Sequenz mehrerer Ebenen getauscht, die bereits beim *ResNeXt50* zum Einsatz kam (3.1). Die Initialisierung der Gewichte wurde durch eine Trainingszeit von 10 Epochen auf dem *ImageNet100*-Datensatz mit der *Batch size*=64 und mit der Lernrate=0.01 durchgeführt. Alle anderen Hyperparameter blieben im Vergleich zum darauf folgenden Training unverändert. Der Datensatz beherbergt insgesamt 135.000 Bilder unterschiedlicher Auflösungen aus 100 Klassen, die zufällig aus dem gesamten *ImageNet*-Datensatz [5] ausgewählt wurden. Es wurde die Mitte der Bilder mit einer Kantenlänge von 224 Pixeln verwendet. Die Augmentationen der Daten wurden wie in 3.1.1 beschrieben vorgenommen. Dieser Schritt wurde aufgrund mangelnder Verfügbarkeit vortrainierter Netzwerke mit dieser *i-ResNet*-Architektur auf einer NVIDIA RTX 3090-GPU ausgeführt.
Die klassifizierenden Schichten wurden anschließend für das Training mit nur zwei Klassen neu aufgesetzt, die restlichen Gewichte blieben jedoch unverändert.
Die Hyperparameter für das Training, bei dem wieder nur der Zustand des Netzwerks mit der höchsten Genauigkeit gespeichert wurde, waren wie folgt:

| Hyperparameter | |
|---|---|
| Epochen: | 40 |
| Lernrate: | 0.005 |
| Batch Size: | 20 |
| Aktivierungsfunktion: | ELU, ActNorm=True |
| Loss-Funktion: | Cross-Entropy |
| Lernraten-Aktualisierung: | Exponentiell, Schrittweite=5, Gamma=0.1 |
| Spektralnormalisierungskoeffizient C: | 0.8 |
| # Spektralnormalisierungen: | 5 |
| Zusätzliche Kanäle pro Eingabebild: | 5 (mit 0 initialisiert) |
| Optimierer: | ADAMAX |

TABELLE 3.4: Hyperparameter für das *i-ResNet*

Die Aktivierungsfunktion wurde gegen *ELU*, welche negative Werte zulässt und eine stetige Ableitung besitzt, ausgetauscht.

## 3.3 Counterfactual-Generierung

Um nicht nur eine Klassifizierung der Bilder zu erhalten, wurde auf Grundlage des *i-ResNets* für jedes Eingabebild aus dem Testdatensatz ein *Counterfactual* mit $\alpha_0$ und eines mit $\alpha_1$ generiert. Dafür wurde die Formel 2.9 aus Kapitel 2 angewandt. Diese wurde in Anlehnung an die offizielle Implementierung von *ECINN* [10] implementiert.
Für den dafür notwendigen Vorbereitungsschritt wurde der Trainingsdatensatz erneut mit dem *i-ResNet* evaluiert und dabei der *Latent Space* jedes Bildes zwischengespeichert. Somit konnten die erforderlichen durchschnittlichen Abbildungen ($\mu$ und $\bar{\mu}$) der Daten auf den *Latent Space* berechnet werden.
Eine Bildung der Cluster aller *Latent Spaces* des Testdatensatzes konnte ebenfalls nach einer Hauptkomponentenanalyse durch eine Visualisierung überprüft werden.

# Kapitel 4

# Ergebnisse

## 4.1 Evaluierung der Netzwerke

Für die Evaluierung der Modelle wird neben der Genauigkeit der Klassifizierung der Z'-Faktor berechnet. Die Bilder werden in zwei Gruppen $p$ und $q$ anhand der tatsächlichen Klassenzugehörigkeit mit den dafür vorhergesagten Wahrscheinlichkeiten unterteilt. In diesem konkreten Anwendungsfall beschreibt er die Streuung der binären Vorhersageergebnisse, also in wie weit sich die zu vergleichenden Gruppen anhand ihrer Ergebnisse differenzieren lassen. Dieser Faktor beschreibt die statistischen Effektgröße und die Variation in den Messdaten. Er kommt vor allem bei Hochdurchsatz-Screenings zum Einsatz, um Bioassays direkt verglichen zu können [27]. Dafür werden die Ergebnisse der Vorhersagen des Testdatensatzes mit einer sigmoidalen Funktion in den Wertebereich [0, 1] überführt und die Ergebnisse nach ihrer tatsächlichen Zugehörigkeit zu den Klassen $p$ und $q$ sortiert. Der Z'-Faktor lässt sich als

$$\text{Estimated Z'-factor } = 1 - \frac{3\left(\bar{\sigma}_p + \bar{\sigma}_q\right)}{\left|\bar{\mu}_p - \bar{\mu}_q\right|} \tag{4.1}$$

anhand des Durchschnitts $\bar{\mu}$ und der Standardabweichung $\bar{\sigma}$ der Prädiktionswerte definieren. Der Z'-Faktor kann maximal den Wert 1 annehmen und ab einem Z'-Faktor$\geq$ 0.5 ist eine Brauchbarkeit des zugrundeliegenden Testverfahrens für weitere Experimente gegeben.
Jegliche Evaluierungen wurden auf dem Testdatensatz mit den Klassen DS und TT durchgeführt.

### 4.1.1 ResNeXt

Das *ResNeXt50* wurde für 10 Epochen mit den in Tabelle 3.2 spezifizierten Hyperparametern auf 70% des Datensatzes traininert, da bereits in vorherigen Durchläufen der empirischen Findung geeigneter Hyperparameter ersichtlich wurde, dass für diese vortrainierte Netzwerkarchitektur wenige Epochen ausreichen. Loss und Genauigkeit des Trainingsprozesses sind in Abbildung 4.1 abgebildet. Das Netzwerk hat bereits ab der ersten Epoche bei einer Testgenauigkeit von 99.75% erreicht. Das Netzwerk ist in der zweiten, fünften und sechsten Epoche besser darin, den Trainingsdatensatz zu klassifizieren, als den Testdatensatz, stabilisiert sich jedoch anschliessend wieder. Somit konnte eine Genauigkeit und ein F1-Score von 100% erreicht werden.
Der Z'-Faktor beläuft sich auf 0.924. In Abbildung 4.2 ist die Streuung der Prädiktionswerte dargestellt.

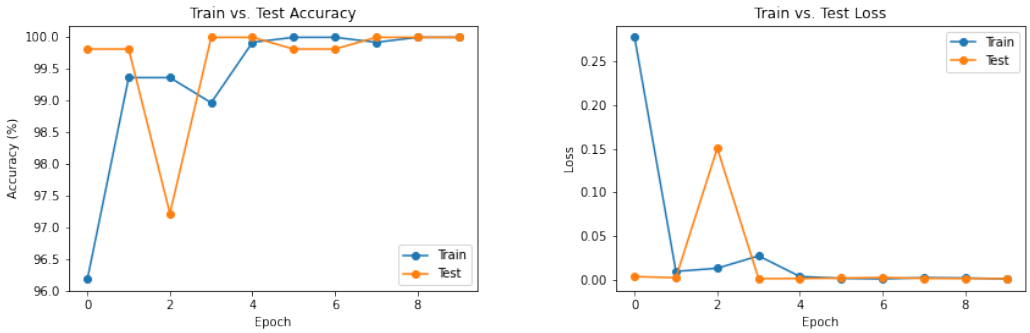


ABBILDUNG 4.1: *ResNeXt* - Loss und Genauigkeit: **Links:** Die Genauigkeit (Accuracy) der Prädiktionen des Netzwerks in Prozent für den gesamten Traingsdatensatz (blau) und den gesamten Testdatensatz (orange) nach jeder Epoche. **Rechts:** Der durch *Cross-Entropy* ermittelte Loss ist nach jeder Epoche dargestellt.

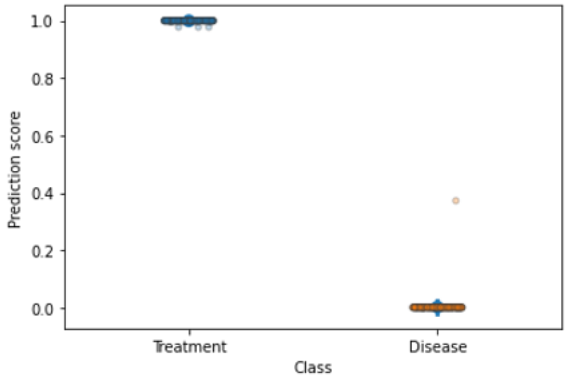


ABBILDUNG 4.2: *ResNeXt* - Verteilung der Prädiktionswerte in Abhängigkeit ihrer tatsächlichen Annotationen. Die Prädiktionen $Y$ für die Bilder der Klasse DS (Disease) sind als $1 - Y$ visualisiert, wodurch sich die Wahrscheinlichkeit der Zugehörigkeit zur Klasse TT (Treatment) ergibt. Die Varianz ist jeweils als vertikale Linie (blau) dargestellt.

### 4.1.2 Attribution

Auf das trainierte *ResNeXt50* wurde die okklusionsbasierte Attribution angewandt. Die daraus resultierenden *Heatmaps* bestätigten, dass das Netzwerk die Unterschiede der Klassen anhand der in den Strukturen der Zellkulturen enthaltenen Informationen definiert. Dies lässt sich daraus ableiten, dass überwiegend die Areale der Bilder einen größeren Einfluss auf die Klassifikation haben, in denen fluoreszierende Marker enthalten sind. Dies ist in Abbildung 4.3 verdeutlicht. Anhand der auf den Durchschnitt und die Standardabweichung aller Pixelwerte des Traingsdatensatzes normalisierten Bilder ist ein deutlicher Unterschied in der Intensität der fluoreszierenden Marker erkennbar. Die Interpretation für Areale, die einen starken Einfluss auf die Klassifikation haben, jedoch großteilig leer sind, erscheint kontraintuitiv (4.3, Links). Solche Areale sind in vielen *Heatmaps* zu finden.

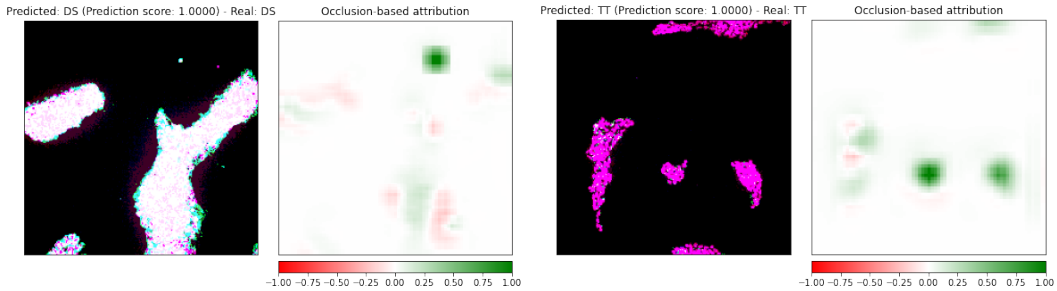


ABBILDUNG 4.3: Okklusionsbasierte Attribution, angewandt auf dem zuvor trainierten *ResNeXt50*. **Links & Mitte rechts:** Normalisierte Eingabebilder aus den Klassen DS und TT mit besonders hohem Prädiktionswert. **Mitte links & rechts:** Die zu den Eingabebildern gehörenden *Heatmaps*. Der Einfluss eines Areals auf das jeweilige Prädiktionsergebnis ist durch die Farbe (von rot zu grün) dargestellt.

### 4.1.3 Autoencoder

Der Autoencoder erreichte nach 1000 Trainingsepochen mit den in Tabelle 3.3 spezifizierten Hyperparametern einen MSE-Loss von 0.46. Die Rekonstruktionen der Bilder aus dem Testdatensatz durch den Autoencoder bilden folglich die Strukturen unter Einbuße eines überzeugenden Detailreichtums ab (4.4). Die extrahierten, latenten Vektoren sind in der Visualisierung nach der Hauptkomponentenanalyse abhängig ihrer ursprünglichen Klassenzugehörigkeit klar differenzierbar (Abbildung 4.5). Eine Interpolation von einem Bild in die Richtung des durchschnittlichen latenten Vektors der anderen Klasse, ändert die Intensität der vorhandenen Marker deutlich. Die zugrundeliegende Struktur der Zellkultur bleibt weiterhin erkennbar (4.6). Für die Klassifizierung der Rekonstruktionen der Interpolierten, latenten Vektoren, bedarf es der Anwendung eines zweiten neuronales Netzwerks.

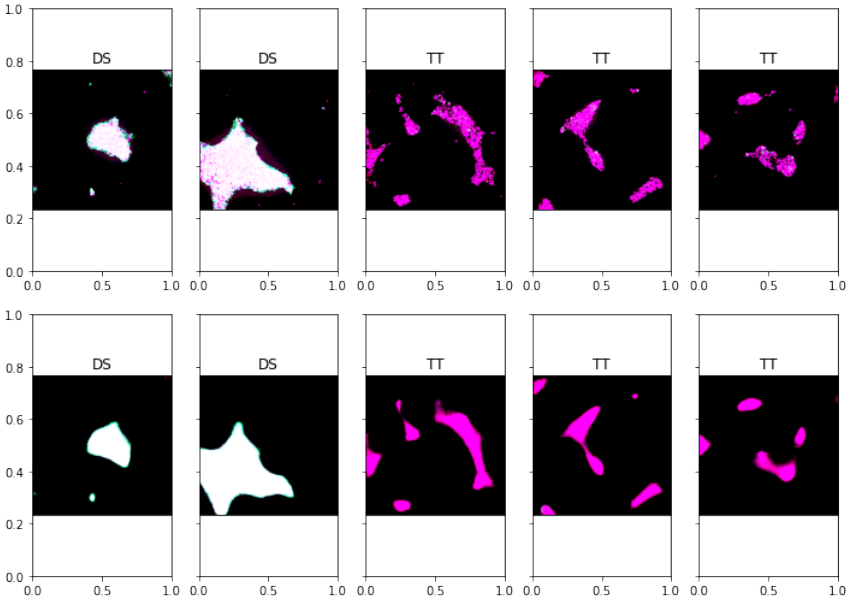


ABBILDUNG 4.4: Rekonstruktionen des Autoencoders. **Oben:** Normalisierte Eingabebilder mit ihrer Klassenzugehörigkeit. **Unten:** Zugehörige Rekonstruktionen der Eingabebilder durch den Autoencoder.

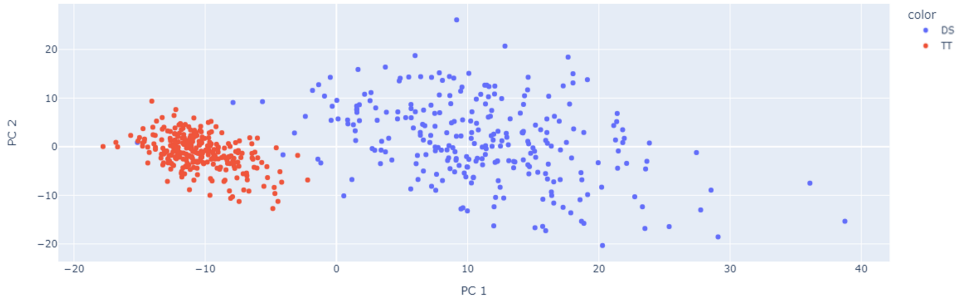


ABBILDUNG 4.5: Autoencoder - Visualisierung des *Latent Space*. Die latenten Vektoren der Testbilder werden anhand ihrer beiden Hauptkomponenten (*PC 1* und *PC 2*) dargestellt. Die zugehörigen Klassen DS (orange) und TT (blau) sind farblich gekennzeichnet.

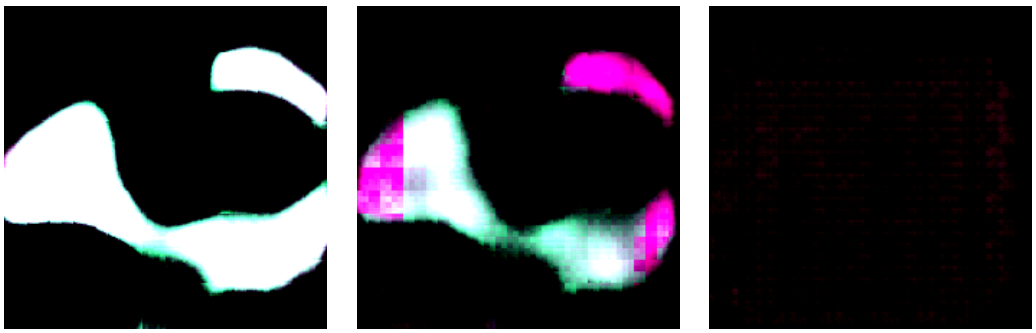

ABBILDUNG 4.6: Autoencoder - Rekonstruktionen von latenten Vektoren. **Links:** Rekonstruktion eines Bildes aus der Klasse DS. **Mitte:** Der *Latent Space* des linken Bildes, verschoben in Richtung des durchschnittlichen *Latent Spaces* der Klasse TT, nach Anwendung des *Decoders*. **Rechts:** Der durchschnittliche *Latent Space* der Klasse TT nach Anwendung des *Decoders*.

### 4.1.4 Invertible Residual Network

Das *i-ResNet* wurde nach der Vorinitialisierung mit dem *ImageNet100*-Datensatz auf dem Trainingsdatensatz mit den in Tabelle 3.4 spezifizierten Parametern weitertrainiert. Der Loss des Testdatensatzes blieb dabei durchgehend geringer als der des Trainingsdatensatzes, wodurch ein *Overtraining* auszuschließen ist (Abbildung 4.7). Dabei konnte eine Genauigkeit und einen F1-Score von 100% erreicht werden.
Der Z'-Faktor belief sich auf 0.74. Die Verteilung der überaus hohen Prädiktionswerte, aus denen sich dieser Wert zusammensetzt, ist in Abbildung 4.8 dargestellt.
Der Betrag der Differenz zwischen den Intensitäten der Pixel des rekonstruierten Bildes und derer des Originals beträgt im Durchschnitt weniger als 1 pro Pixel, ist also in dem Rekonstruierten Bild nicht mit bloßem Auge erkennbar (Abbildung 4.9)

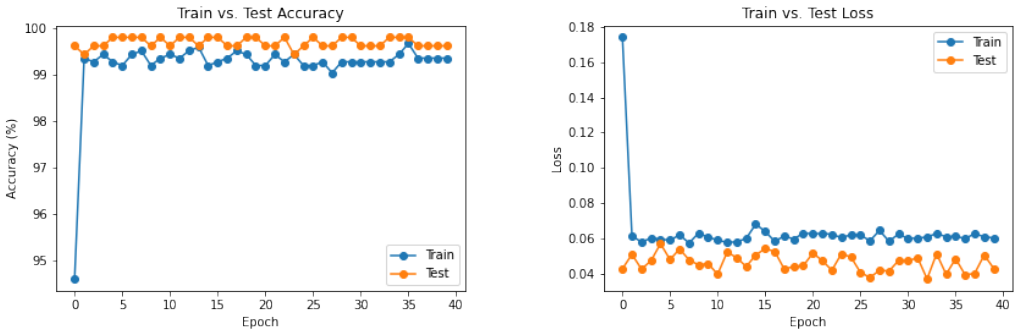


ABBILDUNG 4.7: *i-ResNet* - Loss und Genauigkeit: **Links:** Die Genauigkeit (Accuracy) der Prädiktionen des Netzwerks in Prozent für den gesamten Traingsdatensatz (blau) und den gesamten Testdatensatz (orange) nach jeder Epoche. **Rechts:** Der durch *Cross-Entropy* ermittelte Loss ist nach jeder Epoche dargestellt.

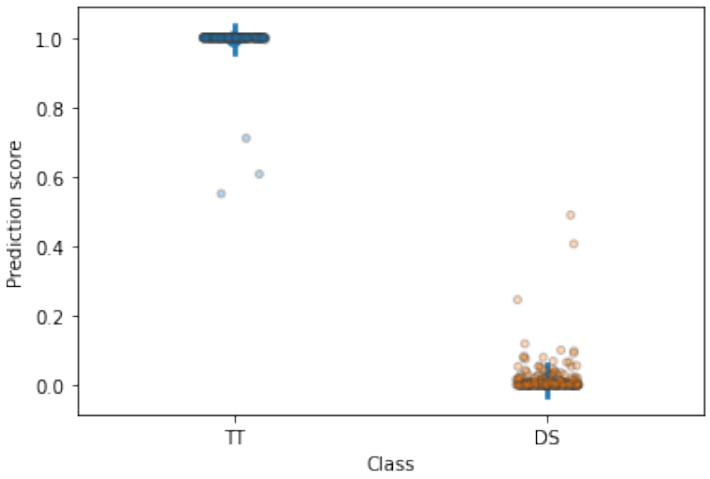


ABBILDUNG 4.8: *i-ResNet* - Verteilung der Prädiktionswerte in Abhängigkeit ihrer tatsächlichen Annotationen. Die Prädiktionen $Y$ für die Bilder der Klasse DS (Disease) sind als $1 - Y$ visualisiert, wodurch sich die Wahrscheinlichkeit der Zugehörigkeit zur Klasse TT (Treatment) ergibt. Die Varianz ist jeweils als vertikale Linie (blau) dargestellt.

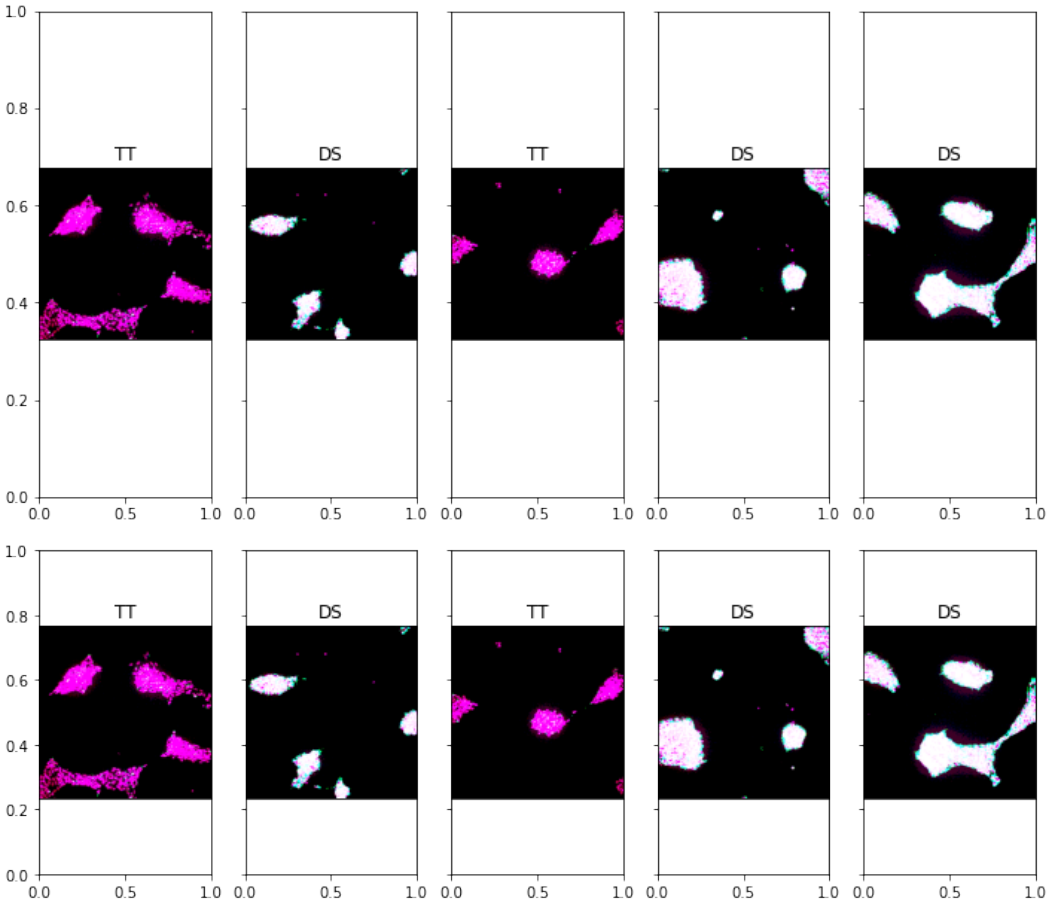


ABBILDUNG 4.9: Rekonstruktionen des *i-ResNets*. **Oben:** Normalisierte Eingabebilder mit ihrer Klassenzugehörigkeit. **Unten:** Zugehörige Rekonstruktionen der Eingabebilder durch den Autoencoder.

### 4.1.5 Counterfactuals

Vor der Generierung der *Counterfactuals* wurde der *Latent Space* nach Anwendung einer Hauptkomponentenanalyse visualisiert. Hier ist anhand der Annotation der tatsächlichen Klassenzugehörigkeit eine eindeutige Differenzierung ersichtlich. Es scheint, als würden die Abbildungen der Testdaten nahezu orthogonal aufeinander stehen (Abbildung 4.10). Dennoch ist eine etwas größere Streuung innerhalb der Klasse DS erkennbar, wie es bereits beim Autoencoder der Fall war (Abbildung 4.5). Die generierten *Counterfactuals*, welche auf der Entscheidungsebene des Klassifizierers abgebildet sind, scheinen minimale Änderungen zu beinhalten, die zwar deutlich erkennbar, aber detailliert sind (Abbildung 4.11).

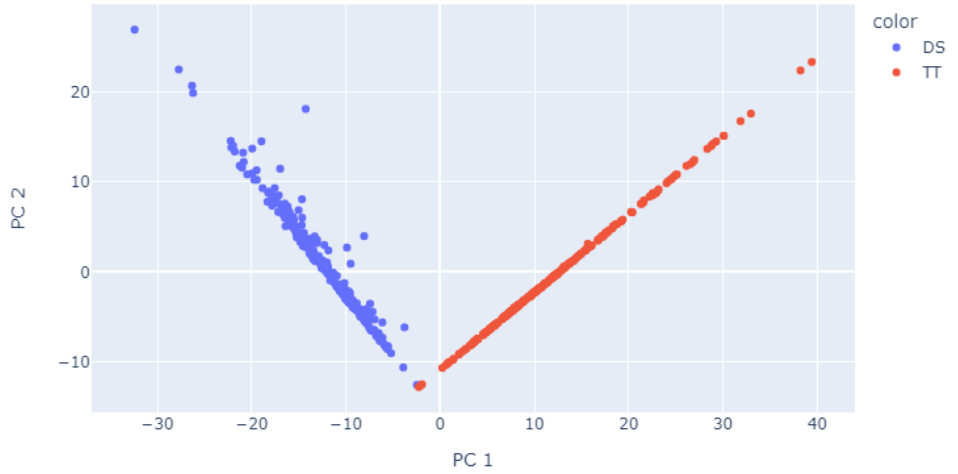


ABBILDUNG 4.10: *i-ResNet* - Visualisierung des *Latent Space*. Die latenten Vektoren der Testbilder werden anhand ihrer beiden Hauptkomponenten (*PC 1* und *PC 2*) dargestellt. Die zugehörigen Klassen DS (orange) und TT (blau) sind farblich gekennzeichnet.

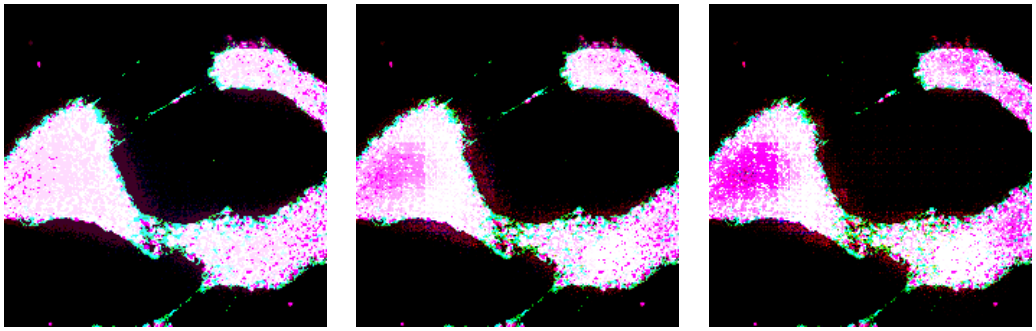

ABBILDUNG 4.11: *i-ResNet* - Rekonstruktionen von latenten Vektoren. **Links:** Rekonstruktion eines Eingabebildes aus der Klasse DS. **Mitte:** Mit 2.9 generiertes *Counterfactual* des Eingabebildes für $\alpha_0$. **Rechts:** Ein überzeugendes *Counterfactual*, welches mit 2.9 und $\alpha_1$ generiert wurde und der Klasse TT zugeschrieben wird.

# Kapitel 5

# Diskussion und Fazit

## 5.1 Diskussion

Die vom *ResNeXt50* erzielte Genauigkeit der Klassifizierungen dient als Grundlage für die Evaluierung der Brauchbarkeit des *i-ResNets*. Da diese bereits bei 100% liegt, und zugleich eine geringe Streuung der Vorhersagewerte aufweist. Da das *ResNeXt50* durch das Vortraining bereits zahlreiche, generalisierende Merkmale erlernen konnte, war die Trainingszeit auf dem problemspezifischen Datensatz sehr kurz. Um ebenfalls mit dem *i-ResNet* gute Ergebnisse zu erzielen, war eine randomisierte Initialisierung der Gewichte nicht ausreichend. Daher wurde es ebenfalls für einige Epochen auf einem Teil des *ImageNet*-Datensatzes vortrainiert. Nach eingehender Anpassung der Hyperparameter des *i-ResNets* konnte dieses ebenfalls eine Genauigkeit von 100% erreichen. Auch wenn der Z'-Faktor um einiges niedriger ist, als der des Vergleichmodells, ist die Streuung der Vorhersagen zwischen den Klassen DS und TT noch immer sehr gering und liegt im erwarteten Rahmen des zugrundeliegenden Experiments.
Eine okklusionsbasierte Attribution lässt sich auf diesem Datensatz anwenden, und kann Aufschluss über die wichtigen Areale liefern, ist jedoch für direkte, visuelle Erklärungen nicht intuitiv anwendbar. Beispielsweise ist dies auf biologischen Datensätzen, bei denen keine klare Unterscheidung in der Intensität der Marker vorliegt, sondern eher auf morphologischen Unterschieden basieren, schwieriger zu interpretieren. Die Abbildung eines Datensatzes auf einen *Latent Space* durch ein invertierbares Netzwerk, welches für die Klassifikation der Daten ausreichend trainiert ist, sollte die klassenabhängigen Merkmale strukturiert abbilden können [6]. Somit ordnen sich die Cluster im Verlauf des Trainings abhängig der Klassifizierung an, anstelle der Informationen für die Rekonstruktion, wie bei dem Autoencoder. Die Rekonstruktionen des Autoencoders erfüllen trotz langer Trainingszeit nicht die Erwartungen, was an dem niedrigdimensionaleren *Latent Space*, sowie der erlernten Parameter des *Decoders* liegen könnte.
Es zeigt sich, dass die approximative, inverse Funktion des *i-ResNets* sehr gute Rekonstruktionen liefern kann. Folglich sind die generierten *Counterfactuals* detailreich und behalten die in den Eingabebildern enthaltenen Strukturen bei, in die die Änderungen eingefügt werden. Zusätzlich lässt sich mit dem selben Netzwerk eine Klassifizierung für jede Änderung einer Abbildung vornehmen. Mit einer Schrittweisen Änderung des Wertes $\alpha_0$ kann mit 2.9 für jeden Schritt ein neues Bild generiert werden und die Änderung somit als Animation visualisiert werden.

## 5.2 Zukünftige Arbeiten

Um die Ergebnisse der Vorhersagen zu optimieren, und diese Methodik auch auf kleineren Datensätzen effizient einzusetzen, wäre ein generalisiertes, vortrainiertes Netzwerk hilfreich. Dieses könnte dann auf die individuellen Problemstellungen hin mittels *Transfer Learning* trainiert werden. Für die Optimierung der Clusterbildung und Generierung neuer Bilder, gerade bei Mehrklassen-Klassifizierungen, kann der Ansatz einer kombinierten *Loss Function* für das *i-ResNet* implementiert werden, welche die Distanz einer Abbildung zur durchschnittlichen Repräsentation der Klasse berücksichtigt, ähnlich zu [1].
Um die klassenspezifischen Änderungen für Datensätze mit weniger prägnanten Merkmalen besser nachzuverfolgen, könnten die Änderungen zwischen den generierten *Counterfactuals*, mit schrittweise modifizierten $\alpha$-Werten, in Abhängigkeit ihres Einflusses auf das Klassifikationsergebnis als *Heatmap* festgehalten werden. Die *Heatmap* kann verwendet werden, um die wichtigen Areale aus der Eingabe zu extrahieren. Auf diesen können dann klassische statistische Methoden angewandt werden, um die Änderung innerhalb der auf das Ergebnis einflussreichen Areale zu quantifizieren. Klassische statistische Quantifikation auf den Arealen, welche wichtige Merkmale enthalten, begleitet von visuellen Veränderung der Eingaben in Richtung einer anderen Klasse könnten helfen, Unterschiede zwischen Klassen zu verdeutlichen.

## 5.3 Fazit

Für die kombinierte Anwendung von neuronalen Netzwerken, die primär auf gute Klassifikationsergebnisse hin trainiert werden, können *Coiunterfactuals* für jede Eingabe generiert werden. Die Anwendung wird dadurch vereinfacht, dass nur ein einziges Netzwerk Trainiert und Evaluiert werden muss, während es eine vergleichbare Leistung zu anderen klassifizierenden Implementationen bietet. Diese Methode kann beispielsweise bei einem Medikamententest auf Zellmodellen zum Einsatz kommen. Somit könnte für jedes Modell und Substanz nicht nur die Änderung der vorhergesagten Klasse ausgewertet werden, sondern ebenso der Unterschied zwischen ihnen aufgezeigt werden. Die Repräsentation der Merkmale der Klassen kann im *Latent Space* ebenfalls visualisiert werden. Dadurch können sich irregulär verhaltende Daten identifiziert werden. Diese Identifikation kann für die Optimierung weiterer Versuchsreihen wichtig sein.

## 5.4 Quellcodes

Die Implementationen dieser Arbeit sind unter
https://github.com/Scaramir/BachelorThesis verfügbar.

# Literatur


[1] Lynton Ardizzone u. a. „Training Normalizing Flows with the Information Bottleneck for Competitive Generative Classification“. en. In: *arXiv:2001.06448 [cs, stat]* (Jan. 2021). arXiv: 2001.06448. URL: http://arxiv.org/abs/2001.06448 (besucht am 10. 05. 2022).

[2] Yusuf Bayraktar und Enes Ayan. „Diagnosis of interproximal caries lesions with deep convolutional neural network in digital bitewing radiographs“. eng. In: *Clinical Oral Investigations* 26.1 (Jan. 2022), S. 623–632. ISSN: 1436-3771. DOI: 10.1007/s00784-021-04040-1.

[3] Jens Behrmann u. a. „Invertible Residual Networks“. en. In: *arXiv:1811.00995 [cs, stat]* (Mai 2019). arXiv: 1811.00995. URL: http://arxiv.org/abs/1811.00995 (besucht am 12. 05. 2022).

[4] Djork-Arné Clevert, Thomas Unterthiner und Sepp Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2015. DOI: 10.48550/ARXIV.1511.07289. URL: https://arxiv.org/abs/1511.07289.

[5] Jia Deng u. a. „Imagenet: A large-scale hierarchical image database“. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, S. 248–255.

[6] Laurent Dinh, Jascha Sohl-Dickstein und Samy Bengio. *Density estimation using Real NVP*. 2016. DOI: 10.48550/ARXIV.1605.08803. URL: https://arxiv.org/abs/1605.08803.

[7] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Kap. 9, S. 327+.

[8] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Kap. 14, S. 499+.

[9] Kaiming He u. a. „Deep Residual Learning for Image Recognition“. In: (Dez. 2015). arXiv:1512.03385 [cs]. DOI: 10.48550/arXiv.1512.03385. URL: http://arxiv.org/abs/1512.03385 (besucht am 20. 09. 2022).

[10] Frederik Hvilshøj, Alexandros Iosifidis und Ira Assent. „ECINN: Efficient Counterfactuals from Invertible Neural Networks“. en. In: *arXiv:2103.13701 [cs]* (Apr. 2021). arXiv: 2103.13701. URL: http://arxiv.org/abs/2103.13701 (besucht am 12. 05. 2022).

[11] Dinesh C. Joshi und Joanna C. Bakowska. „Determination of Mitochondrial Membrane Potential and Reactive Oxygen Species in Live Rat Cortical Neurons“. en. In: *Journal of Visualized Experiments* 51 (Mai 2011), S. 2704. ISSN: 1940-087X. DOI: 10.3791/2704. URL: http://www.jove.com/index/Details.stp?ID=2704 (besucht am 24. 09. 2022).

[12] Diederik P. Kingma und Jimmy Ba. „Adam: A Method for Stochastic Optimization“. In: *CoRR* abs/1412.6980 (2015).

[13] Narine Kokhlikyan u. a. *Captum: A unified and generic model interpretability library for PyTorch*. 2020. arXiv: 2009.07896 [cs.LG].

[14] Pantelis Linardatos, Vasilis Papastefanopoulos und Sotiris Kotsiantis. „Explainable AI: A Review of Machine Learning Interpretability Methods“. en. In: *Entropy* 23.1 (Jan. 2021), S. 18. ISSN: 1099-4300. DOI: 10.3390/e23010018. URL: https://www.mdpi.com/1099-4300/23/1/18 (besucht am 22. 09. 2022).

[15] Jakob J. Metzger u. a. „Deep-learning analysis of micropattern-based organoids enables high-throughput drug screening of Huntington's disease models“. en. In: *Cell Reports Methods* 2.9 (Sep. 2022), S. 100297. ISSN: 2667-2375. DOI: 10.1016/j.crmeth.2022.100297. URL: https://www.sciencedirect.com/science/article/pii/S2667237522001795 (besucht am 27. 09. 2022).

[16] Adam Paszke u. a. „PyTorch: An Imperative Style, High-Performance Deep Learning Library“. In: *Advances in Neural Information Processing Systems 32*. Hrsg. von H. Wallach u. a. Curran Associates, Inc., 2019, S. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[17] Curtis T. Rueden u. a. „ImageJ2: ImageJ for the next generation of scientific image data“. In: *BMC Bioinformatics* 18.1 (Nov. 2017). DOI: 10.1186/s12859-017-1934-z. URL: https://doi.org/10.1186/s12859-017-1934-z.

[18] Fahman Saeed u. a. „Diabetic Retinopathy Screening Using Custom-Designed Convolutional Neural Network“. In: *arXiv:2110.03877 [cs, eess]* (Okt. 2021). arXiv: 2110.03877. URL: http://arxiv.org/abs/2110.03877 (besucht am 12. 05. 2022).

[19] Karl Schulz u. a. „Restricting the Flow: Information Bottlenecks for Attribution“. In: *arXiv:2001.00396 [cs, stat]* (Mai 2020). arXiv: 2001.00396. URL: http://arxiv.org/abs/2001.00396 (besucht am 12. 05. 2022).

[20] Mukund Sundararajan, Ankur Taly und Qiqi Yan. *Axiomatic Attribution for Deep Networks*. en. arXiv:1703.01365 [cs]. Juni 2017. URL: http://arxiv.org/abs/1703.01365 (besucht am 22. 09. 2022).

[21] Christian Szegedy u. a. „Going Deeper with Convolutions“. en. In: (Sep. 2014). arXiv:1409.4842 [cs]. URL: http://arxiv.org/abs/1409.4842 (besucht am 24. 09. 2022).

[22] Esteban G. Tabak und Eric Vanden-Eijnden. „Density estimation by dual ascent of the log-likelihood“. In: *Communications in Mathematical Sciences* 8.1 (2010), S. 217–233. DOI: 10.4310/cms.2010.v8.n1.a11. URL: https://doi.org/10.4310/cms.2010.v8.n1.a11.

[23] Sandra Wachter, Brent Mittelstadt und Chris Russell. „Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR“. In: (2017). DOI: 10.48550/ARXIV.1711.00399. URL: https://arxiv.org/abs/1711.00399.

[24] Saining Xie u. a. „Aggregated Residual Transformations for Deep Neural Networks“. In: (Apr. 2017). arXiv:1611.05431 [cs]. DOI: 10.48550/arXiv.1611.05431. URL: http://arxiv.org/abs/1611.05431 (besucht am 20. 09. 2022).

[25] Muhamad Yani, S Irawan und Casi Setianingsih. „Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail“. In: *Journal of Physics: Conference Series* 1201 (Mai 2019), S. 012052. DOI: 10.1088/1742-6596/1201/1/012052.

[26] Matthew D Zeiler und Rob Fergus. „Visualizing and Understanding Convolutional Networks“. In: (2013). DOI: 10.48550/ARXIV.1311.2901. URL: https://arxiv.org/abs/1311.2901.

[27] Ji-Hu Zhang, Thomas D.Y. Chung und Kevin R. Oldenburg. „A Simple Statistical Parameter for Use in Evaluation and Validation of High Throughput Screening Assays“. In: *SLAS Discovery* 4.2 (Apr. 1999), S. 67–73. DOI: 10.1177/108705719900400206. URL: https://doi.org/10.1177/108705719900400206.