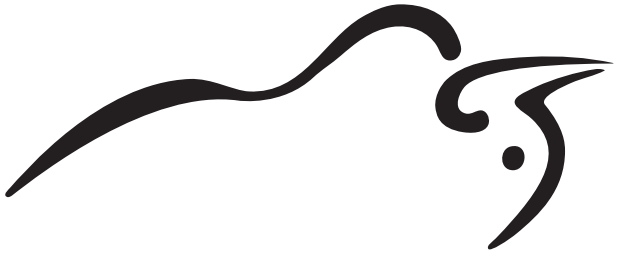


CHAIR FOR BIOINFORMATICS
AND INFORMATION MINING

Deep Learning als Virtual-High-Throughput-Screening-Methode unter Verwendung von gerasterten Molekülstrukturen

Dissertation

zur Erlangung des akademischen Grades
des Doktors der Naturwissenschaften (Dr. rer. nat.)
an der Universität Konstanz
Fachbereich für Informatik und Informationswissenschaft

vorgelegt von

Patrick Winter

Tag der mündlichen Prüfung: 20. April 2020

Referenten:
Prof. Dr. Christian Borgelt
Prof. Dr. Bastian Goldlücke

Patrick Winter:
Deep Learning als Virtual-High-Throughput-Screening-Methode unter Verwendung von gerasterten Molekülstrukturen
Universität Konstanz, 2020.

Abstract

Virtual high-throughput screening is the use of machine learning classification methods to estimate the probability of molecules to show a desired biological activity. The goal in most cases is to preselect the molecules that have the highest probability and thereby reduce the amount of molecules that have to be tested in the lab.

In the past few years, deep learning has become one of the most successful methods to classify data in the area of image recognition. This is usually done with the use of convolutional neural networks that first generate abstract features based on the raw pixel input. These features are then used to classify the data.

The goal of this thesis is to adapt these methods so that they can be used for virtual high-throughput screening. This is done by converting the structure of a molecule into a grid-based format that is similar to an image. The layout of the atoms in this grid is based on the 2D-rendering of the structure. The element symbol and the chemical properties of an atom are utilized as its features. The used network architecture consists of convolutional layers that generate abstract features and dense layers that use these features to classify the molecule. By splitting the network into these two tasks, it is also possible to only use the convolutional part of the network to generate features which are then used in combination with different methods.

Another goal of this thesis is to interpret what the network actually learned from the data. For this saliency maps are used to compute which areas of the input data had the biggest influence on the resulting class. This computed influence can then be used to visualize the substructures that, according to the network, are responsible for the molecules activity. These substructures can then also automatically be extracted by using a threshold-based method. These can also be sorted by relevance, by assigning a score to each substructure.

The software architecture that has been implemented in the course of this thesis is also introduced. It enables efficient use of parallel resources by preprocessing new data using all processor cores while the previous data is used for training in the graphics card. The architecture also saves intermediate results which make it easy to resume experiments that have been halted or did crash.

The experimental part of the thesis searches for good hyperparameters for the presented method. It also compares the method with other virtual high-throughput screening methods. In this comparison the presented method ends up with results similar to those

of existing fingerprints. The automated substructure extraction is also able to find the substructures that were actually responsible for the activity.

Zusammenfassung

Virtual High-Throughput Screening ist die Anwendung von Klassifizierungsmethoden des maschinellen Lernens, um die Aktivität von Molekülen abzuschätzen. Das Ziel dabei ist es oftmals, eine Vorauswahl zu treffen, welche Moleküle am ehesten eine Chance haben, eine gewünschte biologische Aktivität zu zeigen. Dadurch lässt sich die Menge an tatsächlich im Labor durchzuführenden Tests reduzieren.

Im Bereich des maschinellen Sehens hat sich in den letzten Jahren die Anwendung von *Deep Learning* als besonders akkurate Methode der Klassifizierung durchgesetzt. Hierbei werden oft *Convolutional Neural Networks* eingesetzt, die zuerst aus den Pixeln eines Bildes abstraktere Merkmale generieren, welche sich dann zu einer besseren Klassifizierung eignen.

Das Ziel dieser Arbeit ist es, diese Methoden des maschinellen Sehens so zu adaptieren, dass sie sich zum Einsatz für das *Virtual High-Throughput Screening* eignen. Dies wird durch die Umwandlung der Struktur eines Moleküls in ein bildähnliches Rasterformat ermöglicht. Hierbei wird die Anordnung der Atome auf der Basis der 2D-Darstellung der Struktur vorgenommen. Als Merkmale eines jeden Atoms dienen sowohl sein kodiertes Elementsymbol als auch seine chemischen Eigenschaften. Die verwendete Netzwerkarchitektur besteht aus *Convolution*-Schichten zur Generierung abstrakterer Merkmale und aus *Dense*-Schichten, die diese Merkmale nutzen, um ein Molekül zu klassifizieren. Durch diese Aufteilung des Netzwerks ist auch ein separates Generieren von Merkmalen möglich, die dann in Kombination mit anderen Methoden genutzt werden können.

Ein weiteres Ziel dieser Arbeit ist die Interpretation dessen, was ein Netzwerk aus den Daten gelernt hat. Hierzu werden in der vorgestellten Methode *Saliency Maps* genutzt, um zu berechnen, welche Bereiche der Eingabedaten den größten Einfluss auf die gewählte Klasse haben. Unter Verwendung dieser Berechnung ist eine Visualisierung der Substrukturen möglich, die laut Netzwerk für die Aktivität des Moleküls verantwortlich sind. Durch ein schwellenwertbasiertes Verfahren wird außerdem eine automatische Extraktion von wichtigen Substrukturen ermöglicht, die mittels einer Bewertung nach ihrer Relevanz sortiert werden.

Die im Zuge dieser Arbeit implementierte Softwarearchitektur wird ebenfalls vorgestellt. Sie ermöglicht eine effiziente Nutzung paralleler Ressourcen, indem die Vorbereitung neuer Daten auf alle Kerne des Prozessors verteilt wird, während die vorherigen Daten gleichzeitig auf der Grafikkarte zum Training verwendet werden. Ebenfalls spei-

chert die Architektur Zwischenergebnisse, die ein einfaches Fortsetzen von abgebrochenen oder abgestürzten Experimenten ermöglichen.

Im experimentellen Teil der Arbeit werden sinnvolle Hyperparameter für die vorgestellte Methode gesucht. Des Weiteren wird die Methode mit gängigen Methoden des *Virtual High-Throughput Screenings* verglichen. In diesem Vergleich zeigt sich, dass die vorgestellte Methode es mit gängigen *Fingerprints* durchaus aufnehmen kann. Auch das Finden von relevanten Substrukturen wird untersucht. Die tatsächlich für die Aktivität relevanten Substrukturen werden hierbei erfolgreich gefunden.

Danksagung

Ich bedanke mich bei allen, die mich bei der Erstellung dieser Arbeit unterstützt haben.

Mein besonderer Dank gilt Prof. Michael Berthold für die Bereitstellung des Themas. Ich danke Prof. Christian Borgelt, Prof. Bastian Goldlücke, Prof. Thomas Mayer, Prof. Michael Berthold und Dr. Gregory Landrum für ihre Betreuung. Dr. Gregory Landrum danke ich außerdem für die Bereitstellung der in dieser Arbeit genutzten Datensätze.

Zur Durchführung des Dissertationsvorhabens hat Prof. Michael Berthold darüber hinaus mit seiner Unterstützung bei der Suche nach einer geeigneten Finanzierung durch die Konstanzer Graduiertenschule für Chemische Biologie und einem anschließenden Stipendium beigetragen – auch hierfür bedanke ich mich bei ihm.

Zu guter Letzt danke ich allen am Lehrstuhl für ihre Unterstützung und eine angemessene Arbeitsatmosphäre.

Inhaltsverzeichnis

Kapitel 1

Einleitung und Motivation

Bei gesundheitlichen Beschwerden steht heute eine große Auswahl an Medikamenten zur Verfügung, um sie zu lindern oder zu beseitigen. Die Wirkung dieser Stoffe findet oft auf Zellebene statt, wobei kleine Moleküle in die Zelle eindringen und dort mit Proteinen interagieren. Der Prozess, bei dem nach neuen Molekülen gesucht wird, die eine spezifische Wirkung hervorrufen, nennt sich Wirkstoffsuche.

Abbildung 1.1 zeigt die vier Schritte der Suche nach neuen Wirkstoffen [29]. Bei der *Target Identification* wird nach einem Zielmolekül im Organismus gesucht, das am Krankheitsprozess beteiligt ist. Durch Labortests wird dann durch die *Target Validation* dessen Anteil am Krankheitsverlauf bestätigt. Bei der *Lead Identification* geht es darum, Moleküle zu finden, die das Verhalten des *Target* auf die gewünschte Weise beeinflussen. Hierfür werden große Bibliotheken an Molekülen in einem *High-Throughput Screening* im Labor getestet. Dabei lässt sich eine oft sehr kleine Anzahl an Molekülen identifizieren, die die gewünschte Reaktion hervorrufen (die sogenannten *Hits*). Deren Struktur wird im Schritt der *Lead Optimization* noch optimiert, um ihre Wirkung zu verbessern, Nebenwirkungen zu verringern und ihre Eignung als Medikament zu erhöhen.

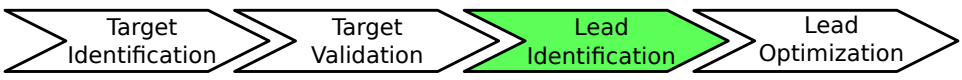


Abbildung 1.1: Die vier wesentlichen Schritte der Wirkstoffsuche.

Im Zentrum der *Lead Identification* steht das *Screening*, bei dem es herauszufinden gilt, welche Moleküle die gewünschte Wirkung zeigen. Dabei wird ein zu testendes Molekül auf das *Target* angewendet und anschließend eine Messung vorgenommen, um zu prüfen, ob die gewünschte Wirkung erzielt wurde. Zur Messung dienen Methoden wie

Fluoreszenz, Lumineszenz und Bildanalyse [46], um einen numerischen Wert für die Stärke der Wirkung zu berechnen. Anschließend wird mittels eines Schwellenwerts bestimmt, welche der Moleküle als *Hits* anzusehen und damit weiter zu erforschen sind.

Die klassische *Lead Identification* wird in Abbildung 1.2 dargestellt. Das *Screening*-Design beschreibt den genauen Aufbau des Experiments. Auf dieser Basis wird dann ein *Screening* anhand der gesamten zur Verfügung stehenden Molekül-Bibliothek ausgeführt. Da das *Screening* hier mit sehr großen Mengen von Molekülen geschieht und dies mit einem sehr hohen Durchsatz einhergeht, der meist durch Automatisierung unter Verwendung von Robotern realisiert wird [27], spricht man von einem *High-Throughput Screening* [31]. Anschließend werden basierend auf den Ergebnissen die *Hits* zur späteren *Lead Optimization* ausgewählt.

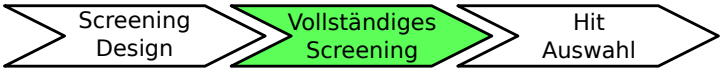


Abbildung 1.2: Bei der klassischen *Lead Identification* wird nach dem *Screening* Design ein vollständiges *Screening* ausgeführt und daraus werden anschließend die *Hits* ausgewählt.

Da ein vollständiges Screening trotz Automatisierung sowohl teuer als auch zeitaufwendig ist, bietet sich die Verwendung von *Virtual High-Throughput Screening* [22] an. Hier wird, wie in Abbildung 1.3 gezeigt, das vollständige *Screening* im Labor durch zwei deutlich kleinere *In-vitro-Screenings* (*Screening im Reagenzglas*) und ein *In-silico-Screening* (*Screening* im Computer) ersetzt. Für das erste, kleinere *In-vitro-Screening* wird eine Menge an möglichst unterschiedlichen Molekülen auf ihr Verhalten im Labor getestet. Die so erhaltenen Ergebnisse werden dann als Trainingsdaten für maschinelles Lernen genutzt, um das *Virtual High-Throughput Screening* auszuführen. Dessen Ziel ist es, die Wahrscheinlichkeit abzuschätzen, ob ein Molekül die gewünschte Wirkung zeigt. Als Ergebnis erhält man eine Rangordnung der noch ungetesteten Moleküle, in der möglichst viele der tatsächlich Wirkung zeigenden Moleküle oben stehen sollten. So kann das Ergebnis als Vorauswahl dienen, um anschließend ein zweites *In-vitro-Screening* mit den n am höchsten eingestuften Molekülen auszuführen. Auf diese Weise hofft man, möglichst viele der Moleküle mit der gewünschten Wirkung zu erhalten, während sich das *Screening* im Labor auf einen Bruchteil der in der Bibliothek enthaltenen Moleküle beschränkt.

Virtual High-Throughput Screening basiert auf der Annahme, dass die Struktur und die Aktivität eines Moleküls zusammenhängen [42]. So können Trainingsdaten mit vor-

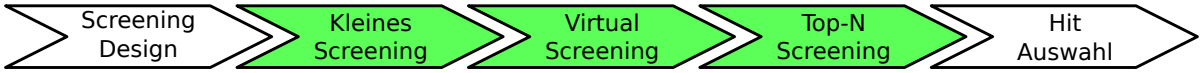


Abbildung 1.3: Bei der Nutzung von *Virtual High-Throughput Screening* wird das vollständige *Screening* durch drei Teilschritte ersetzt. Zuerst wird ein kleines *In-vitro-Screening* von diversen Molekülen ausgeführt, dann erfolgt auf der Basis dieser Daten ein *Virtual High-Throughput Screening* und zum Schluss ein weiteres *In-vitro-Screening* auf den Top-n-Ergebnissen.

liegenden Informationen über deren Struktur und deren tatsächliche Aktivität genutzt werden, um Muster in diesem Zusammenhang zu finden. Anhand dieser Muster lassen sich dann Aussagen über Moleküle mit unbekannter Aktivität treffen. Die Kernaufgabe einer *Virtual-High-Throughput-Screening*-Methode ist es, diese Muster zu erkennen. Eine zusätzliche Schwierigkeit ist dabei, dass in manchen Fällen kleine Änderungen der Struktur große Auswirkungen auf die Aktivität haben können. Deshalb ist eine simple Ähnlichkeitsberechnung hierfür nicht ausreichend.

1.1 Ziele der Forschungsarbeit

Das Ziel der vorliegenden Forschungsarbeit ist es, eine *Virtual-High-Throughput-Screening*-Methode zu entwickeln, die auf die Erfolge von *Deep Learning* insbesondere im Bereich der Bildanalyse aufbaut und die daraus gewonnenen Erkenntnisse für die Analyse von molekularen Strukturen adaptiert. Ein besonderes Augenmerk wird hierbei auf die Generierung von für die Klassifizierung nützlichen Merkmalen gelegt. Des Weiteren wird versucht, das vom *Deep Neural Network* Gelernte zu interpretieren, um die durch das Netzwerk getroffenen Entscheidungen zu erklären und damit auch einen besseren Einblick in die tatsächlichen biologischen Zusammenhänge zu erhalten.

Wesentliche Ergebnisse der vorliegenden Arbeit wurden in folgendem Beitrag zu einem Tagungsband publiziert:

- Patrick Winter, Christian Borgelt und Michael R. Berthold: *Learned Feature Generation for Molecules*. In: *Proceedings of the 17th International Symposium on Intelligent Data Analysis*, S. 380–391, Springer Verlag, 2018. [55]

1.2 Beiträge dieser Forschungsarbeit

Die in dieser Arbeit neu entwickelten Techniken lassen sich wie folgt übersichtlich darstellen:

- Umwandlung der Struktur eines Moleküls in ein Rasterformat unter Verwendung der Positionsberechnung zur Erstellung von 2D-Bildern des Moleküls
- Transformationen der Trainingsdaten basierend auf nötigen Invarianzen für das gewählte Format der Eingabedaten
- Kodierung der Atome und Bindungen als Eingabe-Merkmale
- Hinzufügen von chemischen Eigenschaften als zusätzliche Eingabe-Merkmale
- Spezifisch für diese Methode erstellte Netzwerkarchitektur, die sich in einen merkmalgenerierenden Teil und einen klassifizierenden Teil aufteilen lässt
- Visualisierung von bedeutenden Substrukturen eines Moleküls unter Verwendung von *Saliency Maps*
- Automatische Extraktion und *Scoring* wichtiger Substrukturen basierend auf der *Saliency Map* jedes einzelnen Moleküls im Datensatz
- Entwicklung einer Softwarearchitektur, die eine effiziente und fehlerunanfällige Ausführung von Experimenten ermöglicht

1.3 Aufbau

Die Arbeit ist in drei Teile gegliedert. Im ersten Teil (Kapitel 2–3) werden die bestehenden Grundlagen erklärt, auf die der Inhalt dieser Arbeit aufbaut. Der zweite Teil (Kapitel 4–6) stellt die in dieser Arbeit neu kreierten Methoden vor, die dann im dritten Teil (Kapitel 7) einer experimentellen Evaluation unterzogen werden.

Kapitel 2 führt das nötige Grundwissen zum *Virtual High-Throughput Screening* ein. Hier werden die Informationen über die Struktur von Molekülen erklärt, die in *Screening*-Datensätzen üblicherweise vorhanden sind. Es werden einige weit verbreitete Verfahren gezeigt, wie sich aus der Molekülstruktur Merkmale generieren lassen, die sich zur Klassifikation eignen. Die anschließende Klassifikation wird anhand eines *Random*

Forest erklärt und es wird auf sinnvolle Bewertungsmethoden eingegangen, die sich für die betrachteten Anwendungsfälle eignen.

Die für diese Arbeit relevanten Grundlagen zum *Deep Learning* werden in Kapitel 3 erläutert. Dabei geht ein Überblick auf die Funktionsweise von künstlichen neuronalen Netzen und genauer auf die in dieser Arbeit verwendeten Schichten solcher Netze ein. Anschließend wird die als Inspiration für diese Arbeit verwendete Klassifikation von Bildern dargelegt, um schließlich die Funktionsweise von *Saliency Maps* zu veranschaulichen, die auffällige Muster innerhalb der Eingabedaten aufzeigen, welche die Klassifikation beeinflussen.

Kapitel 4 startet mit der Beschreibung der in dieser Arbeit erstellten Methode. Eine Erläuterung findet dabei die Vorverarbeitung, die aus den vorliegenden Strukturdaten eine rasterbasierte Repräsentation erzeugt. Ebenso wird auf die Transformation der Daten und die in der Repräsentation enthaltenen Merkmale eingegangen. Des Weiteren gilt es, die verwendete Netzwerkarchitektur aufzuzeigen und zu begründen. Zudem wird hier die Aufteilung in einen Teil, der die Generierung von Merkmalen übernimmt, und einen Teil, der für die Klassifizierung verantwortlich ist, aufgezeigt.

Die Nutzung von *Saliency Maps* für die in dieser Arbeit beschriebene Methode erläutert Kapitel 5. Sowohl die Darstellung als auch die automatische Extraktion relevanter Substrukturen wird hier beschrieben und ein Bewertungssystem vorgestellt, um die gefundenen Substrukturen nach ihrer Relevanz zu sortieren.

Kapitel 6 beschreibt die Softwarearchitektur der für Experimente genutzten Implementierung. Insbesondere geht es auf den modularen Aufbau der einzelnen Schritte von Experimenten und das Fortsetzen von Experimenten ein, die zum Teil bereits ausgeführt worden sind. Ebenso werden die effiziente Verwendung paralleler Ressourcen, etwa mehrerer Prozessorkerne, und die parallele Nutzung von Prozessor und Grafikkarte erläutert.

Kapitel 7 stellt die experimentellen Resultate dar und geht dabei auf verschiedene Aspekte der in den vorangehenden Kapiteln beschriebenen Methoden ein. So werden gute Werte für die einzelnen Hyperparameter gesucht, die allgemeine Vorhersagequalität wird evaluiert, der Einfluss verschiedener Vorverarbeitungstechniken beleuchtet, die Anwendbarkeit von übertragendem Lernen überprüft und der Nutzen von *Saliency Maps* betrachtet.

Kapitel 8 fasst die Arbeit zusammen und bietet einen Ausblick auf mögliche zukünftige Forschungsansätze.

Kapitel 2

Virtual High-Throughput Screening

Das Ziel von *Virtual High-Throughput Screening* ist es, die biologische Aktivität von Molekülen abzuschätzen. In dieser Arbeit wird ausschließlich auf Methoden des maschinellen Lernens eingegangen, bei denen auf der Basis von Mustern in bereits bekannten Daten neue Daten klassifiziert werden. Es existieren auch andere Ansätze wie etwa das *Docking* [57], bei dem das chemische Verhalten zwischen dem Zielmolekül und dem zu testenden Molekül simuliert wird.

Bei einem *In-vitro-Screening* werden die im Labor verfügbaren Moleküle darauf getestet, ob sie die gewünschte Wirkung zeigen. Dazu werden *Screening*-Platten mit einer Vielzahl sogenannter *Wells* eingesetzt. Ein *Well* ist ein Schacht, in dem jeweils eines der Experimente stattfindet, also jeweils ein Molekül auf das *Target* angewendet wird. Nachdem dieses Experiment in einer vorbestimmten Zeit und Atmosphäre (insbesondere bei einer bestimmten Temperatur) die Gelegenheit hatte zu wirken, wird es ausgewertet. Dabei kommen verfahren wie Fluoreszenz, Lumineszenz und Bildanalyse zur Anwendung. Deren Ergebnis ist ein numerischer Wert, der besagt, wie stark die gewünschte Wirkung zu beobachten war. Diese Daten werden anschließend von einem Biologen beurteilt. Grundsätzlich erfolgt eine Einteilung der Moleküle in die Klassen Aktiv und Inaktiv gemäß einem gewählten Schwellenwert. Anschließend werden die Ergebnisse begutachtet und durch Erfahrungswerte insbesondere falsch positive Moleküle herausgefiltert. Diese können zum Beispiel auftreten, wenn das Molekül nicht die gewünschte Reaktion mit dem Zielprotein, sondern direkt eine Reaktion mit dem Indikator hervorruft. Auch können Verunreinigungen die Messungen eines ganzen Bereichs einer Platte verfälschen. In diesem Fall sind in *Wells*, die physisch nahe beieinander liegen, ähnliche Werte zu beobachten. Aus diesen Gründen werden die Messwerte nicht direkt als Regressionsproblem

(Modellierung einer numerischen Zielgröße) behandelt, sondern es wird ein Klassifikationsproblem (Modellierung einer nominalen Zielgröße) formuliert, das auf der durch den Biologen vorgenommenen Klassifizierung der Moleküle basiert.

Diese durch den Biologen erstellten Klassen werden zusammen mit den Strukturinformationen der Moleküle als Trainingsdaten genutzt. Für die Moleküle werden anschließend Merkmale generiert, die meistens auf der Struktur basieren. Mit diesen Merkmalen lässt sich dann ein Klassifikator erzeugen, der die Trainingsdaten möglichst korrekt in aktive und inaktive Moleküle aufteilt. Er kann in der Folge auf Moleküle angewendet werden, deren Aktivität bisher unbekannt ist. In den meisten Fällen wird die vom Klassifikator berechnete Wahrscheinlichkeit der Klassenzugehörigkeit zur aktiven Klasse genutzt, um eine sortierte Liste zu erhalten. Es folgt die Auswahl einer bestimmten Anzahl Moleküle, die in dieser Liste am höchsten eingestuft sind. Zu beachten ist, dass in *High-Throughput-Screening*-Daten oftmals nur wenige Moleküle (1–5 %) aktiv und die restlichen inaktiv sind. Bei einigen Klassifikatoren ist es nötig, wegen dieses Ungleichgewichts entweder je nach Methode ausgleichende Klassengewichte festzulegen oder die Gleichgewichtung schon bei der Auswahl der Trainingsdaten zu erzeugen. Geschieht dies nicht ist ein Klassifikator, der ausschließlich die Mehrheitsklasse voraussagt, bereits in den meisten Fällen korrekt. Mit ihm lassen sich allerdings Moleküle nicht voneinander unterscheiden.

2.1 Anwendungsfälle

Drei wichtige Anwendungsfälle von *Virtual High-Throughput Screening* sind (1) die Verringerung der *in vitro* zu testenden Moleküle, (2) die Voreinschätzung von Molekülen, bevor diese synthetisiert oder gekauft werden, und (3) die Prüfung von *in vitro* erhaltenen Ergebnissen, um Hinweise auf mögliche Messfehler zu erhalten.

Der häufigste Anwendungsfall von *Virtual High-Throughput Screening* ist die Reduktion von tatsächlich im Labor zu testenden Molekülen. Hierbei wird in einem ersten Schritt eine kleine Menge möglichst unterschiedlicher Moleküle auf ihre Aktivität im Labor (*in vitro*) getestet, um Trainingsdaten zu erhalten. Anschließend wird ein *Virtual High-Throughput Screening* (*in silico*) durchgeführt, um eine Vorauswahl für einen zweiten Labortest zu treffen. Die so ausgewählten Moleküle werden dann ebenfalls im Labor (*in vitro*) getestet. Je besser die *Virtual-High-Throughput-Screening*-Methode funktioniert, umso mehr der tatsächlich aktiven Moleküle landen in dieser Auswahl. Dadurch wird das Verhältnis der aktiven Moleküle zu den inaktiven erhöht und es können Kosten

gespart werden, ohne dass zu viele der aktiven Moleküle verloren gehen. Es ist hierbei also wichtig, dass in den Top-n-Molekülen möglichst viele der aktiven Moleküle enthalten sind.

Ein weiterer Anwendungsfall ist die Nutzung von *Virtual High-Throughput Screening*, um die Aktivität von Molekülen abzuschätzen, die im Labor gar nicht verfügbar sind. So können ganze Molekül-Bibliotheken auf Aktivität getestet werden, um nur die vielversprechendsten Moleküle dann tatsächlich zu bestellen oder zu synthetisieren. Auch in diesem Fall kommt es darauf an, das in den Top-n möglichst viele der tatsächlich aktiven Moleküle enthalten sind. Im Unterschied zum ersten Anwendungsfall kann hier unter Umständen allerdings auch ein vollständiges *Screening* vorliegen und als Trainingsdaten dienen. Dieses deutlich größere Trainingsdatenset kann, je nach verwendetem Klassifikator, zu einer deutlichen Verbesserung der Vorhersagequalität führen.

Ein dritter möglicher Anwendungsfall ist es, *Virtual High-Throughput Screening* zur Bestätigung von Laborergebnissen beziehungsweise zum Aufdecken von Messfehlern zu nutzen. Da im Labor oftmals viele Störfaktoren auftreten, die zu verfälschten Ergebnissen führen, kann es sinnvoll sein, die gewonnenen Ergebnisse nochmals zu hinterfragen. *Virtual High-Throughput Screening* kann hier eine günstige Methode sein [56]. Wenn für einzelne Moleküle eine allzu große Diskrepanz zwischen dem Laborergebnis und dem durch das *Virtual High-Throughput Screening* abgeschätzten Ergebnis besteht, lohnt es sich, diese nochmals zu testen, um eventuelle Messfehler aufzudecken. In diesem Anwendungsfall kommt es nicht nur darauf an, möglichst viele der tatsächlich aktiven Moleküle in den Top-n zu haben, vielmehr ist auch die Vorhersagequalität für den gesamten Datensatz von Relevanz. Hier ist es ebenfalls möglich, durch Anwendung der *Leave-one-out*-Partitionierung eine große Menge an Trainingsdaten zur Erstellung des Klassifikators zu nutzen. Bei der *Leave-one-out*-Partitionierung werden alle Moleküle außer dem zu klassifizierenden zum Training des Modells verwendet. Das erhaltene Modell kann dann ausschließlich das ausgelassene Molekül klassifizieren. Bei dieser Methode werden maximal viele Daten $(n-1)$ genutzt, um ein möglichst genaues Modell zu bilden.

2.2 Datenquellen

Neben den Datenbanken von großen Pharmazieunternehmen, die üblicherweise strengster Geheimhaltung unterliegen, gibt es auch Datenbanken von Forschungsergebnissen, die

durch öffentliche Mittel finanziert wurden. Zwei besonders nützliche Quellen von Daten über die Aktivität von Molekülen sind PubChem [32, 7] und ChemBL [21, 1].

PubChem ist eine Datenbank, die direkte *Screening*-Ergebnisse aus über 1,2 Millionen *Screenings* enthält. Die Größe dieser *Screenings* kann von wenigen Dutzend bis hin zu mehreren Hunderttausend Molekülen reichen. Insgesamt enthält die Datenbank Ergebnisse für über 94 Millionen unterschiedliche Moleküle. Durch die Nutzung von eindeutigen Bezeichnern können einem Molekül alle zugehörigen Ergebnisse aus unterschiedlichen *Screenings* zugeordnet werden.

Im Gegensatz zu PubChem ist ChemBL eine Datenbank, die auf Ergebnissen in Veröffentlichungen basiert. Es handelt sich um eine kuratierte Datenbank, in der Moleküle, die in Veröffentlichungen erwähnt wurden, mit dem in der Veröffentlichung behandelten *Target* in Verbindung gebracht werden. Insofern beruhen die hier enthaltenen Daten nur indirekt auf *Screening*-Ergebnissen. Auch sind keine Informationen über Moleküle verfügbar, die für das jeweilige *Target* inaktiv sind, da die Veröffentlichungen üblicherweise nur die wenigen aktiven Moleküle behandeln. Der Inhalt der ChemBL Datenbank beruht auf über 67 000 Veröffentlichungen, die über 11 000 unterschiedliche *Targets* behandeln. Über 1,7 Millionen unterschiedliche Moleküle sind in dieser Datenbank aufgeführt.

2.3 Datenstruktur eines Moleküls

Die Struktur eines Moleküls entspricht einem ungerichteten, attributierten Graph. Hierbei sind die Atome die Knoten und die Bindungen die Kanten. Die Knoten haben als Attribut das jeweilige Atomsymbol (chemisches Element, zum Beispiel `C` = Kohlenstoff, `O` = Sauerstoff, `N` = Stickstoff, ...) und die Kanten den jeweiligen Bindungstyp (Einfach-, Zweifach-, Dreifach-, Vierfach- oder aromatische Bindung).

In Datenbanken ist diese Graphstruktur oftmals eindimensional als einfache Zeichenkette abgespeichert, also in einer linearen Darstellung. Eines der häufigsten Formate ist die *Simplified Molecular Input Line Entry Specification* [53, 12], kurz *SMILES*. Die Vorteile von *SMILES* sind die kurze Schreibweise und die leichte Lesbarkeit für den Menschen. *SMILES* beschreibt dabei häufig lediglich die im Molekül enthaltenen Atome und deren Bindungen, kann allerdings auch weitere Informationen wie zum Beispiel deren Ladung enthalten.

Da für diese Arbeit das *SMILES*-Format für die Eingabedaten genutzt wurde, wird dieses im Folgenden genauer beschrieben. Ziel ist es, einen Überblick über die im *SMI-*

LES-Format enthaltenen Informationen zu geben. Die Beschreibung ist nicht ausführlich genug, um als Spezifikation für die Implementierung eines *SMILES*-Parsers zu dienen. Hierfür bietet sich die *OpenSMILES specification* [12] an.

In *SMILES* werden Atome durch ihr jeweiliges Elementsymbol repräsentiert, also durch einen Einzelbuchstaben (zum Beispiel `C` für Kohlenstoff) oder ein Buchstabenpaar (zum Beispiel `Cl` für Chlor). Das Elementsymbol wird im Allgemeinen in eckige Klammern gesetzt. In dieser Klammerdarstellung können optional weitere Informationen über das Isotop, die Ladung und die Anzahl verbundener Wasserstoffatome enthalten sein. Die chemischen Elemente `B`, `C`, `N`, `O`, `F`, `P`, `S`, `Cl`, `Br` und `I`, aus denen organische Moleküle bestehen, können bei Nichtangabe weiterer Informationen auch ohne eckige Klammern, also nur durch ihr Elementsymbol dargestellt werden.

Isotope werden durch eine Zahl vor dem Elementsymbol bezeichnet. Wasserstoffatome gibt *SMILES* üblicherweise nicht mit an, sie werden daher anhand der Valenz implizit angenommen. Sie können allerdings auch explizit mit einem `H` hinter dem Elementsymbol gekennzeichnet werden. Die Zahl hinter dem `H` gibt an, wie viele Wasserstoffatome mit dem jeweiligen Atom verbunden sind; steht hier keine Zahl, wird von einem einzigen Wasserstoffatom ausgegangen. Die Ladung eines Atoms kann durch ein Plus (`+`) oder Minus (`-`) angegeben werden. Wenn auf das Zeichen eine Zahl folgt, zeigt sie die Stärke der Ladung an; fehlt sie, wird von einer Ladung von 1 ausgegangen. Wird gar keine Ladung spezifiziert, ist das Atom elektrisch neutral.

Ein Atom ist jeweils mit den Atomen verbunden, die direkt davor oder danach stehen. Im Normalfall ist eine Einfachbindung impliziert. Sie kann auch explizit mit dem Symbol `-` zwischen den beiden Atomen dargestellt werden. Für Mehrfachbindungen existieren die Symbole `=` (Zweifachbindung), `#` (Dreifachbindung) und `$` (Vierfachbindung). Des Weiteren existiert auch das Symbol `.` (keine Bindung, auch Nullbindung genannt), das genutzt werden kann, um *SMILES* zu erstellen, die mehrere nicht miteinander verbundene Strukturen enthalten.

Eines der wichtigsten Konzepte von *SMILES* ist die Syntax für die nicht lineare Bindung von Atomen durch Verzweigungen und Markierungen. Verzweigungen werden durch den Einsatz von runden Klammern dargestellt. Ein Atom vor einer Verzweigung ist jeweils mit dem ersten Atom in der Verzweigung (erstes Atom nach der öffnenden Klammer) und dem ersten Atom nach der Verzweigung (erstes Atom nach der schließenden Klammer) verbunden. Da mit diesem Konzept noch keine zyklischen Verbindungen dargestellt werden können, gibt es zusätzlich das Konzept der Markierungen. Diese tre-

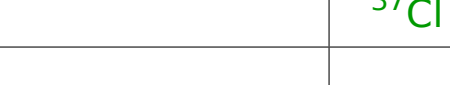
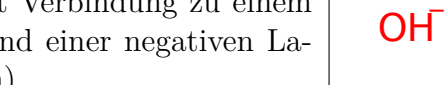
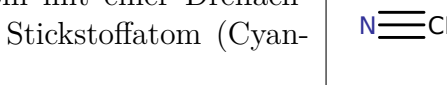
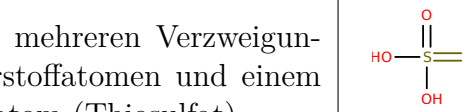
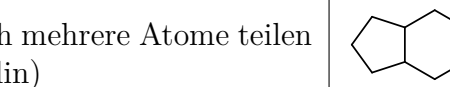
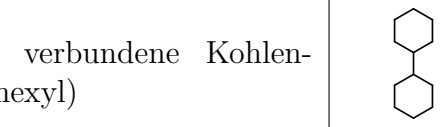
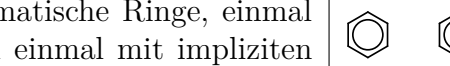
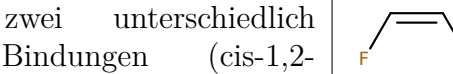
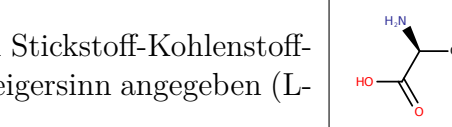
ten immer in Paaren auf. Die jeweils vor einem der beiden Markierungen stehenden Atome sind miteinander verbunden. Eine Markierung wird durch eine ein- oder zweistellige Zahl repräsentiert. Sollte eine Markierung nicht einstellig sein, muss vor der Zahl ein Prozentzeichen (`%`) stehen. Andernfalls würde bei einer zweistelligen Zahl von zwei unabhängigen Markierungen ausgegangen. Ist eine Markierung innerhalb eines *SMILES* bereits zweifach aufgetaucht, gilt sie als geschlossen und kann anschließend erneut verwendet werden.

Aromatizität (das Vorhandensein einer Bindung mit der Eigenschaft von Aromaten) kann sowohl durch sich abwechselnde Ein- und Zweifachbindungen (Kekulé-Formel) als auch durch einen Doppelpunkt (`:`) für aromatische Bindungen dargestellt werden. Außerdem können die Elementsymbole der Atome, die sich in einem aromatischen Ring befinden, auch durch Kleinschreibung gekennzeichnet werden. In diesem Fall ist keine Angabe der Bindung nötig, und es wird implizit von einer aromatischen Bindung ausgegangen statt wie sonst von einer Einfachbindung.

Stereochemische Eigenschaften, also Eigenschaften, die aus der räumlichen Struktur des Moleküls, der dreidimensionalen Anordnung seiner Atome folgen, können ebenfalls angegeben werden. So kann mit den Zeichen `/` und `\` die Richtung von Bindungen außerhalb einer nicht rotierbaren Substruktur angegeben werden. Damit eine Substruktur nicht rotierbar ist, müssen die inneren Bindungen Mehrfachbindungen sein. Ein `/` deutet darauf hin, dass das folgende Atom über seinem Vorgänger liegt, und `\`, dass es unter ihm liegt. Wird die *SMILES*-Zeichenkette also in einer Linie von einem äußeren Atom durch die nicht rotierbare innere Struktur bis hin zum anderen äußeren Atom angegeben, dann deuten zwei gleich gerichtete Bindungen darauf hin, dass sich die beiden äußeren Atome auf unterschiedlichen Seiten befinden. Sind die beiden gerichteten Bindungen unterschiedlich, zeigt dies auf, dass sich beide äußeren Atome auf derselben Seite befinden. Ebenso kann mit dem `@`-Zeichen angegeben werden, ob die folgenden Atome aus Sicht eines Atompaars entgegen dem Uhrzeigersinn (`@`) oder im Uhrzeigersinn (`@@`) aufgelistet sind. Hierbei wird bei dem Atom, mit dem die folgenden Atome verbunden sind, hinter dem Elementsymbol `@` oder `@@` angegeben.

In Tabelle 2.1 werden verschiedene Beispiele für die beschriebenen Konstrukte in *SMILES* wiedergegeben. Abbildung 2.1 zeigt eine *SMILES*-Zeichenkette mit entsprechender Netzwerk- und Bild-Darstellung.

Tabelle 2.1: *SMILES*-Beispiele.

SMILES	**Beschreibung**	**Struktur**
`[Mg]`	Einzelnes Magnesiumatom	
`Cl`	Einzelnes Chloratom	
`[37Cl]`	^{37}Cl Isotop	
`[OH-]`	Sauerstoffatom mit Verbindung zu einem Wasserstoffatom und einer negativen Ladung (Hydroxidion)	
`C#N`	Ein Kohlenstoffatom mit einer Dreifachbindung zu einem Stickstoffatom (Cyanwasserstoff)	
`OS(=O)(=S)O`	Schwefelatom mit mehreren Verzweigungen zu drei Sauerstoffatomen und einem weiteren Schwefelatom (Thiosulfat)	
`N1CC2CCCC2CC1`	Zwei Ringe, die sich mehrere Atome teilen (Perhydroisoquinolin)	
`C1CCCCC1C1CCCCC1`	Zwei miteinander verbundene Kohlenstoffringe (Bicyclohexyl)	
`C1:C:C:C:C:C:1.c1ccccc1`	Zwei separate aromatische Ringe, einmal mit expliziten und einmal mit impliziten aromatischen Bindungen (Benzolringe)	
`F/C=C\F`	Molekül mit zwei unterschiedlich ausgerichteten Bindungen (cis-1,2-Difluoroethene)	
`N[C@@H](C)C(=O)O`	Die Atome an dem Stickstoff-Kohlenstoff-Paar sind im Uhrzeigersinn angegeben (L-Alanin)	

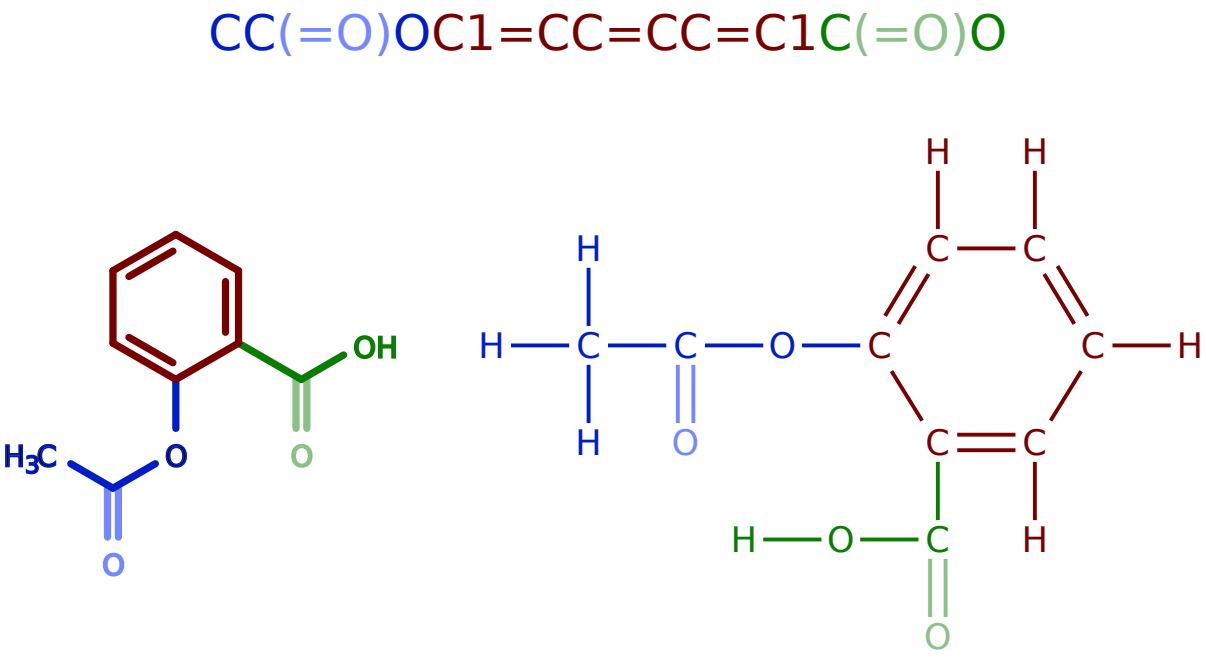


Abbildung 2.1: Der *SMILES* für Aspirin (oben) mit der Bild-Darstellung (links) und der Netzwerk-Darstellung (rechts). Dieselben Teilstrukturen sind in allen Darstellungen mit gleicher Farbe hervorgehoben.

2.4 Merkmalsgenerierung für Moleküle

Ein Großteil der existierenden Methoden maschinellen Lernens benötigt numerische Merkmale. Um diese auf Molekül-Daten anzuwenden, ist es daher erforderlich, aus der Struktur numerische Merkmale zu generieren. Diese Merkmale sollten die für die jeweilige Aufgabe wichtigen Eigenschaften möglichst gut beschreiben. Im Fall von *Virtual High-Throughput Screening* heißt das, dass anhand dieser Merkmale eine gute Unterscheidung zwischen aktiven und inaktiven Molekülen möglich sein soll.

Eine einfache Option, um numerische Merkmale für molekulare Strukturen zu erhalten, ist die Extraktion von molekularen Deskriptoren [52]. Diese können sowohl auf Struktureigenschaften basieren, wie zum Beispiel der Anzahl enthaltener Ringe, als auch auf chemischen Eigenschaften wie dem molekularen Gewicht. Durch die Kombination aus einer Auswahl dieser Deskriptoren erhält man einen Vektor, der das jeweilige Molekül beschreibt. In Tabelle 2.2 werden einige molekulare Deskriptoren vorgestellt.

Eine Alternative zur Verwendung einer Auswahl an molekularen Deskriptoren sind *Fingerprints*. Die Zielsetzung eines *Fingerprints* ist es, einen das Molekül beschreibenden Vektor zu erzeugen, der möglichst einzigartig für das jeweilige Molekül ist. Dabei

Tabelle 2.2: Einige Beispiele molekularer Deskriptoren.

Name	**Beschreibung**
Number Atoms	Anzahl der enthaltenen Atome
Number Heavy Atoms	Anzahl der enthaltenen Atome ausschließlich Wasserstoffatome
Number Rings	Anzahl der enthaltenen Ringe
Number Aromatic Rings	Anzahl der enthaltenen aromatischen Ringe
Number Rotatable Bonds	Anzahl der Bindungen, die rotiert werden können, da keine andere Bindung innerhalb des Moleküls dies verhindert
Hydrogen Bond Donors	Anzahl an Wasserstoffbrücken-Donatoren (Atome, die mit einem Wasserstoffatom gebunden sind, das eine Wasserstoffbrücke aufbauen kann)
Hydrogen Bond Acceptors	Anzahl an Wasserstoffbrücken-Akzeptoren (Atome, die sich mit einem Wasserstoffbrücken-Donator verbinden können)
Molecular Weight	Gesamtgewicht des Moleküls
Accessible Surface Area	Oberfläche des Moleküls, die mit anderen Molekülen interagieren kann
logP	Verhältnis zwischen Fettlöslichkeit und Wasserlöslichkeit

beschreiben die einzelnen Werte des Vektors oftmals (aber nicht zwangsweise) die Anwesenheit oder Abwesenheit bestimmter Substrukturen. Im Folgenden werden beispielhaft zwei unterschiedliche Verfahren zur Erzeugung von *Fingerprints* vorgestellt.

Der *MACCS-Fingerprint* [19] (*Molecular ACCess System Fingerprint*) besteht aus einem 166 Bit großen Vektor, in dem jeder Position eine spezifische Abfrage zugeordnet ist. Welche 166 Abfragen gestellt werden sollen und welche Aspekte eines Moleküls hier somit als die wichtigsten angenommen werden, wurde durch chemisches Expertenwissen festgelegt. Die Erstellung des *Fingerprints* ist dadurch besonders einfach. Jede Abfrage bestimmt, ob eine bestimmte Teilstruktur im Molekül vorkommt oder nicht. Ist sie vorhanden, ist das Bit an der jeweiligen Stelle 1, ansonsten 0.

Einige Beispiele für solche Abfragen sind in Tabelle 2.3 zu finden. Sie basieren auf der Implementierung in *RDKit* [35] und werden in Form eines *SMARTS (SMILES arbitrary target specification)* [11] definiert. *SMARTS* ist eine Abfragesprache für Moleküle, die auf *SMILES* aufbaut. Nachfolgend werden einige der zum Verständnis von Tabelle 2.3 benötigten Sprachkonstrukte erklärt. Wie in *SMILES* werden einzelne Atome in eckigen Klammern mit zusätzlichen Informationen beschrieben. Atome können mittels

ihres Symbols geschrieben werden. Bei Kleinschreibung handelt es sich um ein aromatisches Atom (Teil eines aromatischen Rings) und bei Großschreibung um ein aliphatisches (nicht Teil eines aromatischen Rings). Um sich nicht auf eine dieser beiden Eigenschaften festzulegen, kann das Atom auch mittels seiner Ordnungszahl mit voranstehendem `#` geschrieben werden. Um ein beliebiges Atom anzugeben, lassen sich ein `*` oder `A` (für aliphatisch) oder ein `a` (für aromatisch) verwenden. Bindungen nutzen dieselben Zeichen wie in *SMILES*. Es existiert zusätzlich das Zeichen `~`, das für eine beliebige Bindung steht. Um eine Liste von möglichen Atomen anzugeben, können diese mit einem `,` aufgelistet werden. Mit einem `;` werden Bedingungen für ein jeweiliges Atom angegeben. So steht zum Beispiel ein `R` dafür, dass das Atom innerhalb eines Rings liegt. Weitere Informationen über *SMARTS* lassen sich in [11] nachlesen.

Tabelle 2.3: Einige Beispiele für *MACCS*-Abfragen.

Position	**SMARTS**	**Beschreibung**
11	`*1~*~*~*~1`	Ring der Größe 4
20	`[#14]`	Silizium
24	`[#7]-[#8]`	Stickstoff mit einer Einfachbindung zu Sauerstoff
47	`[#16]~*~[#7]`	Schwefel mit Bindung auf ein beliebiges Atom, das eine Bindung zu Stickstoff hat
121	`[#7;R]`	Stickstoff innerhalb eines Rings
134	`[F,Cl,Br,I]`	Halogen
162	`a`	Aromatisches Atom

Ein weiterer häufig genutzter *Fingerprint* ist der *Extended Connectivity Fingerprint* [41]. In diesem werden Substrukturen innerhalb des Moleküls extrahiert, um daraufhin eine zugehörige Position innerhalb des Vektors zu berechnen. Es gibt vier Parameter für diesen *Fingerprint*. Zum einen lässt sich die Anzahl der Werte innerhalb des Vektors konfigurieren. Ebenso kann man wählen, ob die Werte binär sind, also nur die An- oder Abwesenheit beschreiben, oder ob sie Ganzzahlen sind, die die Häufigkeit des Auftretens repräsentieren. Im Falle eines Bitvektors wird die Kurzform *ECFP* genutzt, im Falle eines zählenden Vektors die Kurzform *ECFC*. Mit dem Radius-Parameter wird bestimmt, wie groß die betrachteten Substrukturen sein können. Dabei beschreibt der Radius die Entfernung von Atomen basierend auf deren Nachbarschaft. In einem Radius von 2 liegen zum Beispiel alle direkten Nachbarn und deren direkte Nachbarn. Der Radius kann in der gängigen Kurzform wie zum Beispiel *ECFP4* abgelesen werden. Hierbei ist allerdings zu beachten, dass die Zahl am Ende den Durchmesser und nicht den Radius beschreibt. Ein *ECFP4* verwendet also einen Radius von 2. Der vierte Parameter beinhaltet die

Eigenschaften der Substruktur, die genutzt werden, um ihre Position im Vektor zu bestimmen. Der Vektor wird berechnet, indem über alle Atome außer Wasserstoff iteriert wird. Das jeweilige Atom wird als Zentrum der Substruktur gesehen, und es werden weitere Atome innerhalb eines Radius hinzugefügt. Hierbei wird jeweils der Radius von 0 bis hin zu dem als Parameter angegeben Radius verwendet. Für jede Substruktur lassen sich die per Parameter gewählten Eigenschaften berechnen, daraus lässt sich wiederum ein *Hash*-Wert bilden, der durch eine Modulooperation in einen Wert umgewandelt wird, der innerhalb der Größe des Vektors liegt. Dieser Wert stellt die dieser Substruktur zugeordnete Position dar. Mit dieser Vorgehensweise landen gleiche Substrukturen immer in der gleichen Position. Es ist aber zu beachten, dass durch die Modulooperation auch andere Substrukturen dieselbe Position bekommen können. Das heißt entsprechend auch, dass gleiche Moleküle immer denselben *Fingerprint* bekommen, ein gleicher *Fingerprint* allerdings nicht garantiert, dass es sich um dasselbe Molekül handelt. Jedoch ist wahrscheinlich, dass sie sich zumindest ähnlich sind. Abbildung 2.2 zeigt, wie die Auswahl der Substrukturen um ein bestimmtes Atom für die Radien 0 bis 2 funktioniert. In Listing 2.1 ist die grundsätzliche Vorgehensweise bei der Berechnung eines *Extended Connectivity Fingerprints* zu ersehen.

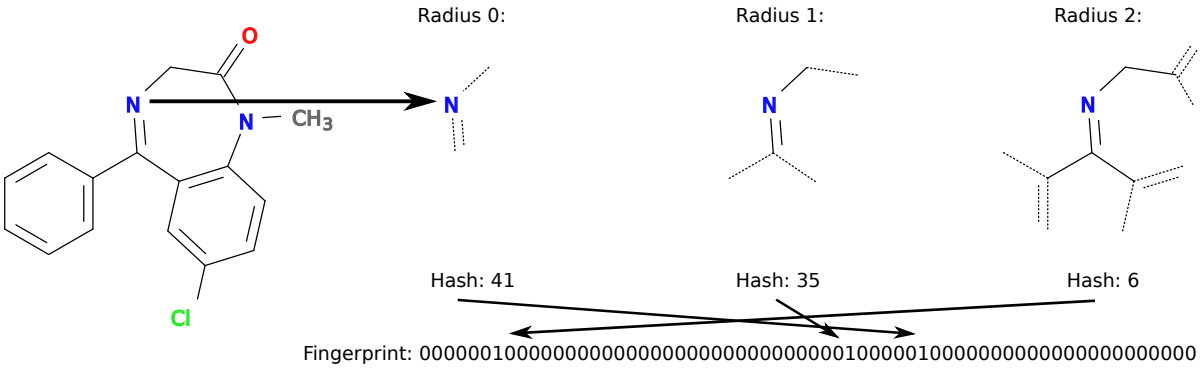


Abbildung 2.2: Auswahl der Substrukturen um das gewählte Stickstoffatom. Es wird ein *ECFP4* betrachtet. Die Radien um das Atom gehen also von 0 bis 2. Für die extrahierten Substrukturen wird ein *Hash*-Wert berechnet. Das Auftreten einer Substruktur mit diesem *Hash*-Wert wird anschließend im *Fingerprint* an dieser Position durch eine 1 markiert.

Eine weitere Möglichkeit ist es, häufige Substrukturen aus einem Datensatz zu extrahieren, die sich zum Unterscheiden von aktiven und inaktiven Molekülen eignen. Ein solcher Ansatz ist *MoSS* (*Molecular SubStructure Miner*) / *MoFa* (*Molecular Fragment*

```
def ecfp(molecule, size, binary, radius, attributes):
  fingerprint = numpy.zeros(size)
  for atom in molecule.GetAtoms():
    for r in range(radius):
      substructure_atoms = set()
      substructure_atoms.add(atom)
      for i in range(r):
        for substructure_atom in substructure_atoms:
          for neighbor in substructure_atom.GetNeighbors():
            substructure_atoms.add(neighbor)
      substructure = extract_substructure(molecule, substructure_atoms)
      hash_value = calculate_attribute_hash(substructure, attributes)
      position = hash_value % size
      if binary:
        fingerprint[position] = 1
      else:
        fingerprint[position] += 1
  return fingerprint
```

Listing 2.1: Python [8]-Code, der die Erstellung eines *Extended Connectivity Fingerprints* verdeutlicht.

Miner) [13]. Hier werden häufige Substrukturen gefunden, und es wird gezählt, wie häufig diese unter den aktiven und wie häufig sie unter den inaktiven Molekülen auftreten. Substrukturen, bei denen der Unterschied dieser beiden Metriken besonders hoch ist, eignen sich entsprechend gut als Merkmal zur Klassifizierung. Das Interessante an diesem Ansatz ist, dass bereits bei der Generierung von Merkmalen die Klasseninformationen genutzt werden, um Merkmale zu erzeugen, die sich für diese Aufgabe besonders eignen.

2.5 Klassifikator

Die Aufgabe eines Klassifikators ist es, Datenpunkten, deren Klasse unbekannt ist, die korrekte Klasse zuzuweisen. Dazu benötigt ein Klassifikator während der Trainingsphase einen Trainingsdatensatz mit bereits bekannten Klassen. Anhand der für diesen Datensatz bekannten Merkmale versucht er, Muster zu lernen, die für die jeweilige Klasse typisch sind. Einem Datenpunkt mit unbekannter Klasse wird dann die Klasse zugewiesen, deren gelerntes Muster am ehesten in den Merkmalen wiederzufinden ist. Entsprechend

dieser Ähnlichkeit kann üblicherweise auch eine Wahrscheinlichkeit der Klassenzugehörigkeit berechnet werden.

Ein Klassifikator, der häufig für das *Virtual High-Throughput Screening* eingesetzt wird, ist der *Random Forest* [16]. Die Vorteile eines *Random Forest* sind seine hohe Vorhersagequalität [20] und der Umstand, dass er auch ohne große Anpassungen der Parameter oft schon zu sehr guten Resultaten führt.

Ein *Random Forest* ist ein *Ensemble* von Entscheidungsbäumen. Für jeden zu klassifizierenden Datenpunkt berechnet jeder Baum des *Ensembles* einzeln eine Vorhersage. Diese Vorhersagen werden dann zu einer Klassifizierung aggregiert. Eine Möglichkeit ist es, die am häufigsten gewählte Klasse als Ergebnis zu wählen und entsprechend die Klassen-Wahrscheinlichkeit anhand der Stimmen pro Klasse festzulegen. Dieses Verfahren wird *Hard Voting* oder auch *Majority Voting* genannt. Die Alternative besteht darin, den Durchschnitt der einzelnen Klassen-Wahrscheinlichkeiten zu nutzen und entsprechend die Klasse mit der höchsten Durchschnittswahrscheinlichkeit zu wählen. Dieses Verfahren nennt sich *Soft Voting*.

Ein Entscheidungsbaum (siehe Abbildung 2.3) ist ein Baum, der aus hierarchischen, aufeinander folgenden Entscheidungen besteht. Zur Klassifizierung eines Datenpunktes wird der Baum von der Wurzel bis zu einem der Blätter durchlaufen. Jeder innere Knoten beschreibt eine Abfrage einer Eigenschaft des zu klassifizierenden Objektes/Datenpunktes. Bei numerischen Daten ist dies typischerweise, ob der Wert eines bestimmten Merkmals einen bestimmten Schwellenwert erreicht oder überschreitet oder ob er unter dem Schwellenwert liegt. Je nach dem Ergebnis der Abfrage wird zu dem zugehörigen Kind des inneren Knotens übergegangen. Auf diese Weise durchläuft ein Datenpunkt den Baum, bis er an einem Blatt angekommen ist. Diesem Blatt sind Klassenwahrscheinlichkeiten zugeordnet, die auf den Trainingsdaten basieren. Die Klasse mit der höchsten Wahrscheinlichkeit wird dem Datenpunkt zugewiesen.

Um einen Entscheidungsbaum zu erzeugen, wird an jedem Knoten die bestmögliche Aufteilung berechnet. Das Ziel einer Aufteilung ist es, Folgeknoten zu erzeugen, deren Klassenverteilung möglichst homogen ist. Dazu werden alle zur Verfügung stehenden Merkmale betrachtet und es wird der beste Schwellenwert für dieses Merkmal ermittelt, anhand dessen anschließend alle Datenpunkte aufgeteilt werden. Mittels eines Maßes wie *Information Gain*, *Information Gain Ratio* [39] oder des Gini-Indexes [23, 17] wird dann die Qualität der Aufteilung beurteilt. Anschließend werden für die neu erzeugten Knoten wieder Aufteilungen berechnet, bis keine weitere Aufteilung mehr sinnvoll oder

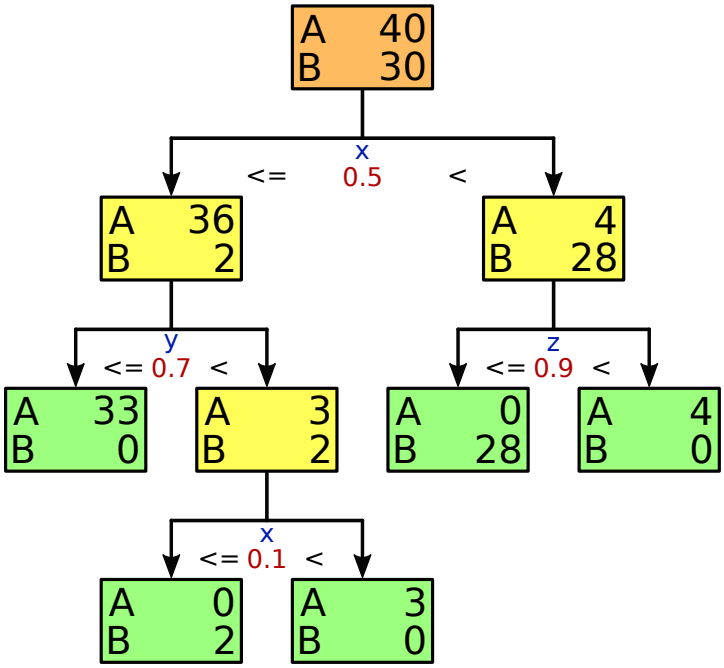


Abbildung 2.3: Ein Entscheidungsbaum, der Daten die Klassen A oder B zuweist. Für einen Datenpunkt wird bei jeder Aufteilung geprüft, ob der Wert des Merkmals (blau) eines Datenpunkts unter oder über dem Schwellenwert (rot) liegt. Entsprechend durchläuft der Datenpunkt angefangen bei der Wurzel (orange) die inneren Knoten (gelb), bis er in einem Blatt (grün) endet. Die dort dominierende Klasse wird ihm dann zugewiesen.

möglich ist. Wird ein Baum zu Ende gelernt, dann sind in jedem Blatt jeweils nur noch Datenpunkte derselben Klasse vorhanden. Das Lernen kann durch Parameter allerdings auch früher gestoppt werden. So lässt sich festlegen, dass der Baum nur bis zu einer bestimmten Tiefe wachsen darf. Auch kann die Anzahl an Datenpunkten pro Blatt auf ein Minimum gesetzt werden, so dass Aufteilungen, die zu kleineren Knoten führen, nicht mehr erlaubt sind.

Da durch das Lernen auf einem bestimmten Datensatz immer derselbe Entscheidungsbaum erzeugt wird, ist es im *Random Forest* nötig, einen Zufallsfaktor einzubringen, damit sich die enthaltenen Bäume unterscheiden. Dies wird zum einen dadurch erreicht, dass für jeden Baum nur eine zufällig gewählte Teilmenge der Daten zum Training genutzt wird. Zum anderen steht einem Baum beim Training auch nur eine zufällige Teilmenge an Merkmalen für eine Aufteilung zur Verfügung. Diese Teilmenge wird für jede einzelne Aufteilung neu bestimmt. Mit dieser Strategie ist gewährleistet, dass sich die Bäume unterschiedlich bilden. Das Ziel ist es, dass die unterschiedlichen Bäu-

me möglichst unterschiedliche Fehler produzieren. Diese einzelnen Fehler werden dann idealerweise durch die anderen Bäume überstimmt und somit die richtige Klasse gewählt.

Die Parameter eines *Random Forest* sind also die Anzahl an Entscheidungsbäumen, der Umstand, ob ein *Hard* oder *Soft Voting* genutzt wird, die vom Entscheidungsbaum geerbte minimale Anzahl an Datenpunkten innerhalb eines Blatts und die Limitierung der Baumtiefe.

2.6 Bewertungsmethoden

Die für das *Virtual High-Throughput Screening* genutzten Bewertungsmethoden sind sowohl durch die ungleiche Klassenverteilung als auch durch den Anwendungsfall begründet. Die sonst häufig genutzte Metrik der Klassifizierungsgenauigkeit, also wie viel Prozent der zugeteilten Klassen der Grundwahrheit entsprechen, eignet sich nicht für stark ungleich gewichtete Daten. Bei 1 % aktiven und 99 % inaktiven Molekülen wäre ein Klassifikator, der ausschließlich inaktiv voraussagt, mit 99 % Genauigkeit fast nicht zu schlagen, würde aber in der Praxis keinerlei nützliche Informationen liefern. Aus diesem Grund sollte die genutzte Metrik eine Sensibilität auf die wenigen, aber wichtigen aktiven Moleküle vorweisen. Des Weiteren ist in manchen der Anwendungsfälle gar nicht von Interesse, wie gut die gesamte Klassifikation ist, sondern nur, dass in den Top-n ausgewählten Molekülen möglichst viele aktive vorhanden sind. Eine aus diesen Gründen viel genutzte Evaluierungsmethode ist der *Enrichment Plot*.

Der *Enrichment Plot* ist ein Liniendiagramm, dass die Anzahl gewählter Moleküle (x-Achse) gegen die Anzahl aktiver Moleküle (y-Achse) darstellt. Dabei sind die Moleküle nach ihrer Wahrscheinlichkeit, aktiv zu sein, sortiert. Das y an einem bestimmten Punkt x entspricht also der Anzahl aktiver Moleküle in den Top-x klassifizierten Daten. In diesem Plot entspricht die Diagonale vom Nullpunkt zum jeweiligen Maximum der x-Achse und y-Achse dem Erwartungswert einer zufälligen Auswahl. Das Optimum entspricht der Diagonalen vom Nullpunkt zum Maximum von y auf beiden Achsen, da in diesem Fall alle aktiven vor allen inaktiven Molekülen stehen und dadurch nach der Wahl von y Molekülen (x-Achse) y aktive Moleküle (y-Achse) gefunden wurden.

Aus den y-Werten des Erwartungswertes der zufälligen und der Top-n-Auswahl an einer jeweiligen x-Position lässt sich der *Enrichment Factor* [26] berechnen. Es handelt sich bei ihm um den Faktor, mit dem das y der zufälligen Auswahl multipliziert wird, um

das y der Top-n-Auswahl zu erhalten. Er bezieht sich dabei immer auf einen bestimmten Prozentsatz der Gesamtzahl an Molekülen. Mit

$$\mathrm{EF}\,p = \frac{y^{\text{TOP-}x}}{y^{\text{RANDOM-}x}}, \qquad x = \mathrm{round}\left(\frac{n \cdot p}{100}\right) \tag{2.1}$$

wird der *Enrichment Factor* berechnet. Hierbei ist EFp der *Enrichment Factor* bei $p\,\%$ der gesamten Anzahl Moleküle n. y ist die Anzahl aktiver Moleküle in der Auswahl der besten x (TOP-x) beziehungsweise in der zufälligen Auswahl von x (RANDOM-x) Molekülen. Abbildung 2.4 zeigt einen *Enrichment Plot* mit eingezeichnetem *Enrichment Factor* bei 10 %. Der *Enrichment Factor* ist ein Maß, das direkt die Qualität von Top-n misst. Da ein gutes Top-n zu einem frühen Finden von aktiven Molekülen bei einem Durchgehen der Rangliste von oben führt, spricht man hier auch von der Früherkennung. Diese ist gerade in Anwendungsfällen wichtig, in denen später ausschließlich die Top-n-Moleküle für weitere Experimente genutzt werden.

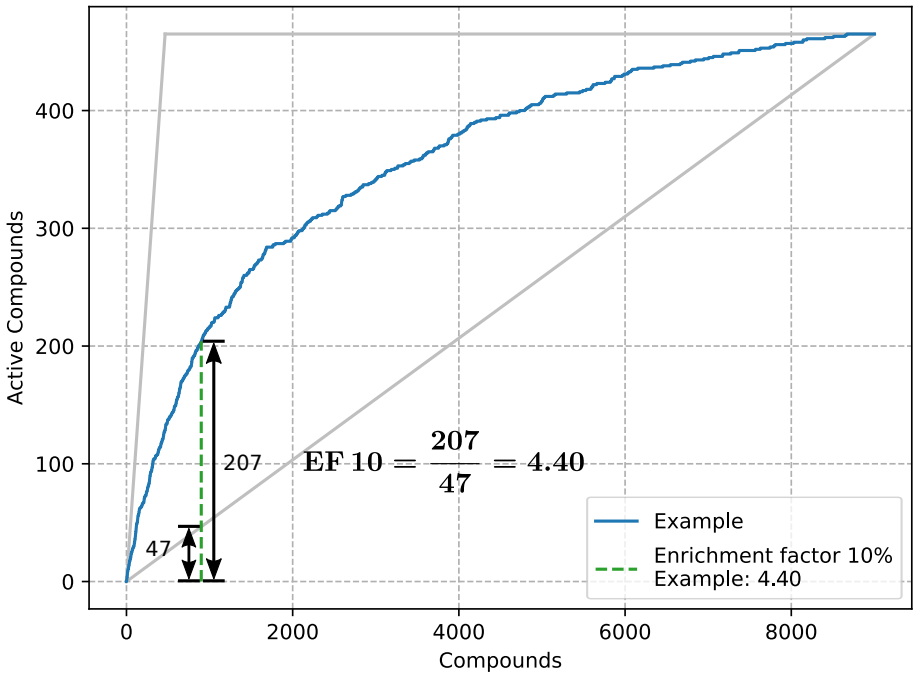


Abbildung 2.4: Ein *Enrichment Plot* mit eingezeichnetem *Enrichment Factor* bei 10 % (*EF10*). Es wird aufgezeigt, wie der *Enrichment Factor* mittels der Formel am konkreten Beispiel berechnet wird.

In Anwendungsfällen, in denen die Qualität der gesamten Klassifikation wichtig ist, bietet es sich an, die *Area under the Curve* als Metrik zu nutzen. Da beim *Enrichment*

Plot allerdings die maximal mögliche *Area under the Curve* vom Verhältnis zwischen aktiven Molekülen und der Gesamtzahl der Moleküle abhängt, wird hier stattdessen meist die *Area under the Curve* der *ROC Curve* (*Receiver Operating Characteristic Curve*) [14] genutzt. Diese hat klarere Grenzen mit 1.0 als Maximum und 0.5 als theoretischem Minimum (Erwartungswert der Leistung von zufälliger Auswahl). Der Unterschied zum *Enrichment Plot* ist im Grunde der, dass die x-Achse nur die inaktiven statt alle Moleküle zählt. Somit erreicht eine perfekte Methode, bei der alle aktiven Moleküle vor alle inaktiven sortiert werden, eine *Area under the Curve* von 1.0. Des Weiteren sind die Achsen auf einen Wert von 0 bis 1 normalisiert. Dies hat auf die *Area under the Curve* aber keinen Einfluss. Ein Beispiel *ROC Curve Plot* ist in Abbildung 2.5 zu sehen.

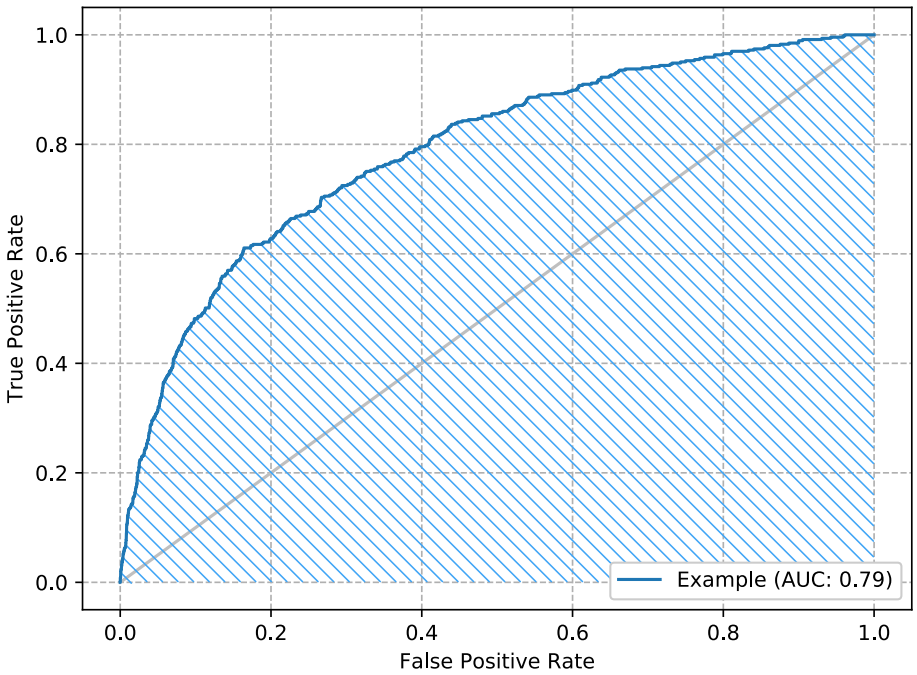


Abbildung 2.5: Ein *ROC Curve Plot* mit markierter *Area under the Curve*.

In den Experimenten dieser Arbeit werden der *Enrichment Factor* (kurz *EF*) und die *Area under the ROC Curve* (kurz *AUC*) als Metriken genutzt. So kann sowohl die Früherkennung als auch die Qualität der gesamten Klassifizierung beurteilt werden.

2.7 Granularität der Klassenwahrscheinlichkeit

Im Zuge dieser Arbeit hat sich herausgestellt, dass die Granularität der Klassenwahrscheinlichkeit eine bedeutende Auswirkung auf das Ergebnis eines *Virtual-High Throughput Screenings* haben kann. Viele Klassifikatoren haben einen Fokus auf eine möglichst korrekte Klassifizierung. Die Klassenwahrscheinlichkeiten werden meist nur als durch die Methode entstehende Zusatzinformationen angesehen. Da sie allerdings beim *Virtual High-Throughput Screening* die eigentlich genutzte Information sind, gilt es hier besonders aufzupassen. Besonders die Granularität der Klassenwahrscheinlichkeit spielt für ein gutes Ergebnis eine wichtige Rolle. Dies wird im Folgenden am Beispiel eines *Random Forest* erläutert.

Obwohl ein *Random Forest* wie bereits erwähnt in den meisten Fällen schon ohne Parameteranpassung zu sehr guten Ergebnissen führt, kann er gerade beim *Virtual High-Throughput Screening* in Probleme laufen. Mit Standardparametern werden die Bäume im *Random Forest* so trainiert, dass jedes Blatt nur noch Datenpunkte einer Klasse enthält. Damit kann ein einzelner Baum als Wahrscheinlichkeit auch nur 0 % oder 100 % ausgeben. Des Weiteren nutzt ein *Random Forest* zur Aggregation der Einzelergebnisse mit Standardparametern ein *Hard Voting*. Hierbei werden feiner granulare Wahrscheinlichkeiten der einzelnen Bäume ignoriert und die Wahrscheinlichkeit allein an der Anzahl der für die Klasse stimmenden Bäume festgemacht. Das heißt, eine Klasse kann von einer Anzahl von 0 bis n Bäumen gewählt werden, womit $n + 1$ verschiedene Werte möglich sind. Des Weiteren wird der Wertebereich wegen der ungleichen Klassenverteilung nicht gleichmäßig ausgenutzt. So treten geringe Wahrscheinlichkeiten der Aktivität sehr viel häufiger auf. Das Ergebnis eines mit Standardparametern trainierten *Random Forest* ist in Abbildung 2.6 zu sehen. Es fällt auf, dass sich 98 % der Moleküle auf lediglich zwei Wahrscheinlichkeitswerte verteilen. Alle Moleküle mit derselben Wahrscheinlichkeit sind in der Sortierung zufällig angeordnet, was die geraden Diagonalen erklärt, die der zufälligen Auswahl entsprechen.

Eine Lösungsmöglichkeit ist es, die Parameter des *Random Forest* gezielt so anzupassen, dass eine höhere Granularität der Wahrscheinlichkeiten erzielt wird. Dazu müssen die Bäume beim Training früher gestoppt werden, damit nicht immer nur Datenpunkte derselben Klasse in einem Blatt landen. Sowohl der Parameter zur maximalen Baumtiefe als auch der Parameter zur minimalen Anzahl an Datenpunkten pro Blatt kann für diesen Zweck genutzt werden. Durch dieses frühere Stoppen können Blätter entstehen, die immer noch Datenpunkte von mehr als einer Klasse enthalten und somit eine Wahr-

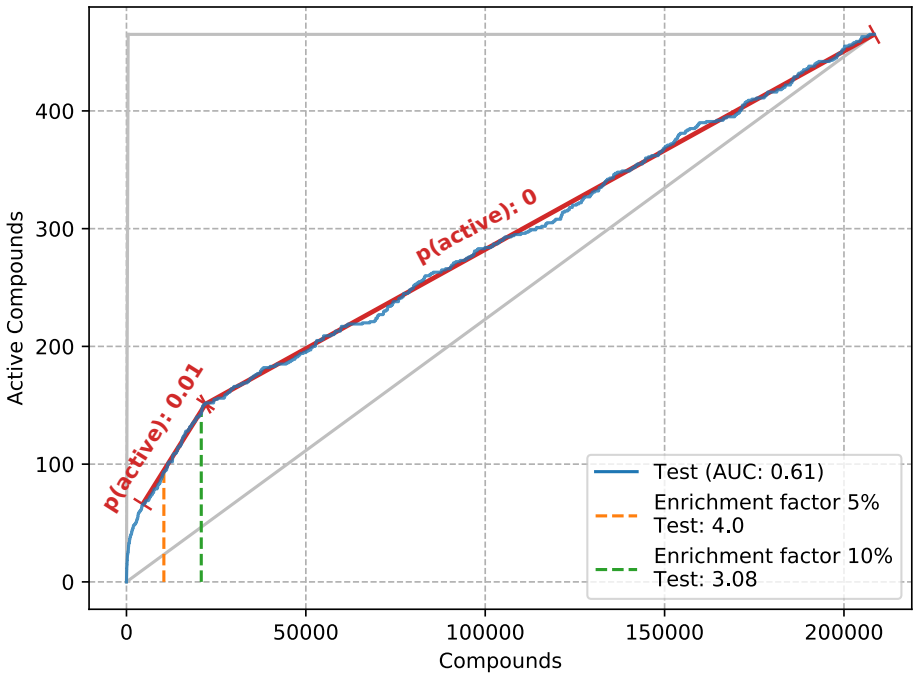


Abbildung 2.6: Ein *Random Forest* ohne Parameteranpassung mit geringer Granularität. 90 % der Moleküle haben eine Wahrscheinlichkeit von 0, aktiv zu sein, und sind entsprechend zufällig sortiert.

scheinlichkeit ausgeben, die diesem Klassenverhältnis entspricht. Damit die so gewonnenen feiner granularen Wahrscheinlichkeiten aber nicht durch das *Hard Voting* wieder ignoriert werden, muss ein *Soft Voting* genutzt werden, das die Gesamtwahrscheinlichkeiten aus dem Mittelwert der Wahrscheinlichkeiten aller einzelnen Bäume berechnet. In Abbildung 2.7 sind die Ergebnisse eines *Random Forest* zu sehen, bei dem genau diese beiden Anpassungen vorgenommen wurden. Im Vergleich zum vorherigen *Random Forest* hat sich die Anzahl verschiedener Wahrscheinlichkeitswerte von 73 auf 132 130 erhöht. Es treten keine größeren Gruppen mit derselben Wahrscheinlichkeit mehr auf, und das Ergebnis im Sinne der *AUC* hat sich deutlich verbessert.

Die vorgeschlagenen Parameter dienen spezifisch dazu, die Granularität der Ergebnisse zu verbessern. Im Allgemeinen wirkt das Nicht-Auslernen der Blätter der Idee des *Random Forest* von spezialisierten Bäumen entgegen und kann somit auch einen negativen Einfluss auf die allgemeine Klassifikationsgüte haben. Die Verwendung von *Hard Voting* oder *Soft-Voting* wiederum bewirkt in den meisten Fällen keinen großen

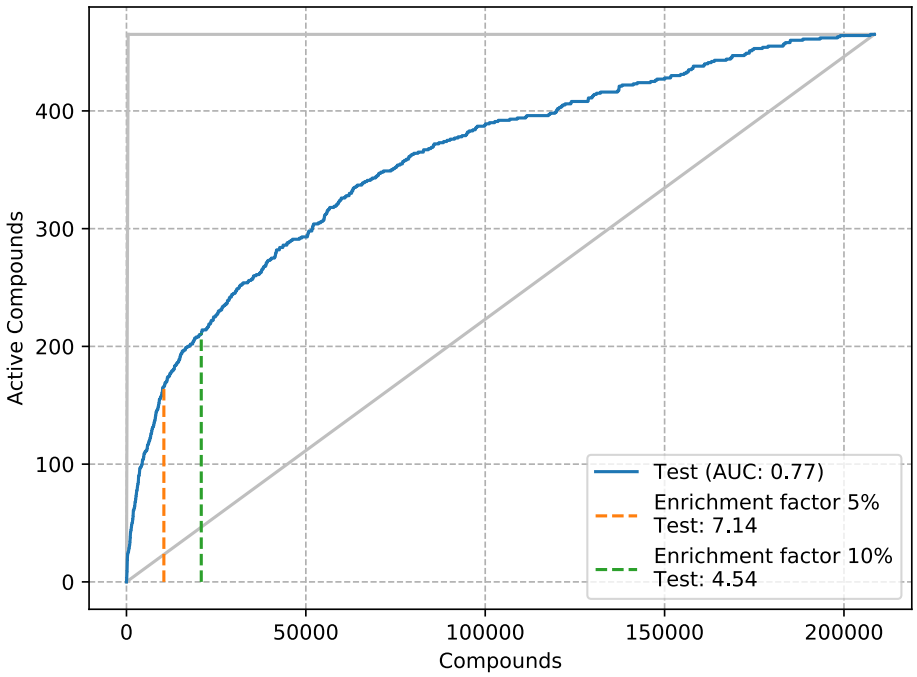


Abbildung 2.7: Ein *Random Forest* mit Parameteranpassung für höhere Granularität. Die minimale Größe von Blattknoten wurde auf 10 gesetzt und es wird *Soft Voting* genutzt.

Unterschied, weshalb unterschiedliche Bibliotheken für maschinelles Lernen auch unterschiedliche Standardwerte nutzen.

Das hier beschriebene Problem tritt durch die Kombination der im Folgenden genannten Faktoren auf: (1) Das gewünschte Ergebnis ist eine Sortierung, die auf der Klassenwahrscheinlichkeit basiert. (2) Die verwendete Klassifizierungsmethode hat eine geringe Granularität der Klassenwahrscheinlichkeit. (3) Der Effekt wird durch eine ungleiche Klassengewichtung verstärkt, da der volle Wertebereich der möglichen Wahrscheinlichkeiten nicht ausgenutzt wird. Sowohl (1) als auch (3) sind beim *Virtual High-Throughput Screening* gegeben. Somit ist darauf zu achten, dass eine Klassifizierungsmethode mit hoher Granularität gewählt wird.

Die in dieser Arbeit vorgestellte Methode basiert auf neuronalen Netzen, die durch ihre Funktionsweise grundsätzlich eine hohe Granularität aufweisen. In Experimenten, die Vergleiche zu einem *Random Forest* ziehen, werden die hier genannten Techniken genutzt, um ebenfalls eine hohe Granularität zu erreichen.

2.8 Zusammenfassung

In diesem Kapitel wurden die nötigen Grundlagen zum *Virtual High-Throughput Screening* erklärt. Die Methode wird dazu eingesetzt, die Wahrscheinlichkeit der Aktivität eines Moleküls abzuschätzen. Es werden bereits im Labor getestete Moleküle genutzt, um einen Klassifikator zu trainieren. Die vom Klassifikator berechnete Wahrscheinlichkeit der Klassenzugehörigkeit kann dann verwendet werden, um noch nicht getestete Moleküle zu sortieren. Somit wird eine Menge an Top-n-Molekülen erzeugt, die möglichst viele der tatsächlich aktiven Moleküle enthält.

Zwei öffentlich verfügbare Datenquellen sind PubChem und ChemBL. Als Daten steht neben der Aktivität üblicherweise die Struktur eines Moleküls zur Verfügung. *SMILES* ist eines der häufig verwendeten Datenformate, in dem diese Struktur kodiert wird. Um sie in numerische Merkmale umzuwandeln, werden häufig *Fingerprints* wie der *Extended Connectivity Fingerprint* genutzt. Dieser kann dann einem Klassifikator wie dem *Random Forest* als Eingabe dienen. Zur Beurteilung der Klassifikationsergebnisse können Metriken wie der *Enrichment Factor* (zur Beurteilung der Früherkennung) und die *Area under the ROC Curve* (zur Beurteilung der gesamten Klassifikation) genutzt werden. Da das Ziel eines *Virtual High-Throughput Screenings* oftmals eine Sortierung der Moleküle anhand der Wahrscheinlichkeit ihrer Aktivität ist, sollte beachtet werden, dass die berechneten Wahrscheinlichkeiten möglichst feingranular sind.

Kapitel 3

Deep Learning

Deep Learning [36] ist die Verwendung von künstlichen neuronalen Netzen mit mehr als einer verborgenen Schicht. Durch die Steigerung der Rechenleistung, insbesondere durch die Nutzung von Grafikprozessoren, ist *Deep Learning* in den letzten Jahren zunehmend zu einer praktikablen Technik geworden. Die höhere Anzahl an Schichten führt dazu, dass neue Typen von Schichten entwickelt werden, deren Kombination in Form einer so genannten Netzwerkarchitektur sich jeweils zur Lösung von spezifischen Problemen eignet. So haben sich *Convolutional Neural Networks* [37] für maschinelles Sehen durchgesetzt, während sich *Recurrent Neural Networks* [25] besonders zur Verarbeitung von natürlicher Sprache eignen.

Tabelle 3.1 erklärt die Symbole, die in den Formeln dieses Kapitels genutzt werden.

3.1 Künstliche neuronale Netze

Der Grundbaustein eines künstlichen neuronalen Netzes ist das Neuron. Neuronen sind in Netzwerken oft in Form von Schichten gruppiert. In dieser Arbeit werden Netzwerke immer in der Reihenfolge Eingabeschicht (oben) bis Ausgabeschicht (unten) dargestellt. Ein vorwärts betriebenes Netz besteht aus mehreren Schichten, wobei die Neuronen einer Schicht nur mit Neuronen von vorherigen Schichten verbunden sind. Im einfachsten Fall sind die Neuronen einer Schicht jeweils mit allen Neuronen ihrer direkten Vorgängerschicht verbunden (*Dense*-Schicht). Hierbei bildet die erste Schicht die Eingabewerte für das Netzwerk ab. Die Neuronen der weiteren Schichten erhalten ihre Eingabe von der Ausgabe ihrer vorherigen Schicht. Die Ausgabe des Netzwerks ist die Ausgabe der letzten Schicht.

Tabelle 3.1: Verwendete Symbolik

Symbol	**Erklärung**
l	Index einer Schicht
$o_{l,i}$	Ausgabewert des Neurons i der Schicht l
$w_{l,j,i}$	Gewicht zwischen dem Neuron i der Schicht $l-1$ und dem Neuron j der Schicht l
$b_{l,i}$	Bias des Neurons i der Schicht l
$z_{l,i}$	Netzeingabe des Neurons i der Schicht l
$A_l(z_{l,i})$	Aktivierungsfunktion, die für das Neuron i der Schicht l aus dem Eingabewert $z_{l,i}$ den Ausgabewert $o_{l,i}$ berechnet
t_i	Wahrer Zielwert des Datenpunktes i
p_i	Vorhergesagter Wert des Datenpunktes i
$E(t,p)$	Fehlerfunktion, berechnet aus der Menge an wahren Zielwerten t und der Menge an Vorhersagen p
$\frac{\partial y}{\partial x}$	Einflussfaktor von x auf y
η	Lernrate
s	Index eines Lernschritts
g_s	Vektor der Gradienten für Lernschritt s
θ	Vektor an Parametern (Gewichte und Bias)
$c(x)$	Anzahl Elemente des Vektors x

Ein Neuron berechnet für eine Eingabe die zugehörige Ausgabe anhand der Gewichte zu den einzelnen Eingabewerten, eines Bias und einer Aktivierungsfunktion. Die Ausgabe des Neurons wird durch

$$o_{l,j} = A_l \left(\sum_{i=1}^{n} (o_{l-1,i} \cdot w_{l,j,i}) + b_{l,j} \right) \tag{3.1}$$

errechnet. Hierbei wird jeder der Ausgabewerte $o_{l-1,i}$ der Neuronen der vorherigen Schicht $l-1$ mit dem jeweiligen Gewicht $w_{l,i,j}$, das das Neuron i der Schicht $l-1$ mit dem Neuron j der Schicht l verbindet, multipliziert. Anschließend werden alle Ergebnisse aufsummiert und ein Bias $b_{l,j}$ wird addiert. Aus diesem Wert, oft als Netzeingabe des Neurons j bezeichnet, wird anschließend mit der Aktivierungsfunktion A_l die Ausgabe $o_{l,j}$ berechnet. Abbildung 3.1 zeigt ein einfaches Beispiel eines vorwärts betriebenen Netzes.

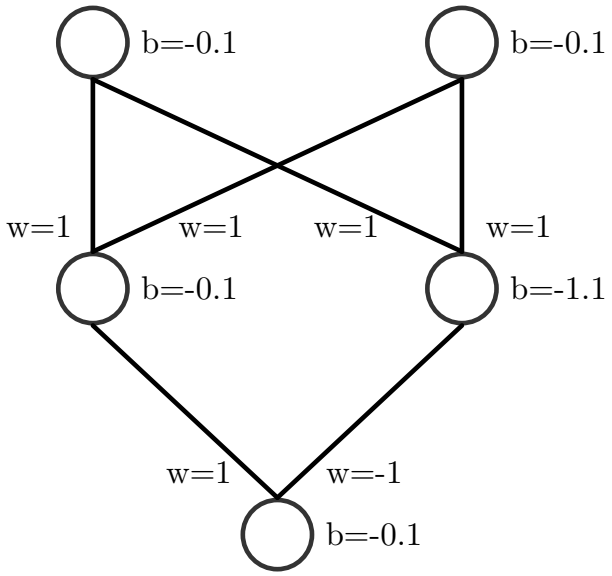


Abbildung 3.1: Ein neuronales Netz, das XOR berechnet. Es wird die Schwellenwertfunktion als Aktivierungsfunktion genutzt.

3.2 Schichttypen

Die meisten Netzwerke bestehen aus Schichten, deren Neuronen auf die gleiche Art mit ihren Vorgängern verbunden sind. Somit lässt sich der Aufbau eines Netzwerks, auch Netzwerkarchitektur genannt, leicht durch eine Auflistung der aufeinander folgenden

Schichten erklären. Auch Berechnungsschritte wie Aktivierungsfunktionen, *Pooling* [45] (das Zusammenfassen mehrerer Neuronen auf der Basis einer Funktion) oder *Dropout* [49] (der zufällige Signalausfall einer bestimmten Quote an Neuronen) können wahlweise als Schicht ausgedrückt werden. Um Platz zu sparen, werden Aktivierungsfunktionen in dieser Arbeit nicht als Schicht dargestellt, sondern als Hyperparameter von Schichten, die eine Aktivierungsfunktion nutzen. Im Folgenden werden solche Schichten erklärt, die dann im weiteren Verlauf der Arbeit Verwendung finden.

Die Eingabeschicht spiegelt lediglich die Werte der Eingabedaten als Neuron wider. Sie bildet die Schnittstelle zwischen den Eingabedaten und dem Netzwerk und besitzt keine trainierbaren Parameter und keine Aktivierungsfunktion.

Die *Dense*-Schicht ist eine Schicht, bei der alle Neuronen jeweils mit allen Neuronen der Vorgängerschicht verbunden sind. Sie dient vor allem zum Lernen der Klassifikation von Daten. Durch die Verbindung mit allen Eingangsdaten können beliebige Muster erkannt werden. Da jede Verbindung ihr eigenes Gewicht hat, stellen *Dense*-Schichten in den meisten Netzwerken den Großteil der trainierbaren Gewichte (je Schicht: Anzahl Neuronen der Vorgängerschicht · Anzahl Neuronen der aktuellen Schicht). Dadurch sind *Dense*-Schichten am meisten an der Überanpassung eines Netzwerks beteiligt. In Abbildung 3.2 ist ein Beispiel einer *Dense*-Schicht zu sehen. Netzwerke, die aus mehreren *Dense*-Schichten bestehen, werden auch als *Multilayer Perceptron* bezeichnet.

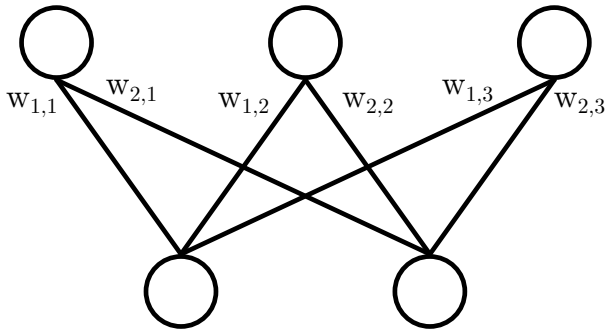


Abbildung 3.2: *Dense*-Schicht mit 2 Neuronen und $3 \cdot 2$ Gewichten.

Durch eine *Convolution*-Schicht wird eine Umwandlung von einer Repräsentation in eine andere, abstraktere Repräsentation gelernt. Dies wird durch ein *Sliding Window* implementiert. Dafür werden die Dimensionen der Daten aufgeteilt in Dimensionen, die eine Position im Raum und damit eine Nachbarschaftsbeziehung ausdrücken, und Dimensionen, die Eigenschaften an dieser Position beschreiben. Im Beispiel eines Farbbildes bestimmen die x- und die y-Dimension die Position eines Pixels, während die

z-Dimension die Farbe des Pixels beschreibt. Die Anzahl der Dimensionen, die als Position angesehen werden und über die damit das *Sliding Window* iteriert, werden durch die Dimensionalität der *Convolution*-Schicht ausgedrückt. Im Beispiel des Farbbildes käme eine 2D-*Convolution* zum Einsatz. Mittels einer Schrittgröße kann bestimmt werden, um wie viele Positionen innerhalb dieser Dimensionen das *Sliding Window* verschoben wird, bevor die nächste Ausgabeposition berechnet wird. An einer Ausgabeposition wird eine festgelegte Anzahl an Neuronen, sogenannte Filter, gelernt. Jedes dieser Filter-Neuronen ist vollständig mit allen Neuronen innerhalb des *Sliding Window* verbunden. Die Gewichte, die alle Neuronen innerhalb des *Sliding Window* mit dem jeweiligen Filter-Neuron verbinden, sind dabei immer dieselben, so dass unabhängig von der Position in den Eingabedaten immer dieselbe Umwandlung stattfindet (Positionsinvarianz). Durch diese Eigenschaften eignet sich die *Convolution*-Schicht besonders zum Generieren von Merkmalen, da sie eine Umwandlung von rohen Eingabedaten hin zu einer nützlicheren Repräsentation der Daten erlernen kann. Als Hyperparameter können die Anzahl an Filtern, die Größe des *Sliding Window* und die Schrittgröße bestimmt werden. Da für eine *Sliding-Window*-Position jeweils nur eine Position in der Ausgabe erzeugt wird, führt eine *Sliding-Window*-Größe über 1 zu einer Verringerung an Positionen. Um dem entgegenzuwirken, können optional weitere Neuronen mit einem Wert von 0 am Rand der Eingabepositionen angenommen werden, so dass die Anzahl der Positionen gleich bleibt. Abbildung 3.3 zeigt ein Beispiel einer eindimensionalen *Convolution*-Schicht, bei der diese Technik angewandt wird.

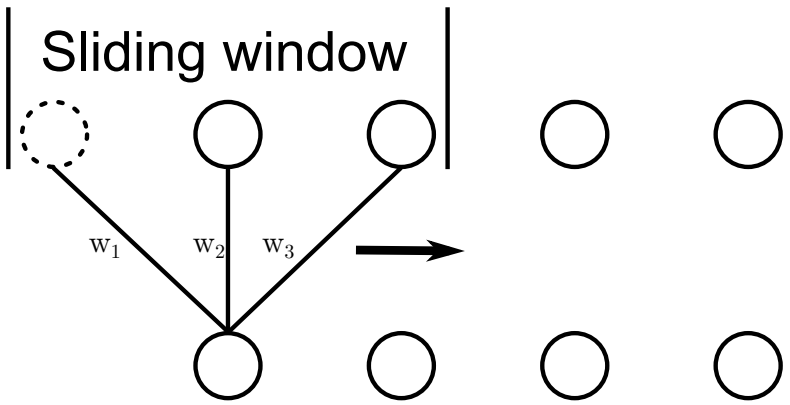


Abbildung 3.3: 1D-*Convolution*-Schicht mit einem *Sliding Window* der Größe 3. Durch die Schrittgröße von 1, die Anzahl der Filter von 1 und das Auffüllen der Neuronen innerhalb des *Sliding Window* werden genauso viele Neuronen wie in der vorherigen Schicht erzeugt.

Eine *Pooling*-Schicht dient vor allem dazu, die Anzahl der Positionen zu verringern, während die signifikantesten Aspekte eines Bereichs beibehalten werden. Dies funktioniert wiederum mittels eines *Sliding Window*. Diesmal werden jedoch keine Gewichte gelernt, sondern es wird anhand einer festen Funktion bestimmt, welches der Neuronen die wichtigste Information enthält und somit für die Daten innerhalb des *Sliding Window* repräsentativ ist. Die am häufigsten verwendete Variante ist das Max-*Pooling*, bei dem jeweils das Neuron mit der höchsten Aktivierung gewählt wird. Der Vorteil des Poolings besteht darin, dass sich die Menge an Daten verringern lässt und dabei gleichzeitig die wichtigsten Informationen beibehalten werden. Ein Beispiel einer Max-*Pooling*-Schicht ist in Abbildung 3.4 zu sehen.

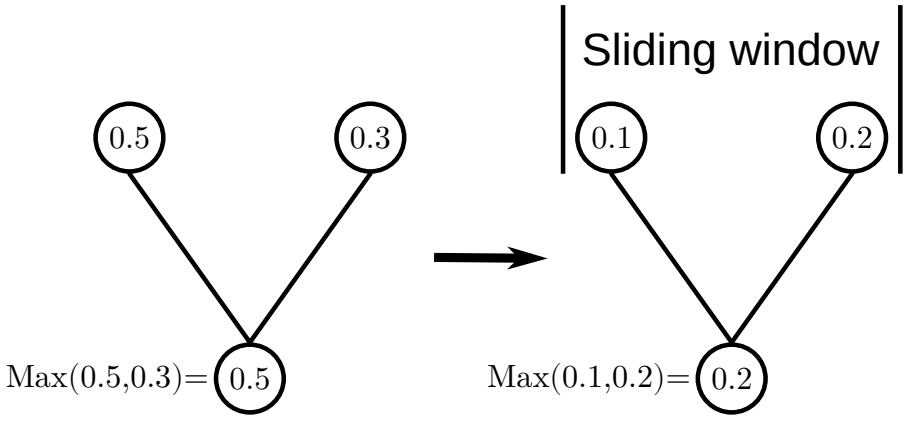


Abbildung 3.4: Max-*Pooling*-Schicht mit einem *Sliding Window* der Größe 2 und einer Schrittgröße von 2.

Die *Dropout*-Schicht dient dazu, das Netzwerk während des Trainings zu zwingen, auch mit unvollständigen Daten vernünftige Vorhersagen zu geben. Hierfür werden während des Trainings zufällig Teile der Eingabedaten auf 0 gesetzt. So kann das Netzwerk sich nicht nur auf ein dominantes Merkmal in den Daten konzentrieren und muss lernen, mit den verfügbaren Daten zum Ziel zu kommen. Dies führt zu einer besseren Generalisierung und wirkt somit einer Überanpassung entgegen, da das Netzwerk nie die vollständigen Trainingsdaten sieht und diese somit schwer auswendig lernen kann. Die *Dropout*-Schicht ist nur während des Trainings aktiv. Werden Vorhersagen getätigt, werden keine Informationen auf 0 gesetzt. Während des Trainings wurden die Ausgabewerte entsprechend skaliert, so dass die Summe der Ausgabewerte zwischen Training und Vorhersage gleich gehalten wird. Der Bruchteil der auf 0 zu setzenden Neuronen ist einstellbar. Abbildung 3.5 veranschaulicht den Effekt einer *Dropout*-Schicht während des Trainings.

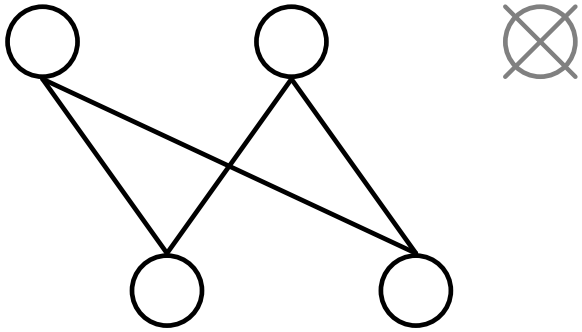


Abbildung 3.5: Darstellung von *Dropout* während des Trainings mit einer *Dense*-Schicht als Folgeschicht. Es wird ein Drittel der Neuronen inaktiv gesetzt. Welches der Neuronen inaktiv wird, wird immer wieder neu zufällig gewählt.

3.3 Aktivierungsfunktionen

Aktivierungsfunktionen dienen dazu, einem Netzwerk nicht lineare Berechnungen zu ermöglichen. Zu diesem Zweck muss die Aktivierungsfunktion selbst nicht linear sein. Die Anpassung des Netzwerks während des Trainings findet mittels Gradientenabstieg auf der Fehlerfunktion statt. Dabei wird der Gradient mittels *Backpropagation* berechnet und das Netzwerk dahingehend angepasst, dass sich der Fehler verkleinert. Zur Berechnung des Gradienten ist die Ableitung der verwendeten Funktionen notwendig. Aus diesem Grund wird im Folgenden auch jeweils auf die Ableitung eingegangen. Das Training mittels Gradientenabstieg und die Verwendung der Ableitungen werden in Kapitel 3.4 erklärt.

Die Schwellenwertfunktion (Abbildung 3.6) gibt für Werte kleiner 0 einen Wert von 0 aus und für positive Werte, inklusive 0, eine 1. Sie ist durch die Biologie inspiriert und eignet sich besonders zur Abbildung logischer Operationen. Die Aktivierung wird mit

$$A_l(z_{l,i}) = \begin{cases} 0, & \text{if } z_{l,i} < 0 \\ 1, & \text{if } z_{l,i} \geq 0 \end{cases} \tag{3.2}$$

berechnet, kann allerdings nicht abgeleitet werden. Aufgrund dessen ist die Schwellenwertfunktion nicht zum Training mittels *Backpropagation* geeignet.

Die logistische Funktion (Abbildung 3.7) bildet die Eingabe auf ein offenes Intervall zwischen 0 und 1 ab. Durch das Abflachen in beide Richtungen wird die Wirkung von hohen Netzeingaben reduziert. Durch den Wertebereich eignet sich die Funktion, um die Wahrscheinlichkeiten voneinander unabhängiger Klassen abzubilden. Dabei wird die

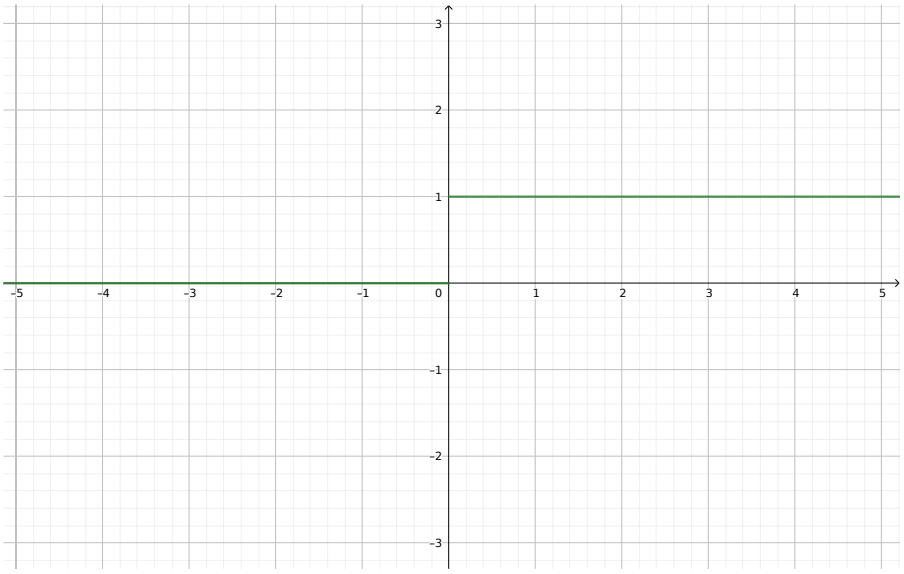


Abbildung 3.6: Schwellenwertfunktion.

Zugehörigkeit zu jeder einzelnen Klasse durch ein eigenes Ausgabeneuron abgebildet. Die Aktivierung wird mit

$$A_l(z_{l,i}) = \frac{1}{1 + \mathrm{e}^{-z_{l,i}}} \tag{3.3}$$

berechnet und hat die Ableitung

$$A_l'(z_{l,i}) = \frac{1}{1 + \mathrm{e}^{-z_{l,i}}} \cdot \left(1 - \frac{1}{1 + \mathrm{e}^{-z_{l,i}}}\right). \tag{3.4}$$

Die *Tanh*-Funktion (Abbildung 3.8) ist eine lineare Transformation der logistischen Funktion und liegt im offenen Intervall zwischen -1 und 1. Die Aktivierung wird mit

$$A_l(z_{l,i}) = \mathrm{Tanh}(z_{l,i}) \tag{3.5}$$

berechnet und hat die Ableitung

$$A_l'(z_{l,i}) = 1 - \mathrm{Tanh}(z_{l,i})^2. \tag{3.6}$$

Die *ReLU-(Rectified-Linear-Unit-)*Funktion [24] (Abbildung 3.9) gibt den Eingabewert $z_{l,i}$ aus, falls er positiv ist. Bei negativen Werten wird eine 0 ausgegeben. Die Aktivierung wird mit

$$A_l(z_{l,i}) = \mathrm{Max}(0, z_{l,i}) \tag{3.7}$$

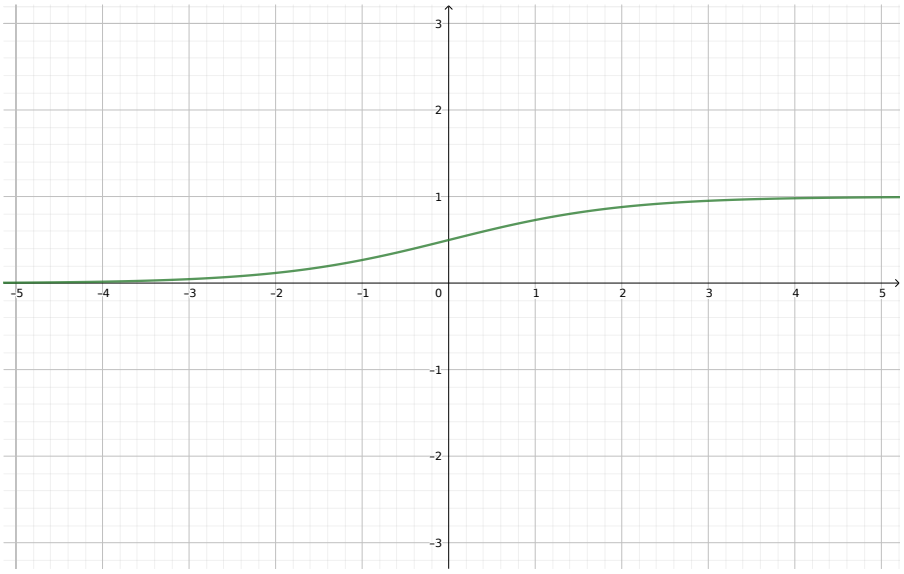


Abbildung 3.7: Logistische Funktion.

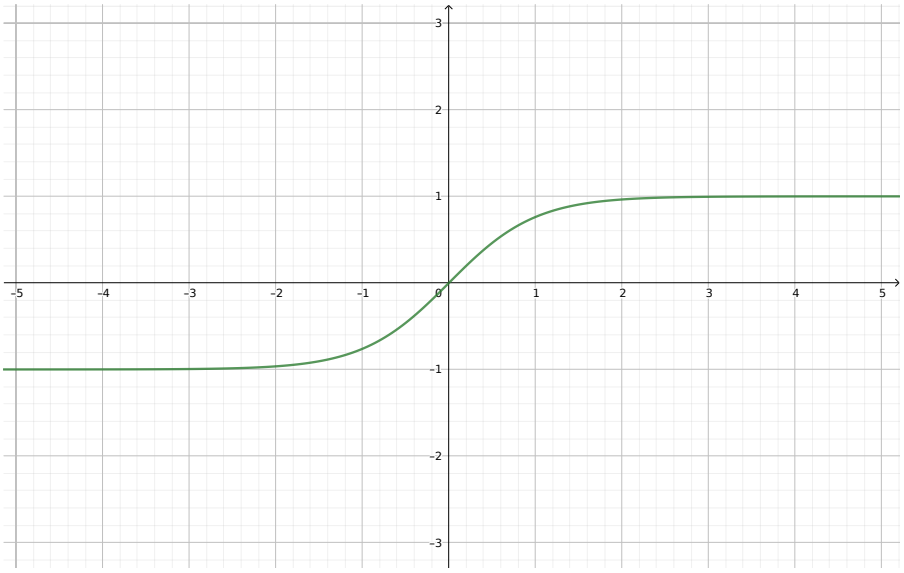


Abbildung 3.8: *Tanh*-Funktion.

berechnet und hat die Ableitung

$$A_l'(z_{l,i}) = \begin{cases} 0, & \text{if } z_{l,i} \leq 0 \\ 1, & \text{if } z_{l,i} > 0 \end{cases}. \tag{3.8}$$

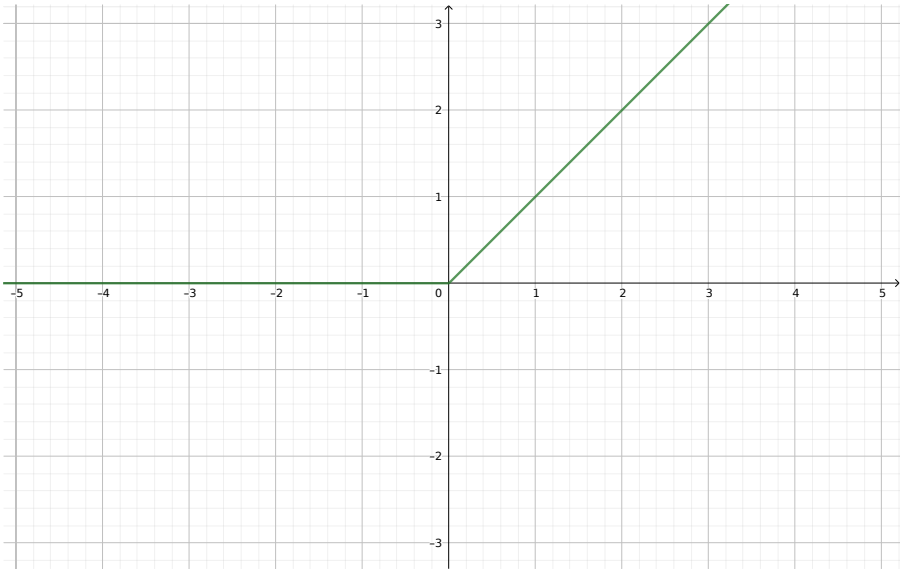


Abbildung 3.9: *ReLU*-Funktion.

Die *Softmax*-Funktion wird in der Ausgabeschicht eingesetzt, um die Wahrscheinlichkeit sich gegenseitig ausschließender Klassen abzubilden. Es wird sichergestellt, dass

$$\sum_{i=1}^{n} A_l(z_{l,i}) = 1 \tag{3.9}$$

gilt. Hierfür müssen erst die Werte aller Neuronen innerhalb der Schicht berechnet werden. Anschließend wird der Ausgabewert von Neuron i durch

$$A_l(z_l, i) = \frac{\mathrm{e}^{z_{l,i}}}{\sum_{j=1}^{n} \mathrm{e}^{z_{l,j}}} \tag{3.10}$$

berechnet. Die Ableitung davon ist

$$\frac{\partial A_l(z_{l,j})}{\partial z_{l,i}} = \begin{cases} A_l(z_{l,j}) \cdot (1 - A_l(z_{l,i})), & \text{if } i = j \\ -A_l(z_{l,i}) \cdot A_l(z_{l,j}), & \text{if } i \neq j \end{cases}. \tag{3.11}$$

3.4 Fehlerfunktionen

Beim überwachten Lernen von neuronalen Netzwerken können stets die vorhandenen Daten genutzt werden, um die Richtigkeit der Ausgabe des Netzwerks zu überprüfen. Dies lässt sich mit einer Fehlerfunktion formalisieren. Es ist das Ziel des Trainings, den

durch die Fehlerfunktion berechneten Fehler möglichst gering zu bekommen. Sie basiert darauf, die gewünschte Ausgabe, die auf der bekannten Grundwahrheit des Trainingsdatensatzes beruht, und die aktuelle Ausgabe des Netzes zu vergleichen. Durch die Wahl der Fehlerfunktion wird bestimmt, wie stark bestimmte Fehler gewichtet werden.

Eine übliche Fehlerfunktion für Regression ist der *Mean Squared Error*, der mit

$$E(t,p) = \frac{1}{n} \cdot \sum_{i=1}^{n} (p_i - t_i)^2 \tag{3.12}$$

berechnet wird. Hierbei ist n die Anzahl an Ausgaben, t_i ist der wahre Wert und p_i der vorhergesagte Wert an Position i. Die Ableitung davon für ein Neuron ist

$$\frac{\partial E(t_i, p_i)}{\partial E(p_i)} = 2 \cdot (p_i - t_i). \tag{3.13}$$

Eine zur Klassifizierung genutzte Fehlerfunktion ist die *Cross Entropy*. Diese wird durch

$$E(t,p) = -\sum_{i=1}^{n} (t_i \cdot \mathrm{Log}(p_i) + (1 - t_i) \cdot \mathrm{Log}(1 - p_i)) \tag{3.14}$$

berechnet. Die Ableitung der *Cross Entropy* für ein Neuron ist

$$\frac{\partial E(t_i, p_i)}{\partial E(p_i)} = \frac{-t_i}{p_i} + \frac{1 - t_i}{1 - p_i}. \tag{3.15}$$

Der *Mean Squared Error* eignet sich vor allem für Regressionsprobleme, um Fehler zu bestrafen, die stark vom richtigen Wert abweichen. Die *Cross Entropy* hingegen eignet sich zum Maximieren der Log-Likelihood, also des Maximierens der Wahrscheinlichkeit, eine richtige Aussage zu treffen.

3.5 Training

Das Training von neuronalen Netzwerken basiert auf der Methode des Gradientenabstiegs. Dabei ist es das Ziel, den durch die Fehlerfunktion berechneten Fehler durch die Berechnung des Gradienten und eine entsprechende Anpassung der Parameter zu verringern. Es wird für jeden Parameter anhand von dessen Beteiligung am Gesamtfehler des Netzwerks ein Gradient berechnet und der Parameter entsprechend angepasst. Korrekterweise wird zur Berechnung des Gradienten der gesamte Trainingsdatensatz genutzt.

Ein gängiges Verfahren, um den Gradientenabstieg zu beschleunigen, ist es jedoch, jeweils nur kleine Teile der Trainingsdaten, sogenannte *Mini-Batches* [30], zur Berechnung zu nutzen. Dadurch lässt sich der korrekte Gradientenabstieg immer noch grob abschätzen und die Berechnung wird dank weniger Datenpunkte deutlich beschleunigt. Damit werden die einzelnen Schritte ungenauer, aber auch erheblich schneller.

Anhand des Fehlers eines Neurons wird berechnet, wie stark dessen eingehende Gewichte an diesem Fehler beteiligt waren. Durch diese Beteiligung an den Fehlern seiner Nachfolger kann damit auch ein Fehler für ein inneres Neuron berechnet werden. Dieser Vorgang zieht sich vom Ende bis hin zum Anfang des Netzes durch und wird *Backpropagation* [43] genannt. Mit Hilfe der *Backpropagation* können also die nötigen Anpassungen (Gradienten) an den Gewichten berechnet werden, um auf dem ausgewerteten Datenpunkt den Fehler beim nächsten Durchlauf zu verringern. Damit das Netzwerk bei der Anpassung das Minimum nicht überspringt, wird die Anpassung nur zu einem kleineren Teil vorgenommen. Hierfür wird der Wert, mit dem das jeweilige Gewicht angepasst wird, durch die Multiplikation mit der Lernrate η (< 1) verringert. Zur Berechnung der Anpassung für eine *Mini-Batch* wird die Anpassung für jeden einzelnen Datenpunkt berechnet und anschließend für alle gemittelt.

Die konkrete Berechnung der Anpassung basiert darauf, die Sensitivität des Fehlers $E(t,p)$ auf das jeweilige Gewicht $w_{l,j,i}$ zu ermitteln. Es wird also die Berechnung

$$\frac{\partial E(t,p)}{\partial w_{l,j,i}} \tag{3.16}$$

durchgeführt.

Da dieser Wert nicht direkt berechnet werden kann, sondern das Gewicht nur einen indirekten Einfluss auf den Fehler hat, muss dieser Einfluss schrittweise mit Hilfe der Kettenregel bis zum Fehler berechnet werden. Dies ist in Abbildung 3.10 dargestellt und wird mit

$$\frac{\partial E(t,p)}{\partial w_{l,j,i}} = \frac{\partial z_{l,j}}{\partial w_{l,j,i}} \cdot \frac{\partial o_{l,j}}{\partial z_{l,j}} \cdot \frac{\partial E(t,p)}{\partial o_{l,j}} = o_{l-1,i} \cdot A'_l(z_{l,j}) \cdot \frac{\partial E(t,p)}{\partial o_{l,j}} \tag{3.17}$$

berechnet. Hierbei wird also der direkte Einfluss des Gewichts $w_{l,j,i}$ auf den Eingabewert $z_{l,j}$ berechnet. Dieser entspricht dem Ausgabewert $o_{l,i}$ des vorherigen Neurons. Anschließend wird der direkte Einfluss des Eingabewerts $z_{l,j}$ auf den Ausgabewert $o_{l,j}$ berechnet. Dieser wird durch die Ableitung $A'_l(z_{l,j})$ der Aktivierungsfunktion ermittelt. Anschließend ist noch der direkte Einfluss des Ausgabewertes $o_{l,j}$ auf den gesamten Fehler $E(t,p)$

zu berechnen. Dies geschieht durch die Berechnung von

$$\frac{\partial E(t,p)}{\partial o_{l,i}} = \begin{cases} E'(t_i, o_{l,i}), & \text{if } l = c(o) \\ \sum_{j=1}^{c(o_{l+1})} \frac{\partial z_{l+1,j}}{\partial o_{l,i}} \cdot \frac{\partial o_{l+1,j}}{\partial z_{l+1,j}} \cdot \frac{\partial E(t,p)}{\partial o_{l+1,j}}, & \text{if } l \neq c(o) \end{cases}. \tag{3.18}$$

Ist das Neuron i also bereits Teil der letzten Schicht, dann entspricht der Einfluss der Ableitung $E'(t_i, o_{l,i})$ der Fehlerfunktion. Befinden sich hinter dem Neuron noch weitere Schichten, dann muss diese Berechnung rekursiv geschehen. Hierbei existieren zwei Unterschiede zur vorherigen Berechnung des Einflusses des Gewichts auf den Fehler. Zum einen wird hier statt des Einflusses des Gewichts auf den Eingabewert $z_{l+1,j}$ des folgenden Neurons der Einfluss des Ausgabewerts $o_{l,i}$ berechnet. Dieser entspricht dem Gewicht $w_{l+1,j,i}$, welches das Neuron mit seinem Nachfolger verbindet, also

$$\frac{\partial z_{l+1,j}}{\partial o_{l,i}} = w_{l+1,j,i}. \tag{3.19}$$

Der zweite Unterschied ist, dass der Ausgabewert über mehrere Gewichte mit mehreren Folgeneuronen verbunden sein kann. Die Einflüsse auf alle Neuronen werden einfach aufsummiert. Sollte zu einem bestimmten Folgeneuron keine Verbindung bestehen, ist das Gewicht $w_{l+1,j,i}$ 0, womit auch der Gesamteinfluss auf dieses Neuron 0 ergibt.

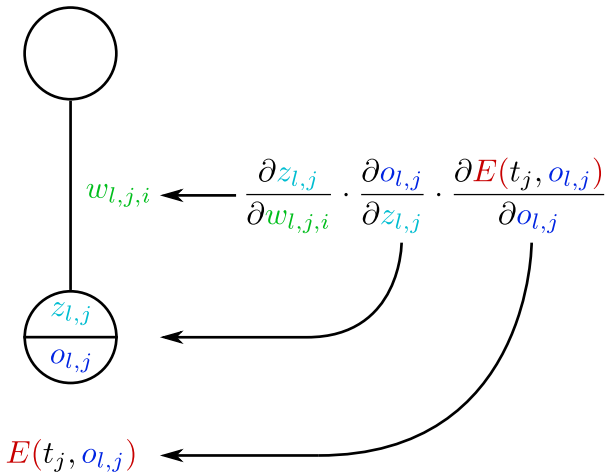


Abbildung 3.10: Kettenregel des Einflusses eines Gewichts auf den Fehler. In diesem einfachen Beispiel ist das Folgeneuron Teil der letzten Schicht und hat damit direkten Einfluss auf den Fehler.

Die Anpassung für den Bias berechnet sich durch

$$\frac{\partial E(t,p)}{\partial b_{l,i}} = \frac{\partial z_{l,i}}{\partial b_{l,i}} \cdot \frac{\partial o_{l,i}}{\partial z_{l,i}} \cdot \frac{\partial E(t,p)}{\partial o_{l,i}} = 1 \cdot A'_l(z_{l,i}) \cdot \frac{\partial E(t,p)}{\partial o_{l,i}}. \tag{3.20}$$

Auf diese Weise werden die Anpassungen für einen spezifischen Datenpunkt berechnet. Ist das für alle Datenpunkte innerhalb der zu verarbeitenden *Mini-Batch* geschehen, wird für jedes Gewicht und jeden Bias (die trainierbaren Parameter) der Mittelwert der jeweiligen Anpassungen berechnet. Diese Werte ergeben den Gradientenvektor g_s für den jeweiligen Lernschritt s.

3.6 Optimierer

Der Einsatz eines Optimierers ermöglicht die Modifikation von Anpassungen vor deren Anwendung mit dem Ziel, den Gradientenabstieg noch zu beschleunigen. Er gestattet es, dass vorherige Anpassungen einen Einfluss auf die aktuelle Anpassung haben. Damit können Datenpunkte aus einer vorherigen *Mini-Batch* die aktuelle Anpassung beeinflussen und den negativen Effekt des *Mini-Batch*-Verfahrens abschwächen. Manche Optimierer erlauben außerdem eine dynamische Anpassung der Schrittweite, teilweise auch für jeden Parameter einzeln.

Eines der möglichen Probleme ist ein langsames Lernen, wenn Anpassungen nur zu sehr kleinen Verbesserungen führen. Sollten diese Anpassungen immer in dieselbe Richtung gehen, ist es sinnvoll, die Schrittweite zu vergrößern, um schneller das Ziel zu erreichen. Gleichzeitig kann es passieren, dass eine Anpassung über das Ziel hinausgeht und entsprechend eine Anpassung in die Gegenrichtung erforderlich ist. Hier wäre eine kleinere Schrittweite zu bevorzugen, um ein Pendeln über das Optimum hinaus zu verhindern. Beide Probleme werden bis zu einem gewissen Grad durch den Momentum-Optimierer [38] gelöst. Er berechnet die Anpassung der Parameter mittels

$$m_0 = 0; \tag{3.21}$$

$$m_s = \alpha \cdot m_{s-1} + g_s; \tag{3.22}$$

$$\theta_{s+1} = \theta_s - m_s. \tag{3.23}$$

Dabei wird das Moment aus dem aktuellen Gradienten und zu einem Faktor α aus dem vorherigen Moment berechnet. Dies führt dazu, dass die Schrittweite vergrößert wird,

wenn mehrfach eine Anpassung in dieselbe Richtung erfolgt, während sie sich verkleinert, wenn die vorherige Anpassung in die Gegenrichtung geschah. Als α wird häufig $0,9$ gewählt. Durch den Moment wird ein Einfluss der vorherigen Trainingsdaten ermöglicht, ohne dass diese individuell mit eingerechnet werden müssen. Dieser Einfluss klingt exponentiell ab.

RMSprop (*Root Mean Squared Gradients*) [51] verfolgt das Ziel, eine angepasste Lernrate individuell für die einzelnen Parameter zu berechnen. Dies geschieht durch

$$v_0 = 0; \tag{3.24}$$

$$v_s = \beta \cdot v_{s-1} + (1 - \beta) \cdot g_s^2; \tag{3.25}$$

$$\theta_{s+1} = \theta_s - \frac{\eta}{\sqrt{v_s} + \epsilon} \cdot g_s. \tag{3.26}$$

Hierbei wird für jeden Lernschritt s ein Vektor v_s berechnet, der für jeden Parameter ein exponentiell abklingendes Mittel des quadrierten Gradienten enthält. Dieser wird dann genutzt, um auf der Basis der festen Lernrate η eine angepasste Lernrate zu bestimmen. Der Faktor des Abklingens wird durch das β (Standardwert 0,9) gesteuert. Das ϵ ist eine sehr kleine Zahl (in der verwendeten Implementierung 10^{-7}) und soll lediglich garantieren, dass der Divisor nicht 0 ist. Durch den Vektor v_s erhöht sich die angepasste Lernrate für Parameter, deren Gradienten längere Zeit klein waren. Dies ist der Fall für flache Regionen des Fehlerraums, die sonst nur sehr langsam durchquert werden. Bei Parametern, die über längere Zeit einen großen Gradienten hatten, verringert sich die angepasste Lernrate. Dadurch wird die Chance, dass ein Minimum übersprungen wird, geringer.

Adam (*Adaptive Moment Estimation*) [33] kombiniert das Moment des Momentum-Optimierers und die individuellen Lernraten von RMSProp. Hierbei wird allerdings das Moment ebenfalls als exponentiell abklingendes Mittel berechnet, während die Berechnung des Vektors zur Anpassung der Lernraten der von RMSProp entspricht. Daraus ergibt sich die Berechnung

$$m_0 = 0; v_0 = 0; \tag{3.27}$$

$$m_s = \alpha \cdot m_{s-1} + (1 - \alpha) \cdot g_s; \tag{3.28}$$

$$v_s = \beta \cdot v_{s-1} + (1 - \beta) \cdot g_s^2. \tag{3.29}$$

Dabei ist m_s der Vektor für das Moment und v_s der Vektor für die Lernratenanpassung. Der Standardwert für α ist 0,9 und für β 0,999. Aufgrund der Initialisierung mit 0 sind die Werte zu Beginn nahe 0. Um das auszugleichen, wird dieser Bias durch

$$\hat{m}_s = \frac{m_s}{1 - \alpha^s} \tag{3.30}$$

und

$$\hat{v}_s = \frac{v_s}{1 - \beta^s} \tag{3.31}$$

kompensiert. Anschließend werden die Parameter ähnlich wie bei RMSProp durch

$$\theta_{s+1} = \theta_s - \frac{\eta}{\sqrt{\hat{v}_s} + \epsilon} \cdot \hat{m}_s \tag{3.32}$$

berechnet. Damit kombiniert Adam die Vorteile des Momentum- und des RMSProp-Optimierers. In der Praxis hat sich Adam als Standardoptimierer für *Deep Learning* durchgesetzt und wird auch in dieser Arbeit verwendet.

3.7 Ungleiche Klassenverteilung

Wenn ein Netzwerk die Klassifizierung ungleich gewichteter Klassen lernen soll, führt dies zu Problemen. Das Netzwerk wird durch zu viele Anpassungen für die dominante Klasse darauf trainiert, sich nur auf diese zu konzentrieren. Die selteneren Anpassungen mit Blick auf die Minderheitsklasse werden damit leicht übertönt.

Eine Möglichkeit, dieses Problem zu lösen, ist die Verwendung von Klassengewichten. Hierbei werden Anpassungen der Minderheitsklasse vergrößert, um die Seltenheit wieder auszugleichen. Ist die Klassenungleichgewichtung zu stark, führt dies allerdings zu einem neuen Problem. Es besteht darin, dass Anpassungen des Netzwerks für die Minderheitsklasse genauso wirken wie viele kleine Anpassungen für dieselbe Klasse. Es wird also für längere Zeit immer dieselbe Klasse trainiert und danach wieder die andere, was das Training stark erschwert.

Wünschenswert wäre eine abwechselnde Anpassung an die verschiedenen Klassen, wie es automatisch bei zufällig sortierten, gleich gewichteten Klassen der Fall ist. Dies ließe sich durch *Oversampling* erreichen. Dabei werden Datenpunkte der Minderheitsklasse so häufig dupliziert, dass das Verhältnis unter den Klassen gleich ist. Bei einer anschließenden zufälligen Wahl von Trainingsdaten ist die Wahrscheinlichkeit dann sehr hoch, dass

gleichmäßig viele Anpassungen für die unterschiedlichen Klassen stattfinden und diese sich regelmäßig abwechseln.

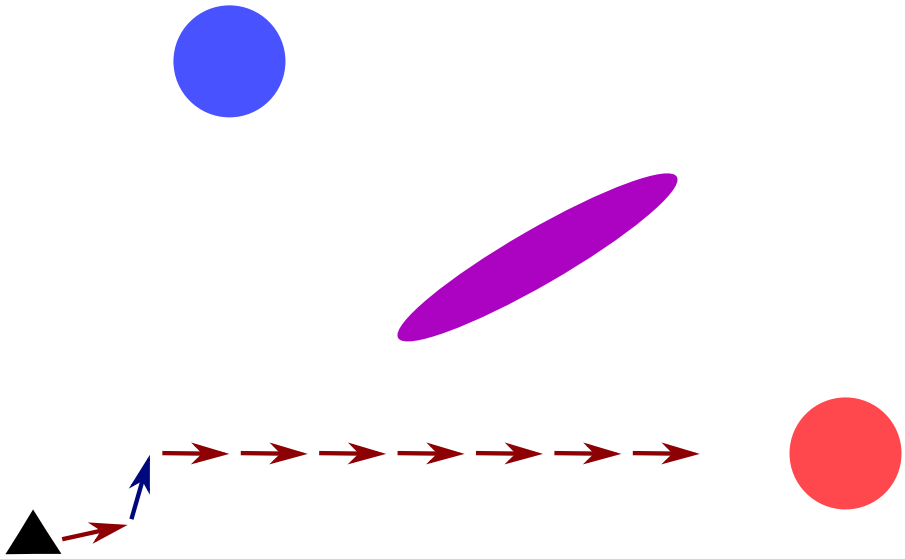


Abbildung 3.11: Verhalten bei ungleicher Klassenverteilung ohne Anpassungen. Die eine Anpassung für die blaue Klasse fällt kaum ins Gewicht und das Modell fokussiert sich auf die rote Klasse.

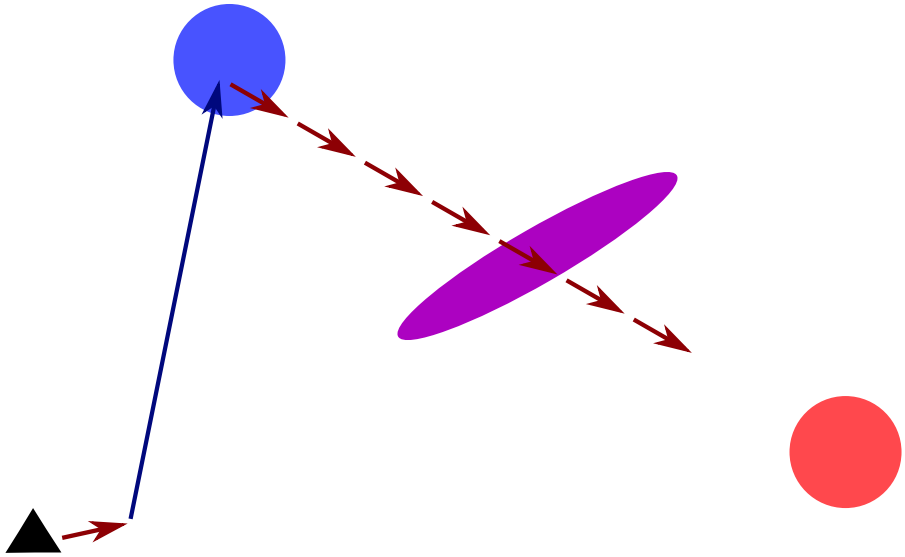


Abbildung 3.12: Verhalten bei ungleicher Klassenverteilung mit gewichtetem Training. Durch die einmalige starke Anpassung lernt das Modell, mit Daten der blauen Klasse umzugehen. Danach wird allerdings zu einseitig gelernt und die Klassifizierung blauer Datenpunkte wird wieder verlernt.

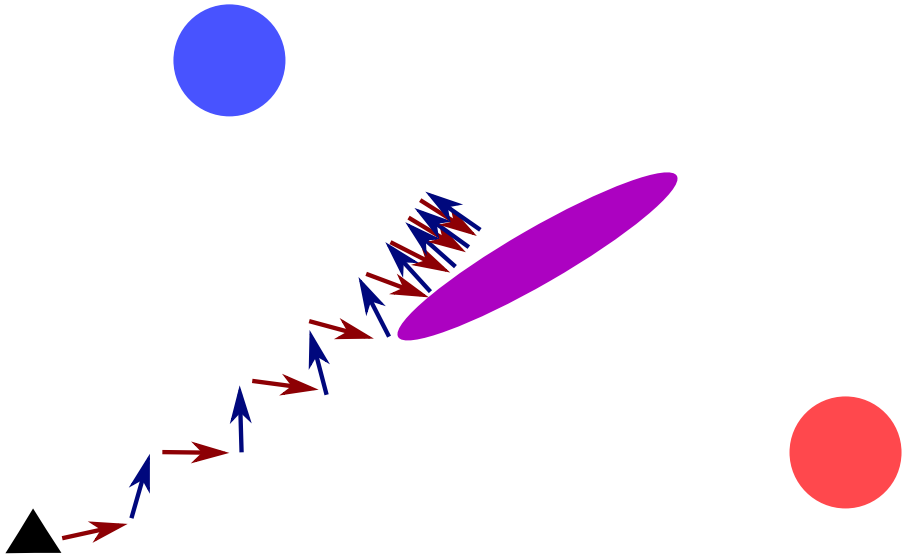


Abbildung 3.13: Verhalten bei ungleicher Klassenverteilung unter Verwendung von *Oversampling*. Durch ein regelmäßiges Abwechseln der zu lernenden Klassen wird ein passender Mittelweg gefunden, bei dem beide Klassen gut klassifiziert werden.

Die Abbildungen 3.11, 3.12 und 3.13 bieten eine vereinfachte Darstellung der Bewegungen des Modells im Lösungsraum bei den genannten Strategien. Hierbei wird ein Modell (schwarzes Dreieck) mit Blick auf die Klassifizierung zweier Klassen (blau und rot) trainiert. Der blaue und rote Kreis zeigen an, dass die jeweilige Klasse hier am besten klassifiziert wird, während die violette Ellipse den Bereich markiert, in dem beide Klassen gut erkannt werden. Zum Training stehen 8 Datenpunkte der roten und nur ein Datenpunkt der blauen Klasse zur Verfügung. Die Pfeile markieren eine Anpassung des Modells durch einen Datenpunkt der jeweiligen Farbe.

Im Zuge dieser Arbeit hat gewichtetes Training zu dem hier dargestellten Problem geführt, welches durch *Oversampling* gelöst wurde.

3.8 Klassifikation von Bildern mittels Convolutional Neural Networks

Der Einsatz von *Convolutional Neural Networks*, also künstlichen neuronalen Netzen, deren Hauptbestandteil *Convolution*-Schichten sind, ist besonders im Bereich des maschinellen Sehens erfolgreich. Dies ist zum Beispiel bei den sich ständig verbessernden Ergebnissen der ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [44, 4] zu

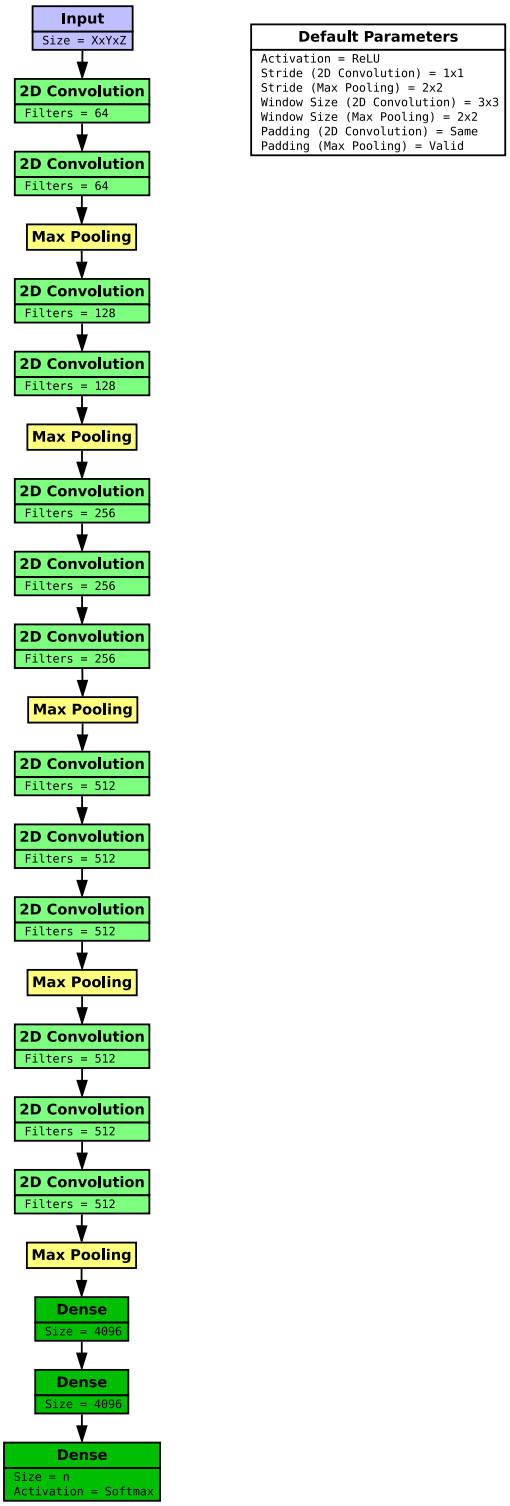


Abbildung 3.14: Netzwerkarchitektur des VGG-16-Netzwerks. Die namensgebenden 16 trainierbaren Schichten sind grün markiert.

sehen, die die Klassifikation von Bildern bei einer Anzahl von 1000 Klassen zur Aufgabe hat.

Ein gutes Beispiel eines typischen *Convolutional Neural Network* sind die Netzwerke der Visual Geometry Group (VGG) [48]. Besonders die beiden Netzwerkarchitekturen VGG 16 und VGG 19, wobei die Nummer jeweils für die Anzahl trainierbarer Schichten steht, werden häufig verwendet. In Abbildung 3.14 ist die VGG-16-Netzwerkarchitektur zu sehen, die auch den Startpunkt für die Entwicklung des in dieser Arbeit beschriebenen Netzwerks darstellt. Als Eingabe dienen die Pixel eines Bildes. Diese werden dann durch die Nutzung von *Convolution*-Schichten in immer abstraktere Repräsentationen umgewandelt. Dabei werden zunehmend mehr Filter generiert. Gleichzeitig werden *Max-Pooling*-Schichten genutzt, um die Anzahl der Positionen in den Daten zu verringern und sich auf die wichtigsten Informationen zu beschränken. Am Ende des Netzwerks werden *Dense*-Schichten genutzt, um aus den so erzeugten Merkmalen die Klassifikation zu lernen. Neben den VGG-Netzwerkarchitekturen gibt es auch noch spezialisiertere wie AlexNet [34] und GoogLeNet [50]. Da in dieser Arbeit allerdings nicht mit Bilddaten gearbeitet wird, bietet es sich an, eine allgemeinere Netzwerkarchitektur wie die der VGG-Netzwerke als ersten Ansatz zu nutzen.

3.9 Saliency Map

Eine *Saliency Map* [47] gibt an, wie stark jede einzelne Position innerhalb einer gewählten Eingabe zum Wert einer gewählten Klasse beigetragen hat. Die Idee dahinter ist, durch *Backpropagation*, den Einfluss jedes einzelnen Neurons der Eingabeschicht $o_{0,i}$ auf das Neuron der gewählten Klasse p_x zu berechnen, also

$$\frac{\partial p_x}{\partial o_{0,i}}. \tag{3.33}$$

Der Unterschied zum Training ist hier lediglich, dass der Einfluss auf den Ausgabewert und nicht auf den Fehler berechnet wird. Somit ist auch kein wahrer Wert zur Berechnung einer *Saliency Map* notwendig, da sie lediglich angibt, welche Teile der Eingabe am meisten zur Klassifizierung beigetragen haben, ungeachtet dessen, ob die Klassifizierung korrekt ist. Mit der *Saliency Map* kann herausgefunden werden, welche Regionen in der Eingabe am meisten für die gewählte Klasse verantwortlich ist. Dadurch lässt sich die Position der Objekte bestimmen, die für eine Klasse besonders relevant sind. An

dieser Technik ist vor allem interessant, dass eine Lokalisierung wichtiger Bereiche in den Eingabedaten möglich ist, obwohl für die Trainingsdaten lediglich die Klasse ohne Kennzeichnung der wichtigen Bereiche vorhanden ist. Die *Saliency Map* kann in Form einer Heatmap in Kombination mit den Eingabedaten dargestellt werden. Durch diese Visualisierung ist es möglich zu interpretieren, warum das Netzwerk sich für die gewählte Klasse entschieden hat.

3.10 Zusammenfassung

Deep Learning ist die Verwendung von künstlichen neuronalen Netzen mit vielen Schichten, die dank der Fortschritte in der Rechenleistung heutiger Computer möglich sind. Besondere Erfolge wurden bisher im Bereich des maschinellen Sehens erzielt. Durch den Einsatz vieler Schichten innerhalb des Netzwerks können spezialisierte Schichten genutzt werden, die eine bestimmte Aufgabe zum Ziel haben. So werden zum Beispiel im Bereich des maschinellen Sehens oft *Convolution*-Schichten eingesetzt, um Abstraktionen der Eingabedaten zu lernen, die für das spätere Netzwerk nützlich sind. Anschließend werden *Dense*-Schichten genutzt, um aus Zusammenhängen innerhalb der so abstrahierten Daten neue Informationen zu generieren und somit eine Klassifikationsaufgabe zu lösen. Um mit Daten mit ungleicher Klassenverteilung zu trainieren, kann *Oversampling* eine sinnvolle Strategie sein. *Saliency Maps* sind ein Ansatz, um Entscheidungen des Netzwerks zu interpretieren und gleichzeitig eine Lokalisierung der für eine Klasse relevanten Objekte innerhalb der Eingabedaten zu erhalten.

Kapitel 4

Rasterrepräsentation von Molekülen

Die in dieser Arbeit vorgestellte Methode basiert auf der Idee, *Deep Learning* für die Nutzung als *Virtual-High-Throughput-Screening*-Methode zu adaptieren. Hierbei dient spezifisch der bei Bildern so erfolgreiche Einsatz von *Convolutional Neural Networks* als Vorbild. Dabei sollte die molekulare Struktur in einer möglichst direkten Repräsentation als Eingabe dienen, so dass die *Convolution*-Schichten des Netzwerks dann eine auf das Lernziel passende Generierung von Merkmalen erlernen können. Zudem muss eine Netzwerkarchitektur verwendet werden, die mit der gewählten Repräsentation der Daten gut arbeitet und auf dieser Basis das Klassifikationsproblem möglichst gut lösen kann.

Die hier vorgestellte Methode basiert auf einer 2D-Rasterdarstellung der Moleküle. Dabei ist zwischen der original berechneten Position von Atomen (im Folgenden auch *pos* genannt) und dem daraus berechneten Index im Raster (im Folgenden auch *idx* genannt) zu unterscheiden. Die zur Erzeugung des Rasters verwendeten Schritte werden in den folgenden Abschnitten genauer vorgestellt, zunächst aber in einer Übersicht aufgeführt:

1. Berechnung der Positionen für die enthaltenen Atome
2. Skalierung der Positionen
3. Horizontale Spiegelung der Positionen
4. Rotation der Positionen um ihren Mittelpunkt
5. Gleichmäßige Verschiebung aller Positionen
6. Übersetzung der Positionen in die dazugehörigen Indizes des Rasters
7. Berechnung der Merkmale eines Atoms für dessen berechneten Rasterindex

4.1 Anordnung

Bestehende Methoden basieren auf Eingabedaten, die in einem Raster angeordnet sind. Deshalb wurde auch hier eine Methode gewählt, die die molekulare Struktur in ein Rasterformat bringt. Hierzu dient die 2D-Repräsentation von Molekülen, wie sie in Bildern zur Anwendung kommt, als Basis.

Eine 2D-Repräsentation wurde gewählt, da sich mit einer geringeren oder höheren Dimensionalität folgende Probleme ergeben. In einer 1D-Repräsentation, etwa bei einem *SMILES*-String, ist es nicht möglich, alle Atome, die sich im Molekül nahe sind, auch in der Repräsentation zusammenzubringen (siehe Abbildung 4.1). Dies ist allerdings für den korrekten Einsatz von *Convolution*-Schichten nötig, da diese auf der Annahme basieren, dass sich innerhalb des *Sliding Window* jeweils das zentrale Objekt und seine nächsten Nachbarn befinden. Eine 3D-Repräsentation wäre vermutlich eine bessere Wahl, da sie auch der Realität näher kommt. Sie führt allerdings zu einer zu großen Datenmenge und damit auch zu einem zu großen Netzwerk. Dies ist bei der heutigen Limitierung von Rechenleistung und Speicherplatz ein Problem und hätte zu hohe Laufzeiten zur Folge. Sobald diese Limitierung kein Problem mehr darstellen, wäre der Umstieg auf eine 3D-Repräsentation eine mögliche Verbesserung. Fast alle kleinen Moleküle lassen sich problemlos in 2D darstellen, ohne dass es zu Überschneidungen kommt. Das heißt, es gehen zwar Informationen über die tatsächliche, räumliche Lage der Atome verloren, deren Bindungen untereinander wird jedoch fehlerfrei abgebildet. Aus den genannten Gründen wird in der hier beschriebenen Methode mit einer 2D-Repräsentation gearbeitet.

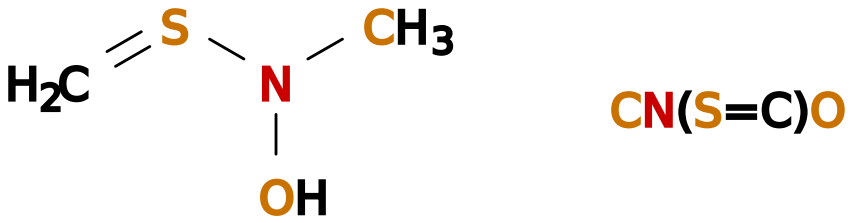


Abbildung 4.1: Während in der 2D-Repräsentation (links) alle Nachbarn (orange) des Stickstoffs (rot) denselben Abstand zu diesem haben, ist der Abstand in der 1D-*SMILES*-Repräsentation (rechts) unterschiedlich. In einer 1D-Repräsentation können immer nur maximal 2 Objekte (eines links, eines rechts) denselben Abstand zu einem zentralen Objekt haben.

In der gewählten 2D-Repräsentation wird jedem Atom eine Position im Raster zugewiesen. Für diesen Zweck wird dieselbe Positionsberechnung angewendet, die auch zur Erstellung von 2D-Zeichnungen von Molekülen genutzt wird. Sie hat den Vorteil, dass die

Abstände zwischen den Atomen etwa gleich sind und möglichst keine Überschneidungen bei den Bindungen entstehen. In der genutzten Implementierung wurden die von *RDKit* berechneten Positionen verwendet (siehe Abbildung 4.2). Sie haben die Eigenschaft, auf 0 zentriert zu sein, das heißt, die Mitte der berechneten Positionen liegt auf dem 0-Punkt. Um diese Fließkommazahlen in einen Index für das Raster umzuwandeln, werden sie durch Rundung in eine natürliche Zahl konvertiert. Die auf 0 zentrierten Positionen müssen außerdem noch um die Hälfte der maximalen Breite beziehungsweise Höhe des Rasters verschoben werden, um auf die Mitte des Rasters zentriert zu sein. Die Rastergröße selbst wird so gewählt, dass alle Moleküle des Datensatzes darin Platz haben. Das heißt, dass die Größe des Rasters für alle Moleküle gleich ist und diese letztlich jeweils in der Mitte des Rasters liegen. Für jedes Paar von Atomen, zwischen denen eine Bindung besteht, wird außerdem eine gerade Linie zwischen ihren Positionen berechnet. Sie kann unter Verwendung des Bresenham-Algorithmus [18] ebenfalls ins Raster eingepasst werden. Dabei dienen die beiden Atome einer Bindung als Eckpunkte und die dazwischen liegenden freien Zellen werden durch die Bindung gefüllt. Eine Visualisierung eines so gefüllten Rasters ist in Abbildung 4.3 zu sehen.

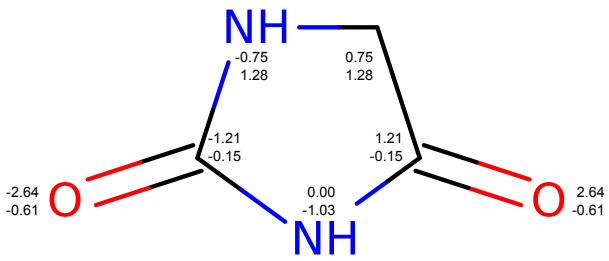


Abbildung 4.2: Das von *RDKit* gezeichnete Molekül und die dazu berechneten Positionen (x oben, y unten) der einzelnen Atome.

Eine Skalierung der Positionen wird ebenfalls vorgenommen. Bei einer zu kleinen Skalierung kann es passieren, dass nicht miteinander verbundene Atome direkt nebeneinander liegen oder sich sogar überlagern. Im Gegensatz führt eine zu große Skalierung zu unnötig großen Datenmengen und damit auch zu einem zu großen Netzwerk. Aus diesem Grund werden die Positionen noch vor ihrer Rundung mit einem Skalierungsfaktor multipliziert. Dieser Skalierungsfaktor hängt von der genutzten Positionsberechnung ab. Bei *RDKit* existiert zwischen zwei verbundenen Atomen eine euklidische Distanz von 1,5. In Abbildung 4.4 wird dasselbe Molekül mit einem unterschiedlichen Skalierungsfaktor gezeigt.

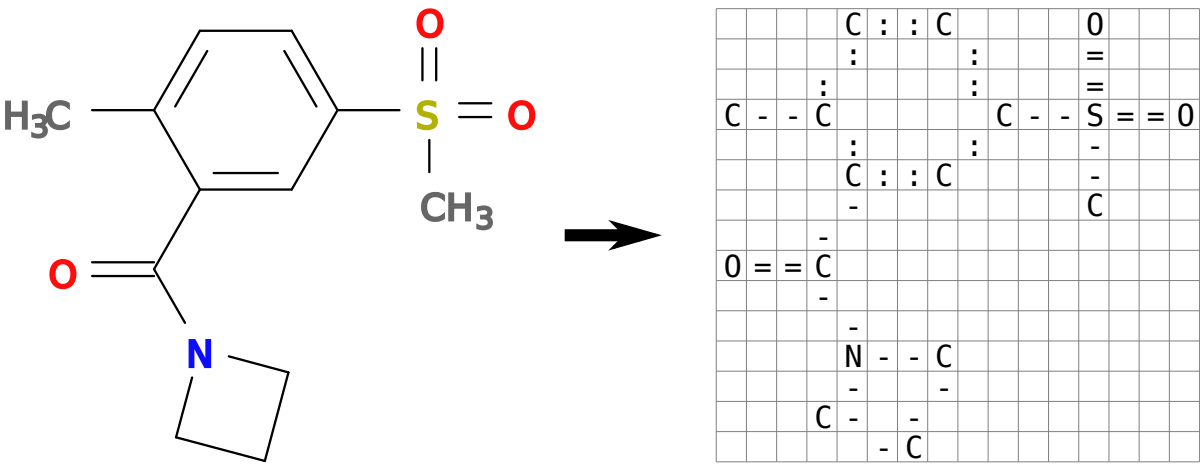
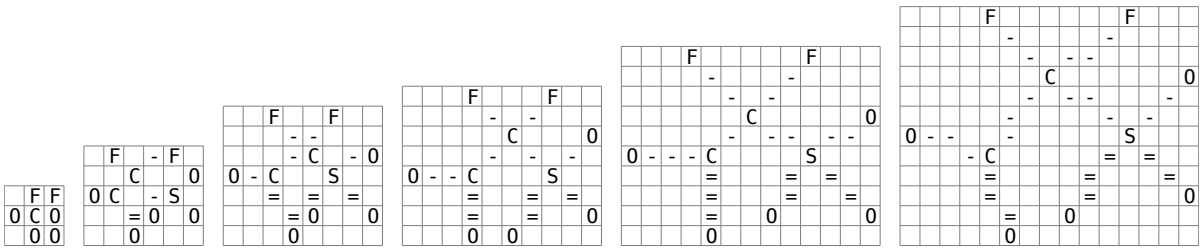


Abbildung 4.3: Dieselben Positionen, die zur Anordnung der Atome in einer 2D-Zeichnung des Moleküls (links) genutzt werden, werden auch zur Positionierung innerhalb des Rasters (rechts) verwendet.



s=0.5 s=1.0 s=1.5 s=2.0 s=2.5 s=3.0

Abbildung 4.4: Dasselbe Molekül mit einem unterschiedlichen Skalierungsfaktor s. Bei einer Skalierung von 0,5 liegen Atome so eng beieinander, dass sie dieselbe Zelle zugeteilt bekommen und sich somit gegenseitig überschreiben. Dieses Problem tritt bei einer Skalierung von 1,0 nicht mehr auf, allerdings sind die Atome immer noch so eng beieinander, das nicht ersichtlich ist, welche tatsächlich miteinander verbunden sind. Durch eine Skalierung von 1,5 tritt dieses Problem nur noch in einem Sonderfall auf. Ab einer Skalierung von 2,0 werden alle Bindungen korrekt repräsentiert und es gibt keinen weiteren Mehrwert bei einer Erhöhung der Skalierung.

Die folgenden Formeln zeigen die Berechnung der Indizes im Raster auf der Basis der Positionen. Da die verwendeten Positionen auf 0 zentriert sind, müssen sie mit Hilfe eines *Offsets* erst auf einen Wertebereich größer/gleich 0 gebracht werden. Dies geschieht durch die Berechnung von

$$x\text{-}offset = -\text{Min}(x\text{-}pos). \tag{4.1}$$

Hierbei wird der *Offset* x-*offset* auf der Basis des kleinsten Werts im Vektor x-*pos* berechnet, der die Positionen aller Atome im Datensatz enthält. Damit berechnen sich die

Indizes mit

$$x\text{-}idx_i = \text{Round}((x\text{-}pos_i + x\text{-}offset) \cdot s). \tag{4.2}$$

Der Index $x\text{-}idx_i$ des Atoms i wird berechnet, indem die Position $x\text{-}pos_i$ mit dem *Offset* $x\text{-}offset$ verschoben, dann mit dem Skalierungsfaktor s skaliert und zum Schluss auf eine Ganzzahl gerundet wird. Die Gesamtgröße des Rasters kann mit

$$x\text{-}size = \text{Round}((\text{Max}(x\text{-}pos) - \text{Min}(x\text{-}pos)) \cdot s) + 1 \tag{4.3}$$

berechnet werden. Dabei wird der Index der höchsten Position berechnet und anschließend eine weitere Zelle addiert, da die besetzten Indizes von 0 bis zum maximalen Index reichen. Die Berechnung der y-Dimension erfolgt auf die gleiche Weise, wobei sie auf dem Vektor aller y-Positionen $y\text{-}pos$ basiert. So berechnet man

$$y\text{-}offset = -\text{Min}(y\text{-}pos); \tag{4.4}$$

$$y\text{-}idx_i = \text{Round}((y\text{-}pos_i + y\text{-}offset) \cdot s); \tag{4.5}$$

$$y\text{-}size = \text{Round}((\text{Max}(y\text{-}pos) - \text{Min}(y\text{-}pos)) \cdot s) + 1. \tag{4.6}$$

4.2 Transformationen

Es kann leicht vorkommen, dass zwei gleiche Substrukturen in zwei sich ähnlichen Molekülen ganz unterschiedlich angeordnet sind. Eine immer gleiche Anordnung ist gar nicht möglich. Dies kann man sich an folgendem Beispiel leicht deutlich machen: Man hat zwei Substrukturen, bei denen jeweils ein bestimmtes Atom an oberster Stelle ist. Werden diese beiden Substrukturen in einem Molekül vereint, ist es je nachdem, an welcher Stelle sie verbunden werden, oftmals nicht möglich, dass bei beiden Substrukturen weiterhin dasselbe Atom oben ist (siehe Abbildung 4.5). Aus diesem Grund können manchmal schon kleine Änderungen, wie das Hinzufügen eines einzelnen Atoms, zu einer großen Änderung in der Positionierung innerhalb des Rasters führen. Abbildung 4.6 zeigt dies anhand eines konkreten Beispiels unter Verwendung von *RDKit*. Um mit diesem Problem umzugehen, wird das Netzwerk darauf trainiert, die möglichen unterschiedlichen Repräsentationen der Struktur zu lernen. Wie auch bei Bildern können hier Transformationen helfen, um unterschiedliche, ebenfalls valide Repräsentationen desselben Objekts zu erzeugen. Während des Trainings wird bei jedem Abruf eines Moleküls dieses zufällig transformiert. So

lernt das Netzwerk in mehreren Epochen nicht immer wieder mit genau derselben Repräsentation des Moleküls, sondern mit unterschiedlichen Varianten. Auch bei Daten, die durch *Oversampling* vervielfacht wurden, wird dadurch die Diversität erhöht. Der Vorteil dieser Transformationen ist es, dass sie dem Netzwerk eine Überanpassung erschweren und damit eine bessere Generalisierung fördern. Die angewandten Transformationen sind das Drehen, die Spiegelung und das Verschieben. Alle Transformationen werden auf den Positionen der Atome ausgeführt, bevor diese in einen Rasterindex umgerechnet werden.

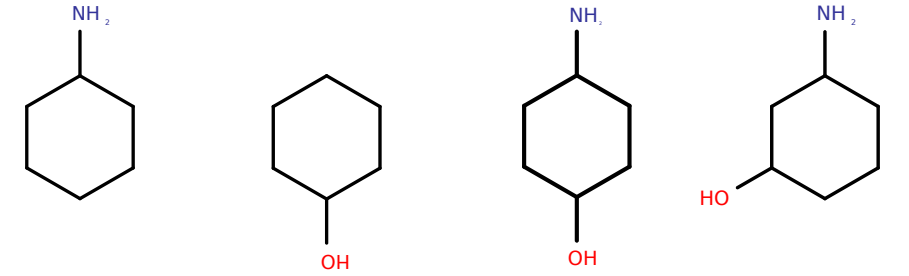


Abbildung 4.5: Substruktur 1: Das Stickstoffatom wird immer an der oberen Spitze dargestellt. Substruktur 2: Das Sauerstoffatom wird immer an der unteren Spitze dargestellt. Kombination 1: Beide Regeln können eingehalten werden. Kombination 2: Es kann nur eine der Regeln angehalten werden. Werden die Regeln angepasst, tritt dasselbe Problem in Kombination 1 auf.

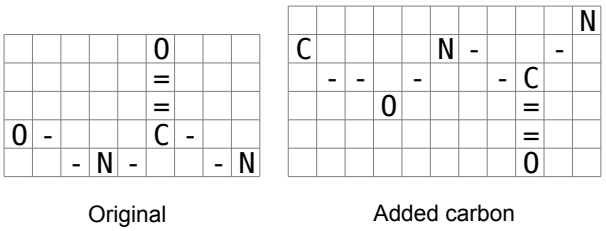


Abbildung 4.6: Durch kleine Änderungen wie das Hinzufügen eines einzelnen Kohlenstoff-Atoms kann sich die Anordnung der anderen Atome drastisch ändern. Neben dem Anfügen des neuen Atoms auf der linken Seite wurde das Molekül vertikal gespiegelt und die Positionen der Atome haben sich leicht verschoben, so dass sich nach dem Runden andere Abstände ergeben.

Die Drehungstransformation wird berechnet, indem jede Position um den Mittelpunkt des Moleküls gedreht wird. Da sich dadurch die Maximalwerte ändern können, wird bei der Größenberechnung des Rasters immer das Maximum beider Achsen verwendet, um ausreichend Platz zu gewährleisten. Des Weiteren wird ein zusätzlicher Spielraum hinzugefügt, da ein langer, diagonal gelegener Arm bei einer Drehung sonst aus den

Grenzen des Rasters herausragen kann. Im schlimmsten Fall ist dazu eine Größe von $\sqrt{x^2 + y^2}$ nötig. In der Realität empfiehlt es sich allerdings, mit Erfahrungswerten zu arbeiten, die deutlich kleiner ausfallen können, um die Menge an Daten und den damit verbundenen Rechenaufwand nicht unnötig zu erhöhen. Ein Beispiel einer Drehung, bei der sich die benötigte Höhe vergrößert, ist in Abbildung 4.7 zu sehen.

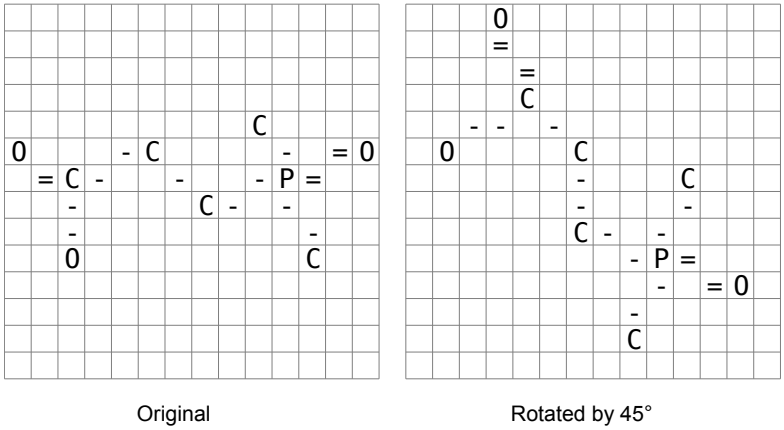


Abbildung 4.7: Dasselbe Molekül im Original (links) und um 45° gedreht (rechts).

Um das Raster in ein quadratisches Format zu bringen und den gewünschten Spielraum hinzuzufügen, werden die Formeln durch

$$x\text{-}size = \text{Round}((\text{Max}(x\text{-}pos) - \text{Min}(x\text{-}pos) + 2 \cdot margin) \cdot s) + 1; \tag{4.7}$$

$$y\text{-}size = \text{Round}((\text{Max}(y\text{-}pos) - \text{Min}(y\text{-}pos) + 2 \cdot margin) \cdot s) + 1 \tag{4.8}$$

angepasst. Hierbei wird die Rastergröße *x-size* beziehungsweise *y-size* wie zuvor berechnet, vor der Skalierung allerdings noch der Spielraum *margin* zweifach hinzugefügt (einfacher Spielraum davor plus einfacher Spielraum dahinter). Die tatsächlich für beide Dimensionen des quadratischen Rasters genutzte Größe *size* ist dann das Maximum der einzelnen Dimensionen, also

$$size = \text{Max}(x\text{-}size, y\text{-}size). \tag{4.9}$$

Bei der Berechnung der *Offsets x-offset* und *y-offset* wird der Spielraum *margin* jeweils nur einfach (davor) hinzugefügt. Dies wird mit

$$x\text{-}offset = -\text{Min}(x\text{-}pos) + margin \tag{4.10}$$

und

$$y\text{-}offset = -\text{Min}(y\text{-}pos) + margin \tag{4.11}$$

berechnet. Verwendet wird allerdings nur das *Offset offset* der größeren Dimension, um weiterhin auf die Mitte des verfügbaren Platzes zu zentrieren, also

$$offset = \begin{cases} y\text{-}offset, & \text{if } x\text{-}size < y\text{-}size \\ x\text{-}offset, & \text{if } x\text{-}size \geq y\text{-}size \end{cases}. \tag{4.12}$$

Abgesehen von der Verwendung des allgemeinen *Offsets offset* bleibt die Formel zur Berechnung der Indizes gleich. Entsprechend findet die Berechnung durch

$$x\text{-}idx_i = \text{Round}((x\text{-}pos_i + offset) \cdot s) \tag{4.13}$$

und

$$y\text{-}idx_i = \text{Round}((y\text{-}pos_i + offset) \cdot s) \tag{4.14}$$

statt.

Zur Drehungstransformation werden die Positionen vor der Rasterung mit den folgenden Formeln angepasst. Dazu müssen die Zentren *x-center* und *y-center* durch

$$x\text{-}center = \text{Min}(x\text{-}pos) + \frac{\text{Max}(x\text{-}pos) - \text{Min}(x\text{-}pos)}{2} \tag{4.15}$$

und

$$y\text{-}center = \text{Min}(y\text{-}pos) + \frac{\text{Max}(y\text{-}pos) - \text{Min}(y\text{-}pos)}{2} \tag{4.16}$$

berechnet werden. Dabei wird zur kleinsten Position die Hälfte der Differenz zwischen größter und kleinster Position hinzuaddiert, um die Mitte des Wertebereichs zu erhalten. Die Positionen müssen dann basierend auf diesem Zentrum um den 0-Punkt zentriert werden. Dies berechnet sich durch

$$x\text{-}0\text{-}pos_i = x\text{-}pos_i - x\text{-}center \tag{4.17}$$

und

$$y\text{-}0\text{-}pos_i = y\text{-}pos_i - y\text{-}center. \tag{4.18}$$

Die so um 0 zentrierten Positionen x-0-*pos* beziehungsweise y-0-*pos* können dann mit der folgenden Formel um den Winkel α gedreht und anschließend mit der Addition des

jeweiligen Zentrums wieder um ihr ursprüngliches Zentrum zentriert werden. Dies wird durch

$$x\text{-}pos'_i = \mathrm{Cos}(\alpha) \cdot x\text{-}0\text{-}pos_i - \mathrm{Sin}(\alpha) \cdot y\text{-}0\text{-}pos_i + x\text{-}center \tag{4.19}$$

und

$$y\text{-}pos'_i = \mathrm{Sin}(\alpha) \cdot x\text{-}0\text{-}pos_i + \mathrm{Cos}(\alpha) \cdot y\text{-}0\text{-}pos_i + y\text{-}center \tag{4.20}$$

berechnet. Sind die Positionen bereits um 0 zentriert, wie dies bei *RDKit* der Fall ist, kann die Anpassung des Zentrums entsprechend entfallen, da dann *x-center* und *y-center* immer 0 sind.

Die Spiegelungstransformation ermöglicht es, die 2D-Repräsentation von der anderen Seite zu betrachten. Ist das Molekül symmetrisch, dann ist eine Spiegelung nicht nötig, da sie auch durch eine 180°-Drehung erreicht werden kann. Ohne Symmetrie ist das Ergebnis einer Spiegelung sonst nicht erreichbar. Ein Beispiel hierfür ist in Abbildung 4.8 zu sehen. Da eine vertikale Spiegelung einer horizontalen mit anschließender 180°-Drehung gleichkommt, ist lediglich eine dieser Spiegelungen nötig.

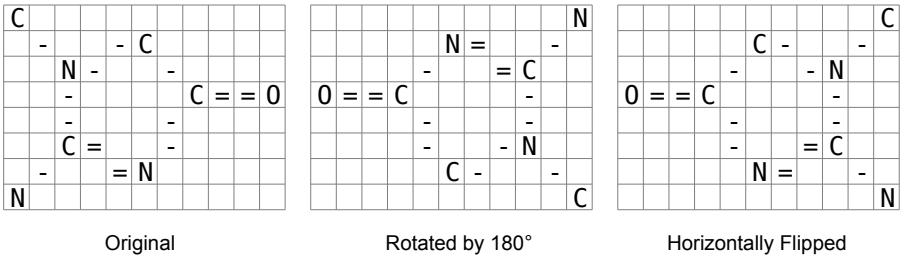


Abbildung 4.8: Dasselbe Molekül im Original (links) um 180° gedreht (Mitte) und horizontal gespiegelt (rechts). Bei nicht symmetrischen Strukturen ist eine 180°-Drehung einer horizontalen Spiegelung nicht gleich.

Die horizontale Spiegelung wird durch

$$x\text{-}pos'_i = -x\text{-}pos_i + 2 \cdot x\text{-}center \tag{4.21}$$

berechnet. Dabei wird durch die Änderung ihres Vorzeichens die Position $x\text{-}pos_i$ um die y-Achse gespiegelt. Durch das zweifache Aufaddieren als *Offset* (einmal links von der y-Achse und einmal rechts davon) wird das Zentrum *x-center* wieder in den Wertebereich der Positionen verschoben.

Auch bei Rundungsoperationen, mit deren Hilfe die Positionen in Indizes innerhalb des Rasters umgerechnet werden, ist eine Verschiebungstransformation sinnvoll. Da eine Positionsinvarianz durch die Verwendung von *Convolution*-Schichten bereits vorhanden

ist, wird bei dieser Transformation lediglich auf den Effekt eingegangen, dass ein einzelnes Atom durch eine kleine Verschiebung von ±0,5 über den Schwellenwert hinauskommen kann, der bestimmt, welcher von zwei beieinander liegenden Zellen es zugeordnet wird. Da sich dies von Atom zu Atom ändern kann, ist es möglich, dass zwei Atome, die vorher direkt untereinander lagen, nun schräg untereinander liegen. Ein Beispiel für diesen Effekt ist in Abbildung 4.9 zu sehen. Aus diesem Grund wird die Verschiebung um ±0,5 ausgeführt, was maximal zu einer Verschiebung um eine Zelle innerhalb des Rasters führen kann. Entsprechend ist bereits ein Spielraum der Größe 1 am Rand des Rasters ausreichend, um eine Überschreitung der Rastergrenzen zu verhindern.

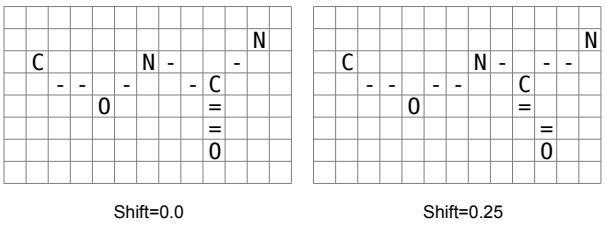


Abbildung 4.9: Dasselbe Molekül ohne Verschiebung (links) und mit einer ¼-Verschiebung (rechts) der x-Positionen. Die beiden Stickstoffatome und das rechte Sauerstoffatom haben sich nach rechts verschoben, während die restlichen Atome in ihren ursprünglichen Zellen geblieben sind.

Zur Berechnung der verschobenen Positionen *x-pos′* beziehungsweise *y-pos′* muss lediglich die Verschiebung *x-shift* beziehungsweise *y-shift* zur ursprünglichen Position hinzuaddiert werden. Dies geschieht durch

$$x\text{-}pos'_i = x\text{-}pos_i + x\text{-}shift \tag{4.22}$$

und

$$y\text{-}pos'_i = y\text{-}pos_i + y\text{-}shift. \tag{4.23}$$

Während des Trainings werden die Daten mit diesen drei Techniken zufällig transformiert. Dieselbe Transformation wird auf die Positionen aller Atome angewendet. Während einer Transformation wird um 0°–359° gedreht, mit einer 50 %-Wahrscheinlichkeit horizontal gespiegelt und die x- und y-Positionen werden um jeweils ±0–0,5 verschoben.

4.3 Merkmale

Jede Position innerhalb eines Rasters besitzt einen Vektor an Merkmalen. Bei Bildern sind dies üblicherweise die verschiedenen Farbkanäle. Für Moleküle kodieren diese Merkmale die Eigenschaften des an der betreffenden Stelle befindlichen Atoms beziehungsweise der dort vorhandenen Bindung. Sind an dieser Stelle weder ein Atom noch eine Bindung vorhanden, sind die Werte aller Merkmale 0. Es werden zwei Arten von Merkmalen verwendet: kodierte Symbole und chemische Eigenschaften.

Bei den kodierten Symbolen wird für die Atome das jeweilige Elementsymbol verwendet. Für Bindungen werden dieselben Symbole genommen, die auch bei *SMILES* verwendet werden, um den jeweiligen Bindungstyp zu repräsentieren. Um diese Symbole numerisch zu kodieren, wird ein 1-aus-n-kodierter Vektor genutzt. Das heißt, jedem im Datensatz auftretenden Symbol wird ein Index in diesem Vektor zugewiesen. Das zu kodierende Symbol wird dann an seinem zugehörigen Index mit einer 1 markiert, während der Rest der Symbole auf 0 gesetzt ist. Im Gegensatz zu einer Kodierung, bei der jedem Symbol ein ganzzahliger Wert zugewiesen wird, ist auf diese Weise keine unterschiedliche Distanz zwischen den einzelnen Symbolen impliziert, die von dem Netzwerk fälschlicherweise genutzt werden könnte.

Die Idee hinter der Nutzung von chemischen Eigenschaften ist es, dem Netzwerk nicht nur eine Unterscheidung zwischen den verschiedenen Atomen zu ermöglichen, sondern die Eingabedaten auch mit chemischem Wissen anzureichern. So können hier Ähnlichkeiten zwischen zwei unterschiedlichen Atomtypen erlernt werden, die möglicherweise mit deren biologischer Aktivität zusammenhängen. Hierfür werden nur Eigenschaften genommen, die leicht für einzelne Atome berechenbar sind. Tabelle 4.1 listet die verwendeten chemischen Eigenschaften auf, die in der Implementierung mit Hilfe von *RDKit* berechnet werden. Sie wurden aufgrund ihrer leichten Berechnung gewählt und enthalten teilweise miteinander korrelierende Informationen. Da die verschiedenen Eigenschaften einen sehr unterschiedlichen Wertebereich haben können, ist zu empfehlen, sie vorher zu normalisieren. Für Bindungen werden keine chemischen Eigenschaften berechnet.

Der Vektor, der die Symbole kodiert, und der Vektor der chemischen Eigenschaften werden dann am Ende zu einem Merkmalsvektor verknüpft. Letzten Endes besteht die Repräsentation jedes Moleküls aus einem 3D-Tensor mit zwei Dimensionen für die Position und einer Dimension mit den Merkmalen des Atoms beziehungsweise der Bindung an dieser Position. Dies ist anhand eines Beispiels in Abbildung 4.10 nochmals illustriert.

Tabelle 4.1: Chemische Eigenschaften.

Eigenschaft	**Beschreibung**
`aromatic`	Ob das Atom Teil eines aromatischen Rings ist
`atomic_number`	Ordnungszahl des Atoms
`formal_charge`	Formalladung des Atoms
`in_ring`	Ob das Atom Teil eines Rings ist
`isotope`	Isotop, zu dem das Atom gehört
`mass`	Masse des Atoms
`mol_log_p`	Beteiligung des Atoms am Octanol-Wasser-Verteilungskoeffizienten des Moleküls
`mol_mr`	Beteiligung des Atoms an der molaren Masse des Moleküls
`number_hs`	Anzahl an benachbarten Wasserstoffatomen
`number_neighbors`	Anzahl an benachbarten Atomen (ausgenommen Wasserstoffatome)
`valence`	Valenz des Atoms

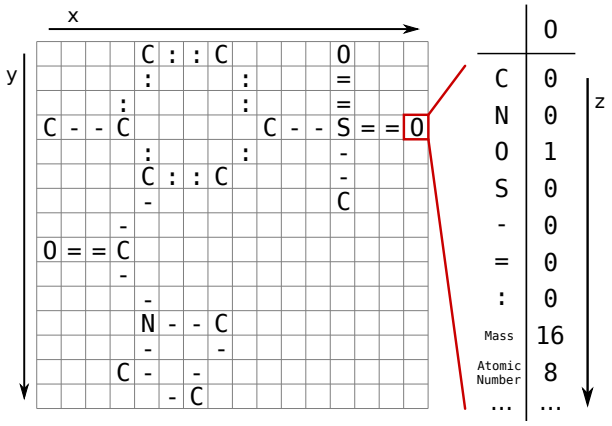


Abbildung 4.10: Die Repräsentation eines Moleküls besteht aus zwei Dimensionen für die Position (x, y) und einer Position für die Merkmale (z).

4.4 Verwendete Netzwerkarchitektur

Die Architektur des verwendeten Netzwerks ist an erfolgreiche Architekturen angelehnt, die für maschinelles Sehen konzipiert wurden. Der grundsätzliche Aufbau eines VGG-16-Netzwerks besteht aus *Convolution*-Blöcken und einem abschließenden Block an *Dense*-Schichten. Die *Convolution*-Blöcke bestehen dort aus mehreren *Convolution*-Schichten und einer abschließenden Max-*Pooling*-Schicht. Dabei erhöht sich die Anzahl an gelernten Filtern pro Block, während durch das Max-*Pooling* die Anzahl der Positionen verringert

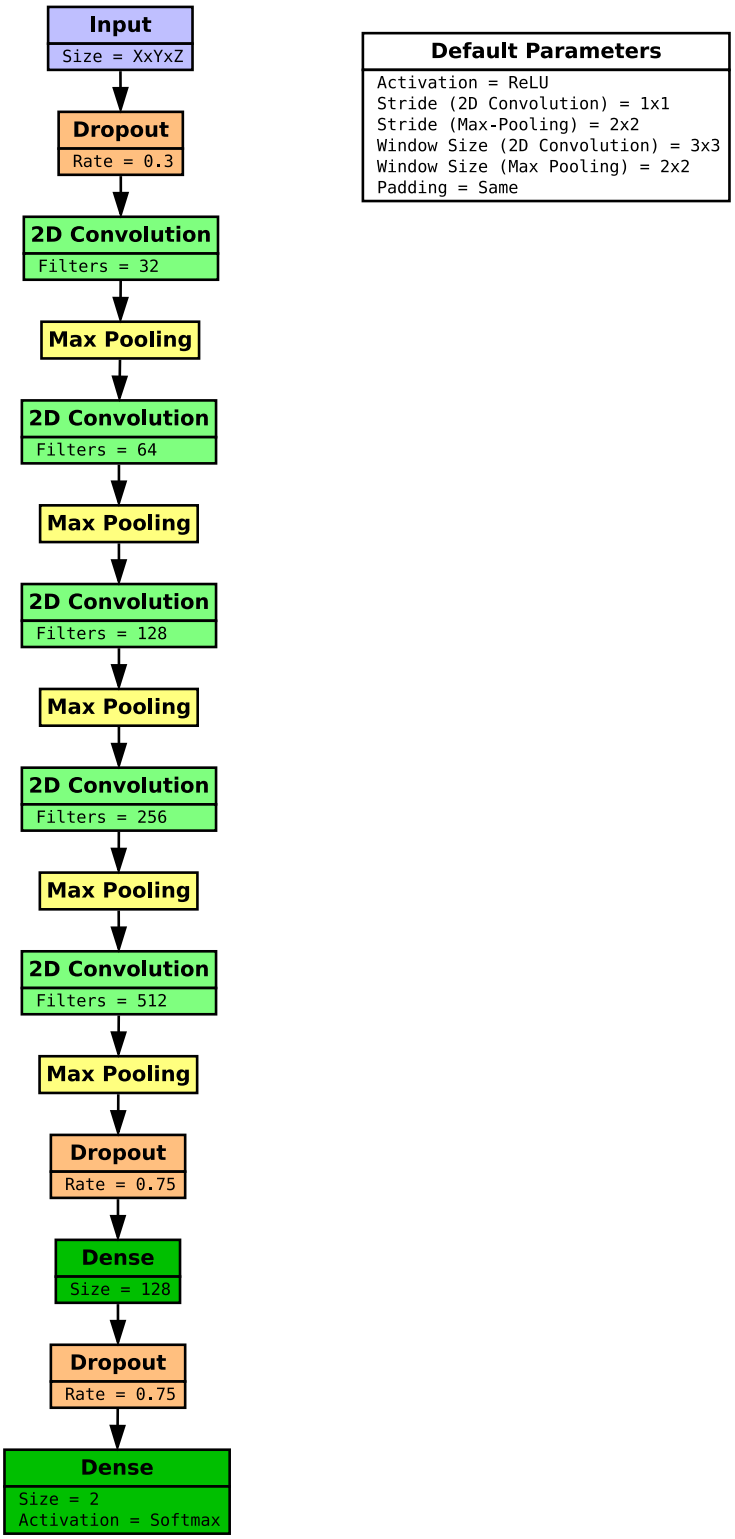


Abbildung 4.11: Netzwerkarchitektur zum Lernen mit Moleküldaten im Rasterformat.

wird. Der am Ende befindliche Block an *Dense*-Schichten hat die Klassifizierung zur Aufgabe und entspricht einem *Multi-Layer Perceptron*.

Auf der Suche nach einer Architektur, die sich für die gewählte Repräsentation der Moleküle und die zu erfüllenden Aufgaben am besten eignet, wurden verschiedene Kombinationen aus *Convolution*-, Max-*Pooling*-, *Dense*- und *Dropout*-Schichten händisch ausprobiert. Unter anderem wurden auch *Dropout*-Schichten zwischen den *Convolution*-Blöcken und mehrere *Convolution*-Schichten hintereinander versucht. Ebenso erfolgte eine Optimierung der jeweils zur Verfügung stehenden Parameter jeder Schicht. Die am Ende in ihrer Vorhersagequalität als auch in der Beständigkeit ihrer Ergebnisse am besten abschneidende Architektur ist in Abbildung 4.11 zu sehen. Hier bestehen die fünf *Convolution*-Blöcke aus jeweils nur einer *Convolution*- und einer Max-*Pooling*-Schicht. Die Anzahl gelernter Filter verdoppelt sich mit jedem Block, während die Anzahl der Positionen pro Dimension halbiert wird. Als *Multi-Layer Perceptron* zur Klassifizierung werden zwei *Dense*-Schichten eingesetzt. Um einer Überanpassung des Netzwerks vorzubeugen, befinden sich *Dropout*-Schichten sowohl direkt hinter dem Input als auch vor den *Dense*-Schichten.

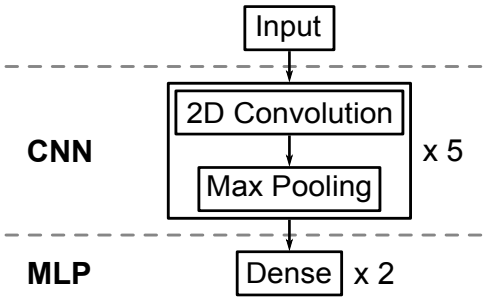


Abbildung 4.12: Das Netzwerk besteht aus zwei Teilen: einem *Convolutional Neural Network* aus fünf *Convolution*-Blöcken zur Generierung von Merkmalen und einem *Multi-Layer Perceptron* aus zwei *Dense*-Schichten zur Klassifikation.

Das Netzwerk kann, wie in Abbildung 4.12 gezeigt, in zwei funktionelle Teile aufgegliedert werden. Das *Convolutional Neural Network* lernt die Generierung nützlicher Merkmale. Das anschließende *Multi-Layer Perceptron* lernt anhand dieser Merkmale, die Moleküle zu klassifizieren. Diese Aufteilung ermöglicht es, den bereits trainierten Teil des *Convolutional Neural Network* auch einzeln als Merkmalsgenerator zu verwenden. Die so erzeugten Merkmale können dann in Kombination mit anderen Methoden, die auf der Verwendung von numerischen Merkmalen basieren, genutzt werden. So wäre es zum

Beispiel möglich, auf der Basis der von dem *Convolutional Neural Network* erzeugten Merkmale einen *Random Forest* zu trainieren, der dann die Klassifikation übernimmt.

Zum Training des Netzwerks wird *Cross Entropy* als Fehlerfunktion und Adam (mit einer Lernrate von 0,0001) als Optimierer eingesetzt. Zur Initialisierung findet die Methode von He et al. [28] Verwendung.

4.5 Zusammenfassung

Die Moleküle werden in eine rasterbasierte 2D-Repräsentation konvertiert. Zur Anordnung der Atome innerhalb des Rasters werden dieselben Positionen verwendet, die auch zur Anfertigung von 2D-Zeichnungen der Moleküle genutzt werden. Durch die ausgewählte Positionierung und eine passende Skalierung werden ein gleichmäßiger Abstand zwischen den Atomen und möglichst wenige Überschneidungen der Bindungen gewährleistet. Da ein Molekül auf mehr als eine Art korrekt repräsentiert werden kann, werden Transformationen während des Trainings herangezogen, um das Netzwerk entsprechend zu trainieren. Die eingesetzten Transformationen sind: Drehen, Spiegeln und Verschieben. Es werden zwei Arten von Merkmalen eingesetzt. Die kodierten Symbole bestehen aus einem Vektor, der die Elementsymbole von Atomen und das Bindungssymbol von Bindungen kodiert. Mit chemischen Eigenschaften kann das Netzwerk außerdem ausgehend von chemischem Wissen über die Atome lernen. Die eingesetzte Netzwerkarchitektur ähnelt der von gängigen Netzwerken für maschinelles Sehen. Das Netzwerk besteht aus zwei Teilen: einem *Convolutional Neural Network* zur Generierung von Merkmalen und einem *Multi-Layer Perceptron* zur Klassifikation. Das fertig trainierte *Convolutional Neural Network* kann auch zur Generierung von Merkmalen eingesetzt werden, die sich dann für andere, auf numerischen Merkmalen basierende Methoden nutzen lassen.

Kapitel 5

Finden relevanter Substrukturen

Unter Einsatz von *Saliency Maps* ist es möglich, die Entscheidung eines Netzwerks nachzuvollziehen und zu sehen, welche Regionen der Eingabedaten für die zugewiesene Klasse verantwortlich sind. Im Fall von Molekülen sind Regionen innerhalb der Eingabedaten Substrukturen des Moleküls.

5.1 Visuelle Erkennung wichtiger Substrukturen

Durch die *Saliency Map* kann in Form einer Heatmap visualisiert werden, welche Substrukturen das Netzwerk dazu veranlasst haben, ein Molekül als aktiv zu klassifizieren. Unter der Annahme, dass das Netzwerk das richtige Konzept hinter der Aktivität gelernt hat, heißt das, dass mit einer *Saliency Map* gezeigt werden kann, welche Substrukturen tatsächlich für die Aktivität des Moleküls verantwortlich sind. Dies kann ein enormer Wissensgewinn für den Biologen sein, dem sich damit ein besseres Verständnis der biologischen Vorgänge eröffnet. Dieses Wissen kann im nächsten Schritt, der *Lead Optimization*, dabei helfen, das Molekül noch weiter zu optimieren, um eine bessere Wirkung zu erhalten und Nebenwirkungen zu minimieren. In Abbildung 5.1 ist ein Beispiel einer *Saliency Map* für ein Molekül zu sehen.

Da die *Saliency Map* normal nur aufzeigt, welche Positionen innerhalb der Eingabe den größten Einfluss auf die Klassifizierung genommen haben, wird hier mittels einer Min-Max-Normalisierung pro Molekül immer der volle Wertebereich ausgenutzt. Diese

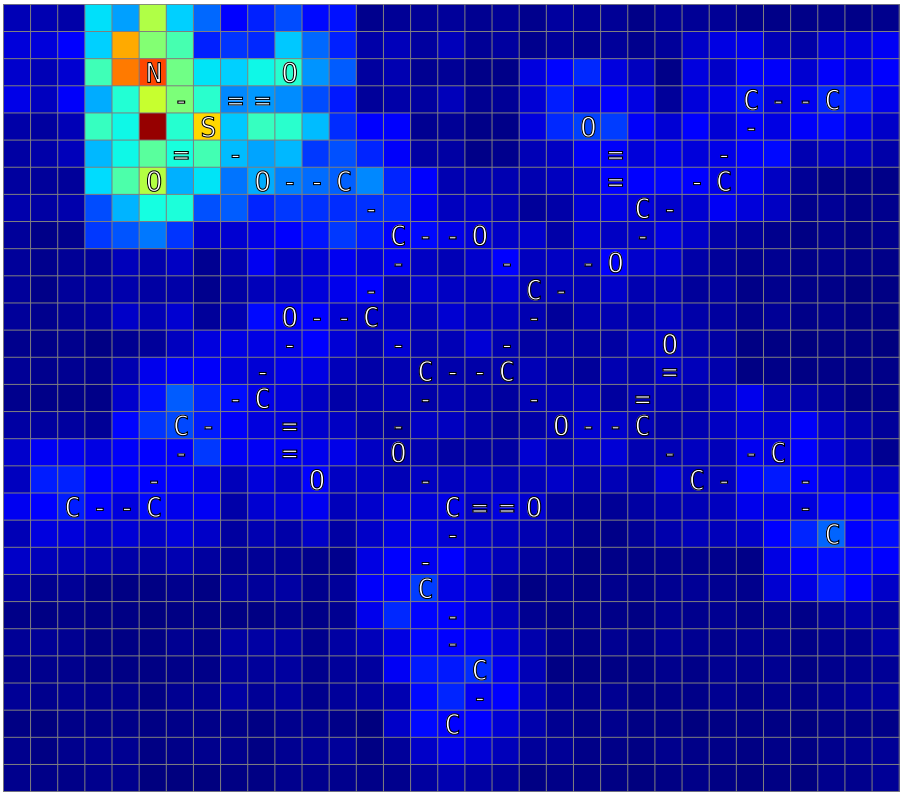


Abbildung 5.1: Eine *Saliency Map* für ein aktives Molekül eines α-Carboanhydrase-Datensatzes. Für diesen Datensatz ist bekannt, dass die Substruktur `S(=O)(=O)N` für die Aktivität verantwortlich ist. Diese ist in der *Saliency Map* klar hervorgehoben.

Normalisierung wird durch

$$\mathrm{Norm}(v, i) = \frac{v_i - \mathrm{Min}(v)}{\mathrm{Max}(v) - \mathrm{Min}(v)} \tag{5.1}$$

berechnet. Es wird der normalisierte Wert des ursprünglichen Wertes v_i berechnet, indem dieser um das Minimum aller Werte v verschoben (*Offset*) und anschließend durch den maximalen, ebenfalls verschobenen Wert skaliert wird. Dadurch gibt die *Saliency Map* an, welche Teile der Eingabe am ehesten zu einer Aktivität beitragen würden, ungeachtet dessen, wie wahrscheinlich eine Aktivität überhaupt ist.

Um diese Werte mit der vorhergesagten Aktivität des Moleküls ins Verhältnis zu setzen, werden sie zudem mit der Klassenwahrscheinlichkeit p der aktiven Klasse multipliziert, also

$$v'_i = \mathrm{Norm}(v, i) \cdot p. \tag{5.2}$$

Ein weiterer Vorteil der *Saliency Map* ist es, dass das Verhalten des Modells damit visualisierbar wird. So kann zum Beispiel die durch Transformationen erlernte Rotationsinvarianz betrachtet werden, indem *Saliency Maps* für unterschiedliche Transformationen desselben Moleküls berechnet werden. Abbildung 5.2 zeigt drei verschiedene Rotationen desselben Moleküls. Auf allen *Saliency Maps* ist zu erkennen, dass dieselbe Substruktur gefunden wird.

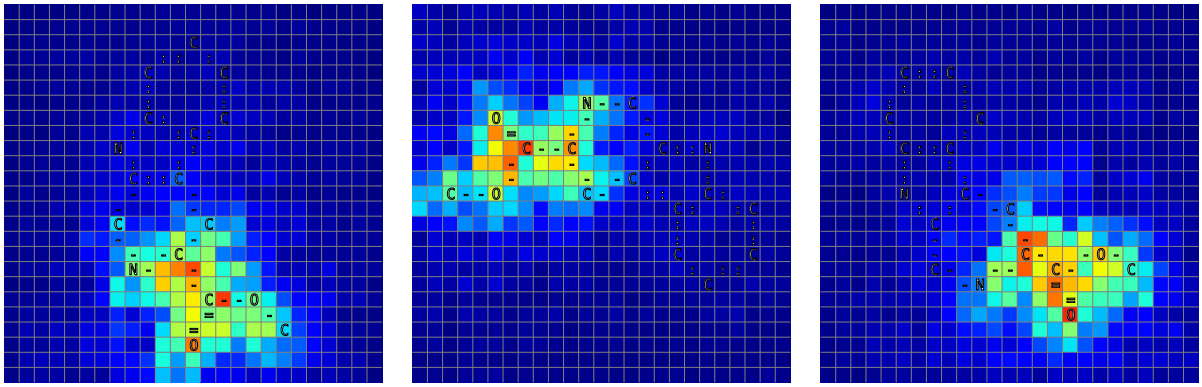


Abbildung 5.2: *Saliency Maps* für dasselbe Molekül mit unterschiedlicher Rotation. Wie zu sehen, wird dieselbe Substruktur gefunden, obwohl die Datenrepräsentation eine andere ist.

5.2 Automatisierte Substruktur-Extraktion

Es wäre wünschenswert, einen Überblick über die wichtigsten Substrukturen innerhalb des gesamten Datensatzes zu erhalten. Da hierfür ein händisches Durchsehen der *Saliency Maps* allerdings unrealistisch ist, ist der Einsatz einer automatisierten Substruktur-Extraktion sinnvoll. Auf diese Weise kann eine Liste der Substrukturen erstellt werden, die für die Klassifizierung am wichtigsten sind.

Ähnlich der Segmentierung von Vordergrund und Hintergrund in der Bildanalyse kann ein Schwellenwert auf den Werten der *Saliency Map* genutzt werden, um wichtige Strukturen von unwichtigen zu trennen. Auf diese Weise erhält man zusammenhängende Substrukturen. Ein Wert, der sich erfahrungsgemäß als sinnvoller Schwellenwert herausgestellt hat, ist 0,25. In Abbildung 5.3 ist ein Beispiel dieses „Ausschneidens“ von Substrukturen zu sehen. Die so ausgeschnittenen Substrukturen werden auf ihre Atome reduziert und dann anhand der Bindungen untereinander wieder zu einer Strukturbeschreibung (kanonische *SMILES*-Zeichenkette) umgewandelt. Auf diese Weise erhält

man vergleichbare Substrukturen. Gleichzeitig werden, durch diesen Ansatz, nicht miteinander verbundene Substrukturen voneinander getrennt. Um den Wert für die jeweilige Substruktur zu berechnen, wird der Durchschnittswert der enthaltenen Atome genutzt.

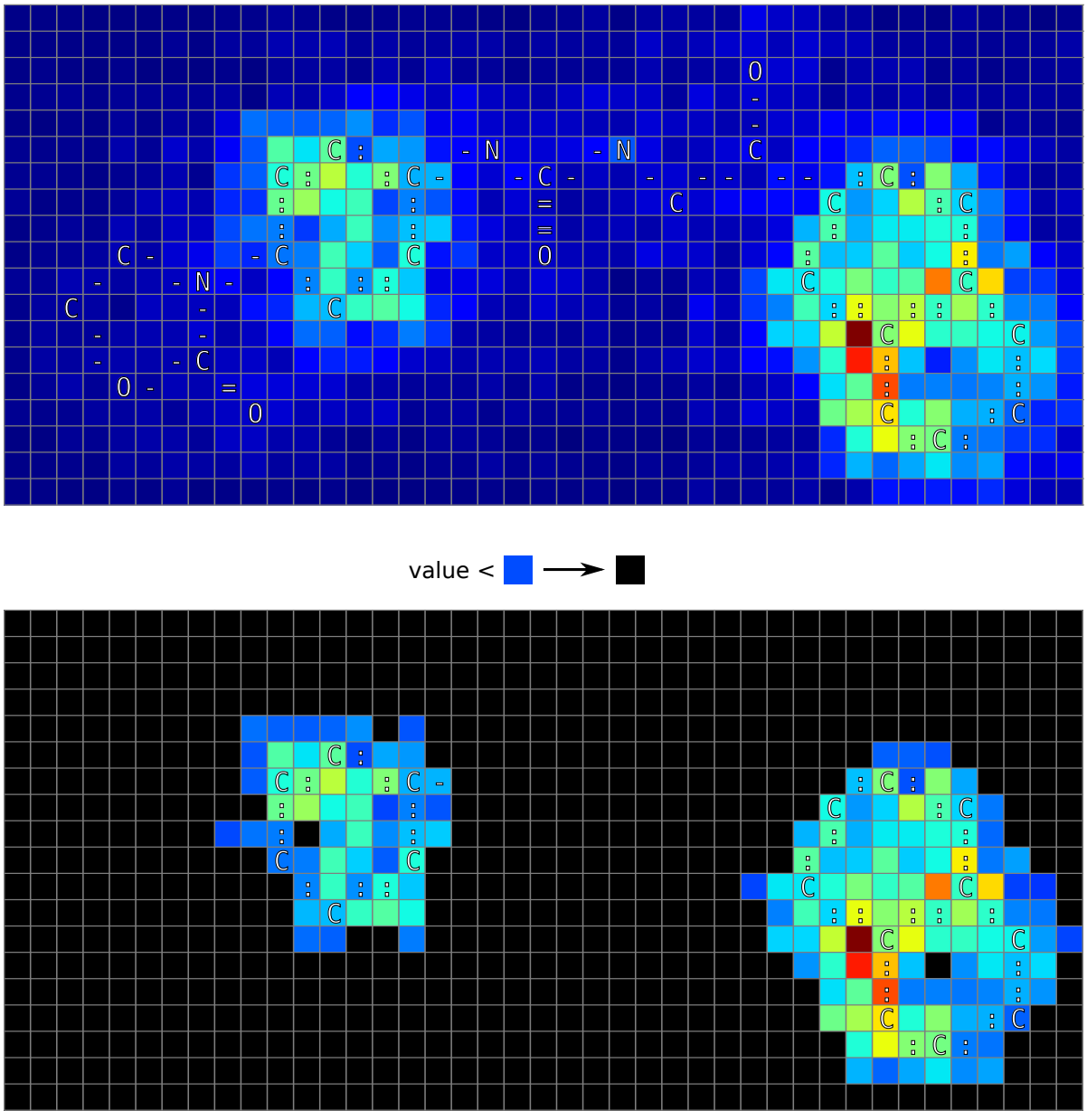


Abbildung 5.3: Die *Saliency Map* eines Moleküls vor (oben) und nach (unten) der Anwendung eines Schwellenwerts. Alle übrig gebliebenen zusammenhängenden Atome bilden jeweils eine Substruktur.

Die so erhaltenen Substrukturen werden dann über den gesamten Datensatz gesammelt und verschiedene Statistiken für jede einzigartige Substruktur berechnet. *vs* ist

dabei der Durchschnitt des *Saliency*-Werts für jedes Auftreten der Substruktur. f ist die Häufigkeit des Auftretens und n die Anzahl enthaltener Atome. Basierend auf diesen Statistiken kann für jede Substruktur eine Bewertung s durch

$$s_i = vs_i \cdot f_i \cdot n_i \tag{5.3}$$

berechnet werden.

vs beschreibt, wie sehr die Substruktur zu der Entscheidung, ob ein Molekül aktiv ist, beigetragen hat, und ist damit das primäre Argument, warum eine Substruktur als wichtig angesehen werden sollte.

Mit der Häufigkeit f wird bewertet, wie oft eine Substruktur überhaupt als wichtige Substruktur (also als eine, die über dem Schwellenwert liegt) erkannt wurde. Ohne diesen Faktor könnte eine Substruktur, die nur ein einziges Mal, dann aber mit einem hohen *Saliency*-Wert gefunden wurde, über alle anderen Substrukturen im restlichen Datensatz dominieren.

Durch die Verwendung von n werden komplexe Substrukturen bevorzugt. Andernfalls würden kleine Substrukturen dominieren. Diese haben zum einen eine größere Wahrscheinlichkeit aufzutreten und können zum anderen leicht das Resultat eines Ausschneidens durch Anwendung eines Schwellenwerts sein, bei dem nur wenige der Atome über diesem Schwellenwert liegen. Während bei der Segmentierung in der Bildanalyse häufig eine Methode zum Entfernen von Rauschen genutzt wird, ist dies hier nicht sinnvoll. In manchen Fällen kann schon eine kleine Anzahl Atome der Auslöser für Aktivität sein und darf deshalb nicht als Rauschen direkt herausgefiltert werden, wie das mit einzelnen Pixeln geschieht. Deshalb ist ein Benachteiligen kleiner Strukturen durch das Einrechnen als Faktor eine bessere Lösung.

Für eine bessere Interpretierbarkeit der Bewertung kann es sinnvoll sein, diese noch durch eine Normalisierung auf einen Wert zwischen 0 und 1 zu bringen. Das Ergebnis ist eine sortierte Tabelle an Substrukturen zusammen mit deren Bewertungen, die einen Hinweis darauf geben können, wie viel Einfluss die jeweilige Substruktur auf das zu lösende Klassifizierungsproblem hat.

5.3 Zusammenfassung

Indem die durch eine *Saliency Map* gefundenen Regionen im Fall von Molekülen direkt Substrukturen zugeordnet werden können, haben *Saliency Maps* hier einen besonders hohen Nutzen. Zum einen können sie für einen Experten zum Verständnis des biologischen Verhaltens eines spezifischen Moleküls beitragen. Zum anderen eignen sie sich, um automatisiert eine Liste der wichtigsten Substrukturen zu erstellen. Diese können einem Experten als wichtige Informationen zum Verständnis der biologischen Zusammenhänge dienen.

Kapitel 6

Softwarearchitektur

Im Folgenden werden Entwurfsprinzipien für die Implementierung der in dieser Arbeit entwickelten Methode vorgestellt. Dabei liegt ein Schwerpunkt auf der Modularität, dem Fortsetzen bereits begonnener Experimente und der möglichst effizienten Nutzung von parallelen Ressourcen. Die Implementierung [54] steht auf GitHub [2] unter GPLv3 [3] zur Verfügung.

Da die existierende Implementierung auf Python [8] und NumPy [6] basiert, sei zudem erwähnt, dass es in diesem Fall wichtig ist, Operationen auf den Tensoren möglichst unter Verwendung der NumPy-Methoden in C auszuführen. Bei der Verwendung von NumPy liegen die Daten in C vor, und auch die NumPy-Methoden führen die Operationen direkt in C aus. Dadurch müssen weder die Daten in Python übertragen werden, noch verlangsamt Python die Ausführung. Somit dient Python lediglich zur Beschreibung der auszuführenden Operationen, während die schnellere C-Implementierung diese dann tatsächlich ausführt.

Gegenüber der anfänglichen naiven Implementierung haben die Code-Optimierung und die volle Ausnutzung der parallelen Ressourcen, wie sie im Folgenden beschrieben wird, zu einer Geschwindigkeitsverbesserung geführt, durch die dasselbe Experiment in einem Fünftel der Zeit berechnet werden kann. Datensätze, die in ihrer Tensordarstellung nicht mehr in den Hauptspeicher passen, werden nun sogar in einem Zehntel der Zeit berechnet. Grund ist, dass die hier beschriebene Implementierung nicht die vollständigen Daten an einem Stück vorverarbeitet und die Daten somit nicht auf die Festplatte ausgelagert werden müssen, wenn der Hauptspeicher nicht mehr genügend Platz bietet.

6.1 Schrittweiser Aufbau von Experimenten

Um beim Aufbau der Experimente Modularität zu ermöglichen, sind diese in Schritte aufgeteilt. Ein Schritt ist eine Gruppe von konkreten Implementierungen, die denselben Typ von Aufgabe erfüllen. Durch eine Aneinanderreihung von Schritten ist es möglich, diese weitgehend frei miteinander zu kombinieren.

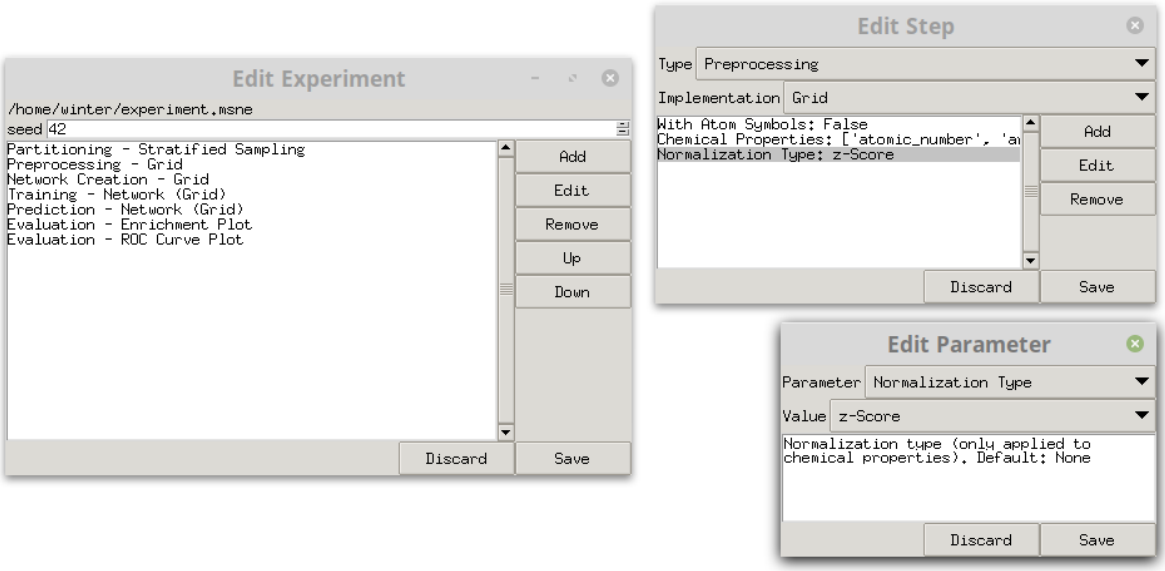


Abbildung 6.1: Benutzeroberfläche zum Bearbeiten eines Experiments. Ein Experiment besteht aus einer Liste von Schritten (links). Jede Implementierung eines Schritts hat eine Liste an Parametern (rechts oben), die konfiguriert werden können (rechts unten). Dank der vorhandenen Informationen über Parameter sind nur valide Werte wählbar, und es existiert auch eine Beschreibung. Schritte können dem Experiment hinzugefügt, entfernt und in der Ausführungsreihenfolge sortiert werden.

Schritte werden immer hintereinander ausgeführt, so dass ein Schritt üblicherweise die nötigen Daten für den nächsten Schritt berechnet. Die Kommunikation zwischen den einzelnen Schritten funktioniert über Dateien, die deren Ergebnisse beinhalten, und über globale Parameter. Jede Implementierung eines Schrittes setzt das Vorhandensein bestimmter Dateien voraus. Des Weiteren hat jede Implementierung ihre eigenen Parameter. Die Parameter können Restriktionen wie Minimum und Maximum besitzen und haben immer auch einen Beschreibungstext. Diese Informationen ermöglichen eine leicht zu bedienende Benutzeroberfläche (implementiert in Tkinter [10], siehe Abbildung 6.1), die eine schnelle Erstellung von Experimenten ermöglicht. Sie dient zur Erzeugung einer JSON-Konfigurationsdatei (JavaScript Object Notation) [15], wie sie in Listing 6.1

```
{   "seed": 42,
    "steps":
    [   {   "id": "stratified_sampling",
            "parameters": { "train_percentage": 10 },
            "type": "partitioning" },
        {   "id": "tensor_2d",
            "parameters":
            {   "atom_symbols": false,
                "chemical_properties":
                [   "atomic_number",
                    "aromatic",
                    "mass",
                    "number_neighbors",
                    "number_hs",
                    "valence",
                    "in_ring",
                    "mol_log_p",
                    "mol_mr" ],
                "normalization": "z-Score" },
            "type": "preprocessing" },
        {   "id": "tensor_2d",
            "type": "network_creation" },
        {   "id": "tensor_2d",
            "parameters": { "epochs": 20 },
            "type": "training" },
        {   "id": "tensor_2d",
            "type": "prediction" },
        {   "id": "enrichment_plot",
            "type": "evaluation" },
        {   "id": "roc_curve_plot",
            "type": "evaluation" } ] }
```

Listing 6.1: JSON-Konfigurationsdatei für das in Abbildung 6.1 in der Benutzeroberfläche abgebildete Experiment. Die Parameter der einzelnen Schritte enthalten nur die Abweichungen von den Standardparametern.

dargestellt ist. In Tabelle 6.1 sind alle verfügbaren Schritte und deren Implementierung aufgelistet.

Schritte können frei kombiniert werden. Es ist lediglich zu beachten, dass sie jeweils entsprechende Daten benötigen. Zum Beispiel kann ein rasterbasiertes Netzwerk erst

Tabelle 6.1: Verfügbare Schritte.

Schritt	**Implementierung**
Target Generation	• Substructure
Partitioning	• Stratified Sampling
Preprocessing	• Grid
Feature Generation	• Learned Features (Grid) • ECFP Fingerprint • MACCS Fingerprint • Saliency Map Substructures • MoSS • Combined Features
Network Creation	• Grid
Training	• Network (Grid) • Random Forest
Prediction	• Network (Grid) • Random Forest
Evaluation	• Enrichment Plot • ROC Curve Plot
Interpretation	• Calculate Saliency Maps • Render Saliency Maps • Extract Saliency Map Substructures • Calculate Substructure Locations • Render Substructure Locations • Saliency Map Evaluation

trainiert werden, wenn sowohl die Rastervorverarbeitung vollzogen als auch ein entsprechendes Netzwerk erstellt worden ist. Fehlen entsprechende Daten, bemerkt der Schritt vor der Ausführung dies und gibt es als Fehler aus.

Ein Experiment beschreibt jeweils nur die Vorgehensweise, nicht aber, welche Daten genutzt werden. Diese werden beim Start des Experiments als Argument übergeben und liegen dann intern als globale Parameter vor.

6.2 Fortsetzen bereits angefangener Experimente

In der Forschung stehen Rechenressourcen nicht immer nur einem Nutzer zur Verfügung. Das heißt, sie müssen eventuell freigegeben und das eigene Experiment zu einem späteren Zeitpunkt fortgesetzt werden. Auch können in der Entwicklung neuer Methoden

schnell Fehler zu einem Absturz führen. Aus diesen beiden Gründen ist es wünschenswert, Experimente möglichst in dem Setting wieder fortzusetzen, wo sie abgebrochen wurden beziehungsweise das System abgestürzt ist.

Da die einzelnen Schritte über Dateien kommunizieren, ist das Fortsetzen eines Experiments leicht möglich. Jeder Schritt überprüft als Erstes, ob die Daten, die er berechnen soll, bereits existieren. In diesem Fall werden die Berechnungen einfach übersprungen. Sollten die Daten noch nicht vorliegen, werden sie berechnet und gegebenenfalls zunächst in eine temporäre Datei geschrieben. Erst wenn der Schritt erfolgreich zu Ende geführt worden ist, wird diese Datei an ihren eigentlichen Ort verschoben. Auf diese Weise lässt sich vermeiden, dass eine unfertige Datei gefunden und aufgrund dessen der Schritt übersprungen wird. Da das Training eines Netzwerks besonders zeitintensiv ist, wird das trainierte Netzwerk nach jeder Epoche gespeichert, wobei das alte Netzwerk überschrieben wird. Die aktuelle Epoche wird ebenfalls herausgeschrieben, um an dieser Stelle das Training wieder fortsetzen zu können. Die einzelnen Parameter eines Schrittes werden zu einem *Hash*-Wert kombiniert, der der Ausgabedatei als Namenssuffix dient.

Mehrere Experimente können sich einige Dateien auch teilen. Ein Beispiel dafür sind die berechneten *Fingerprints*, die unabhängig vom eigentlichen Experiment für denselben Datensatz bei gleichen Parametern immer gleich ausfallen.

6.3 Parallele Vorverarbeitung

Das Training von Netzwerken ist trotz der Verwendung schneller GPU-Prozessoren immer noch sehr zeitaufwendig. Auch kann die Vorverarbeitung einiges an Zeit und Rechenleistung in Anspruch nehmen. Deshalb sind besonders diese zwei Punkte so zu optimieren, dass eine effiziente Ausnutzung paralleler Ressourcen gegeben ist. Dabei wird das effiziente Training des Netzwerks bereits von der verwendeten *Deep-Learning*-Bibliothek (in der existierenden Implementierung Keras [5] und TensorFlow [9]) implementiert. Da das Training auf der Grafikkarte rechnet, sind die Kerne des Hauptprozessors noch frei. Gerade bei einer intensiven Vorverarbeitung sollte diese dann nicht sequentiell vor, sondern parallel zum Training stattfinden und die freien Prozessoren möglichst alle ausnutzen.

Die Daten für die gerade trainierte *Mini-Batch* müssen allerdings zum Start des Trainings bereits vorliegen. Aus diesem Grund ist es sinnvoll, dass die Vorverarbeitung die *Mini-Batches* in der zu trainierenden Reihenfolge vorbereitet und in eine FIFO-Warteschlange einreiht. Das Training kann aus dieser Warteschlange die nächsten Trai-

ningsdaten holen und mit diesen trainieren, während gleichzeitig die nächsten Daten vorverarbeitet werden. Dieses Konzept ist in Abbildung 6.2 dargestellt.

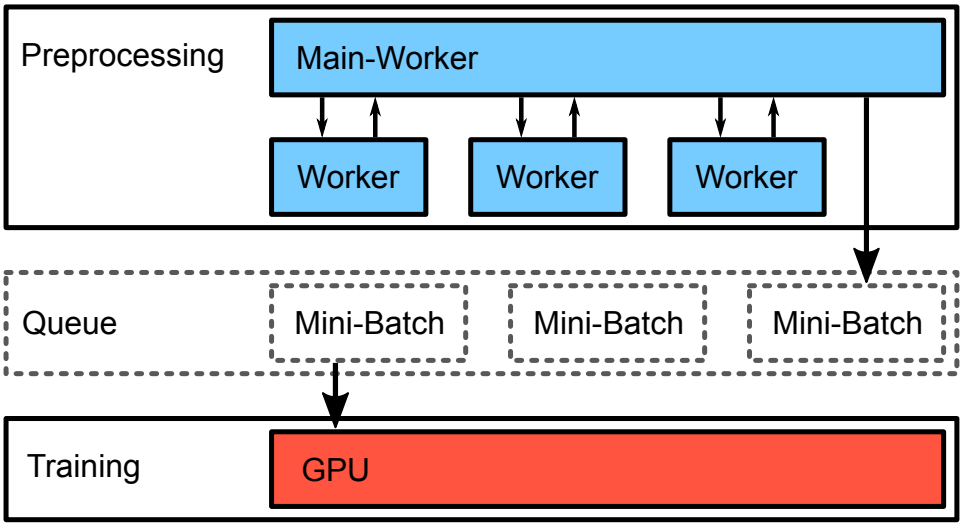


Abbildung 6.2: Während das Training auf der Grafikkarte (rot) mit einer *Mini-Batch* trainiert, erzeugt die Vorverarbeitung bereits die nächsten *Mini-Batches* mit Hilfe mehrerer Arbeiterprozesse auf dem Hauptprozessor (blau) und legt diese in die Warteschlange.

Die Vorverarbeitung selbst teilt ihre Daten gleichmäßig auf mehrere Arbeiterprozesse auf. So werden alle zur Verfügung stehenden Prozessoren genutzt. Jeder Arbeiterprozess bekommt die Liste an Molekülen, die vorverarbeitet werden sollen, im *SMILES*-Format. Die Prozesse berechnen lediglich die Positionen der Atome und deren chemische Eigenschaften. Beides wird als Liste an den Hauptprozess zurückübermittelt und erst dort in den Tensor, der die gesamte *Mini-Batch* enthält, eingefügt. Dies ermöglicht es, die Menge an zu übertragenden Daten zu minimieren, da das Raster eines Moleküls ohnehin sehr dünn besetzt ist. Weil es bei einer kleinen *Mini-Batch*-Größe und einer größeren Anzahl an Prozessorkernen schnell zu einem unverhältnismäßigen Overhead kommen kann, macht es Sinn, tatsächlich größere Blöcke an Daten vorzuverarbeiten und diese erst zum Schluss auf *Mini-Batch*-Größe zu unterteilen.

Die Informationen, die zur Vorverarbeitung nötig sind, wie zum Beispiel *Offsets* oder Statistiken zur Normalisierung, werden im Vorhinein berechnet. Die tatsächliche Vorverarbeitung findet allerdings erst statt, wenn die Daten benötigt werden. Ein zusätzlicher Vorteil dieses Vorgehens ist, dass die großen Datenmengen der vorverarbeiteten Daten weder auf die Festplatte geschrieben und anschließend wieder gelesen werden müssen, noch müssen sie alle in den Speicher passen. Erst kurz bevor sie verbraucht werden,

werden sie erstellt. Durch die Limitierung der verwendeten Warteschlange auf eine maximale Größe kann verhindert werden, dass mehr Daten vorverarbeitet werden, als in den Speicher passen. Beim Einfügen in die Warteschlange kann diese den Arbeiterprozess blockieren, bis Daten konsumiert wurden und entsprechend für neu erzeugte Daten wieder Platz ist.

6.4 Zusammenfassung

Die verwendete Softwarearchitektur besitzt einen modularen Aufbau, der ein leichtes Austauschen verschiedener Implementierungen ermöglicht. Durch die Verwendung von Dateien als Zwischenergebnisse können abgebrochene oder abgestürzte Experimente fortgesetzt und müssen so nicht wieder von Anfang an durchgeführt werden. Auch können sich mehrere Experimente bestimmte Zwischenergebnisse teilen, falls diese ohnehin dieselben wären. Dank der Nutzung paralleler Ressourcen können die Vorverarbeitung und das Training gleichzeitig laufen und dabei sowohl alle Kerne des Hauptprozessors als auch des Grafikprozessors ausnutzen. Des Weiteren kommen keine zu großen Datenmengen auf, da diese erst direkt vor ihrer Verwendung generiert werden.

Kapitel 7

Experimentelle Resultate

Im Folgenden werden Resultate von Experimenten präsentiert, die sowohl die allgemeine Leistung der beschriebenen Methoden also auch die optimalen Hyperparameter evaluieren. Die Ergebnisse zeigen jeweils nur den Trend, der anhand der hier verwendeten Datensätze zu erkennen ist, und können für andere Datensätze anders ausfallen.

In den Ergebnissen werden jeweils die Früherkennung mit Hilfe des *Enrichment Factor* und die Vorhersagequalität auf der Basis des gesamten Datensatzes unter Verwendung der *Area under the ROC Curve* betrachtet.

Die 141 verwendeten Datensätze stammen von PubChem und wurden von Riniker et al. [40] zur Evaluierung von biologischen *Fingerprints* zusammengestellt. Bei den angegebenen IDs handelt es sich um die *Screening*-AID. Auf folgender Website können weitere Informationen zu jedem *Screening* gefunden werden: `https://pubchem.ncbi.nlm.nih.gov`.

Bei der Datenbereinigung wurden folgende Moleküle aus den Datensätzen entfernt:

- Moleküle, die nicht in allen Datensätzen vorkommen
- Moleküle, die in einem der Datensätze zu uneindeutigen Ergebnissen geführt haben
- Moleküle, die die Plausibilitätsprüfung von *RDKit* nicht überstanden haben

Dadurch ergibt sich eine Menge von 231 644 Molekülen. Für jedes der Moleküle existiert ein *Screening*-Resultat zu jedem der 141 verschiedenen *Screenings*. Pro *Screening* sind jeweils zwischen 34 und 6859 aktive Moleküle vorhanden.

Tabelle 7.1: Die 10 häufig genutzten PubChem-Datensätze.

AID	Titel	Aktiv	Inaktiv
1899	TR-FRET-based primary biochemical high-throughput screening assay to identify inhibitors of Hepatitis C Virus (HCV) core protein dimerization	517	231127
1906	QFRET-based counterscreen for PFM18AAP inhibitors: biochemical high throughput screening assay to identify inhibitors of the Cathepsin L proteinase (CTSL1)	1049	230595
1947	Fluorescence polarization-based counterscreen for RBBP9 inhibitors: primary biochemical high throughput screening assay to identify inhibitors of the serine hydrolase family member Fam108B	537	231107
1950	Fluorescence polarization-based primary biochemical high throughput screening assay to identify inhibitors of the Epstein-Barr virus nuclear antigen 1 (EBNA-1)	699	230945
2057	Fluorescence polarization-based primary biochemical high throughput screening assay to identify inhibitors of myeloid cell leukemia sequence 1 (MCL1) interactions with BIM-BH3 peptide	1132	230512
2129	Primary biochemical high throughput screening assay to identify inhibitors of BCL2-related protein, long isoform (BCLXL)	1182	230462
2130	Fluorescence polarization-based primary biochemical high throughput screening assay to identify inhibitors of Protein Phosphatase Methylesterase 1 (PME-1)	1066	230578
2174	Counterscreen for PME1 inhibitors: fluorescence polarization-based primary biochemical high throughput screening assay to identify inhibitors of lysophospholipase 1 (LYPLA1)	296	231348
2177	Counterscreen for PME1 inhibitors: fluorescence polarization-based primary biochemical high throughput screening assay to identify inhibitors of lysophospholipase 2 (LYPLA2)	775	230869
2234	Counterscreen for inhibitors of EBNA-1: fluorescence polarization-based biochemical high throughput primary assay to identify inhibitors of the Epstein-Barr virus-encoded protein, ZTA	248	231396

Zum Partitionieren der Datensätze wird jeweils ein *Stratified Sampling* genutzt, um das Verhältnis zwischen aktiven und inaktiven Molekülen beizubehalten. Des Weiteren wird anschließend immer ein *Oversampling* herangezogen, um das Klassenverhältnis gleichzusetzen, und die Anordnung wird zum Schluss randomisiert, um einer Ordnung in den Daten vorzubeugen.

Die Experimente wurden jeweils so konstruiert, das jedes Experiment in etwa 4–5 Tagen auf der zur Verfügung stehenden Hardware (CPU 2,6 GHz · 28 Cores, GPU mit 9,3 TeraFLOPS und 16 GiB Memory, 192 GiB RAM) ausgeführt werden konnte. Entsprechend variiert die Anzahl an evaluierten Datensätzen. Details zu den zehn Datensätzen, die in den meisten Experimenten genutzt wurden, sind in Tabelle 7.1 zu finden. In Experimenten, die besonders rechenaufwändig sind, wurden lediglich die ersten beiden Datensätze verwendet. Sie eignen sich gut, da erfahrungsgemäß Vorhersagen für den

Datensatz 1899 nur wenige Fehler zeigen, während die Vorhersagen für Datensatz 1906 oft viele Fehler beinhalten.

Falls nicht anders angegeben, werden als Standard-Hyperparameter die in Tabelle 7.2 aufgeführten Werte verwendet. Sie haben sich bei vorläufigen Experimenten mit kleineren Datensätzen als die besten erwiesen. Um eine Konsistenz und Vergleichbarkeit innerhalb aller hier durchgeführten Experimente zu gewährleisten, werden sie auch dann beibehalten, wenn sich in den Ergebnissen andere Hyperparameter als besser herausstellen. Die Größe des Rasters liegt bei Verwendung der Standard-Hyperparameter für das verwendete Set an PubChem-Molekülen bei $121 \times 121 \times 13$.

Tabelle 7.2: Standard-Hyperparameter.

Hyperparameter	**Wert**	**Erklärung**
Trainingspartition	10 %	10 % Trainingsdaten, entsprechend den kleinen Mengen die man auch in echtem VHTS zur Verfügung hat
Skalierungsfaktor	2,0	Doppelte Abstände zwischen Atomen basierend auf den von *RDKit* berechneten Positionen
Elementsymbole	`False`	Es werden keine 1-aus-n kodierten Elementsymbole genutzt

Chemische Eigenschaften	`atomic_number`
	`aromatic`
	`mass`
	`number_neighbors`
	`number_hs`
	`valence`
	`in_ring`
	`mol_log_p`
	`mol_mr`
		Eine Auswahl an Eigenschaften die als Merkmal eines Atoms dienen

Normalisierung	z-Score	Normalisierung der chemischen Eigenschaften
Mini-Batch-Größe	100	100 Moleküle pro *Mini-Batch*
Epochen	20	Training für 20 Epochen, danach ist mit Überanpassung zu rechnen

Bei Experimenten, bei denen ein Trend untersucht wird, werden die Ergebnisse als Liniendiagramm angezeigt. Hier wird oftmals der *Moving Average* (bei einer Fenstergröße von 5) gezeigt und in schwächerer Darstellung die konkreten Ergebnisse.

7.1 Streuung der Ergebnisse durch Zufallseffekte

In vielen Aspekten der Erstellung der Trainingsdaten und des Trainings des Modells spielt der Zufall eine Rolle. In Tabelle 7.3 werden alle Aspekte, die vom Zufall beeinflusst werden, aufgeführt. Um einen besseren Eindruck davon zu gewinnen, wie stark dieser Zufallseffekt sich auf die Ergebnisse auswirkt, wurde für zwei Datensätze jeweils 41-mal ein Modell trainiert und evaluiert. Dabei wurde jedes Mal ein anderer *Random-Seed* (Startwert für den (Pseudo-)Zufallszahlengenerator) genutzt, der den Zufall für die genannten Aspekte beeinflusst.

Tabelle 7.3: Aspekte, die durch den Zufall beeinflusst werden.

Schritt	**Beeinflusste Aspekte**
Partitioning	• Aufteilen in Training- und Testpartition • Zufällige Reihenfolge
Network Creation	• Initialisierung des Netzwerks
Training	• Transformationen
Evaluation	• Zufällige Sortierung (Tie-Breaker)

Es ist zu erwarten, dass der größte Einflussfaktor die zufällige Auswahl der Trainingsdatenpunkte ist. Deshalb wurde ein zweites Experiment durchgeführt, bei dem wieder jeweils 41 weitere Modelle trainiert wurden. Diesmal blieb die Auswahl der Trainingsdaten jedes Mal gleich, während alle anderen Aspekte weiterhin durch den Zufall beeinflusst wurden.

In den Ergebnissen (Abbildungen 7.1 und 7.2) ist zu sehen, dass sich der *Enrichment Factor* bei 5 % um maximal 2,925 unterscheidet. Bei der *Area under the ROC Curve* ist der maximale Unterschied 0,085. Wie erwartet vermindert sich die Streuung bei Verwendung der gleichen Trainingsdaten deutlich. Hier ist der maximale Unterschied des *Enrichment Factors* bei 5 % bei 1,592 und der *Area under the ROC Curve* bei 0,031. Es ist zu beachten, dass für die Experimente mit fester Partitionierung eine beliebige Partition gewählt wurde. Da es unwahrscheinlich ist, dass sie gerade dem Durchschnitt entspricht, hat sich auch der Median im Vergleich zu den vorherigen Ergebnissen verschoben. Es geht in diesen Experimenten aber ohnehin um die mögliche Streuung der Ergebnisse statt der absoluten Werte.

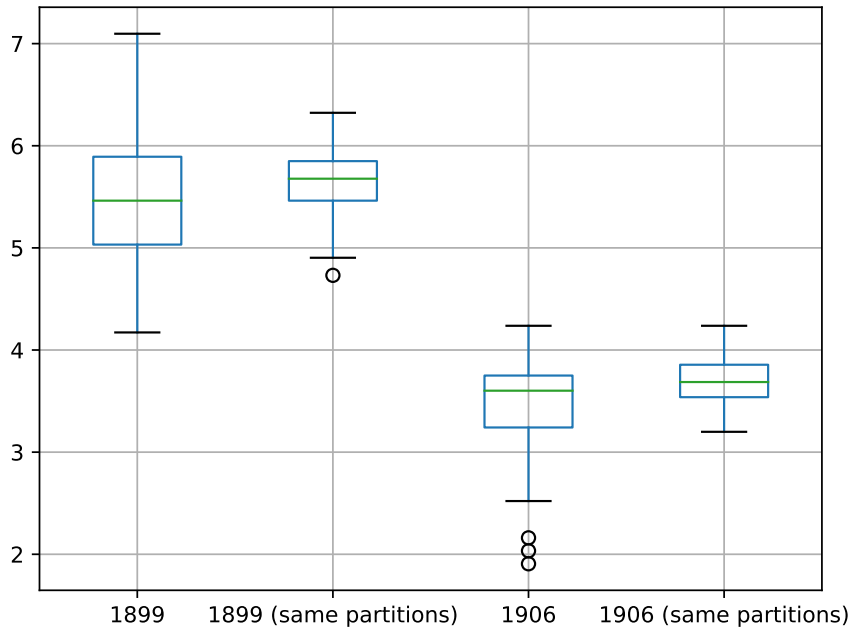


Abbildung 7.1: Streuung des *Enrichment Factor* bei 5 %, jeweils für zufällig gewählte Partitionen und unter Verwendung derselben Partitionen.

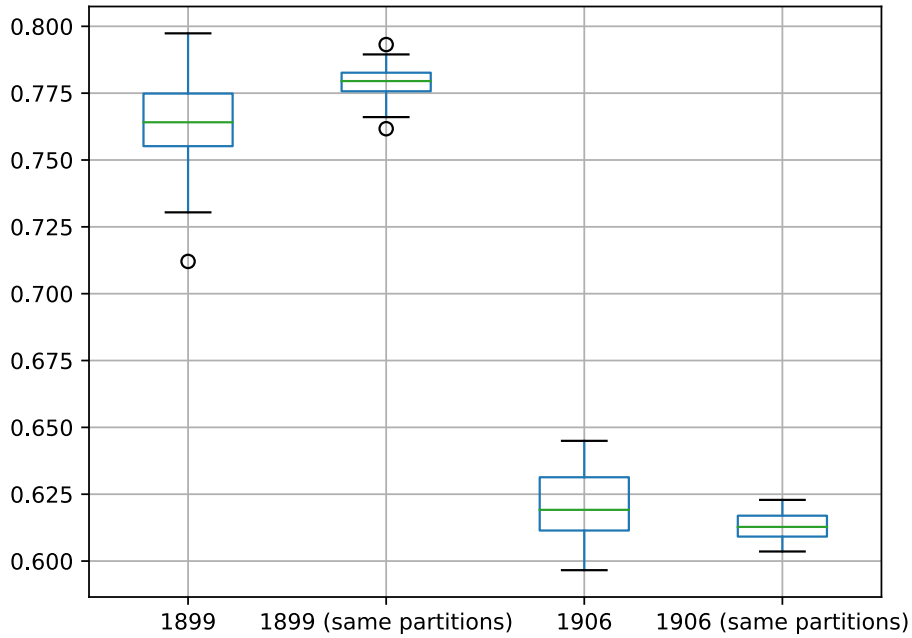


Abbildung 7.2: Streuung der *Area under the ROC Curve*, jeweils für zufällig gewählte Partitionen und unter Verwendung derselben Partitionen.

7.2 Nötige Anzahl an Trainingsdaten

Um herauszufinden, wie viele Trainingsdaten nötig sind, um ein ausreichend gutes Netzwerk zu trainieren, wurde ein Experiment durchgeführt, das die Vorhersagequalität bei verschiedenen Größen der Trainingspartition evaluiert. Es ist anzunehmen, dass durch eine zu geringe Menge an Daten eine Unteranpassung stattfindet, da in den wenigen Daten noch nicht genug Informationen enthalten sind, um sie auf ein Muster zurückzuführen. Gleichzeitig kann es auch zu einer Überanpassung auf die wenigen zur Verfügung stehenden Datenpunkte kommen.

Für das Experiment wurden zwei der PubChem-Datensätze genutzt. Jeder Datensatz wurde einmalig in 50 % Trainingsdaten und 50 % Testdaten unterteilt. Während für jeden Durchlauf die vollständigen Testdaten zur Auswertung genutzt wurden, kam zum Training jeweils nur ein Teil der Trainingsdaten zum Einsatz. Mit jedem Schritt wurde immer ein weiteres Prozent der Trainingsdaten (also 0,5 % des Gesamtdatensatzes) hinzugefügt, bis alle Trainingsdaten genutzt wurden. Jedes damit trainierte Model wurde für 20 Epochen trainiert. Durch die Nutzung der gleichen Anzahl an Epochen, aber einer unterschiedlichen Anzahl an Datenpunkten, wird das Netzwerk auch unterschiedlich häufig angepasst, da eine Epoche dadurch mehr *Mini-Batches* enthält. Ein Hochsetzen der Epochen, um dies auszugleichen, würde allerdings gerade bei wenigen Datenpunkten in erheblichem Maße eine Überanpassung fördern. Deshalb wurde hier eine feste Anzahl an Epochen gewählt.

In Abbildung 7.3 und 7.4 sind die Ergebnisse zu sehen. Wie zu erwarten, bringen mehr Trainingsdaten insbesondere zu Beginn eine deutliche Verbesserung. Sind bereits viele Trainingsdaten vorhanden, bringen weitere Daten einen geringeren Mehrwert. Dies ist an der abflachenden Kurve am Ende zu sehen. Trotzdem ist auch bei 50 % des Gesamtdatensatzes immer noch ein Trend nach oben zu erkennen, insbesondere in der Früherkennung. Der schwierigere der beiden Datensätze (1906) profitiert deutlich von mehr Daten. Hervorzuheben ist, dass in Datensatz 1899 lediglich 0,22 % der Moleküle aktiv sind, während dies bei Datensatz 1906 immerhin 0,45 % sind, womit bei derselben Partitionsgröße etwa doppelt so viele aktive Moleküle zum Training zur Verfügung stehen. Für die hier untersuchten Datensätze ist eine Trainingspartitionsgröße von etwa 10–20 % zu empfehlen, was etwa 20 000–40 000 Molekülen entspricht. Es ist zu beachten, dass für die anderen Experimente als Standard-Hyperparameter 10 % Trainingspartition gewählt wurde. Wie sich hier andeutet, ist es aber durchaus möglich, dass mit einer größeren Trainingspartition deutlich mehr Datensätze gute Ergebnisse erreichen.

Abbildung 7.3: Einfluss der Größe der Trainings-Partition auf den *Enrichment Factor* bei 5 %. Die Größe ist sowohl in Prozent als auch in der Anzahl unterschiedlicher Moleküle (ohne *Oversampling*) angegeben.

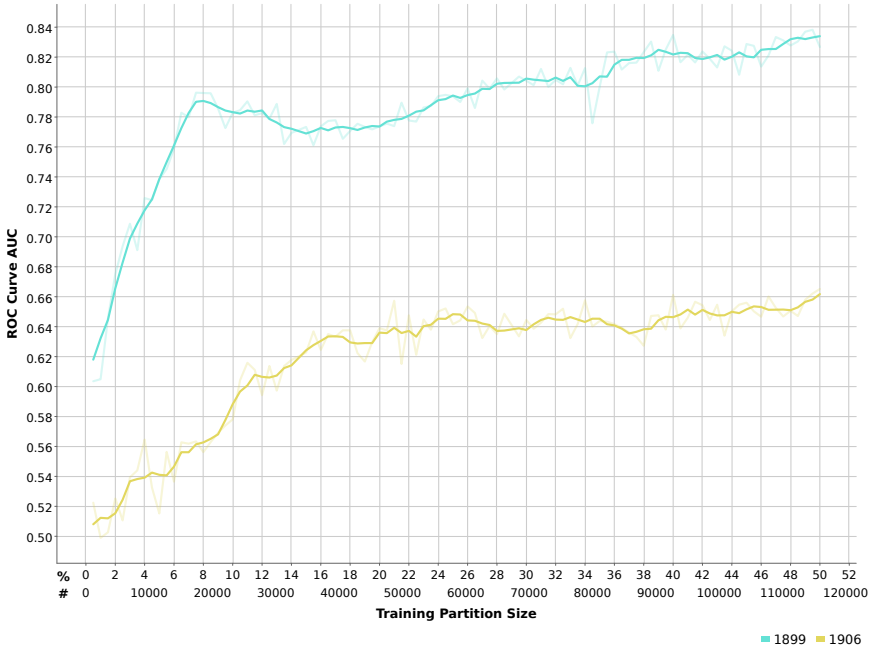


Abbildung 7.4: Einfluss der Größe der Trainings-Partition auf die *Area under the ROC Curve*. Die Größe ist sowohl in Prozent als auch in der Anzahl unterschiedlicher Moleküle (ohne *Oversampling*) angegeben.

7.3 Entwicklung des Netzwerks über mehrere Epochen

Um die nötige Anzahl an Epochen herauszufinden, wurde für zehn Datensätze jeweils ein Modell trainiert und dieses nach jeder einzelnen Epoche evaluiert.

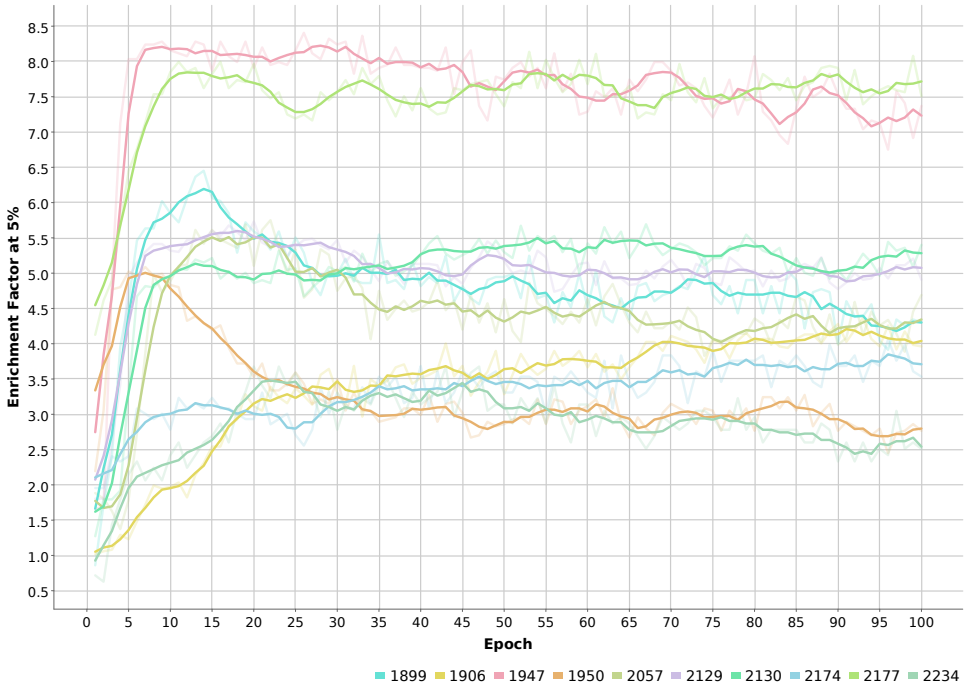


Abbildung 7.5: Entwicklung des Netzwerks nach jeder Epoche. Es wird der *Enrichment Factor* bei 5 % auf den Testdaten gezeigt.

In den Abbildungen 7.5 und 7.6 sind die Ergebnisse zu sehen. Wie zu erwarten, bringen die ersten Epochen eine deutliche Verbesserung. Viele der Datensätze haben bereits nach 10 Epochen ihren Höchstwert erreicht, während manche erst bei etwa 20 Epochen so weit sind. Für die meisten der Datensätze zeichnet sich der Trend ab, dass sie nach dem Erreichen ihres recht frühen Höchstwerts langsam etwas schlechter werden. Dies deutet auf einen Übergang von Anpassung zu Überanpassung hin. Bei einigen der Datensätze scheint das Problem der Überanpassung besonders stark zu sein. Grundsätzlich aber dürfte die Vorhersagequalität für den gesamten Datensatz stabiler sein als in der Früherkennung.

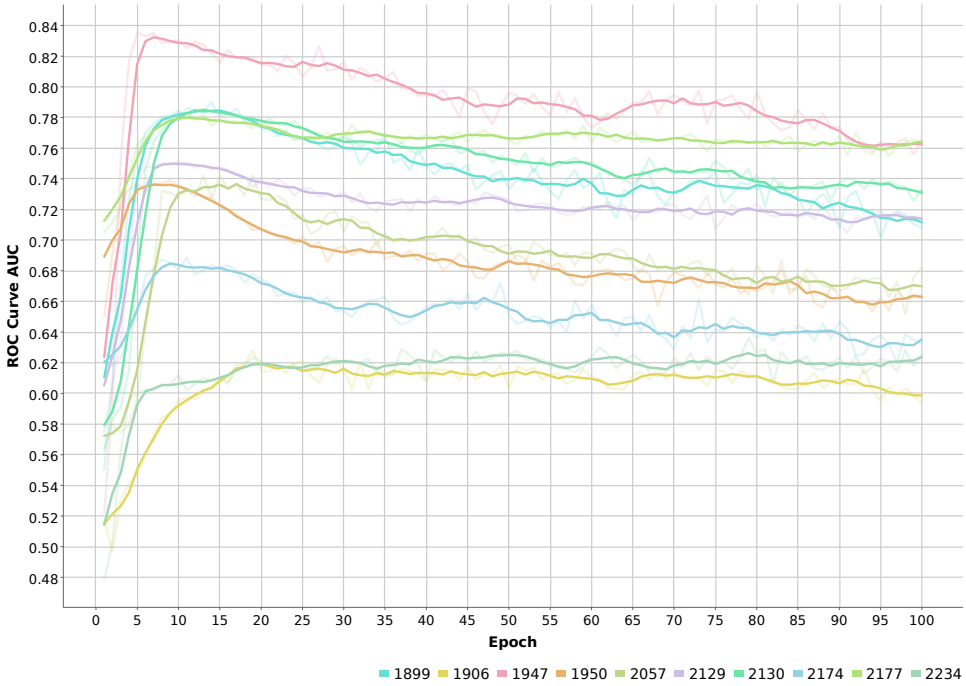


Abbildung 7.6: Entwicklung des Netzwerks nach jeder Epoche. Es wird die *Area under the ROC Curve* auf den Testdaten gezeigt.

7.4 Einfluss der Skalierung

Die Skalierung der Positionen der einzelnen Atome im Raster beeinflusst sowohl, wie gut diese voneinander getrennt sind, als auch die Größe der Eingabedaten und damit des Netzwerks selbst. Bei einer zu kleinen Skalierung kann es passieren, dass mehrere Atome in derselben Zelle landen und sich damit gegenseitig überschreiben. Auch können sie so nahe beieinander liegen, dass nicht zu sehen ist, ob und wenn ja welche Bindung sie zueinander haben. Die Skalierung sollte also groß genug ausfallen, um solche Probleme zu vermeiden. Da sie aber auch die Größe der Eingabedaten sowohl in der Höhe als auch in der Breite beeinflusst, kann eine zu große Skalierung zu einer zu hohen Laufzeit führen.

Die Ergebnisse in Abbildung 7.7 und 7.8 zeigen, dass bereits ein sehr kleiner Skalierungsfaktor zu Ergebnissen führen kann, die besser als bei einem zufälligen Raten ausfallen. Wie erwartet, führt ein größerer Faktor allerdings meist zu besseren Ergebnissen. Ab einem gewissen Abstand der Atome zueinander sollten die erwähnten Probleme nicht mehr auftauchen und damit kein größerer Unterschied mehr festzustellen sein. Dies ist auch bei den meisten Datensätzen zu sehen, deren Ergebnisse sich ab einem gewis-

sen Faktor nicht mehr stark ändern. Ein weiterer Faktor, der einer kleinen Skalierung zugutekommt, ist, dass bei ihr weniger Variationsmöglichkeiten dafür bestehen, wie eine Substruktur im Raster dargestellt wird. Es ist zu vermuten, dass die beste Skalierung davon abhängt, wie gut die für die Aktivität verantwortliche Substruktur mit dieser Skalierung abgebildet werden kann.

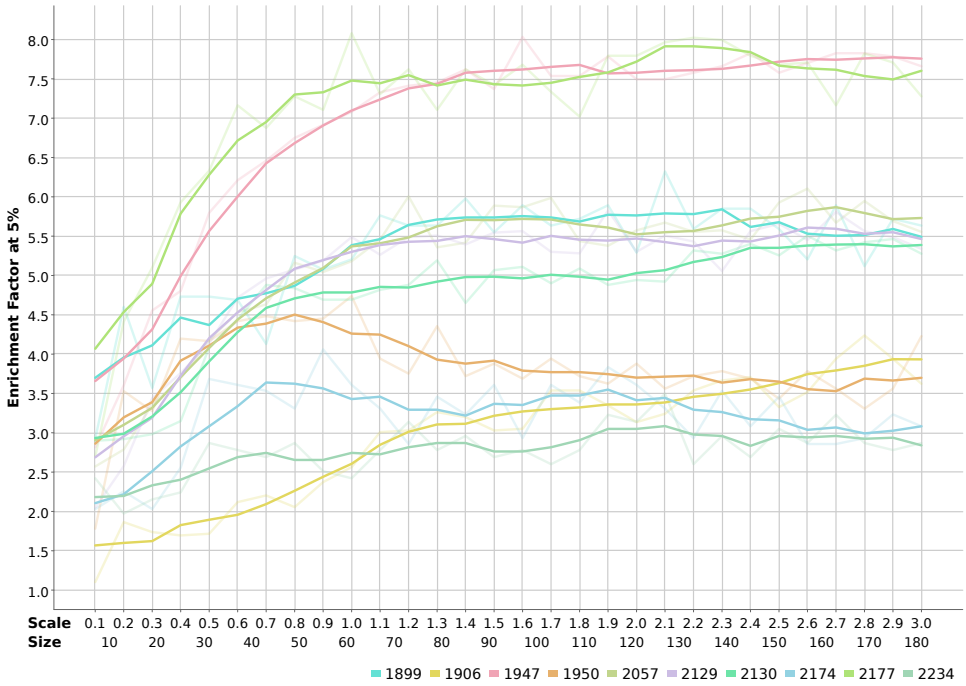


Abbildung 7.7: Einfluss der Skalierung auf den *Enrichment Factor* bei 5 %. Es sind sowohl der Skalierungsfaktor als auch die daraus resultierende Größe (Anzahl Zellen pro Positionsdimension) angegeben.

In Abbildung 7.9 ist die Laufzeit des Trainings dargestellt. Sie ist über alle Datensätze sehr konstant und zeigt kaum Abweichungen. Es ist zu erkennen, dass die Skalierung zu Beginn keinen Einfluss auf die Laufzeit hat. Dies liegt an der Softwarearchitektur der Implementierung. Da die Vorverarbeitung parallel zum Training ausgeführt wird, wird bei einem schnelleren Training dieses von der Vorverarbeitung ausgebremst. Die Skalierung wirkt sich kaum auf die Vorverarbeitung aus, da die Anzahl Atome und deren zu berechnende Eigenschaften sich nicht ändern und lediglich die Allokation des Rasters von der Größe beeinflusst wird. Sobald das Training länger braucht als die Vorverarbeitung, ist ein Anstieg der Laufzeit zu erkennen, der etwas stärker als ein linearer Anstieg ist. Interessant ist, dass bei manchen Größen eine Optimierung zu greifen scheint, die

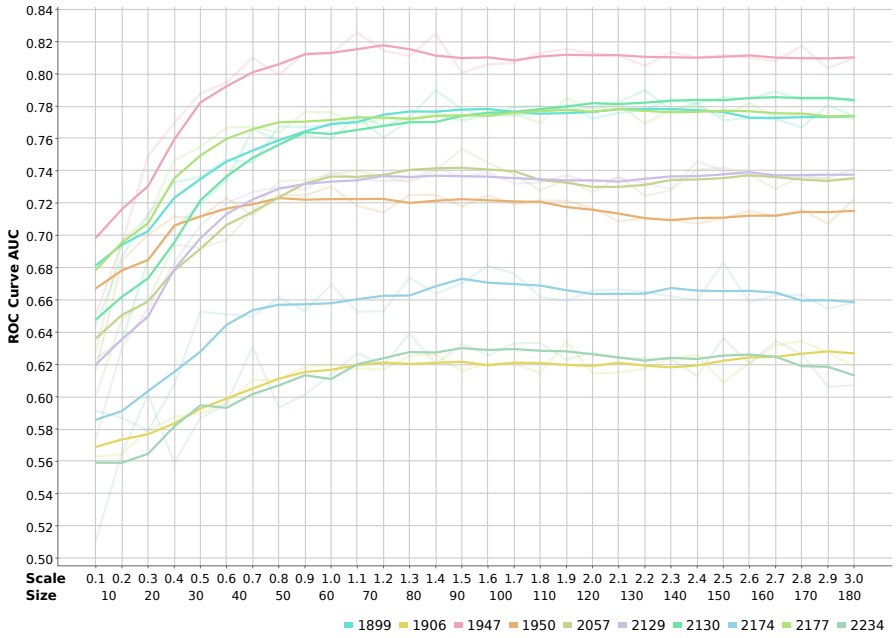


Abbildung 7.8: Einfluss der Skalierung auf die *Area under the ROC Curve*. Es sind sowohl der Skalierungsfaktor als auch die daraus resultierende Größe (Anzahl Zellen pro Positionsdimension) angegeben.

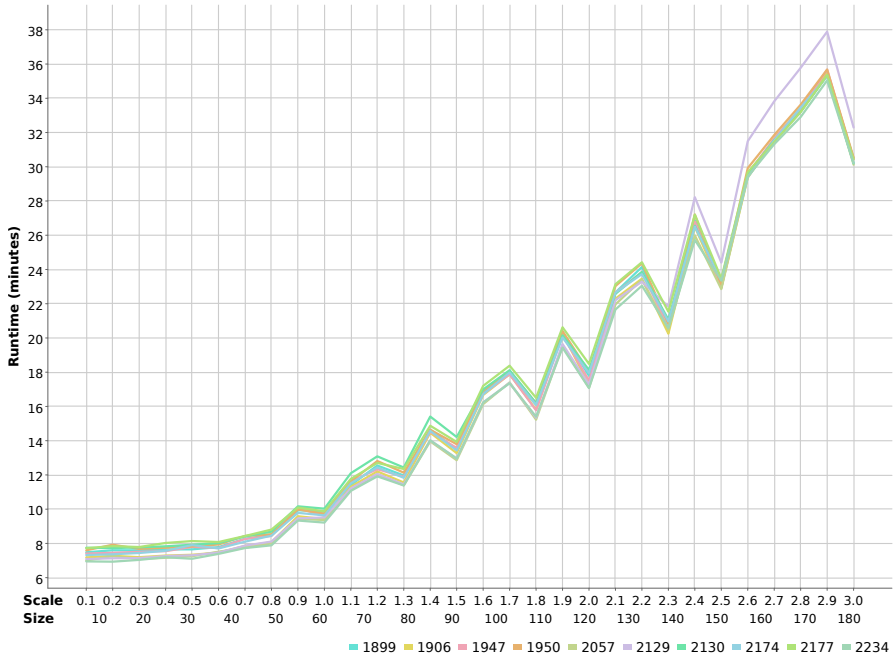


Abbildung 7.9: Laufzeit des gesamten Trainings (20 Epochen) in Minuten, basierend auf der Größe der Eingabedaten. Es sind sowohl der Skalierungsfaktor als auch die daraus resultierende Größe (Anzahl Zellen pro Positionsdimension) angegeben.

zu einer schnelleren Laufzeit führt. Trotz Bemühungen war es nicht in akzeptabler Zeit herauszufinden, woher dieser Unterschied kommt.

7.5 Nützlichkeit von chemischen Eigenschaften

Es wurde geprüft, wie die Hinzunahme einzelner chemischer Eigenschaften sich auf die Vorhersagequalität auswirkt. Dazu wurde die symbolbasierte Kodierung als Basismerkmal genommen. In jedem weiteren Experiment wurde jeweils eine chemische Eigenschaft zu diesem Basismerkmal hinzugefügt, um abzuschätzen, welchen Mehrwert diese bringt.

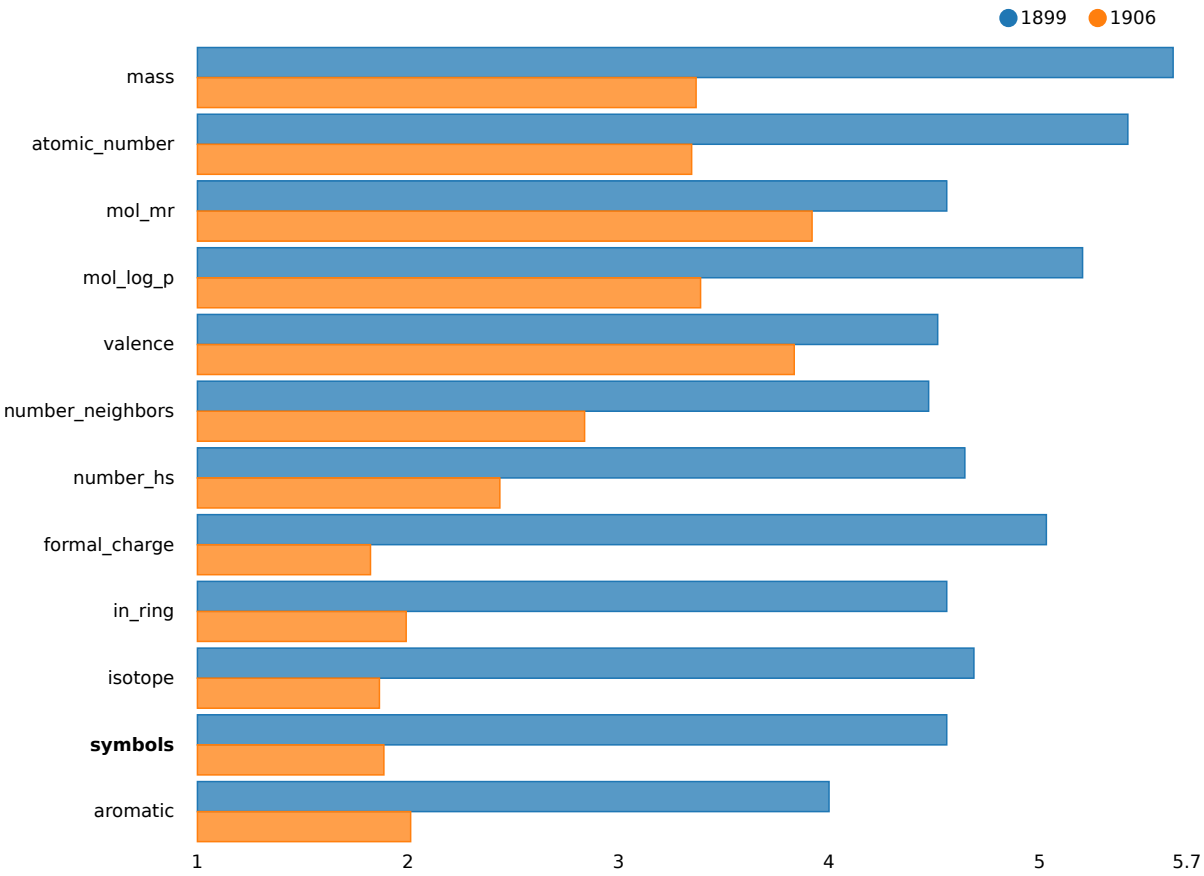


Abbildung 7.10: Der *Enrichment Factors* bei 5 % für jeweils eine chemische Eigenschaft in Kombination mit den kodierten Symbolen. Die Eigenschaften sind nach ihrer durchschnittlichen Effektivität sortiert. `symbols` ist der Wert, der ohne chemische Eigenschaften nur anhand der kodierten Symbole erzielt wird.

In Abbildung 7.10 und 7.11 sind die Ergebnisse der zwei getesteten Datensätze zu sehen. Die Effektivität der chemischen Eigenschaften unterscheidet sich zwischen diesen

beiden Datensätzen erheblich. Während zum Beispiel `formal_charge` für den einen Datensatz zu den besten gehört, bringt es für den anderen keine Besserung im Vergleich zur Verwendung allein der kodierten Symbole. Es zeichnen sich trotzdem vier Eigenschaften ab, die bei beiden Datensätzen deutlich die besten sind: `mass`, `atomic_number`, `mol_mr` und `mol_log_p`. Aufgrund der Unterschiede je nach Datensatz ist es zu empfehlen, alle Eigenschaften zu verwenden.

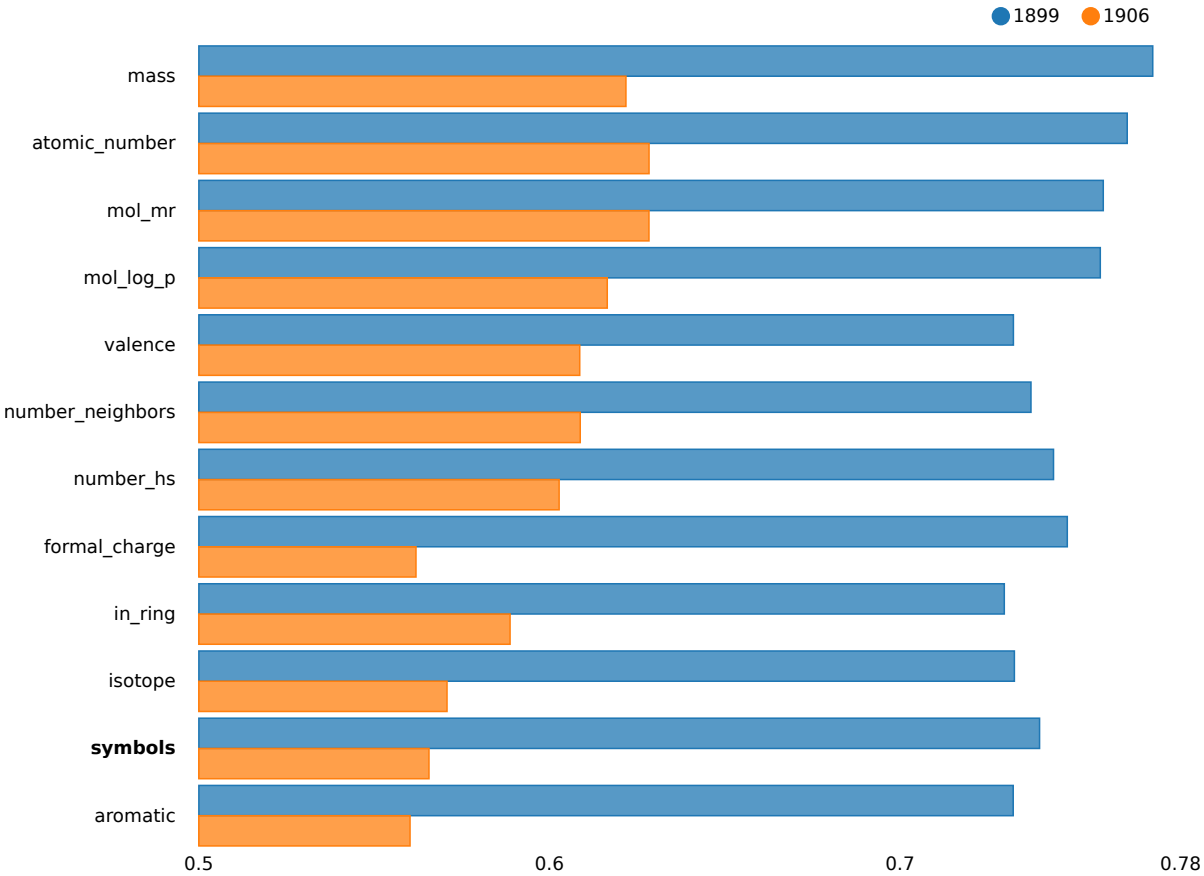


Abbildung 7.11: Die *Area under the ROC Curve* für jeweils eine chemische Eigenschaft in Kombination mit den kodierten Symbolen. Die Eigenschaften sind nach ihrer durchschnittlichen Effektivität sortiert. `symbols` ist der Wert, der ohne chemische Eigenschaften nur anhand der kodierten Symbole erzielt wird.

7.6 Einfluss von Transformationen

Um dem Netzwerk Invarianzen in Drehung, Spiegelung und Verschiebung beizubringen, werden Transformationen der Trainingsdaten vorgenommen. Damit der Effekt dieser

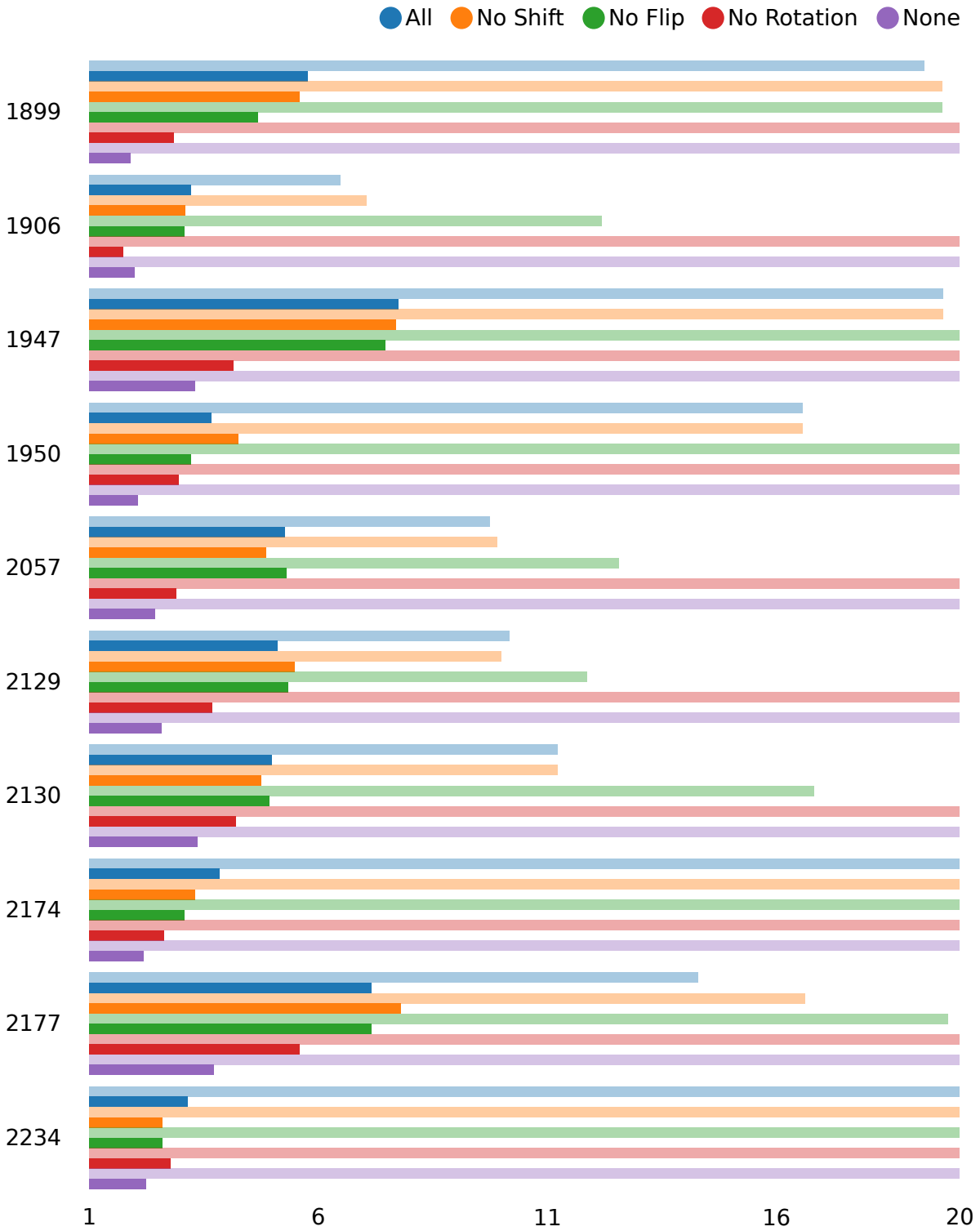


Abbildung 7.12: *Enrichment Factor* bei 5 % unter Verwendung der angegebenen Transformationen für Trainingsdaten (schwache Farben) und Testdaten (kräftige Farben).

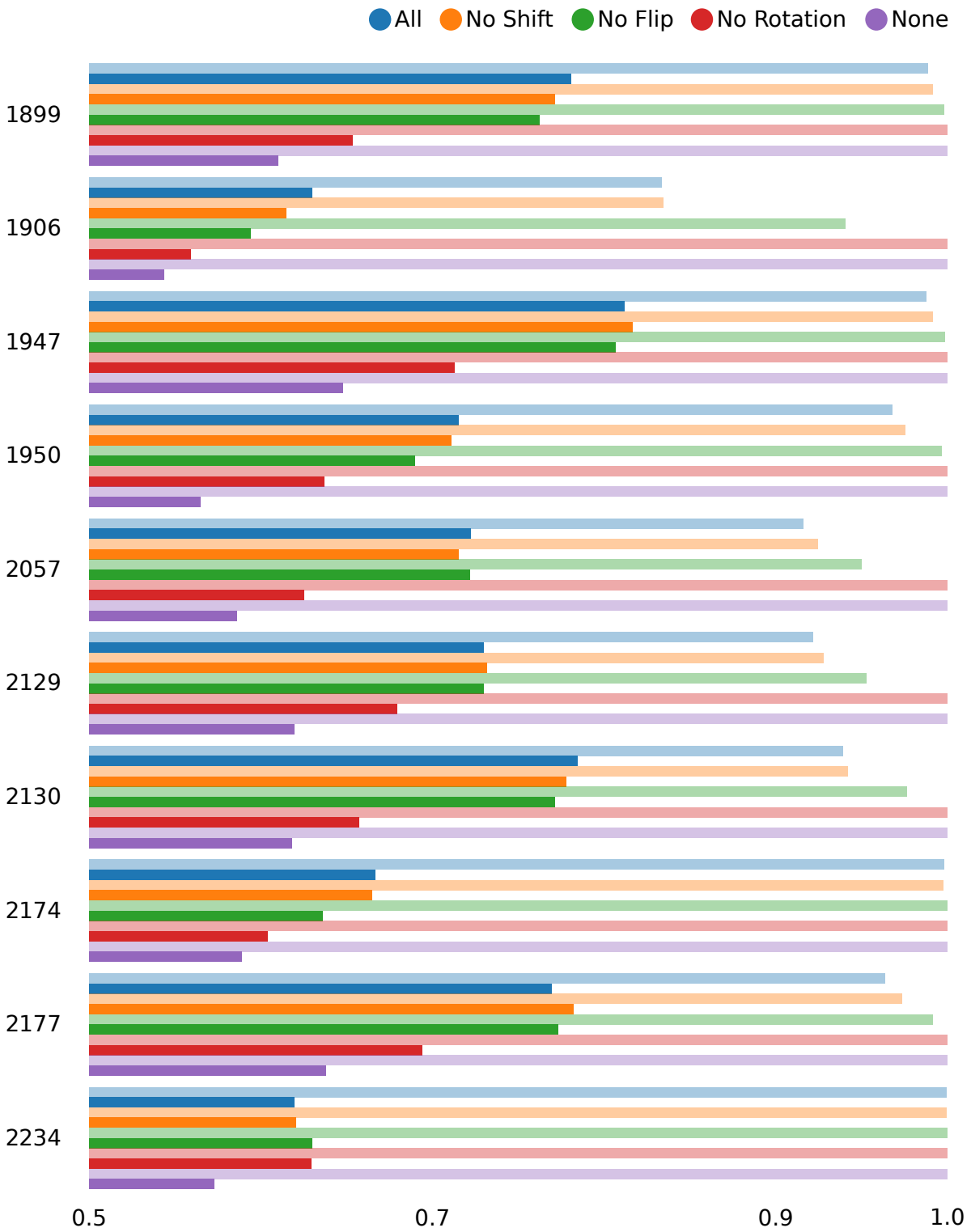


Abbildung 7.13: *Area under the ROC Curve* unter Verwendung der angegebenen Transformationen für Trainingsdaten (schwache Farben) und Testdaten (kräftige Farben).

Transformationen beurteilt werden kann, wurden Experimente durchgeführt, bei denen jeweils eine dieser Transformationen deaktiviert wurde. Sie wurden dann verglichen mit Experimenten, die keine oder alle Transformationen nutzten.

In den Ergebnissen (Abbildung 7.12 und 7.13) ist ein klarer Trend zu sehen, welche Transformationen den größten Nutzen bringen. So ist die Nutzung aller Transformationen meistens die beste Variante. Unter den einzelnen Transformationen scheint das Drehen deutlich am wichtigsten zu sein, während das Spiegeln und das Verschieben einen weniger deutlichen Unterschied bringen. Alle Transformationen wirken durch ihre Variabilität einer Überanpassung entgegen, was gut anhand der Trainingsdaten zu erkennen ist. Die perfekten Ergebnisse zeigen deutlich, dass die Trainingsdaten auswendig gelernt wurden. Durch die Transformationen kann die Überanpassung somit deutlich verringert werden.

7.7 Vorhersagequalität im Vergleich zu existierenden Methoden

Um die Vorhersagequalität zwischen gängigen *Fingerprints* (die durch Expertenwissen definiert wurden) und *Convolutional Neural Networks* (die ihre Merkmale automatisch lernen) zu vergleichen, erfolgte ein Experiment mit allen 141 PubChem-Datensätzen. Dabei wurden der binäre *Extended Connectivity Fingerprint* mit einem Radius von 2 (*ECFP4*), der zählende *Extended Connectivity Fingerprint* mit einem Radius von 0 (*ECFC0*) und der *MACCS Fingerprint* mit dem *Convolutional Neuronal Network* (*CNN*) verglichen. Für die *Fingerprints* wurde jeweils ein *Random Forest* mit 10 000 Bäumen und *Soft Voting* als Klassifikator genutzt. Beim *Convolutional Neural Network* diente das damit trainierte *Multi-Layer Perceptron* als Klassifikator.Abbildung 7.14 und 7.15 zeigen die Ergebnisse.

Des Weiteren wurde für die vier Methoden jeweils pro Datensatz ein Rang vergeben und diese Ränge wurden über alle Datensätze gezählt. Das Ergebnis ist in Tabelle 7.4 und 7.5 zu sehen. *ECFP4* sticht klar als die beste Methode heraus. *MACCS* nimmt ebenfalls deutlich die zweite Position ein. *CNN* und *ECFC0* teilen sich die hinteren Plätze. Allerdings ist *CNN* in Einzelfällen auch in den vorderen Plätzen zu finden und ist sogar häufiger die beste Methode als *MACCS*. Grundsätzlich lässt sich sagen, dass das neuronale Netzwerk bereits mit den *Fingerprints* mithalten kann, allerdings noch Verbesserungen benötigt, um konstanter die besten Ergebnisse liefern zu können.

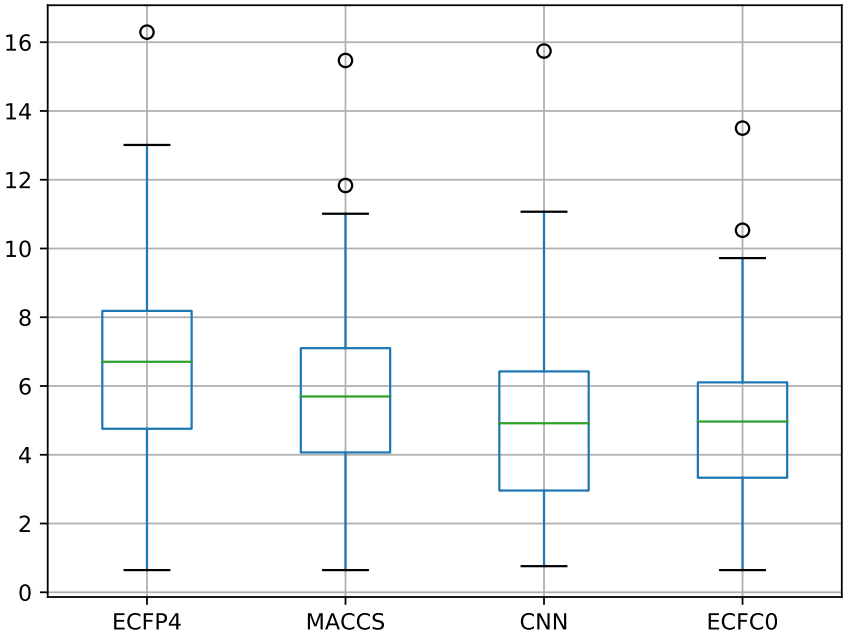


Abbildung 7.14: *Enrichment Factor* bei 5 % über 141 Datensätze.

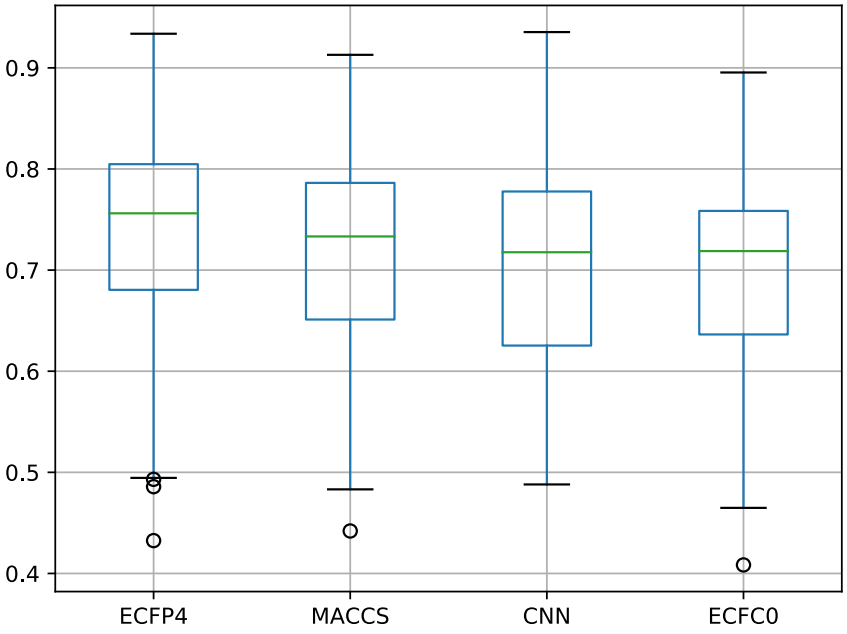


Abbildung 7.15: *Area under the ROC Curve* über 141 Datensätze.

Tabelle 7.4: Anzahl an erreichten Rängen der vier Methoden über 141 Datensätze, hier der *Enrichment Factor* bei 5 %.

Rang	ECFP4	MACCS	CNN	ECFC0
1	106	13	16	10
2	19	87	21	16
3	9	25	49	56
4	7	16	55	59

Tabelle 7.5: Anzahl an erreichten Rängen der vier Methoden über 141 Datensätze, hier die *Area under the ROC Curve*.

Rang	ECFP4	MACCS	CNN	ECFC0
1	88	22	21	10
2	35	68	28	10
3	13	30	48	50
4	5	21	44	71

7.8 Nutzen von gelernten Merkmalen mit einem anderen Klassifikator

Ein weiteres Experiment sollte zeigen, dass sich die durch das *Convolutional Neural Network* generierten Merkmale ebenso gut zur Verwendung mit anderen Methoden eignen. Hierzu wurden die Merkmale aus dem *Convolutional Neural Network* einmal mit dem damit trainierten *Multi-Layer Perceptron* und einmal mit einem *Random Forest* genutzt. Es wurde die Vorhersagequalität anhand aller 141 PubChem-Datensätze verglichen.

Wie in Abbildung 7.16 zu sehen ist, sind die Unterschiede recht gering. Der maximale Unterschied im *Enrichment Factor* bei 5 % beträgt 1,724 und bei der *Area under the ROC Curve* 0,054. Das ist nicht bedeutend höher als die Differenz, die bei Verwendung derselben Methode mit unterschiedlichen *Random-Seeds* erreicht wird. Dies deutet darauf hin, dass die generierten Merkmale sich auch gut in Kombination mit anderen Methoden nutzen lassen und nicht nur zusammen mit dem *Multi-Layer Perceptron*, das ja die Generierung der Merkmale mit beeinflusst hat.

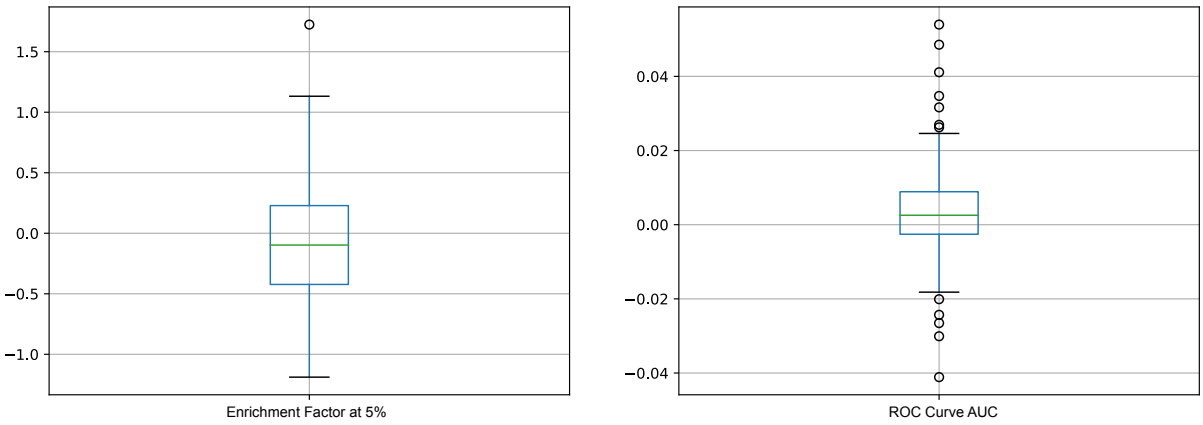


Abbildung 7.16: Differenz (*Random Forest - Multi-Layer Perceptron*) zwischen zwei Klassifikatoren unter Verwendung der Merkmale desselben *Convolutional Neural Network*.

7.9 Finden relevanter Substrukturen

Ein großer Vorteil der in dieser Arbeit vorgestellten Methode ist, dass man zusätzlich zu der Klassifikation auch Hinweise darauf bekommt, welche Substrukturen innerhalb eines Datensatzes mit der Aktivität zu tun haben. Um die Qualität dieser Hinweise zu evaluieren, wurde ein Experiment ausgeführt, das die in den *Saliency Maps* hervorgehobenen Substrukturen auswertet. Da die Substrukturen normalerweise bei echten Datensätzen nicht bekannt sind, wurden Datensätze generiert. Dazu wurden die Moleküle des PubChem-Datensatzes herangezogen. Alle Moleküle, die eine gewählte Substruktur enthalten, wurden als aktiv angesehen und alle anderen als inaktiv. Als Substrukturen dienten jeweils Benzol, Alanin und Acetamid (siehe Abbildung 7.17). Zur Auswertung wurde jeweils der Wert jedes einzelnen Atoms innerhalb der *Saliency Maps* genutzt. Sollte die *Saliency Map* wie erwartet die korrekten Substrukturen hervorheben, dann müssten die Werte für Atome, die Teil der Substruktur sind, hoch sein, während Werte für Atome, die nicht Teil der Substruktur sind, deutlich niedriger ausfallen.

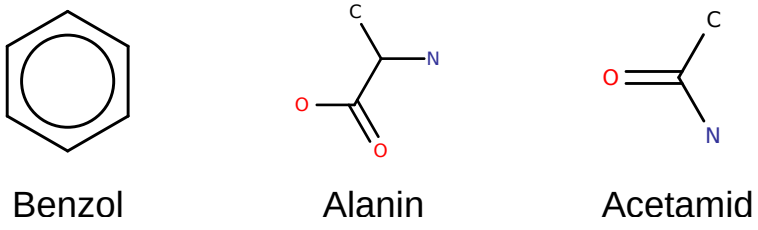


Abbildung 7.17: Zielstrukturen.

In den Ergebnissen (Abbildung 7.18, 7.19 und 7.20) ist zu sehen, dass diese Annahme sich für die getesteten Datensätze als wahr herausgestellt hat. Abgesehen von Ausreißern haben die Boxplots bei Benzol und Alanin keine Überschneidungen. Bei Acetamid liegen die Werte näher beieinander, es sind aber trotzdem deutlich höhere Werte bei der korrekten Substruktur zu finden.

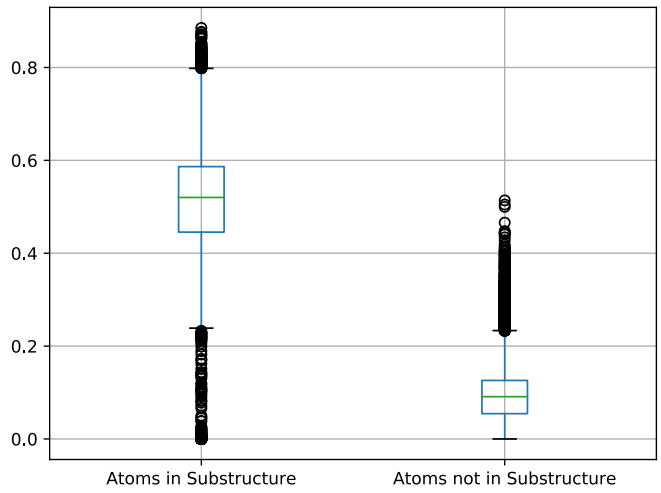


Abbildung 7.18: *Saliency-Map*-Werte der einzelnen Atome, unterteilt in Atome, die Teil, und Atome, die nicht Teil eines Benzols sind.

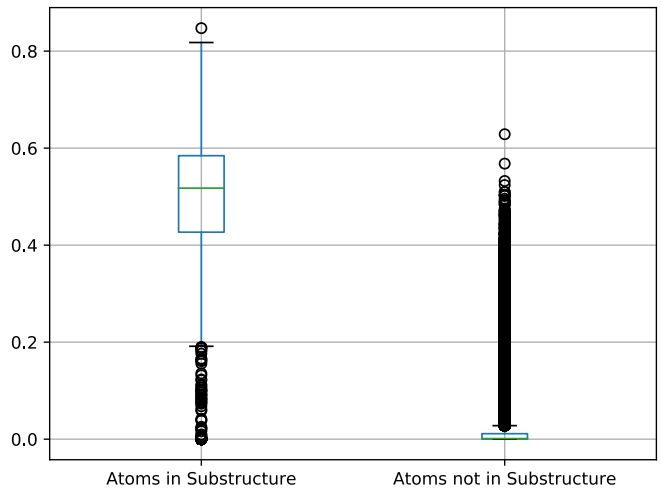


Abbildung 7.19: *Saliency-Map*-Werte der einzelnen Atome, unterteilt in Atome, die Teil, und Atome, die nicht Teil eines Alanins sind.

Um die Extraktion von Substrukturen und die damit verbundene Bewertungsberechnung zu überprüfen, wurden außerdem aus allen *Saliency Maps* die hervorgehobenen Substrukturen extrahiert. Die Top-n-Substrukturen sind in den Tabellen 7.6, 7.7 und

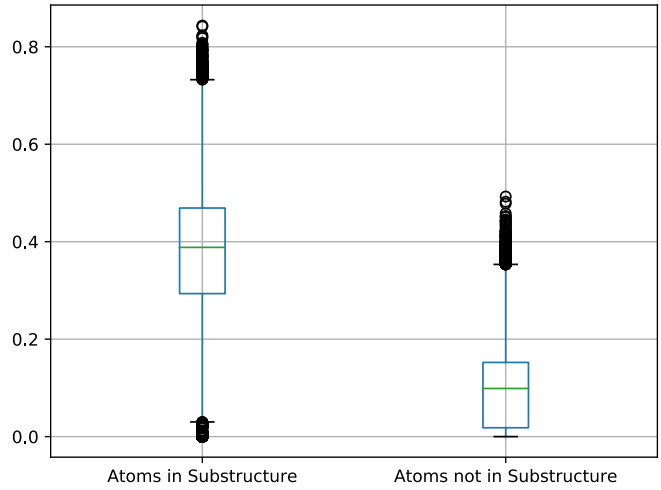


Abbildung 7.20: *Saliency-Map*-Werte der einzelnen Atome, unterteilt in Atome, die Teil, und Atome, die nicht Teil eines Acetamids sind.

Tabelle 7.6: Extrahierte Substrukturen für den Benzol-Datensatz. Angegeben ist der Rang auf der Basis der Bewertung s, die sich zusammensetzt aus dem durchschnittlichen Wert vs der *Saliency Maps*, der Häufigkeit f und der Anzahl der Atome n. Farblich hervorgehoben sind die korrekte Substruktur (grün) und verwandte Substrukturen (gelb), die von der korrekten um nicht mehr als 2 Atome abweichen.

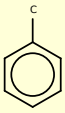
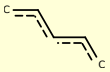
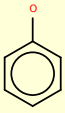
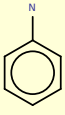
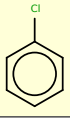
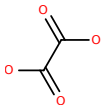
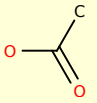
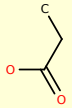
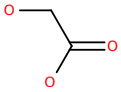
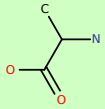
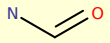
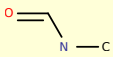
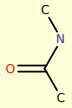
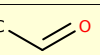
Rang	Substruktur	s	vs	f	n
1		1,0	0,537	102950	6
2		0,252	0,542	21249	7
3		0,136	0,468	23515	5
4		0,13	0,522	11815	7
5		0,109	0,528	9724	7
6		0,078	0,539	6652	7

Tabelle 7.7: Extrahierte Substrukturen für den Alanin-Datensatz. Angegeben ist der Rang auf der Basis der Bewertung s, die sich zusammensetzt aus dem durchschnittlichen Wert vs der *Saliency Maps*, der Häufigkeit f und der Anzahl der Atome n. Farblich hervorgehoben sind die korrekte Substruktur (grün) und verwandte Substrukturen (gelb), die von der korrekten um nicht mehr als 2 Atome abweichen.

Rang	Substruktur	s	vs	f	n
1		1,0	0,472	702	6
2		0,42	0,456	535	4
3		0,395	0,474	345	5
4		0,267	0,46	250	5
5		0,21	0,365	1535	2
...	...	...	...	...	...
22		0,106	0,536	58	6

7.8 zu sehen. Hierbei war die korrekte Substruktur bei allen drei Datensätzen in den obersten 0,25 % der extrahierten Substrukturen zu finden. Benzol landete auf Rang 1 von 7124, Alanin auf Rang 22 von 9423 und Acetamid auf Rang 8 von 9423 extrahierten Substrukturen. Dabei ist zu beachten, dass der durch die Bewertung vergebene Rang jeweils besser war als die Ränge, die durch einzelne Bestandteile der Bewertung vergeben wurden. Eine Substruktur, die in mehreren dieser Aspekte (durchschnittlicher Wert in der *Saliency Map*, Häufigkeit des Auftretens und Anzahl der Atome) einen guten Wert hat, ist also durchaus wichtiger. Es ist auch zu beachten, dass die meisten der hoch bewerteten Substrukturen oftmals ein Teil der korrekten Substruktur sind oder diese enthalten. Auch sie sind bereits ein wertvoller Hinweis auf den Auslöser der Aktivität.

Tabelle 7.8: Extrahierte Substrukturen für den Acetamid-Datensatz. Angegeben ist der Rang auf der Basis der Bewertung s, die sich zusammensetzt aus dem durchschnittlichen Wert vs der *Saliency Maps*, der Häufigkeit f und der Anzahl der Atome n. Farblich hervorgehoben sind die korrekte Substruktur (grün) und verwandte Substrukturen (gelb), die von der korrekten um nicht mehr als 2 Atome abweichen.

Rang	Substruktur	s	vs	f	n
1		1,0	0,484	23356	3
2		0,554	0,491	25126	2
3		0,552	0,482	8682	4
4		0,379	0,474	6192	4
5		0,212	0,476	2574	5
6		0,21	0,465	5371	3

7.10 Zusammenfassung

Es wurden Experimente mit 141 PubChem-Datensätzen durchgeführt. Dabei war zu sehen, dass der Einfluss des Zufalls sich in Grenzen hält, insbesondere wenn die gleichen Partitionen verwendet werden.

Für die betrachteten Datensätze hat sich gezeigt, dass sich wie zu erwarten der Mehrwert von mehr Trainingsdaten in zunehmendem Maße verkleinert. Die Schwierigkeit des Problems hat außerdem einen Einfluss auf die Anzahl benötigter Trainingsdaten. Betrachtet man die Entwicklung des Netzwerks über mehrere Epochen hinweg, so ist zu sehen, dass das Maximum der Vorhersagequalität oft bereits nach wenigen Epochen erreicht wird und danach ein langsamer Übergang in die Überanpassung stattfindet. Eine zu kleine Skalierung führt zu einem schlechten Ergebnis, da sie die klare Trennung zwischen Atomen beeinflusst. Da gleichzeitig die Größe der Eingabedaten und des Netzwerks mit einem größeren Skalierungsfaktor steigt, erhöht sich auch die Laufzeit. Daher ist eine möglichst kleine Skalierung anzustreben, die jedoch so groß sein muss, dass sie die Atome ausreichend voneinander trennt. Bei der Untersuchung der Nützlichkeit chemischer Eigenschaften stellte sich heraus, dass sich diese von Datensatz zu Datensatz teilweise stark unterscheiden kann. Deshalb ist zu empfehlen, alle zur Verfügung stehenden chemi-

schen Eigenschaften zu nutzen. Bei der Untersuchung der Transformationen zeigte sich, dass die Verwendung aller vorgeschlagenen Transformationen empfehlenswert ist. Wie zu erwarten, trägt insbesondere das Drehen zu besseren Ergebnissen bei.

Was die Vorhersagequalität betrifft, konnte im Vergleich mit gängigen *Fingerprints* das *Convolutional Neural Network* durchaus überzeugen, allerdings lieferte es im Durchschnitt schlechtere Ergebnisse als der Top-*Fingerprint*. Für mehr als ein Zehntel der Datensätze erbrachte das *Convolutional Neural Network* allerdings die besten Ergebnisse aller untersuchten Methoden. In einem weiteren Experiment war zu sehen, dass die Merkmale, die durch die *Convolution*-Schichten des Netzwerks gelernt wurden, nicht nur mit dem damit trainierten *Multi-Layer Perceptron* funktionieren, sondern unter Verwendung eines *Random Forest* zu ebenso guten Ergebnissen führen.

In einem Experiment mit generierten Daten wurde geprüft, wie gut sich *Saliency Maps* zum Finden von Substrukturen, die der Grund einer Aktivität sind, eignen. Hierbei waren die Werte der *Saliency Map* für Atome der Substruktur deutlich höher als für andere Atome. Auch erhielt die korrekte Substruktur unter Verwendung der vorgeschlagenen Bewertung jeweils einen der besten Ränge. Damit ist die Verwendung von *Saliency Maps* zum Finden relevanter Substrukturen plausibel.

Kapitel 8

Fazit

Das Ziel dieser Arbeit war es, eine *Virtual-High-Throughput-Screening*-Methode auf der Basis von *Deep-Learning*-Techniken der Bildanalyse zu entwickeln. Dies wurde umgesetzt, indem die Struktur der Moleküle in ein bildähnliches, rasterbasiertes Format umgewandelt wurden, das dann als Eingabe eines *Convolutional Neural Network* diente. Durch die Aufteilung des *Convolutional Neural Network* in die *Convolution*-Schichten zur Merkmalgenerierung und die *Dense*-Schichten zur Klassifizierung können die Merkmale auch losgelöst vom Rest des Netzes genutzt werden. Ein weiteres Ziel war es, einen Einblick in die durch das Netzwerk getroffenen Entscheidungen zu geben. Dies wird mit Hilfe der *Saliency Maps* ermöglicht. Die Substruktur-Extraktion erlaubt es, für einen Datensatz eine Liste an Substrukturen zu erstellen, die vom Netzwerk für die gegebene Klassifizierung als wichtig angesehen werden. Damit kann ein erster Anhaltspunkt für die biologischen Zusammenhänge zwischen der Aktivität und der Struktur gegeben werden.

8.1 Zusammenfassung

Das *Virtual High-Throughput Screening* ist eine Methode, um die biologische Aktivität von Molekülen mittels maschinellen Lernens abzuschätzen. Dabei werden Moleküle, deren Aktivität bekannt ist, als Trainingsdaten genutzt, um einen Klassifikator zu erstellen. Dieser berechnet die Wahrscheinlichkeit eines Moleküls, aktiv zu sein. Somit können aus einer Menge von Molekülen, deren Aktivität unbekannt ist, die mit der höchsten Wahrscheinlichkeit einer Aktivität für echte Tests im Labor ausgewählt werden. Als Eingabedaten dient üblicherweise die Struktur der einzelnen Moleküle. Um die Qualität der Ergebnisse abzuschätzen, gibt es Maße wie den *Enrichment Factor* und die *Area under*

the ROC Curve. Mit dem *Enrichment Factor* wird gemessen, wie viel Mehrwert das Modell im Vergleich zu einer Zufallsauswahl bietet, wenn ein bestimmter Prozentsatz der Top-n-Moleküle gewählt wird. Mit der *Area under the ROC Curve* wird hingegen die gesamte Vorhersage beurteilt.

Deep Learning ist die Verwendung von vielschichtigen neuronalen Netzen, um komplexe Aufgaben zu lösen. Insbesondere im Bereich des maschinellen Sehens wurden mit Hilfe von *Convolutional Neural Networks* große Erfolge erzielt. Hierbei dienen die *Convolution*-Schichten am Anfang des Netzwerks der Generierung von Merkmalen, während die *Dense*-Schichten am Ende die Aufgabe der Klassifikation auf der Basis dieser Merkmale übernehmen. Mit *Saliency Maps* besteht des Weiteren die Möglichkeit, den Einfluss der Eingabedaten auf die resultierende Klasse zu berechnen.

Um *Deep-Learning*-Methoden des maschinellen Sehens für die Verwendung als *Virtual-High-Throughput-Screening*-Methode zu adaptieren, ist insbesondere eine Repräsentation der molekularen Struktur in einem Format nötig, mit dem ein *Convolutional Neural Network* umgehen kann. Dafür wird die Struktur in ein Raster eingepasst, indem den einzelnen Atomen des Moleküls Positionen innerhalb des Rasters zugeordnet werden. An diesen Positionen wird jedes Atom durch sein Elementsymbol und seine chemischen Eigenschaften repräsentiert. Mit Hilfe von Transformationen der Trainingsdaten kann dem Netzwerk außerdem eine Invarianz gegenüber Drehung, Spiegelung und Verschiebung der enthaltenen Strukturen beigebracht werden. Da die verwendete Netzwerkarchitektur leicht in einen Teil, der die Generierung von Merkmalen, und einen Teil, der die Klassifikation zur Aufgabe hat, aufgeteilt werden kann, lässt sich das resultierende Netzwerk auch dazu nutzen, lediglich Merkmale zu generieren, die dann als Eingabedaten einer anderen Methode dienen.

Durch die Verwendung von *Saliency Maps* zur Berechnung der wichtigen Bereiche innerhalb der Eingabedaten ist eine Visualisierung möglich, die für ein spezifisches Molekül jene Substruktur hervorhebt, die laut dem Netzwerk für die Aktivität verantwortlich ist. Des Weiteren können automatisiert anhand der Nutzung eines Schwellenwerts relevante Substrukturen aus den *Saliency Maps* des gesamten Datensatzes ausgeschnitten werden. Durch die Verwendung einer Bewertung auf der Basis des durchschnittlichen *Saliency-Map*-Werts, der Häufigkeit des Auftretens und der Größe können diese Substrukturen sortiert und somit die relevantesten unter ihnen gefunden werden. Diese Informationen können einem Biologen als Anhaltspunkt dienen, welche Substruktur für die tatsächliche biologische Aktivität der Moleküle verantwortlich ist.

Die vorgestellte Softwarearchitektur erlaubt es, parallele Ressourcen möglichst effizient zu nutzen. So wird während des Trainings auf der Grafikkarte bereits die nächste *Mini-Batch* an Daten, verteilt auf mehrere Prozessorkerne, vorbereitet. Des Weiteren wird durch den modularen Aufbau ein einfacher Austausch spezifischer Implementierungen ermöglicht. Durch das Schreiben von Zwischenergebnissen ist ein einfaches Fortsetzen möglich, ohne Experimente gänzlich wieder von vorne beginnen zu müssen.

In den Experimenten wurde insbesondere darauf Wert gelegt, sinnvolle Hyperparameter für die genutzten Datensätze zu finden. Die chemischen Eigenschaften variierten in ihrem Nutzen von Datensatz zu Datensatz. Es hat sich gezeigt, dass die Transformationen ihren Zweck erfüllen und zu besseren Ergebnissen führen. Im Vergleich mit gängigen *Fingerprints* konnte die vorgestellte Methode durchaus bestehen, war allerdings noch nicht so gut wie der beste *Fingerprint*. Die durch das Netzwerk gelernten Merkmale konnten auch zum Lernen eines anderen Klassifikators genutzt werden, der ähnlich gute Ergebnisse lieferte. Des Weiteren war zu sehen, dass sich die *Saliency Maps* eignen, um Substrukturen zu finden, die für die Aktivität von Molekülen verantwortlich sind.

8.2 Ausblick

Der Fokus dieser Arbeit war darauf gerichtet, grundlegend zu erforschen, wie *Deep-Learning*-Methoden des maschinellen Sehens zur Ausführung von *Virtual High-Throughput Screening* genutzt werden können. Es konnte gezeigt werden, dass dies möglich und durchaus sinnvoll ist. Im Weiteren besteht nun das Potential, die Methode zum Beispiel durch Änderungen an der Netzwerkarchitektur zu verfeinern, um die Vorhersagequalität zu verbessern. Damit wären in Zukunft auch Ergebnisse möglich, die jene der gängigen Methoden wie etwa *Fingerprints* übertreffen. Aus dem maschinellen Sehen ist bereits bekannt, dass sich durch eine leichte Verfeinerung über die Jahre immer bessere Ergebnisse erreichen ließen.

Da es sich bei der Struktur eines Moleküls eigentlich um ein 3D-Objekt handelt, würde auch eine 3D-Repräsentation zur Erzeugung der Eingabedaten sinnvoll sein und möglicherweise zu besseren Ergebnissen führen. Dies war zum Zeitpunkt dieser Arbeit nicht möglich, da mit der zusätzlichen Dimension die Daten und damit auch das Netzwerk deutlich größer werden und dies zu Problemen bei Speicher und Rechenleistung führt. Sollte diese technische Limitierung in der Zukunft so nicht mehr bestehen, wäre ein Umstieg auf eine 3D-Repräsentation sinnvoll.

Die vorliegende Arbeit überprüfte die erstellte Methode im Hinblick auf die Wirkstoffmittelsuche. Es wäre interessant, sie auch auf verwandte Themengebiete, etwa die Toxikologie, anzuwenden und ihre Effektivität in diesem Bereich zu untersuchen.

Bisher wurde nur betrachtet, welche Substrukturen der Eingabe sich auf die Wahl der Klasse ausgewirkt haben. Aufschlussreich dürfte auch sein, die einzelnen vom Netzwerk generierten Merkmale zu analysieren, um ein besseres Verständnis zu erhalten, welche grundlegenden Eigenschaften das Netzwerk als Merkmal betrachtet.

Das für die automatisierte Substruktur-Extraktion genutzte Ausschneideverfahren könnte durch eine fortgeschrittenere Methode ersetzt werden, bei der kein globaler Schwellenwert angewandt wird. Auch die genutzte Bewertung ließe sich gegebenenfalls verfeinern. Bisher basiert sie auf einer intuitiven Idee, welche Eigenschaften eine besonders wichtige Substruktur ausmachen. Durch Anpassungen der Formel wären eventuell noch bessere Ergebnisse bei der Suche nach den richtigen Substrukturen möglich. Wünschenswert wäre zudem ein Zusammenfassen von Substrukturen, die sich sehr ähneln, da sie im Grunde von derselben Substruktur stammen und nur ein Atom mehr oder weniger durch das Schwellenwertverfahren abgeschnitten wird.

Die Anwendung des übertragenden Lernens (die Verwendung eines bereits für ein ähnliches Szenario trainierten Netzwerks als Ausgangspunkt) wäre ebenfalls interessant. Damit ließe sich potentiell die Anzahl benötigter Trainingsdaten für ein einzelnes *Screening* verringern und die Methode könnte auch in Fällen zur Anwendung kommen, in denen nicht viele Daten zur Verfügung stehen.

Literaturverzeichnis

[1] *ChEMBL*. https://www.ebi.ac.uk/chembl.

[2] *GitHub*. https://github.com/.

[3] *GPLv3*. http://www.gnu.org/licenses/gpl-3.0.

[4] *ImageNet Large Scale Visual Recognition Challenge*. http://image-net.org/challenges/LSVRC.

[5] *Keras*. https://keras.io.

[6] *NumPy*. https://www.numpy.org.

[7] *PubChem*. https://pubchem.ncbi.nlm.nih.gov.

[8] *Python*. https://www.python.org.

[9] *TensorFlow*. https://www.tensorflow.org.

[10] *TkInter - Python Wiki*. https://wiki.python.org/moin/TkInter.

[11] *SMARTS - A Language for Describing Molecular Patterns*, 2011. http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html.

[12] Apodaca, R., N. O'Boyle, A. Dalke, J. van Drie, P. Ertl, G. Hutchison, C. A. James, G. Landrum, C. Morley, E. Willighagen, H. D. Winter, T. Vandermeersch und J. May: *OpenSMILES specification*, 2016. http://www.opensmiles.org.

[13] Borgelt, C. und M. R. Berthold: *Mining molecular fragments: Finding relevant substructures of molecules*. In: *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, S. 51–58. IEEE, 2002.

[14] Bradley, A. P.: *The use of the area under the ROC curve in the evaluation of machine learning algorithms.* Pattern recognition, 30(7):1145–1159, 1997.

[15] Bray, T.: *The javascript object notation (json) data interchange format.* 2014.

[16] Breiman, L.: *Random forests.* Machine learning, 45(1):5–32, 2001.

[17] Breiman, L., J. H. Friedman, R. A. Olshen und C. J. Stone: *Classification and Regression Trees.* Wadsworth and Brooks, 1984.

[18] Bresenham, J. E.: *Algorithm for computer control of a digital plotter.* IBM Systems journal, 4(1):25–30, 1965.

[19] Durant, J. L., B. A. Leland, D. R. Henry und J. G. Nourse: *Reoptimization of MDL keys for use in drug discovery.* Journal of chemical information and computer sciences, 42(6):1273–1280, 2002.

[20] Fernández-Delgado, M., E. Cernadas, S. Barro und D. Amorim: *Do we need hundreds of classifiers to solve real world classification problems?* The Journal of Machine Learning Research, 15(1):3133–3181, 2014.

[21] Gaulton, A., L. J. Bellis, A. P. Bento, J. Chambers, M. Davies, A. Hersey, Y. Light, S. McGlinchey, D. Michalovich, B. Al-Lazikani *et al.*: *ChEMBL: a large-scale bioactivity database for drug discovery.* Nucleic acids research, 40(D1):D1100–D1107, 2011.

[22] Geppert, H., M. Vogt und J. Bajorath: *Current trends in ligand-based virtual screening: molecular representations, data mining methods, new application areas, and performance evaluation.* Journal of chemical information and modeling, 50(2):205–216, 2010.

[23] Gini, C.: *Variabilità e mutabilità.* Reprinted in Memorie di metodologica statistica (Ed. Pizetti E, Salvemini, T). Rome: Libreria Eredi Virgilio Veschi, 1912.

[24] Glorot, X., A. Bordes und Y. Bengio: *Deep sparse rectifier neural networks.* In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, S. 315–323, 2011.

[25] Graves, A., M. Liwicki, S. Fernández, R. Bertolami, H. Bunke und J. Schmidhuber: *A novel connectionist system for unconstrained handwriting recognition.* IEEE transactions on pattern analysis and machine intelligence, 31(5):855–868, 2009.

[26] Halgren, T. A., R. B. Murphy, R. A. Friesner, H. S. Beard, L. L. Frye, W. T. Pollard und J. L. Banks: *Glide: a new approach for rapid, accurate docking and scoring. 2. Enrichment factors in database screening.* Journal of medicinal chemistry, 47(7):1750–1759, 2004.

[27] Hamilton, S.: *Introduction to screening automation.* In: *High Throughput Screening*, S. 169–189. Springer, 2002.

[28] He, K., X. Zhang, S. Ren und J. Sun: *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.* In: *Proceedings of the IEEE international conference on computer vision*, S. 1026–1034, 2015.

[29] Hillisch, A. und R. Hilgenfeld: *Modern Methods of Drug Discovery*, Bd. 93. Birkhäuser, 2012, ISBN 376436081X.

[30] Hinton, G., N. Srivastava und K. Swersky: *Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.* 14, 2012.

[31] Inglese, J. und D. S. Auld: *High Throughput Screening (HTS) techniques: applications in chemical biology.* Wiley Encyclopedia of Chemical Biology, S. 1–15, 2007.

[32] Kim, S., P. A. Thiessen, E. E. Bolton, J. Chen, G. Fu, A. Gindulyte, L. Han, J. He, S. He, B. A. Shoemaker *et al.*: *PubChem substance and compound databases.* Nucleic acids research, 44(D1):D1202–D1213, 2015.

[33] Kingma, D. P. und J. Ba: *Adam: A method for stochastic optimization.* arXiv preprint arXiv:1412.6980, 2014.

[34] Krizhevsky, A., I. Sutskever und G. E. Hinton: *Imagenet classification with deep convolutional neural networks.* In: *Advances in neural information processing systems*, S. 1097–1105, 2012.

[35] Landrum, G. A. *et al.*: *RDKit: Open-source cheminformatics.* `https://www.rdkit.org/`, 2006.

[36] LeCun, Y., Y. Bengio und G. Hinton: *Deep learning.* Nature, 521(7553):436, 2015.

[37] LeCun, Y., B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard und L. D. Jackel: *Handwritten digit recognition with a back-propagation network.* In: *Advances in neural information processing systems*, S. 396–404, 1990.

[38] Polyak, B. T.: *Some methods of speeding up the convergence of iteration methods.* USSR Computational Mathematics and Mathematical Physics, 4(5):1–17, 1964.

[39] Quinlan, J. R.: *Induction of decision trees.* Machine learning, 1(1):81–106, 1986.

[40] Riniker, S., Y. Wang, J. L. Jenkins und G. A. Landrum: *Using Information from Historical High-Throughput Screens to Predict Active Compounds.* Journal of Chemical Information and Modeling, 54(7):1880–1891, June 2014.

[41] Rogers, D. und M. Hahn: *Extended-connectivity fingerprints.* Journal of chemical information and modeling, 50(5):742–754, April 2010.

[42] Roy, K., S. Kar und R. N. Das: *A primer on QSAR/QSPR modeling: fundamental concepts.* Springer, 2015.

[43] Rumelhart, D. E., G. E. Hinton und R. J. Williams: *Learning representations by back-propagating errors.* nature, 323(6088):533, 1986.

[44] Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg und L. Fei-Fei: *ImageNet Large Scale Visual Recognition Challenge.* International Journal of Computer Vision (IJCV), 115(3):211–252, 2015.

[45] Scherer, D., A. Müller und S. Behnke: *Evaluation of pooling operations in convolutional architectures for object recognition.* In: *International conference on artificial neural networks*, S. 92–101, 2010.

[46] Shariff, A., J. Kangas, L. P. Coelho, S. Quinn und R. F. Murphy: *Automated image analysis for high-content screening and analysis.* Journal of biomolecular screening, 15(7):726–734, 2010.

[47] Simonyan, K., A. Vedaldi und A. Zisserman: *Deep inside convolutional networks: Visualising image classification models and saliency maps.* arXiv preprint arXiv:1312.6034, 2013.

[48] Simonyan, K. und A. Zisserman: *Very Deep Convolutional Networks for Large-Scale Image Recognition.* In: *International Conference on Learning Representations*, 2015.

[49] Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever und R. Salakhutdinov: *Dropout: a simple way to prevent neural networks from overfitting.* The Journal of Machine Learning Research, 15(1):1929–1958, 2014.

[50] Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke und A. Rabinovich: *Going deeper with convolutions.* In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, S. 1–9, 2015.

[51] Tieleman, T. und G. Hinton: *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.* COURSERA: Neural networks for machine learning, 4(2):26–31, 2012.

[52] Todeschini, R. und V. Consonni: *Handbook of molecular descriptors*, Bd. 11. John Wiley & Sons, 2008.

[53] Weininger, D.: *SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules.* Journal of chemical information and computer sciences, 28(1):31–36, 1988.

[54] Winter, P.: *mol-struct-nets.* `https://github.com/patrick-winter-kn/mol-struct-nets`.

[55] Winter, P., C. Borgelt und M. R. Berthold: *Learned Feature Generation for Molecules.* In: *International Symposium on Intelligent Data Analysis*, 2018.

[56] Zhu, T., S. Cao, P. C. Su, R. Patel, D. Shah, H. B. Chokshi, R. Szukala, M. E. Johnson und K. E. Hevener: *Hit identification and optimization in virtual screening: practical recommendations based on a critical literature analysis: miniperspective.* Journal of medicinal chemistry, 56(17):6560–6572, 2013.

[57] Zoete, V., A. Grosdidier und O. Michielin: *Docking, virtual high throughput screening and in silico fragment-based drug design.* Journal of cellular and molecular medicine, 13(2):238–248, 2009.