FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Entwicklung und Untersuchung neuronaler Netze zur Positionsschätzung der Rosettenzentren von Pflanzen

Bachelorarbeit

Max Riedel
MatrNr.: 4011732

Jülich, den 30. August 2017

Fachhochschule Aachen
Campus Jülich
Fachbereich: Medizintechnik und Technomathematik
Studiengang: Scientific Programming

JÜLICH
FORSCHUNGSZENTRUM

Versicherung

Diese Arbeit ist von mir selbständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

(Max Riedel) .....

Unterschrift

Diese Arbeit wurde betreut von

1. **Prüfer:** Prof. Dr. rer. nat. Bodo Kraft
2. **Prüfer:** Dr. habil. Hanno Scharr

Corrigendum

- Gleichung (4.3) auf Seite 26 im Kapitel **4.3 Eigenschaften der Fouriertransformation** muss richtigerweise lauten

$$\mathcal{F}\big[af(t) + bg(t)\big] = a\hat{f}(k) + b\hat{g}(k) \tag{4.8}$$

- Gleichung (6.1) auf Seite 31 im Kapitel **6.3 Vorbereitung der Daten** muss richtigerweise lauten

$$\begin{aligned} \mathrm{xmax} &= \max(x') \\ x &= \frac{x'}{\mathrm{xmax}} \\ \mathrm{ymax} &= \max(y') \\ y &= \frac{y'}{\mathrm{ymax}} \end{aligned} \tag{6.1}$$

Jülich, den 6. November 2017
Max Riedel

Inhaltsverzeichnis

1 Einleitung

Künstliche neuronale Netze sind in den vergangenen Jahren zur *state of the art* Methodik für Klassifizierungsaufgaben avanciert. Vor allem im Bereich der Mustererkennung konnten sie bahnbrechende Erfolge verzeichnen und somit altgediente Verfahren der Bildverarbeitung verdrängen bzw. ersetzen. Auf Bildverarbeitung setzt ebenfalls der Forschungsbereich der Phänotypisierung, in dem die quantitative Erfassung pflanzlicher Merkmale vor allem durch nicht invasive Methoden geschehen soll. Hier stellt sich die Frage, inwiefern künstliche neuronale Netze alternative Lösungsmethoden für Probleme der Phänotypisierung bereitstellen können.

Diese Arbeit untersucht die Anwendbarkeit künstlicher neuronaler Netze auf die Problematik der Positionsschätzung von Merkmalen am Beispiel der Rosettenzentren von Pflanzen. Diese Position ist eine wichtige Information für viele weiterverarbeitende Algorithmen und ihre Bestimmung wird somit oft benötigt. Objektlokalisierung mittels neuronaler Netze wird bisher nur über Umwege gemacht. Ein typischer Ansatz ist die Klassifizierung von Bildausschnitten (z.B. *sliding window*) mittels eines CNN. Der Ausschnitt mit der höchsten Klassifizierungssicherheit definiert dann die Position des Objektes im Bild [Gir+13]. In dieser Arbeit wird die Frage gestellt, ob künstliche neuronale Netze geeignet sind, eine solche Regressionsaufgabe ohne Umwege zu lösen und falls ja, was beachtet werden muss, um ein anwendbares Netz zu erhalten.

Zu Beginn der Arbeit wird präzisiert, was das Schätzen von Pflanzenzentren bedeutet und welche Relevanz es hat. Anschließend wird in einer theoretischen Einführung die Thematik neuronaler Netze erläutert. Hier werden alle für diese Arbeit verwendeten Aspekte erklärt. Ebenfalls wird die Fouriertransformation beschrieben, welche als analytisches Werkzeug von elementarer Bedeutung für die systematischen Untersuchungen im Zuge dieser Arbeit ist.

Das Problem der Positionsschätzung wird in zwei Teilprobleme unterteilt. Zunächst muss das Rosettenzentrum lokalisiert werden, was durch Faltungsnetze geschehen soll. Diese sollen die Positionsinformation in Form einer Wahrscheinlichkeitsdichtefunktion (PDF) oder Heatmap angeben. Bei der Entwicklung dieser Netze fällt auf, dass schnell komplexe Systeme entstehen, deren Training aufwändig ist und viele Trainingsdaten verlangt. Aus diesem Grund wird versucht bereits gelernte Filter eines anderen Netzes zu verwenden, was ein Ansatz des Transfer Learnings ist.

Das zweite Teilproblem ist die Berechnung der Koordinaten aus der PDF. Unter der Voraussetzung, dass die Lokalisierung erfolgreich war und die Information über die Position des gesuchten Merkmals in Form einer PDF vorliegt, muss eine Umrechnung dieser Position in Bildkoordinaten erfolgen. Hier wird ein Ansatz verfolgt, der eine einzelne Schicht benutzt. Für diesen werden Vorversuche gemacht, die anhand der Fouriertransformation analysiert werden.

Schlussendlich werden die Netze für die Teilprobleme zu einem einzelnen Netz kombiniert. Die Schätzungen dieses Netzes zeigen, dass die Position durch die Verwendung von neuronalen Netzen bestimmt werden kann. Allerdings muss der Programmierer das System speziell für diese Anwendung aufsetzen und teilweise in den Lernprozess eingreifen.

2 Schätzen von Pflanzenzentren

In diesem Kapitel wird auf die Thematik der Positionsschätzung von Rosettenzentren von Pflanzen eingegangen. Neben einer Eingrenzung des Themas wird die Problematik genauer erläutert und beschrieben, wo das Problem auftritt und wieso es wichtig zu lösen ist.

2.1 Schätzung

Umgangssprachlich wird der Begriff *Schätzung* oft verwendet, um zu erklären, dass die Bestimmung einer Messgröße auf Intuition oder Erfahrung beruht und somit nicht verlässlich ist. In der Statistik hingegen erfolgt eine Schätzung nach einer vorgegebenen mathematischen Berechnungsvorschrift. Das Ziel ist eine Approximation der real vorliegenden Werte zu erhalten (vgl. [Onlh]).

Schätzungen werden oft durch Stichproben generiert. Dafür wird eine überschaubare Anzahl von Beispielen aus einer Gesamtheit entnommen. Die darin auftauchende Verteilung wird dann auf die Menge der Gesamtheit hochgerechnet. Das Ergebnis ist eine fehlerbehaftete Approximation an die wirklich vorliegende Verteilung.

2.2 Pflanzenzentren

Der Begriff Pflanzenzentrum beschreibt den Punkt, von dem aus die Rosettenpflanze ihre Blätter austreibt. In Abbildung 2.1 auf der nächsten Seite ist das Zentrum mit einem orangen Kreuz markiert. Doch wieso ist die automatische Bestimmung dieser Position relevant?

Das Institut für Bio- und Geowissenschaften – Pflanzenwissenschaften, kurz IBG-2, des Forschungszentrums Jülich beschäftigt sich mit der Phänotypisierung von Pflanzen. In der Biologie ist die Phänotypisierung ein eigenständiger Forschungsbereich, der die Vermessung und Quantifizierung von Pflanzeneigenschaften zum Ziel hat. Der Phänotyp einer Pflanze ist die Gesamtheit aller Merkmale und entsteht aus der Kombination des Genotyps der Pflanze und der Umweltfaktoren, der sie ausgesetzt ist. Anhand des Phänotyps lassen sich Vorhersagen zu wichtigen Eigenschaften der Pflanze, wie zum Beispiel die zu erreichende Biomasse, treffen. Bei der Messung dieser Eigenschaften sind nicht-invasive Methoden invasiven Methoden vorzuziehen, da bei Beschädigung der Pflanze das weitere Wachstum mindestens gestört, wenn nicht sogar beendet wird.

Nicht invasive Methoden sind zumeist bildgebende Verfahren. Diese können dabei unterschiedlicher Art sein. Das IBG-2 verwendet RGB- und Infrarot-Bilder aus handelsüblichen Spiegelreflex- oder Industriekameras, aber auch speziellere Methoden, wie

Abbildung 2.1: Arabidopsispflanze mit markiertem Rosettenzentrum

die flugzeuggestützte Aufnahme von Hyperspektralbildern aus der Luft oder dem Erstellen von 3D Wurzelmodellen mittels Magnet Resonanz Tomographie.

Mittels geeigneter Algorithmen lassen sich aus diesen Bildern Informationen aus den fotografierten Pflanzen ermitteln. Doch diese Algorithmen der Bildverarbeitung brauchen oftmals einen geeigneten Startpunkt, um zum Beispiel die Pflanze zu markieren, falls sich die Blätter verschiedener Pflanzen überdecken, oder um einen Startpunkt für die Bestimmung von Blattlängen zu erhalten. Dafür bietet sich das Rosettenzentrum an. In [GMT15], zum Beispiel, wird das Rosettenzentrum für die Transformation des Bildes von kartesischen in logarithmische Polarkoordinaten gebraucht. Dort wird das Rosettenzentrum über einen Algorithmus bestimmt, der eine Skelettierung der Pflanze voraussetzt. In [Aks+15] wird auf einer Vordergrundsegmentierung der Pflanze der Massenschwerpunkt bestimmt.

3 Künstliche Neuronale Netze

Künstliche neuronale Netze sind ein Teilgebiet des Supervised Learning und mittlerweile in vielen Bereichen im Einsatz. Sie übernehmen die Spracherkennung in Assistenzsystemen wie Amazons Alexa [Onlb] oder die Gesichtserkennung in sozialen Netzwerken wie Facebook. Aber vor allem in der Bildverarbeitung sind die Faltungsnetze in den letzten Jahren immer mehr in den Mittelpunkt gerückt. Dieses Kapitel wird in die Thematik künstlicher neuronaler Netze einführen und die wichtigsten Konzepte von Faltungsnetzen erläutern. Dieses Kapitel folgt in Grundzügen der im Dezember 2016 vom Autor eingereichten Seminararbeit mit dem Thema *Deep Convolutional Neural Networks*. Die Inhalte wurden jedoch auf die Thematik dieser Arbeit angepasst.

3.1 Maschinelles Lernen

Maschinelles Lernen ist ein Forschungsgebiet der Computerwissenschaften, in dem Systeme selbständig die Erfüllung einer Aufgabe lernen, ohne explizit darauf programmiert zu sein. Das verlangt das selbständige Anpassen der Parameter durch das System. Der Stand der Technik ist dabei, dass für verschiedene Problemstellungen bzw. Anforderungen unterschiedliche Systemarchitekturen verwendet werden müssen. Diese müssen vom Programmierer vorgegeben werden, was dazu führt, dass die Systeme hochspezialisiert sind, was sie im Allgemeinen nicht auf andere Probleme anwendbar macht. So wird beispielsweise ein Netz, das für die Erkennung von menschlichen Gesichtern in Bildern entworfen wurde (zum Beispiel FaceNet [SKP15]) nicht unverändert Segmentierungsaufgaben (wie [SLD16]) erlernen können und ebenso nicht für Text- oder Spracherkennung (wie [KH15]) geeignet sein. Eine Universalarchitektur, die rein datengetrieben Problemstellungen erlernt und löst, gibt es aktuell nicht.

3.1.1 Aufgabenstellung

Tabelle 3.1 auf der nächsten Seite zeigt eine Auflistung typischer Machine Learning Aufgaben. Maschinelles Lernen kommt überall da zum Einsatz, wo die explizite Programmierung eines Systems auf die Lösung eines Problems nur schwer möglich ist, da das Problem zu komplex ist. Das ist in der Bildverarbeitung besonders häufig der Fall, da die einzelnen Pixelwerte nur schwer in einen größeren Zusammenhang zu bringen sind. So fällt das hier betrachtete Problem, die Rosettenzentren auf Pflanzenbildern zu bestimmen, zum Beispiel in die Kategorie *Klassifikation*, da jedes Pixel einer der beiden Klassen zugeordnet werden kann, die darstellen, ob es sich um ein Zentrum handelt, oder nicht. Ebenfalls könnte es der Kategorie *Regression* zugeordnet werden,

wenn das gelernte System anhand eines Bildes ableitet, an welcher Pixelposition sich das Rosettenzentrum befindet. Dieser Ansatz wird in dieser Arbeit verfolgt.

Problem	Beschreibung
Klassifikation	Zuordnen einer Eingabe zur richtigen Klasse
Clustering / Aggregation	Einteilung der Eingaben in Gruppen
Charakterisierung	Gemeinsamkeiten der Eingaben finden
Regression	Modellierung von Beziehungen zwischen Eingaben

Tabelle 3.1: Aufgabenbereiche des maschinellen Lernens

3.1.2 Lernklassen

Für die Spezifizierung des Trainings wird beim maschinellen Lernen zwischen verschiedenen Lernklassen unterschieden. Der in dieser Arbeit verfolgte Ansatz fällt unter das *überwachte Lernen* bzw. *Supervised Learning*. Hierbei bestehen die Trainingsdaten aus Ein- und Ausgabepaaren. Die Eingabe wird vom System verarbeitet, welches dann anhand des errechneten Ergebnisses und der gewünschten Ausgabe die Parameter anpassen kann. Beim *unüberwachten Lernen* bzw. *Unsupervised Learning* gibt es diese gewünschte Ausgabe nicht. Das System muss also nur anhand der Eingabedaten selbst lernen. Diese Art des Lernens eignet sich nicht für Klassifikation oder Regression, kann aber für Clustering genutzt werden.

3.1.3 Anpassung der Parameter

Wie bereits erwähnt, lernt das System durch Anpassung seiner Parameter. Dabei stellt das System eine Funktion $f(\vec{e}) = \vec{a}$, mit dem Eingabevektor $\vec{e}$ und der Ausgabe $\vec{a}$, dar. Die Parameter von f werden zu Beginn der Trainingsphase zufällig initialisiert. Nun wird in jeder Trainingsiteration ein Eingabevektor $\vec{e}$ verarbeitet und somit ein Ausgabevektor $\vec{a}$ erzeugt. Beim Supervised Learning gehört zu $\vec{e}$ auch noch der gewünschte Ausgabevektor $\vec{a}'$. Eine *Kosten-Funktion* bzw. *Loss Function* berechnet dann, inwiefern $\vec{a}$ und $\vec{a}'$ voneinander abweichen. Anhand dieses Fehlers werden die Parameter des Systems angepasst und mit dem nächsten Eingabedatum fortgefahren. Es entsteht der in Abbildung 3.1 auf der nächsten Seite dargestellte Kreislauf. Die Trainingsphase kann nach unterschiedlichen Kriterien beendet werden. Üblicherweise endet die Trainingsphase, wenn durch die vorhandenen Trainingsdaten keine Verbesserung des Systems mehr erreicht werden kann. Danach kann das System in die Nutzphase übergehen.

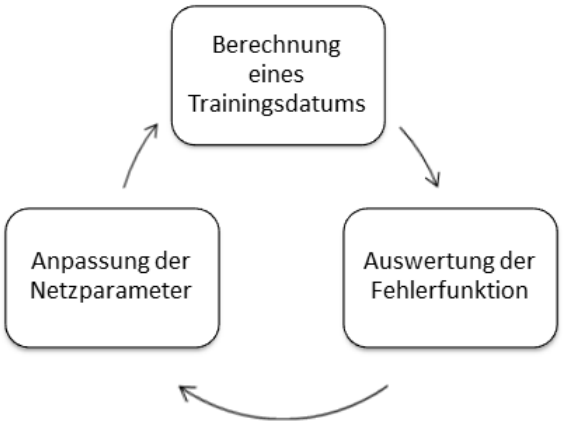

Abbildung 3.1: Iteration während der Trainingsphase

3.1.4 Overfitting

Ein immer wieder auftretendes Problem bei maschinellem Lernen ist das *Overfitting*. Overfitting ist die Überanpassung eines Systems an die Trainingsdaten, die im Allgemeinen dazu führt, dass das System ungesehene neue Daten schlechter verarbeiten kann. Dies ist an einem einfachen Beispiel gut zu zeigen. Angenommen ein Modell folgt einer quadratischen Modellvorschrift und die zur Verfügung stehenden Daten unterliegen einem leichten Rauschen. Trainiert man nun ein System auf diese Daten, das ein Polynom von Grad 5 erlernt, erhält man das in Abbildung 3.2 auf der nächsten Seite abgebildete Verhalten. Die orangen Punkte sind die Trainingsdaten. Die grüne Kurve entspricht der abzubildenden Modellvorschrift $f(x) = x^2$. Die blau gestrichelte Kurve zeigt ein Polynom $p(x)$ mit Grad 5. p interpoliert die Testdaten perfekt. Der Abweichungsfehler des Trainings liegt also bei 0. Doch vor allem in den Bereichen $x < 1$ und $x > 5$ entspricht p nur ungenügend der Modellvorschrift, was alle Vorhersagen des Systems unbrauchbar macht. Das System ist also *überangepasst*. Grund dafür ist, dass die angesetzte Funktion zu komplex für das Problem ist. Ein Polynom von Grad 2 zu verwenden, hätte das bessere Ergebnis geliefert. Das System enthielt also zu viele freie Parameter, die nicht nur für ein schlechteres Ergebnis sorgen, sondern auch Speicherplatz und Rechenleistung verschwenden. Außerdem können diese nicht benötigten Freiheitsgrade Rauschanteile der Daten abbilden, was es zu verhindern gilt. Die Reduzierung dieser Parameter auf ein angebrachtes Maß, ist also eine der Kernaufgaben beim maschinellen Lernen.

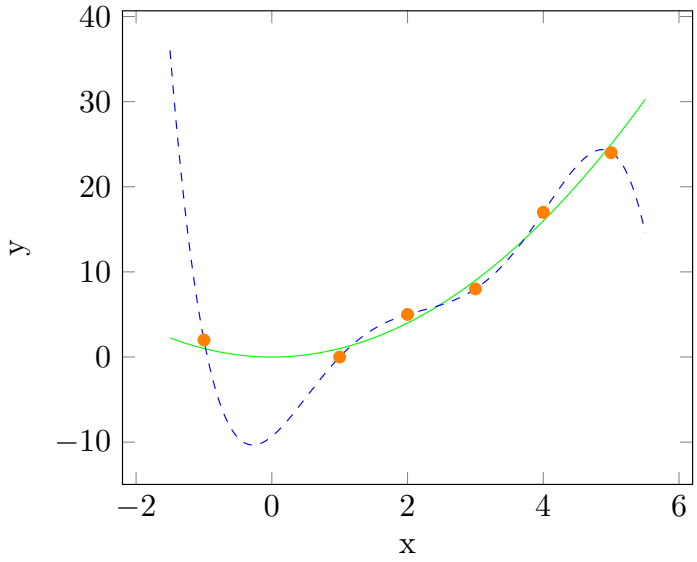

Abbildung 3.2: Regressionskurve und Polynominterpolation

3.2 Idee und Motivation

Künstliche neuronale Netze sind vom biologischen Vorbild des menschlichen Nervensystems inspiriert. Dieses wurde 1943 von Warren McCulloch und Walter Pitts erstmals mathematisch analysiert. In [MP43] beschrieben sie, dass die Neuronen in einer Netzstruktur das Nervensystem bilden. Jedes Neuron erhält von vorgeschalteten Neuronen bestimmte Erregungen. Überschreiten die Summe dieser Erregungen einen bestimmten Grenzwert, gibt das Neuron selbst eine Erregung weiter. Da sich dieses einfache Modell eines biologischen Neurons leicht auf künstliche Neuronen anwenden lies, ebneten sie damit den Weg für künstliche neuronale Netze.

3.3 Künstliche Neuronen

Neuronale Netze lassen sich in Computern also als Netz aus künstlichen Neuronen realisieren. Im ursprünglichen Modell stellen diese eine Funktion $f : \mathbb{R}^n \to \{0, 1\}$ dar. Ein reeller Vektor $\vec{x}$ wird auf eine binäre Ausgabe abgebildet, die der Aktivierung des Neurons entspricht. Erreicht die Summe der Elemente von $\vec{x}$ einen Schwellwert θ, wird eine 1 ausgegeben, ansonsten eine 0. Die Ausgabe eines Neurons mit Schwellwert θ und Eingabevektor $\vec{x}$ sieht also folgendermaßen aus (vgl. [Roj93, S. 31]):

$$f(x) = \begin{cases} 0, & \sum x_i < \theta \\ 1, & \sum x_i \geq \theta \end{cases} \tag{3.1}$$

Mit diesem Neuronenmodell lassen sich bereits einfache logische Funktionen realisieren. Abbildung 3.3 auf der nächsten Seite zeigt dies an den Beispielen AND und OR. Bei

der AND Funktion müssen die Aktivierungen den Schwellwert $\theta = 2$ überschreiten, damit das Neuron aktiviert wird. Es wird also nur eine 1 ausgeben, wenn $x_1 = x_2 = 1$ gilt. Bei der OR Funktion hingegen liegt der Schwellwert bei $\theta = 1$, was dafür sorgt, dass es für eine Aktivierung reicht, wenn eine der beiden Eingaben 1 ist.

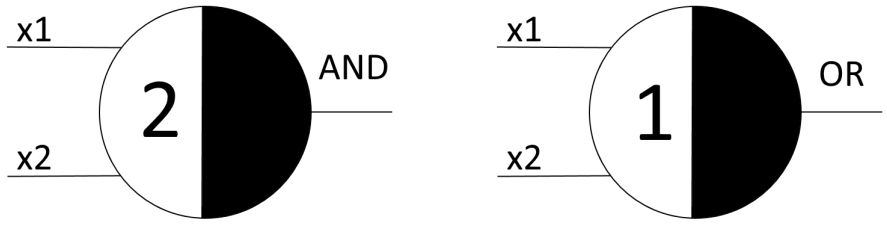

Abbildung 3.3: Künstliche Neuronen für die logischen Funktionen AND und OR

Diese Art von künstlichen Neuronen, die auch McCulloch-Pitts-Zellen genannt werden, bringen einen entscheidenden Nachteil mit sich. Die Kanten genannten Verbindungen zwischen den Neuronen kommen ausschließlich ungewichtet vor. Das heißt, dass Aktivierungen nur 1 oder 0 sein können. Das Lernen in solchen Netzen verlangt allerdings die Veränderung der Netztopologie, was komplexe Algorithmen voraussetzt und schwer zu automatisieren ist [Roj93, S. 51]. Ein Modell mit gewichteten Kanten umgeht diese Schwierigkeit. Um den Schwellwert θ ebenso durch den Lernalgorithmus anpassbar zu machen, geht er nun als Kante mit der standardmäßigen Aktivierung 1 und dem Gewicht $-\theta$ in das Neuron ein. Um eine Aktivierung weiter zu geben, muss das Neuron nun nicht mehr den Schwellwert überschreiten, sondern es gilt mit w_i als Kantengewicht

$$f(x) = \begin{cases} 0, & (\sum w_i x_i) - \theta < 0 \\ 1, & (\sum w_i x_i) - \theta \geq 0 \end{cases} \tag{3.2}$$

Somit lässt sich das in Abbildung 3.3 dargestellte AND Neuron zum Beispiel mit den Werten $w_1 = w_2 = 0,5$ und $\theta = 1$ darstellen. Folgende Formel zeigt die Äquivalenz (vgl. [Roj93, S. 41]):

$$\begin{aligned} 0,5x_1 + 0,5x_2 - 1 &\geq 0 \\ \Leftrightarrow x_1 + x_2 &\geq 2 \end{aligned} \tag{3.3}$$

3.4 Backpropagation

Um das System *lernen* zu lassen, müssen die Parameter an die Trainingsdaten angepasst werden. Dafür wird die Fehlerfunktion betrachtet. Diese berechnet nach jedem Eingabedatum die Abweichung zum gewünschten Ergebnis. In dieser Arbeit wird *Mean*

Squared Error als Fehlerfunktion verwendet. Diese Fehlerfunktion soll nun minimiert werden, wofür *Backpropagation* benutzt wird.

Backpropagation ist ein Gradientenabstiegsverfahren. Der Fehlerraum soll entlang des Gradienten, also des steilsten Abstiegs, abgegangen werden. Um das zu ermöglichen, muss von dem obigen Neuronenmodell Abstand genommen werden, da die Aktivierung dieser Neuronen eine Treppenfunktion ist. Die Ableitung der Aktivierung ist an allen differenzierbaren Stellen 0, was einen Gradientenabstieg verhindert. Somit muss zu einer stetigen Aktivierungsfunktion übergegangen werden, deren Ableitung an keiner Stelle verschwindet. Ebenso muss die Funktion die Treppenfunktion aus Abbildung 3.4 approximieren, um das gewünschte Verhalten des Neurons beizubehalten.

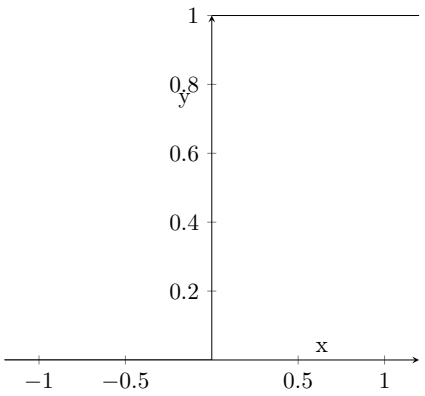

Abbildung 3.4: Treppenfunktion

Abbildung 3.5 zeigt zwei häufig verwendete Aktivierungen. Sowohl die Sigmoide $s_c(x) = \frac{1}{1+e^{-cx}}$, als auch die hyperbolische Tangente $\tanh(x)$ erfüllen die Kriterien.

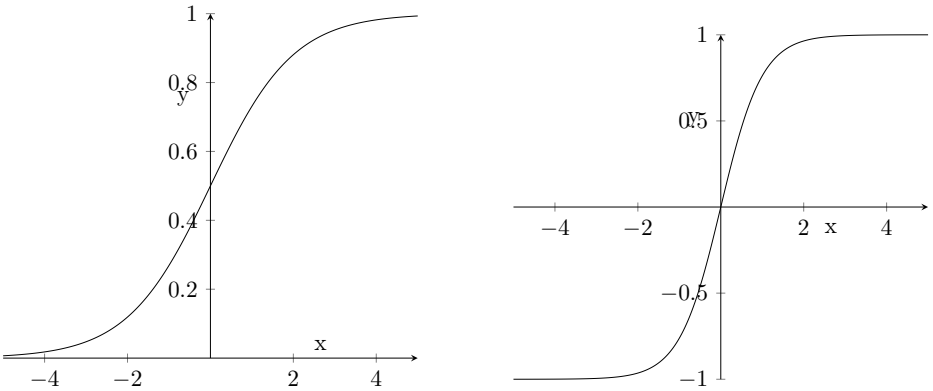

Abbildung 3.5: $s_c(x)$ mit $c = 1$ (links), $\tanh(x)$ (rechts)

In dieser Arbeit wird die 2011 in [GBB11] eingeführte ReLU Aktivierungsfunktion genutzt. ReLU steht für Rectified Linear Unit und beschreibt die einfache Funktion

$f(x) = \max(x, 0)$, die den Vorteil hat, dass die Ableitung entweder den Wert 1, für $x > 0$, oder den Wert 0, für $x \leq 0$, hat. Dadurch wird der Gradientenabstieg bei Netzen mit vielen Schichten beschleunigt. Ebenso fallen Neuronen mit niedriger Aktivierung, also auch niedriger Bedeutung, einfach weg, was Overfitting vorbeugt.

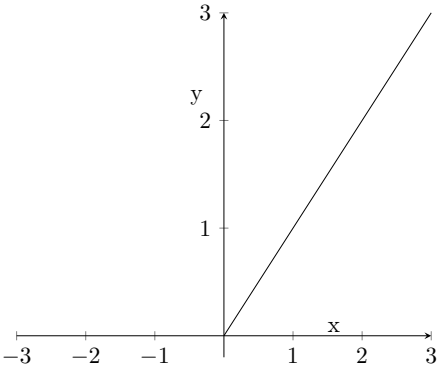

Abbildung 3.6: (ReLU) $f(x) = \max(x, 0)$

Nun können Netze, die aus solchen Neuronen bestehen, mittels Backpropagation trainiert werden. Bei Convolutional Neural Networks wird für gewöhnlich eine Lernrate angegeben, die bestimmt, wie stark bei jeder Iteration angepasst wird.

Bei Gradientenabstiegsverfahren werden große Fehler stärker korrigiert als kleine. Wäre die zweite Hälfte eines Trainingsdatensatzes mit dem Faktor 2 skaliert, wären die Fehler so groß, dass die erste Hälfte des Datensatzes einen vernachlässigbar kleinen Einfluss auf das Training hätte, und somit nicht gelernt werden würde. Um das zu vermeiden wird beim Training neuronaler Netze für gewöhnlich eine Skalierung der Eingabedaten vorgenommen. Dafür werden alle Werte des Datensatzes x durch das Maximum des Datensatzes xmax geteilt. Das gewährleistet, dass alle Daten auf die gleiche Größenordnung skaliert sind.

$$x \to \frac{x}{\text{xmax}} \tag{3.4}$$

Abbildung 3.7 auf der nächsten Seite zeigt den typischen Verlauf einer Lernkurve eines neuronalen Netzes. Dort sieht man in blau die *loss*-Kurve und in orange die *val_loss*-Kurve. Die loss-Kurve beschreibt die Fehlerfunktion der Trainingsdaten und die val_loss-Kurve die der Testdaten. Eine Epoche beschreibt hier eine vollständige Iteration über alle Trainingsdaten.

3.5 Faltungsnetze

Mit diesem Neuronenmodell lassen sich nun Faltungsnetze zusammensetzen. *Convolutional Neural Networks*, kurz *CNN*, setzen sich zusammen aus verschiedenen Schichten von Neuronen. Sie bekommen ihren Namen von den Faltungsschichten, die eine der diskreten Faltung entsprechende Operation durchführen.

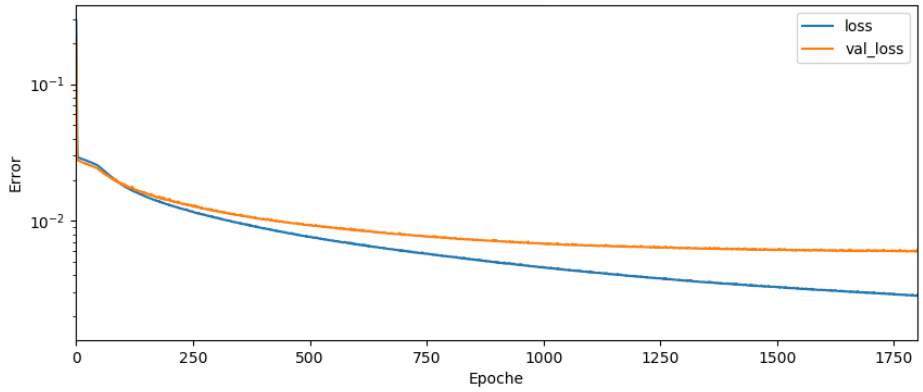

Abbildung 3.7: Typische Lernkurve eines Faltungsnetzes

3.5.1 Diskrete Faltung in der Bildverarbeitung

In der digitalen Bilderverarbeitung benutzt man Faltungsmatrizen, sogenannten Kernel, um Filteroperationen auf Bildern auszuführen. Dazu wird der Kernel über das Bild *geschoben* und die gewichtete Summe der Pixel berechnet. Somit wird jedes Pixel eines Bildes in einen Zusammenhang mit seiner Umgebung gebracht, was zum Beispiel die Hervorhebung von Kanten ermöglicht. Die Gewichte w_1 bis w_n ergeben sich dabei aus dem Kernel. Gleichung (3.5) zeigt einen Kernel der Abmessungen 3×3 (vgl. [GW08, S. 151]).

$$k = \begin{pmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{pmatrix} \tag{3.5}$$

Das errechnete Ergebnispixel I' berechnet sich dann mit dem Kernel k wie in Gleichung (3.6). Dabei sind $\vec{w}$ und $\vec{x}$ n-dimensionale Vektoren, die aus den Matrixkoeffizienten von k und den Pixelwerten des Bildes an der entsprechenden Stelle bestehen.

$$I' = \sum_{i=1}^{n} w_i x_i \tag{3.6}$$

Wie x gebildet wird, veranschaulicht folgendes Beispiel: Für $n = 9$ gilt für die Stelle $(x, y) = (1, 1)$

$$\vec{x} = \begin{pmatrix} I_{0,0} & I_{0,1} & I_{0,2} & I_{1,0} & I_{1,1} & I_{1,2} & I_{2,0} & I_{2,1} & I_{2,2} \end{pmatrix} \tag{3.7}$$

$I_{x,y}$ bezeichnet das Pixel an der Stelle (x, y) im Bild I. Mit verschiedenen Kernels lassen sich jetzt verschiedene Filter auf Bilder anwenden. Typische Filter sind Kantendetektion, Schärfen oder Weichzeichnen. Abbildung 3.8 auf der nächsten Seite zeigt die Anwendung eines Glättungsfilters [Onld] und eines Gaußweichzeichners, sowie eines Kantendetektors auf ein einfaches Beispielbild.

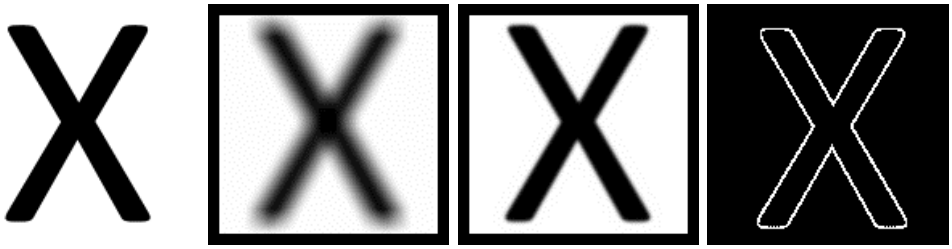

Abbildung 3.8: v.l.n.r.: Original, Weichzeichner, Gauß, Kanten

3.5.2 Schichten in Faltungsnetzen

Wie bereits zuvor erwähnt, bestehen CNNs aus verschiedenen Schichten. Jedes Netz beinhaltet mindestens eine Eingabeschicht und eine Ausgabeschicht. Die Eingabeschicht erhält den Eingabevektor, zum Beispiel das Bild, als Eingabe. Die Ausgabeschicht muss so geformt sein, dass ein Ergebnis der gewünschten Struktur ausgegeben wird. In dieser Arbeit wäre das zum Beispiel die Ausgabe von zwei Zahlen, die die (x, y) Position des Rosettenzentrums darstellen. Zwischen diesen beiden Schichten können sich beliebig viele weitere befinden. Abbildung 3.9 zeigt die typische Struktur eines CNNs. Dieses beinhaltet Faltungs-, Pooling- und Fully-connected Schichten, wobei das Pooling in der Abbildung als Subsampling bezeichnet wird. Auf die Funktion dieser Schichten wird im Folgenden eingegangen.

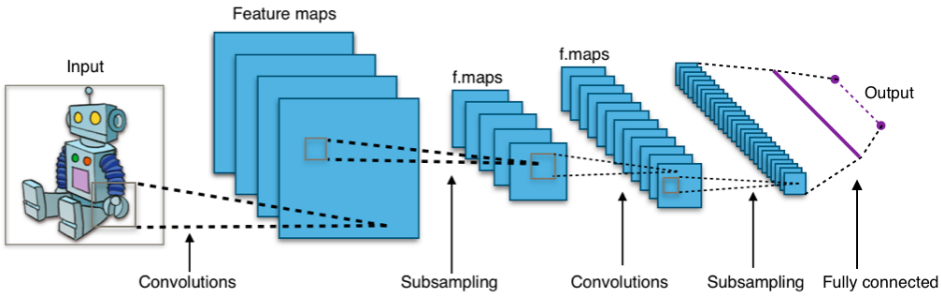

Abbildung 3.9: Beispielhafte CNN Architektur [Aph]

3.5.2.1 Faltungsschicht

Wie leicht zu erkennen ist, findet sich die Formel aus Gleichung (3.6) auf der vorherigen Seite in Gleichung (3.2) auf Seite 17 wieder. Es liegt auf der Hand, dass sich Faltungsoperationen durch künstliche Neuronen darstellen lassen, indem die Gewichte der Eingangskanten den Werten des Filterkernels entsprechen. Ordnet man viele solcher Neuronen mit identischen Gewichten zweidimensional an, so dass ein ganzes Bild abgedeckt werden kann, ist die Faltungsoperation nachgebildet. Diese Schicht bezeichnet man dann als Faltungsschicht, bzw. *convolutional layer*. Da die Nutzung fester Filter allerdings den Prinzipien des Machine Learnings widersprechen würde, werden

die Gewichte der Neuronen zufällig initialisiert und erst während der Lernphase angepasst. Eine Faltungsschicht enthält dabei eine beliebige Anzahl an separaten Filtern, die sich alle unterschiedlich ausprägen. Somit kann in jeder Schicht eine ganze Filterbank ausgeprägt werden (siehe Abbildung 3.10).

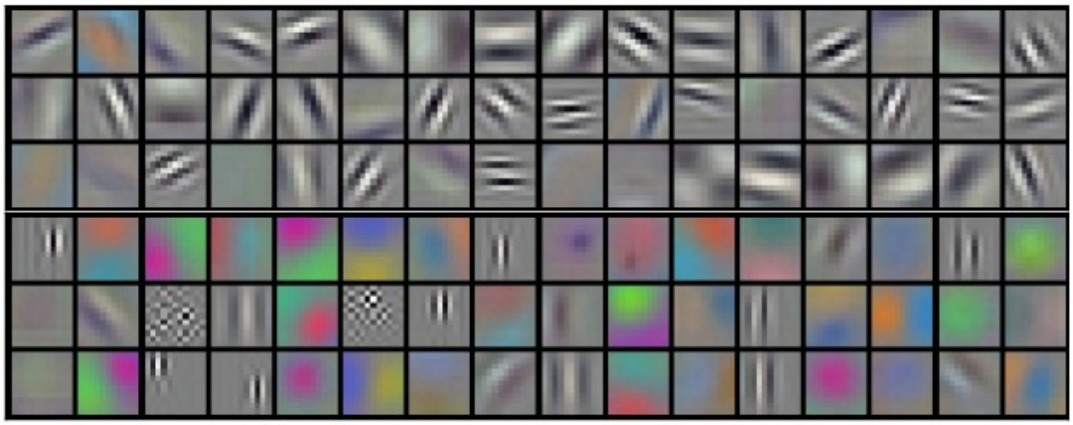

Abbildung 3.10: Von einem CNN gelernte Filter [KSH12]

Tiefere Schichten des Netzes, die nicht direkt auf dem Eingabebild arbeiten, sondern auf den Aktivierungen der vorgeschalteten Schicht, können diese Feature zu höherrangigen und komplexeren Features kombinieren. Abbildung 3.11 zeigt das anhand eines Netzes, das auf Gesichter trainiert wurde. Links sind nur einfache Features zu erkennen, die in der Mitte allerdings schon zu komplexen Strukturen wie einer Nase oder einem Auge kombiniert wurden. Rechts sieht man dann bereits ganze Gesichtsausschnitte.

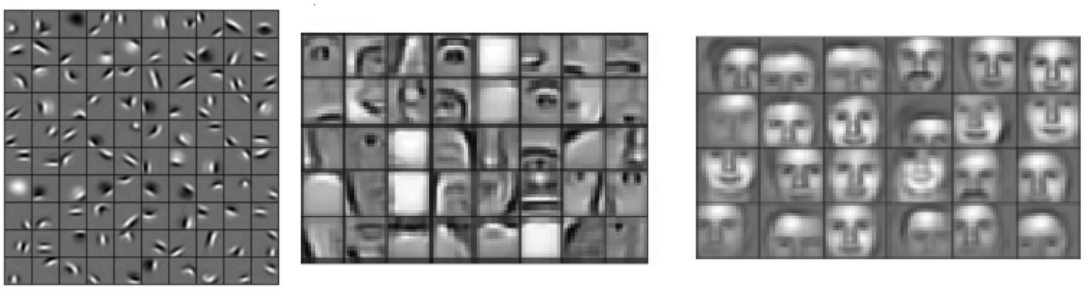

Abbildung 3.11: Kombination der Features zu komplexen Mustern [Lee+11]

3.5.2.2 Pooling

Durch die Verwendung mehrerer Filter vergrößern Faltungsschichten also meistens die dritte Dimension des Datenpakets. Wird auf ein RGB Bild der Größe $256 \times 256 \times 3$ eine Faltungsschicht mit 64 Filtern angewendet, hat das Ergebnis die Größe $256 \times 256 \times 64$. Diese Vergrößerung der Daten sorgt aber auch für eine aufwändigere Berechnung

und kann das System anfällig machen für Overfitting. Um dem vorzubeugen, kann ein *Subsampling* der Schichten vorgenommen werden. In dieser Arbeit werden dafür *Pooling Layer* verwendet, die nach dem Prinzip des *Average Pooling* arbeiten. Beim Pooling werden Bereiche von Neuronen auf unterschiedliche Weise zusammengefasst. Average Pooling bildet den Durchschnitt aller Aktivierungen, während zum Beispiel Max Pooling nur die stärkste Aktivierung berücksichtigt. Pooling arbeitet dabei immer auf einem festgelegten Bereich der vorgelagerten Schicht. Gleichung (3.8) zeigt das 2×2 Average Pooling anhand eines Beispiels. Dabei wird die Datenmenge auf 25% reduziert.

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 8 & 7 & 6 & 5 \\ 3 & 9 & 12 & 15 \\ 0 & 2 & 4 & 6 \end{pmatrix} \xrightarrow{2\times2 \ \textit{Average Pooling}} \begin{pmatrix} 4.5 & 4.5 \\ 3.5 & 9.25 \end{pmatrix} \tag{3.8}$$

3.5.2.3 Fully Connected

Bisher sind nur Schichten betrachtet worden, die lediglich einen Ausschnitt der vorherigen Schicht berücksichtigen. Diese eignen sich gut, um eine auf einen bestimmten Bereich beschränkte Operation für alle Bereiche des Bildes auszuführen, wie zuvor bei Convolutional und Pooling Layern beschrieben wurde. Damit lassen sich die räumlich weit voneinander getrennten Bereiche allerdings nicht in einen Zusammenhang bringen, was für die Abbildung des Netzes auf das gewünschte Ergebnis aber zwingend notwendig ist. Dafür gibt es die so genannten *Fully Connected Layer*, in denen jedes Neuron eine Verbindung zu jedem Neuron in der vorherigen Schicht hat.

3.5.2.4 Activation

Die *Activation Layer* belegt die Neuronen der vorherigen Schicht mit einer Aktivierungsfunktion. Hierbei handelt es sich eigentlich nicht um eine eigenständige Schicht, da die Aktivierung zu jedem Neuron dazu gehören sollte. Aus Anschauungsgründen wird die Aktivierung von einigen Frameworks jedoch als Schicht implementiert.

3.5.2.5 Flatten

Die *Flatten Layer* hat die Funktion aus einem beliebig dimensionalen Vektor einen eindimensionalen Vektor zu machen. Aus der Eingabe der Größe $w \times h \times c$ würde somit zum Beispiel eine Ausgabe der Größe $w * h * c$. Diese Operation ist notwendig, um Fully Connected Layer anzuwenden. Die *Reshape Layer* kann aus einem eindimensionalen Vektor wieder einen mehrdimensionalen Vektor formen.

4 Fouriertransformation

Zur Untersuchung von Beobachtungen in Bezug auf das Lernverhalten von neuronalen Netzen wurde in dieser Arbeit die Fouriertransformation genutzt. Sie ist ein bei der Untersuchung linearer Systeme übliches Werkzeug, zum Beispiel zur Interpretation von Faltungsoperationen in der Signal- oder Bildverarbeitung [Jä93, S. 30]. Um diese Methode nutzen zu können, wird in diesem Kapitel eine Einführung in die Fourieranalysis gegeben. Die Fouriertransformation wird mathematisch und anhand von Beispielen gezeigt. Ebenso wird eine einheitliche, für diese Arbeit geltende Notation eingeführt.

4.1 Fourieranalysis

Unter dem Begriff Fourieranalysis versteht man die Theorie der Fourierreihen. Eine periodische Funktion $f(t)$ mit Periode T lässt sich als Funktionsreihe aus den 2π periodischen Funktionen Sinus und Kosinus darstellen als

$$f(t) = \sum_{k=0}^{\infty} \left[A_k \cos\left(\frac{2\pi kt}{T}\right) + B_k \sin\left(\frac{2\pi kt}{T}\right) \right]. \tag{4.1}$$

Die Koeffizienten A_k und B_k berechnen sich für $k = 1, 2, ..., \infty$ durch

$$A_k = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \cos\left(\frac{2\pi kt}{T}\right) dt \tag{4.2}$$

$$B_k = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \sin\left(\frac{2\pi kt}{T}\right) dt. \tag{4.3}$$

Um zu verdeutlichen, wie Funktionen sich durch die Fourierreihendarstellung annähern lassen, wird nun das Beispiel einer Dreiecksfunktion $f(t)$ auf dem Intervall $[-1, 1]$ betrachtet. Dieses Beispiel stammt aus [Wea89, S. 3].

$$f(t) = \begin{cases} 1 + t, & t \in [-1, 0) \\ 1, & t = 0 \\ 1 - t, & t \in (0, 1] \end{cases} \tag{4.4}$$

Die Koeffizienten lassen sich durch die Gleichungen (4.2) und (4.3) bestimmen.

$$A_k = \frac{2 - 2\cos(\pi k)}{(\pi k)^2} \qquad , k = 1, 2, ..., \infty$$

$$A_0 = \frac{1}{T}\int_{-T/2}^{T/2} f(t)dt = \frac{1}{2} \tag{4.5}$$

$$B_k = 0$$

Die nach N Schritten abgebrochene Reihendarstellung ist jetzt

$$f_N(t) = \frac{1}{2} + \sum_{k=1}^{N}\left[\frac{2 - 2\cos(\pi k)}{(\pi k)^2}\right] cos(\pi k t). \tag{4.6}$$

Abbildung 4.1 zeigt $f(t)$ in blau und die beiden Annäherungen an die Fourierreihe $f_1(t)$ und $f_5(t)$ in orange.

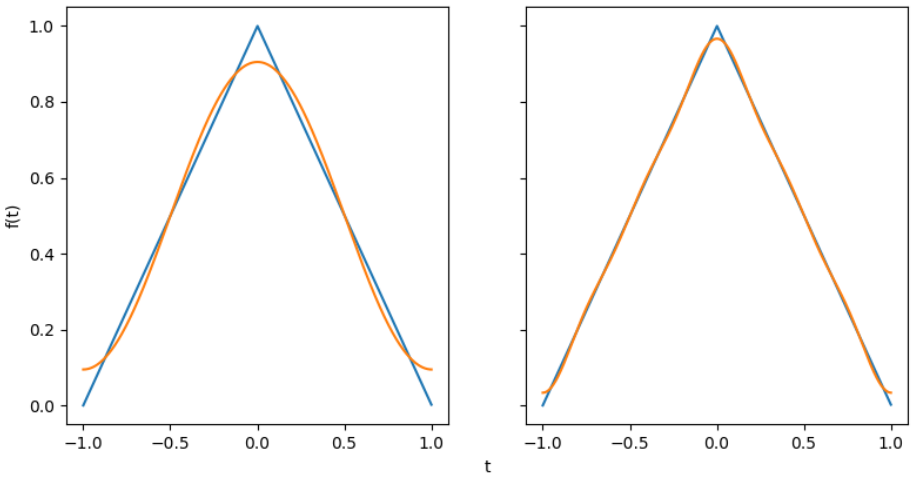

Abbildung 4.1: $f(t)$ mit Fourierreihendarstellung $f_1(t)$ (links) und $f_5(t)$ (rechts)

4.2 Fouriertransformation

Die Bedingung für die Zerlegung einer Funktion in ihr Spektrum, also die Darstellung als Fourierreihe, ist, dass die Funktion periodisch ist. Die Fouriertransformation ist eine Methode, die es ermöglicht, auch aperiodische, kontinuierliche Signale im Frequenzbe-

reich darzustellen. Die Fouriertransformation $\mathcal{F}$ und die inverse Fouriertransformation $\mathcal{F}^{-1}$ sind definiert als

$$\begin{aligned} \mathcal{F}(f(t)) =& \hat{f}(k) = \int_t f(t)e^{-2\pi ikt}dt \\ \mathcal{F}^{-1}(\hat{f}(k)) =& f(t) = \int_k \hat{f}(k)e^{2\pi ikt}dk. \end{aligned} \tag{4.7}$$

Zur vereinfachten Darstellung wird $\hat{f}(k)$ als Fouriertransformierte von $f(t)$ definiert.

$\hat{f}(k)$ ist im allgemeinen eine komplexe Funktion. Sie beschreibt das Frequenzspektrum von $f(t)$. Dabei wird der Realteil von symmetrischen Signalen und der Imaginärteil von antisymmetrischen Signalen bestimmt. Für die in dieser Arbeit verwendete Darstellung wird die Betrachtung des Powerspektrums, bzw. der spektralen Leistungsdichte, benötigt. Das Powerspektrum gibt an, wie hoch der Anteil der einzelnen Frequenzen am gesamten Signal ist [Onla]. Dafür wird das Produkt von $\hat{f}(k)$ und der komplex konjugierten $\hat{f}^*(k)$ gebildet.

4.3 Eigenschaften der Fouriertransformation

Die Fouriertransformation ist eine lineare Operation [Wea89, S. 159], es gilt also

$$af(t) + bg(t) = a\hat{f}(k) + b\hat{g}(k). \tag{4.8}$$

Außerdem besagt das Faltungstheorem [Wea89, S. 189] für die Fouriertransformation, dass die Faltung $*$ im Bildraum

$$f(t) * g(t) \coloneqq \int_{-\infty}^{\infty} f(x)g(t-x)dx \tag{4.9}$$

der punktweisen Multiplikation im Fourierraum entspricht. Die Umkehrung gilt ebenfalls.

$$\begin{aligned} \mathcal{F}[f(t) * g(t)] =& \hat{f}(k)\hat{g}(k) \\ \mathcal{F}[f(t)g(t)] =& \hat{f}(k) * \hat{g}(k) \end{aligned} \tag{4.10}$$

Für die Korrelation $\star$

$$f(t) \star g(t) = \int_{-\infty}^{\infty} f^*(t)g(t+x)dt \tag{4.11}$$

gilt entsprechend [Wea89, S. 196]

$$\mathcal{F}[f(t) \star g(t)] = \hat{f}(k)\hat{g^*}(k). \tag{4.12}$$

5 Verwendete Werkzeuge

5.1 System

Den nachfolgenden Tabellen können die relevanten Systemkomponenten und die relevanten installierten Programme und Pythonpakete entnommen werden, die für diese Arbeit zum Einsatz kamen.

CPU	Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
GPU	NVIDIA GeForce GTX 1080 Ti
RAM	32 GB DDR4 @ 2666MHz
Betriebssystem	Microsoft Windows 10 Pro 64-Bit

Continuum Anaconda	4.3.22
Python	3.5.2
h5py	2.6.0
hdf5	1.8.15.1
Keras	2.0.5
numpy	1.11.3
pandas	0.20.1
protobuf	3.3.0
scikit-learn	0.18.1
scipy	0.19.0
tensorflow-gpu	1.2.1

5.2 Frontend

Zur Entwicklung der Netze wurde *Keras*[Cho+15] verwendet. Keras ist eine öffentlich zugängliche und kostenlose Bibliothek für die Programmiersprache Python. Das Ziel von Keras ist, die Entwicklung neuronaler Netze zu vereinfachen, was durch das

folgende Zitat, welches wörtlich aus der Keras Dokumentation entnommen wurde, verdeutlicht wird.

> Being able to go from idea to result with the least possible delay is key to doing good research. [Onlf]

Keras kann auf Unix, iOS und Windowssystemen verwendet werden. Es funktioniert sowohl mit Python2.7, als auch mit Python3.5, sowie allen nachfolgenden Versionen. Unter Windows funktioniert Keras jedoch nur mit Python3.5. Wichtig ist, dass weitere Pythonpakete wie *numpy* oder *scipy* installiert sind. Zur Nutzung der GPU ist außerdem eine CUDA fähige Grafikkarte von NVIDIA und das Paket *cuDNN* nötig. Keras ermöglicht die Verwendung der gängigen Backends *Theano* und *TensorFlow*, sowie seit 2017 ebenfalls *CNTK*. Diese müssen separat installiert werden (vgl. [Onlf]).

Keras stellt alle gängigen Module zur Verfügung, die benötigt werden, um neuronale Netze zu erstellen. Diese lassen sich dank der Modularität der Bibliothek einfach hintereinander schalten. Die Art der Implementierung entspricht also auch dem intuitiven Verständnis des Schichtenmodells von neuronalen Netzen. Die so erstellten Netze lassen sich dann einfach abspeichern, trainieren, auswerten und nutzen.

Abbildung 5.1 zeigt, mit wie wenig Zeilen ein einfaches neuronales Faltungsnetz mit Keras erstellt werden kann. Dieses Netz besteht aus einer Faltungsschicht (Convolution2D) mit 32 Filtern der Größe 3×3, gefolgt von einer ReLU Aktivierung und einem Flatten layer. Anschließend übernimmt ein Fully Connected Layer (Dense) die Abbildung des Netzes auf die zwei Ausgabeneuronen. Mit dem *compile* Befehl wird das Netz erstellt und die Fehlerfunktion *mse* gewählt, was für *Mean Squared Error* steht. Das macht Keras zu einem leicht zugänglichen Frontend für die Entwicklung neuronaler Netze, was der der Hauptgrund für die Auswahl dieser Bibliothek ist.

```
model = Sequential()
model.add(Convolution2D(32, kernel_size=(3,3),
                        input_shape=(256,256,3)))
model.add(Activation('relu'))
model.add(Flatten())
model.add(Dense(2))
model.compile(loss='mse', optimizer='sgd')
```

Abbildung 5.1: Erstellung eines einfachen CNN mit Keras

5.3 Backend

Keras unterstützt die Backends TensorFlow, Theano und CNTK. Dabei ist das von Google entwickelte TensorFlow [Onlg] die Standardwahl und wurde auch in dieser Arbeit verwendet. Das deutlich ältere Theano ist für diese Arbeit nicht in Betracht

gezogen worden. Microsofts Cognitive Toolkit (CNTK) ist 2017 als Backend hinzugekommen und versucht TensorFlow abzulösen. Laut der Dokumentation von Microsoft schlägt es TensorFlow in allen wichtigen Bereichen (vgl. [Onlc]). Inwiefern CNTK wirklich schneller und genauer ist, wurde für diese Arbeit nicht getestet. Sollte sich ein Wechsel des Backends als lohnenswert erweisen, kann darüber in Zukunft nachgedacht werden.

6 Daten

6.1 Bilder

Alle in dieser Arbeit verwendeten Bilder sind RGB Bilder der Größe 256×256. Diese verwendeten Bilder stammen aus vergangenen Experimenten des IBG-2, sowie aus publizierten Datensätzen [Min+15]. Sie zeigen Nahaufnahmen von Arabidopsispflanzen in unterschiedlichen Stadien. Links in Abbildung 6.1 ist ein sehr frühes Stadium zu sehen, während das rechte Bild eine ältere Pflanze zeigt. Dass die Bilder eine geringe Auflösung haben, ist hierbei nicht hinderlich, da die benötigten Merkmale noch gut zu erkennen sind.

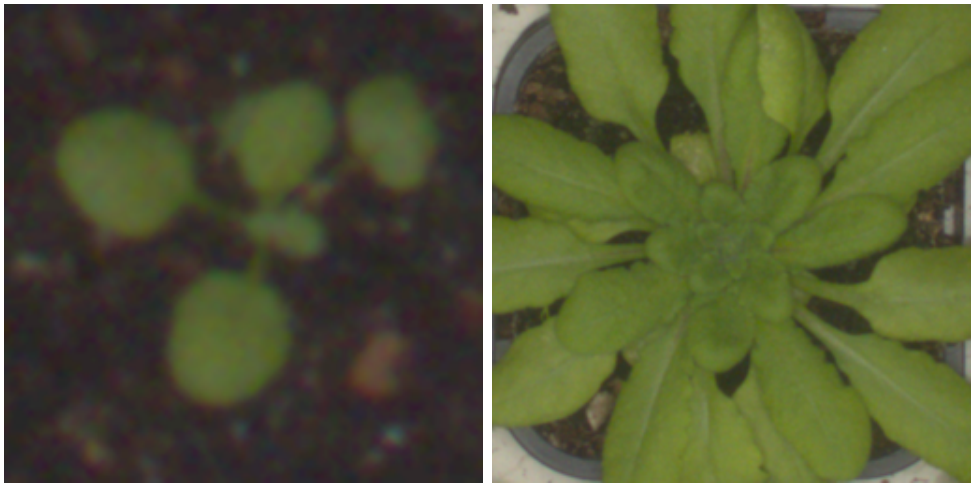

Abbildung 6.1: RGB Bilder von Arabidopsispflanzen in unterschiedlichen Stadien

6.2 Rosettenzentren

Die Ground Truth Daten der Rosettenzentren sind in csv Dateien gespeichert. Diese csv Dateien enthalten den Namen des Bildes und die x und y Koordinate des Rosettenzentrums. Diese wurde händisch durch Klicken auf das Zentrum ermittelt. Zu den Trainingsdaten gab es bereits die Ground Truth Daten. Für die Testdaten mussten diese noch generiert werden. Dafür wurde mittels eines Bildverarbeitungsprogramms per Mausklick ein roter Punkt im Rosettenzentrum platziert. Die Position dieses Punktes wurde dann von einem Pythonskript, das den rot Kanal des RGB Bildes nach dem Maximum absucht, ermittelt. Die roten Punkte in den Bildern wurden nicht gespeichert,

da dies die neuronalen Netze beeinflussen könnte. Abbildung 6.2 zeigt einen Ausschnitt aus einer Ground Truth Datei.

```
plant015_rgb.png;114;112
plant016_rgb.png;107;180
plant017_rgb.png;81;107
plant018_rgb.png;148;101
plant020_rgb.png;110;180
plant021_rgb.png;77;106
plant022_rgb.png;142;102
```

Abbildung 6.2: Auszug einer Ground Truth csv Datei

6.3 Vorbereitung der Daten

Um den Eingabevoraussetzungen neuronaler Netze zu entsprechen, mussten die Daten vorverarbeitet werden. Die Bilder, die als Trainingsdaten dienen, werden dafür zunächst in eine Liste geladen. Mit der Umwandlung dieser Liste in ein numpy Array entsteht ein Datensatz x' der Größe $256 \times 256 \times 3n$, wobei n der Anzahl der geladenen Bilder entspricht. Ebenso enthält ein Datensatz y' der Größe $2 \times n$ die Ground Truth Daten über die Position der Rosettenzentren. Diese Datensätze müssen nun normalisiert werden. So kann verhindert werden, dass Daten aus unterschiedlichen Größenordnungen unterschiedlich stark im Training berücksichtigt werden (s. Abschnitt 3.4 auf Seite 17). Hierfür wird für jeden Datensatz das Maximum über alle Einträge ermittelt und danach jedes Element des Datensatzes durch dieses Maximum geteilt. Dadurch werden alle Werte auf das Interval $[0; 1]$ skaliert.

$$\begin{aligned} \text{xmax} &= \max(x') \\ x &= \frac{x}{\text{xmax}} \\ \\ \text{ymax} &= \max(y') \\ y &= \frac{y'}{\text{ymax}} \end{aligned} \tag{6.1}$$

Anschließend werden die Datensätze in Trainings- und Testdaten unterteilt. Dafür werden 15% zufällig als Testdaten ausgewählt. Diese Testdaten bleiben im Trainingsprozess unbeachtet und werden zwischen den Iterationen zur Validierung benutzt.

Faltungsschichten sind translationsinvariant. Das heißt, dass die gelernten Features nicht von der Position im Bild abhängen. Fully Connected Schichten hingegen sind das nicht. In den hier vorliegenden Datensätzen sind die Rosettenzentren immer mittig abgebildet, was bei herkömmlichem Training (*end to end learning*) dazu führen

kann, dass das Netz diese Eigenschaft erlernt. Da in dieser Arbeit aber der translationsinvariante vom translationsvarianten Teil getrennt wurde, ist diese Eigenschaft der Trainingsdaten kein Problem.

Da jede Faltungsschicht allerdings einen nicht betrachteten Rand verursacht, müssen die Ground Truth Daten diesen Rand mit berücksichtigen. Deshalb wird die Position (x, y) zur neuen Position (x', y') umgerechnet. l_x und l_y sind dabei die Längen des Urpsrungsbildes in x bzw. y Richtung. l'_x und l'_y analog die Längen des Zielbildes. b gibt die Breite des Randes an.

$$\begin{aligned} x' &= \frac{l'_x}{l_x - 2b}(x - b) \\ y' &= \frac{l'_y}{l_y - 2b}(y - b) \end{aligned} \tag{6.2}$$

Faltungsschichten sind zwar translationsinvariant, allerdings sind sie nicht rotationsinvariant. Durch eine Rotation der Bilder können also neue Trainingsdaten generiert werden. Bei einer Rotation entstehen allerdings schwarze Randbereiche, die unerwünschte Effekte auf das Netz haben könnten. Deshalb wurde das Bild vor der Rotation um einen Rand der Breite 100 vergrößert. Dieser wurde gefüllt mit den Werten der Pixel am Rand des Bildes. Nach der Rotation um einen zufälligen Winkel wurde ein mittlerer Bereich aus dem Bild ausgeschnitten. Anschließend wurden die Ränder des Bildes noch geglättet, damit der streifenartige Effekt abgeschwächt wird. Somit entstanden rotierte Bilder ohne Rand (s. Abbildung 6.3).

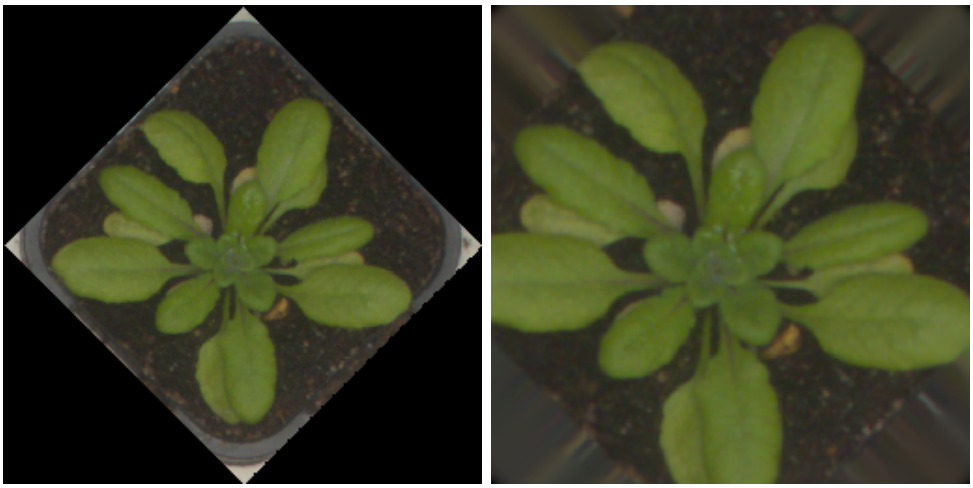

Abbildung 6.3: Um 45° gedrehtes Bild mit (links) und ohne Rand (rechts)

7 Positionsschätzung

Um das Problem der Positionsschätzung mit neuronalen Netzen zu bearbeiten, muss verdeutlicht werden, dass das Problem in zwei Teilprobleme unterteilt werden kann. Diese Teilprobleme sind zum einen das Schätzen der Position des gesuchten Features, zum Beispiel in Form einer Heatmap oder Wahrscheinlichkeitsdichtefunktion (PDF), und zum anderen die Abbildung dieser Information auf Koordinaten. Im folgenden Kapitel werden beide Teilprobleme getrennt voneinander betrachtet. In der *Deep Learning Community* wird meistens der Ansatz des *end to end* Lernens verfolgt, bei dem Lokalisierung und Umrechnung in einem trainiert werden müsste. Das Problem wird somit deutlich komplexer, was dazu führt, dass das Training solcher Netze für den Umfang dieser Arbeit zu lange dauern würde. Die Trennung der beiden Aufgaben verhindert das und ermöglicht somit ein schnelleres Training.

7.1 Abbildung einer Position auf Koordinaten

Ist die Information, an welcher Position sich ein bestimmtes Merkmal befindet, in Form einer PDF oder Heatmap bereits vorhanden, muss diese noch in Koordinaten umgewandelt werden. Diese Transferleistung kann bei künstlichen neuronalen Netzen von Fully Connected Schichten erbracht werden. Um herauszufinden, wie die Umwandlung funktionieren kann, wurde ein Modellexperiment aufgesetzt.

7.1.1 Beschreibung des Modellexperiments

Obwohl auf diskreten Werten gerechnet wurde, wird für die Notation nun der Schritt ins Kontinuierliche gemacht. Der Grund dafür ist die Veranschaulichung der Argumentation durch die einheitliche Verwendung von Funktionen, da die PDF ebenfalls als Funktion vorliegt und ihr Schwerpunkt als Integral bestimmt wird. Außerdem werden wichtige Eigenschaften der Fouriertransformation verwendet, deren diskrete Notation aufwendiger ist.

In diesem Modellexperiment wurde der Wertebereich $0 \leq x < 512$ betrachtet. Alle Gleichungen beziehen sich auf diesen Wertebereich. Ziel des Experiments ist die Entwicklung eines Netzes, das das Maximum der Eingabefunktion $P(x)$ ermittelt. Zunächst wurde als Eingabefunktion der zur Position y verschobene Deltapeak $P_y(x) = \delta(x - y)$ gewählt. Die einfachste Lösung dieses Problems ist die Verwendung einer einzelnen Fully Connected Schicht mit einem Ausgabeneuron, die als Eingabe $P_y(x)$ erhält. Das

Ausgabeneuron hat die Gewichtsfunktion $f(x)$. $\bar{x}(y)$ ist der Schwerpunkt von $P_y(x)$, der ebenfalls die Ausgabe des Netzes sein soll. y sind die Label der Trainingsdaten.

$$\delta(x) = \begin{cases} 1, & x = 0 \\ 0, & sonst \end{cases} \tag{7.1}$$

$$\bar{x}(y) = \int_x P_y(x)x \tag{7.2}$$

Ein Neuron rechnet mit Kantengewichten $f(x)$

$$\bar{x}(y) = \int_x P_y(x)f(x) = y \tag{7.3}$$

Werden die Kanten linear aufsteigend mit $f(x) = x$ gewichtet (siehe Abbildung 7.1) wird genau das Ergebnis aus Gleichung (7.2) erreicht. Um diese Gewichtung zu lernen, muss allerdings die Voraussetzung gemacht werden, dass jede Positionsmöglichkeit Teil der Trainingssets ist. In diesem Beispiel ist das mit 512 Positionen eine Leichtigkeit. In realen Anwendungen, zum Beispiel, wenn auf Fotografien der Größe 10 Megapixel gearbeitet wird, ist das allerdings nicht mehr zu gewährleisten. Deshalb muss auch im Experiment darauf geachtet werden, dass die Anzahl der Trainingsdaten geringer ist, als die Anzahl der möglichen Positionen.

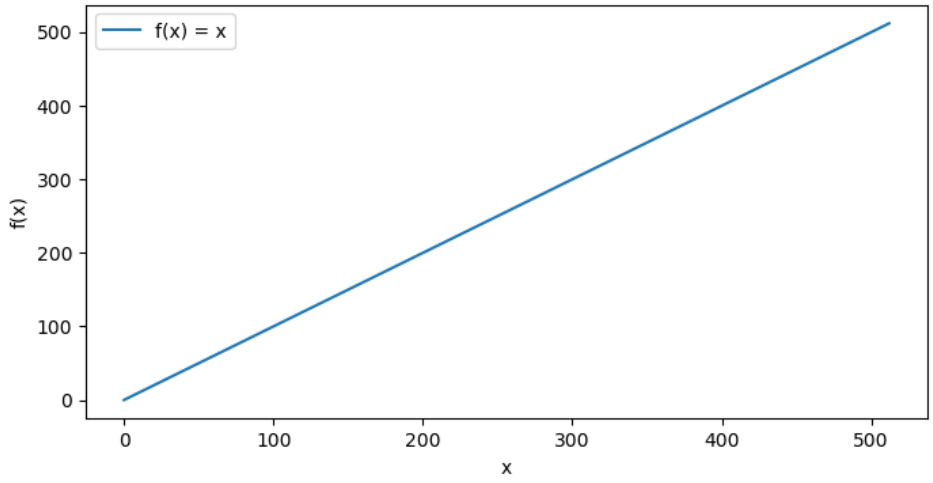

Abbildung 7.1: Gewünschte Gewichtsfunktion $f(x)$ des Ausgabeneurons

Für einen ersten Test wurden 512 Ein- und Ausgabepaare erstellt, so dass jede mögliche Position des Deltapeaks vertreten war. Diesem Set wurden 15% der Paare entnommen, die als Testset dienen. Mit dem Trainingsset, bestehend aus den verbleibenden 85%, wurde das Netz nun trainiert. Die Werte der Fehlerfunktion während des Trainings sind in Abbildung 7.2 auf der nächsten Seite dargestellt. Dort sieht man in

blau die *loss*-Kurve und in orange die *val_ loss*-Kurve. Die loss-Kurve beschreibt die Fehlerfunktion der Trainingsdaten und die val_loss-Kurve die der Testdaten.

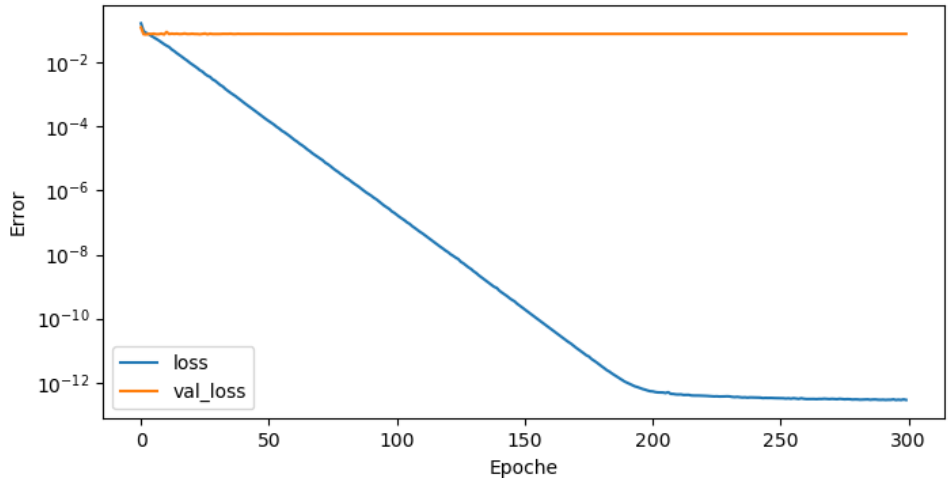

Abbildung 7.2: Error Funktionen von `peak_coord_net_nr`

Wie zu erwarten war, trainiert das Netz die Fully Connected Schicht sehr schnell auf die Testdaten. Der Fehler, der in jeder Epoche gemacht wird, sinkt der angegebenen Lernrate entsprechend exponentiell, bis er in eine Sättigung nahe Null verfällt. Der Fehler, der bei Verwendung des Testsets gemacht wird, sinkt hingegen nicht ab. Offenbar werden die Verbindungen der Neuronen zu den im Testset vorkommenden Fällen durch das Trainingsset nicht trainiert. Dieses Verhalten ist auf die Tatsache zurückzuführen, dass durch die Verwendung der delta-Funktion als Eingabe, die Datensätze so konstruiert sind, dass sie disjunkte Mengen darstellen. Die Kanten des Ausgabeneurons, die auf die Bereiche zugreifen, die lediglich im Testdatensatz angesprochen werden, erfahren nie eine Aktivierung und werden somit auch nicht trainiert. Abbildung 7.3 und Abbildung 7.4 auf der nächsten Seite zeigen die Gewichte der Fully Connected Schicht vor dem Training (randomisiert) und danach. Es ist zu erkennen, dass es tatsächlich Stellen gibt, an denen die Gewichtsfunktion das gewünschte lineare Verhalten nicht gelernt hat.

Das Ergebnis des Trainings ist also ein Netz, das bekannte Fälle bearbeiten kann, unbekannte Fälle aber nicht. Dieses Verhalten macht das Netz unbrauchbar, da in Anwendungsfällen ausschließlich unbekannte Daten verarbeitet werden müssen. Um dies zu umgehen, muss man darauf achten, dass die Trainingsdaten den gesamten Eingaberaum abdecken und alle beteiligten Kanten aktivieren. Dies ist bei Faltungsschichten durch ihre Translationsinvarianz automatisch gewährleistet. Bei räumlich varianten Schichten, zu denen unter anderem auch die Fully Connected Schichten gehören, benötigt man hingegen geeignete Regularisierungen. Dies wird im nächsten Abschnitt exemplarisch untersucht.

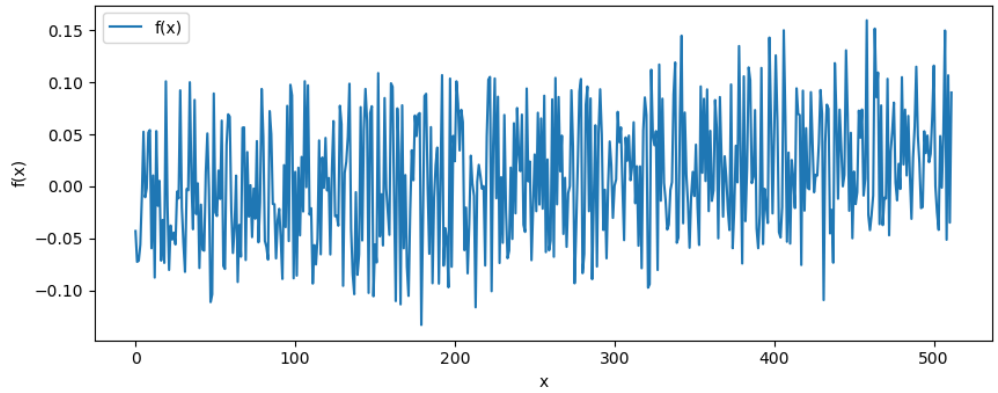

Abbildung 7.3: Zufällig initialisierte Gewichte der Fully Connected Schicht

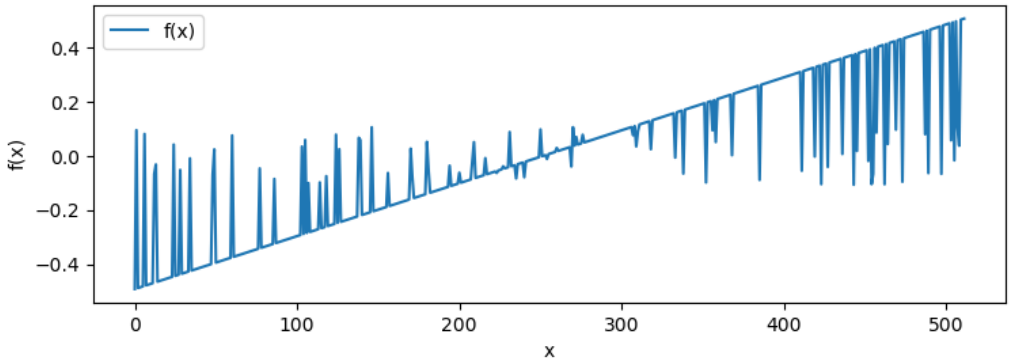

Abbildung 7.4: Gelernte Gewichte der Fully Connected Schicht

7.1.2 Regularisierung der Eingabe

Das Modellexperiment hat gezeigt, dass das Training eines Netzes, bestehend aus einer Fully Connected Schicht, nicht mit Deltapeak Arrays funktionieren kann, wenn weniger Trainingsdaten als abzudeckende Fälle vorhanden sind. Eine Idee, um dieses Problem zu vermeiden, ist eine Regularisierung der Trainingsdaten vorzunehmen. In einem weiteren Experiment wurde ein Netz `peak_coord_net` trainiert, das ebenfalls aus nur einer Fully Connected Schicht besteht. Die Eingabefunktion ist jetzt eine gaußsche Wahrscheinlichkeitsdichte $P(x)$ (Abbildung 7.5 auf der nächsten Seite).

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-0.5(\frac{x-\mu}{\sigma})^2} \tag{7.4}$$

Es gilt $\int_x P(x)dx = 1$. Gewählt wurde $\sigma = 15$. μ stellt das Maximum dar und wird wie zuvor mit jedem Wert von 0 bis 511 einmal belegt.

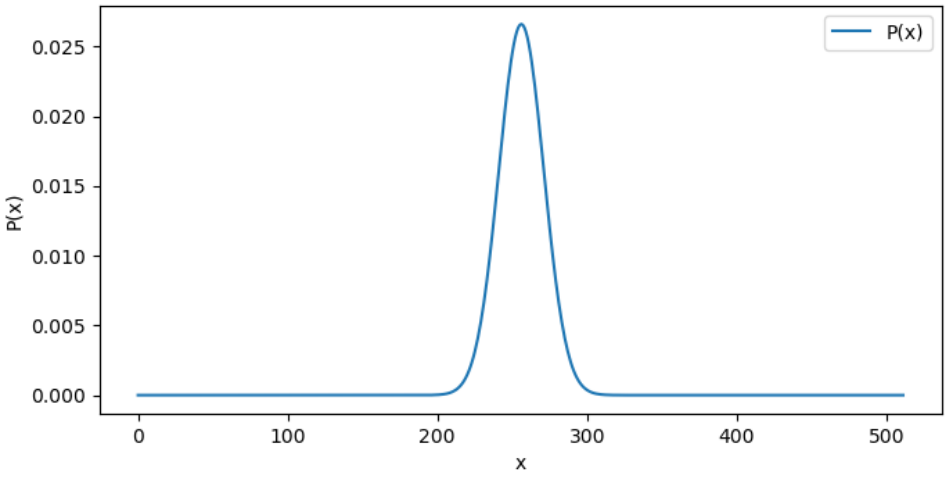

Abbildung 7.5: Gauß-Funktion mit $\sigma = 15$, $\mu = 256$

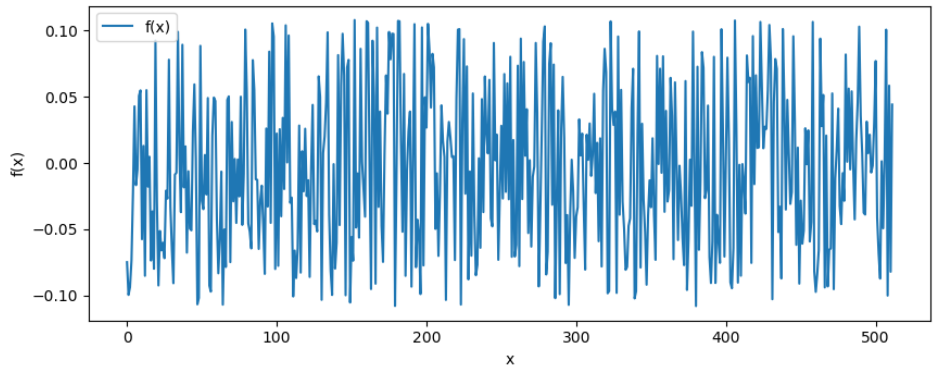

Abbildung 7.6: Zufällig initialisierte Gewichte der Fully Connected Schicht

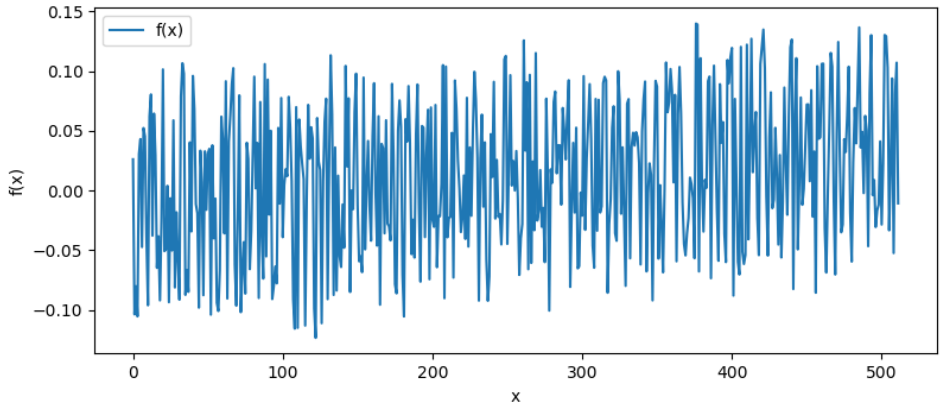

Abbildung 7.7: Gelernte Gewichte der Fully Connected Schicht

Abbildung 7.6 auf der vorherigen Seite zeigt die Gewichtsfunktion $f(x)$ des Ausgabeneurons vor der Trainingsphase. Diese ist randomisiert initialisiert, was den verrauschten Verlauf erklärt. Auch hier wird wieder erwartet, dass $f(x)$ wie in Abbildung 7.4 auf Seite 36 auf einen linearen Verlauf trainiert wird, jedoch ohne die dort beobachteten Ausreißer. In Abbildung 7.7 auf der vorherigen Seite ist aber zu erkennen, dass die Gewichte nach erfolgtem Training nach wie vor stark schwanken. Jedoch ist ebenfalls ein linear aufsteigender Trend zu beobachten, der darauf hinweist, dass niederfrequente Anteile im Signalspektrum gelernt wurden. Das Rauschen hingegen deutet auf Anteile höherer Frequenzen hin, die nicht durch das Training entfernt werden. Diese Beobachtungen geben Anlass zu genaueren Untersuchungen des Spektrums der Gewichtsfunktion.

Das Spektrum der Gewichtsfunktion vor und nach der Trainingsphase wird in Abbildung 7.8 dargestellt. Diese Kurven sind durch die Fouriertransformation (Kapitel 4) entstanden, wobei aus dieser jeweils das Powerspektrum des Eingangssignals berechnet wurde. Hierbei fällt auf, dass die Kurve des Spektrums der gelernten Gewichtsfunktion die Kurve des Spektrums der initialen Gewichtsfunktion bis auf einen Bereich um die 0 herum verdeckt. Die Anteile für höhere Frequenzen scheinen sich also während des Trainings nicht bzw. kaum geändert zu haben.

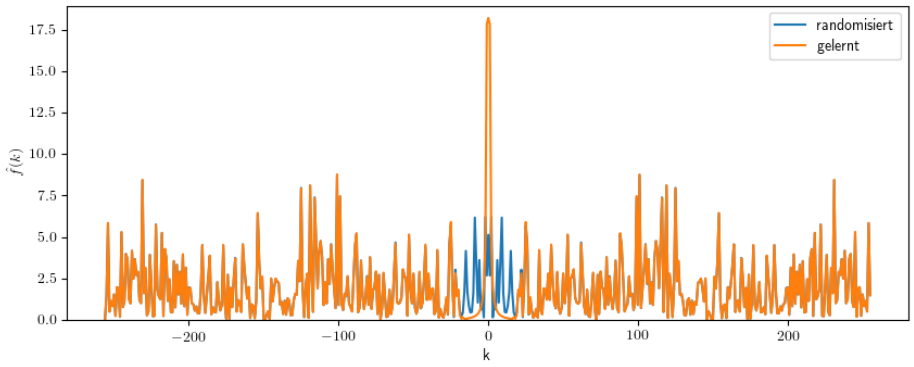

Abbildung 7.8: Spektrum der randomisierten (blau) und gelernten (orange) Gewichte

Um genauer herauszufinden, wie sich das Spektrum der gelernten Gewichte zusammen setzt, werden nun der Realteil und der Imaginärteil der Fouriertransformierten betrachtet (s. Abbildung 7.9 auf Seite 40). Dabei ist wieder zu erkennen, dass die Randbereiche verrauscht sind. Um $k = 0$ herum verlaufen die Funktionen allerdings glatt. Der Imaginärteil des Spektrum repräsentiert die ungeraden Funktionen und der Realteil die geraden (s. Kapitel 4). Deshalb kann vermutet werden, dass sich die gelernte Gewichtsfunktion, abgesehen vom Rauschen, aus einer konstanten und einer linearen Funktion zusammensetzt. Die Ähnlichkeit des Imaginärteils zum Imaginärteil des Spektrums der linearen Funktion $g(x) = \frac{x}{\text{xmax}}$ (s. Abbildung 7.10 auf Seite 40) deutet darauf hin, dass ebendiese Funktion $g(x)$ von der Gewichtsfunktion gelernt wurde.

xmax ist dabei der größte Wert der Eingabefunktion $P(x)$. Sein genauer Wert hängt von der Wahl von σ ab.

Das Powerspektrum von $g(x)$ (s. Abbildung 7.11 auf der nächsten Seite) ähnelt dem der gelernten Gewichte im Bereich um $k = 0$ ebenfalls. Allerdings unterscheiden sich hier die Amplituden des Peaks, was auf den konstanten Anteil, der in $f(x)$ enthalten ist und diese in y-Richtung verschiebt, hinweist.

7.1.3 Lernverhalten bei Regularisierung

Nach Konstruktion des Netzes soll $\bar{x}(y) = y$ gelernt werden. Das wird durch die Verwendung der Fehlerfunktion *mean-squared-error*, die in Gleichung (7.5) gezeigt wird, verdeutlicht.

$$E = \int_y (\bar{x}(y) - y)^2 dy \tag{7.5}$$

Sei $P(x)$ eine um 0 zentrierte Gauß-Funktion, dann ist $P(x-y)$ die um y zentrierte Gauß-Funktion. Diese Darstellung wird in Gleichung (7.6) verwendet, um die Ausgabe des Netzes darzustellen. Diese berechnet sich durch das Integral über den Wertebereich x des Produktes der Testfunktion $P(x-y)$ und der Gewichtsfunktion $f(x)$.

$$\bar{x}(y) = \int_x P(x-y) f(x) dx \tag{7.6}$$

$\bar{x}(y)$ entspricht jetzt genau der Definition der Faltung $*$. Das ermöglicht die Verwendung des Faltungstheorems (s. Abschnitt 4.3 auf Seite 26).

$$\hat{\bar{x}}(y) = \hat{P}(k)\hat{f}(k) \tag{7.7}$$

Der Fehler E lässt sich mit der Definition

$$D(y) = \bar{x}(y) - Y(y), \tag{7.8}$$

als

$$E = \int_y D^2(y) dy \tag{7.9}$$

schreiben, wobei die Notation $Y(y) = y$ verwendet wird. Wird der Fehler E nun als parametrisierte Funktion $E(\xi)$ verstanden, die ausschließlich an der Stelle $\xi = 0$ ausgewertet wird, lässt sich eine Darstellung erzeugen, die der Korrelation $\star$ entspricht.

$$E(\xi)\Big|_{\xi=0} = \int_y D(y) D(y+\xi) dy\Big|_{\xi=0} \tag{7.10}$$

Diese lässt sich wegen des Korrelationstheorems in das Produkt der Fouriertransformierten und der jeweils komplex konjugierten umwandeln (vgl. Abschnitt 4.3).

$$\int_y D(y) D(y+\xi) dy\Big|_{\xi=0} = \mathcal{F}_\xi^{-1}(\hat{D}(k)\hat{D}^*(k))\Big|_{\xi=0} \tag{7.11}$$

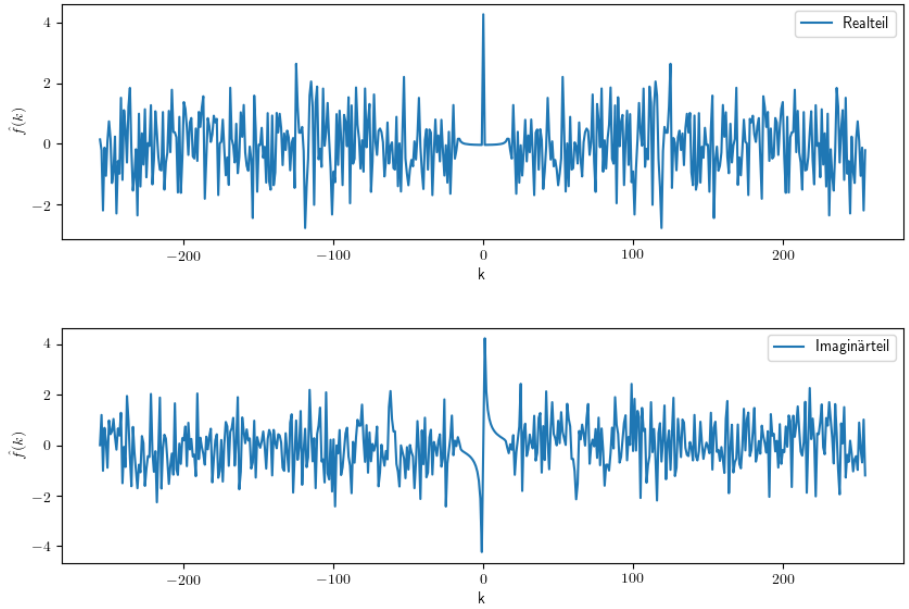

Abbildung 7.9: Real- und Imaginärteil des Spektrums der gelernten Gewichte

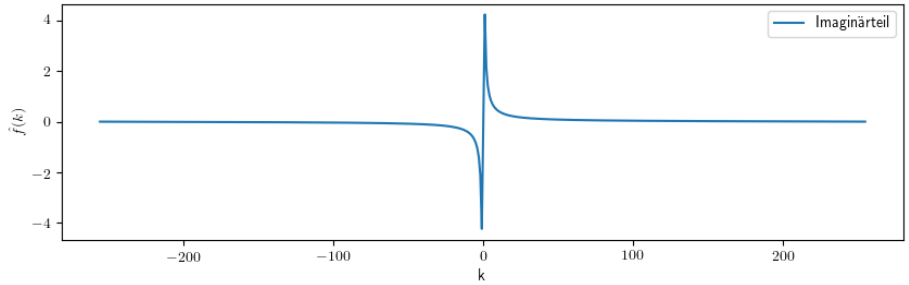

Abbildung 7.10: Imaginärteil des Spektrums von $g(x) = \frac{x}{\text{xmax}}$

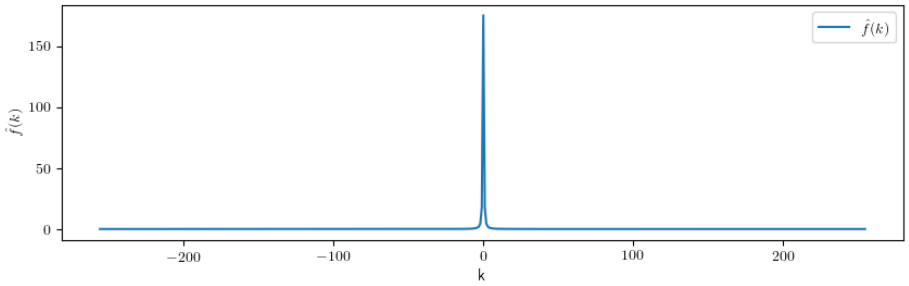

Abbildung 7.11: Spektrum von $g(x) = \frac{x}{\text{xmax}}$

Die Rücktransformation an der Stelle $\xi = 0$ ist aber

$$E = c \int_k \hat{D}(k)\hat{D}^*(k)dk \tag{7.12}$$

Die bei der Anwendung der inversen Fouriertransformation entstandene Konstante c ist für den Gradientenabstieg nicht von Bedeutung. Wir definieren $E(k) = c\hat{D}(k)\hat{D}^*(k)$. An der letzten Gleichung ist zu erkennen, dass Beiträge zu E von den Stellen k, also der entsprechenden Wellenfunktionen, gegen 0 gehen, wenn $\hat{D}(k)$ gegen 0 geht. Mit den Gleichungen 7.7 und 7.8 auf Seite 39 ist $\hat{D}(k)$ gegeben durch

$$\begin{aligned} \hat{D}(k) &= \hat{\tilde{x}}(k) - \hat{Y}(k) \\ &= \hat{P}(k)\hat{f}(k) - \hat{Y}(k). \end{aligned} \tag{7.13}$$

Es gilt also, dass $\hat{D}(k)$ gegen 0 geht, wenn $\hat{P}(k)\hat{f}(k) - \hat{Y}(k)$ gegen 0 geht.

Nach Konstruktion (siehe Abbildung 7.12) ist $\hat{P}(k)$ für hohe Frequenzen nahe Null. Somit ist auch das Produkt $\hat{P}(k)\hat{f}(k)$ für diese Frequenzen nahe Null, falls $\hat{f}(k)$ beschränkt ist, was hier gegeben ist. Abbildung 7.11 auf der vorherigen Seite zeigt, dass $\hat{Y}(k)$ ein spitzer Peak ist, der außerhalb eines engen Bereiches um $k = 0$ herum nur Werte sehr nah bei 0 hat. Das bedeutet, dass $\hat{P}(k)\hat{f}(k) - \hat{Y}(k)$ für hohe Frequenzen unabhängig von $\hat{f}(k)$ Null oder sehr nahe Null ist. An dieser Betrachtung ist zu erkennen, dass der Fehler $E(k)$ für Bereiche höherer Frequenzen unabhängig von den Gewichten $f(x)$ sehr gering bzw. 0 ist. Da das Netz mittels des in Abschnitt 3.4 auf Seite 17 beschriebenen Backpropagation Algorithmus trainiert wird, ist die Konsequenz aus $E(k) = 0$, dass kein Gewichtsupdate an diesen Stellen k durchgeführt wird. Das erklärt, wieso das Rauschen aus Abbildung 7.7 auf Seite 37, das durch das Vorkommen hoher Frequenzen erzeugt wird, nicht durch das Training entfernt werden kann.

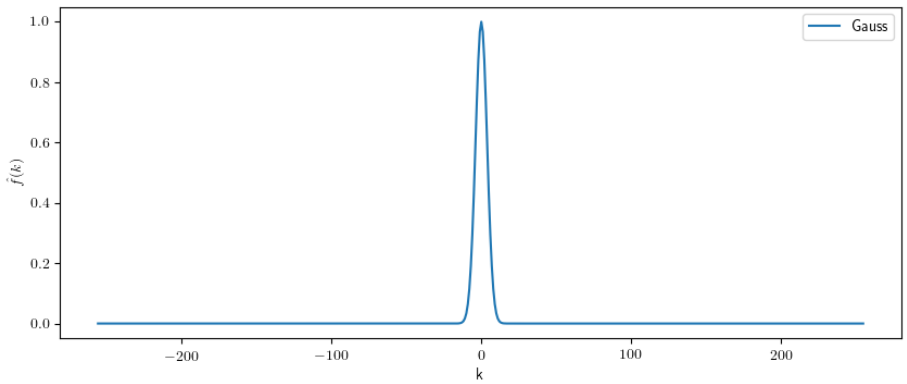

Abbildung 7.12: Powerspektrum der Gaußschen Normalverteilung mit $\sigma = 15$

7.1.4 Vorinitialisierung der Gewichte

Die gemachten Experimente haben gezeigt, dass das Rauschen der Gewichtsfunktion nicht durch das Training entfernt werden kann. Um dieser Beobachtung entgegen zu wirken, soll die Gewichtsfunktion nun manuell vorinitialisiert werden. Das würde vermeiden, dass es zu einem Rauschen in der Gewichtsfunktion kommen kann.

Zum Training wurden wieder 512 Gaußkurven $P(x)$ (Gleichung (7.4) auf Seite 36) mit $\mu_i = i$ für $i = 0, 1, ..., 511$ und $\sigma = 15$ erstellt. Diesem Set wurden wieder zufällig 15% der Ein- und Ausgabepaare entnommen, die das Testset bilden. Die verbliebenen 85% bildeten das Trainingsset. Die Gewichte der Fully Connected Schicht wurden mit der Funktion $f(x) = \frac{x}{512}$ belegt. Zuvor wurde gezeigt, dass eine Vorinitialisierung mit $f(x) = \frac{x}{\text{xmax}}$ vermutlich sinnvoller wäre, doch aus Anschauungsgründen wurde auf diese Optimierung verzichtet. Abbildung 7.13 auf der nächsten Seite zeigt die Gewichtsfunktion $f(x)$ nach der Trainingsphase. Es ist gut erkennen, dass $f(x)$ an den Rändern eine zunehmende Oszillation aufweist. Im mittleren Bereich verläuft $f(x)$ hingegen eher glatt.

Zum Vergleich dazu wurde unter unveränderten Voraussetzungen das Netz mit anderen Breiten der Gaußfunktion trainiert. Getestet wurden $\sigma = 50$ und $\sigma = 5$. Die sich ergebenen Gewichtsfunktionen werden in Abbildung 7.14 auf der nächsten Seite und Abbildung 7.15 auf der nächsten Seite gezeigt. Während sich für $\sigma = 50$ das Oszillationsverhalten auf den gesamten Wertebereich ausgebreitet hat, wurde es für $\sigma = 5$ weiter an den Rand gedrängt. Der Grund für die Überschwinger an den Rändern ist, dass der Schwerpunkt der Eingabefunktion dort nicht korrekt bestimmt werden kann. Dadurch, dass weder die Eingabe, noch die Gewichtsfunktion für Bereiche von $x < 0$ gilt, verschiebt der Schwerpunkt von $P(x)f(x)$ am linken Rand etwas in Richtung der Mitte des Wertebereichs. Analog verschiebt er sich am rechten Rand, weil der Wertebereich bei $x = 511$ endet. Da das Netz für diese Bereiche aber ebenfalls die korrekte Position der Schwerpunkte erhält, muss es die Differenz ausgleichen, indem der Wert der Gewichtsfunktion verkleinert bzw. vergrößert wird. $f(x)$ muss dies aber in der näheren Umgebung auch wieder ausgleichen, was zur beobachteten Oszillation von $f(x)$ führt. Da bei schmaleren Gaußkurven dieser Effekt erst näher am Rand auftritt, ist der glatte Bereich in der Mitte breiter. Allerdings ist die Oszillation am Rand stärker.

Ein weiterer Nachteil eines kleinen σ ist die Tatsache, dass mehr Trainingsdaten benötigt werden. Abbildung 7.16 auf Seite 44 zeigt die mit Gaußkurven mit $\sigma = 1$ trainierte Gewichtsfunktion $f(x)$. Da nur 85% aller möglichen Fälle durch die Trainingsdaten abgedeckt werden, und die Eingaben nicht breit genug sind, alle weiteren Fälle ebenfalls zu erfassen, enthält $f(x)$ erneut Stellen, an denen keine zuverlässige Vorhersage getroffen werden kann. Die vertikalen Linien in Abbildung 7.16 auf Seite 44 markieren die Einträge des Testsets. Es ist zu erkennen, dass die Oszillation zunimmt, wenn die Markierungen näher beieinander liegen.

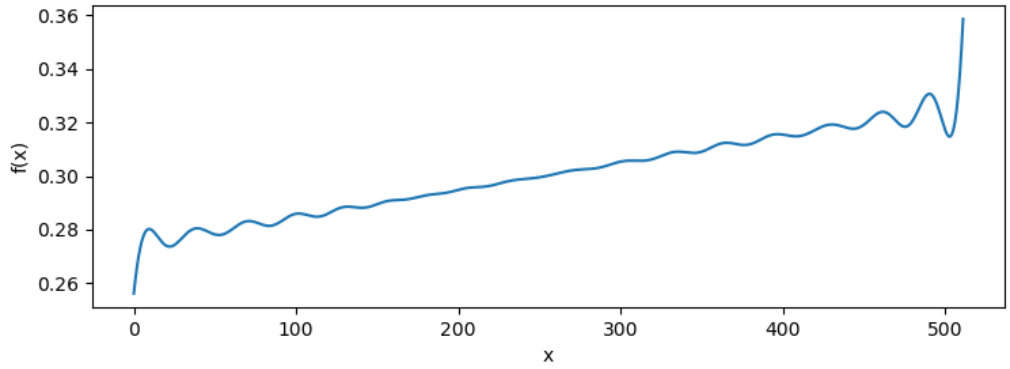

Abbildung 7.13: Gewichtsfunktion $f(x)$ nach Training mit Gaußkurven ($\sigma = 15$)

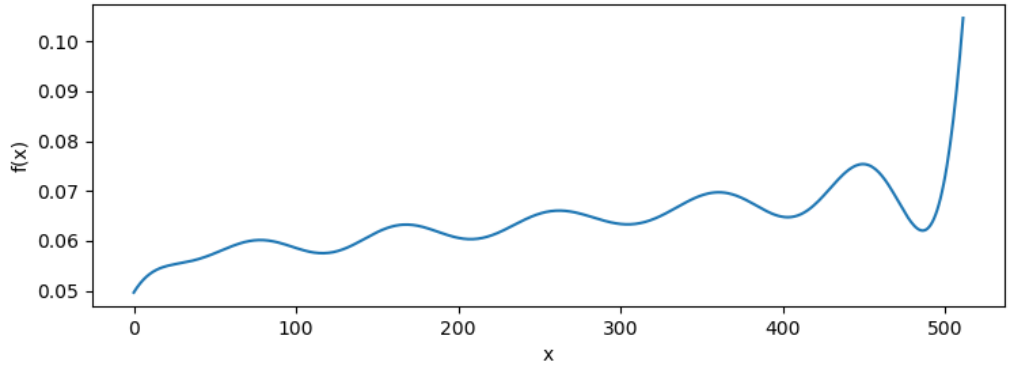

Abbildung 7.14: Gewichtsfunktion $f(x)$ nach Training mit Gaußkurven ($\sigma = 50$)

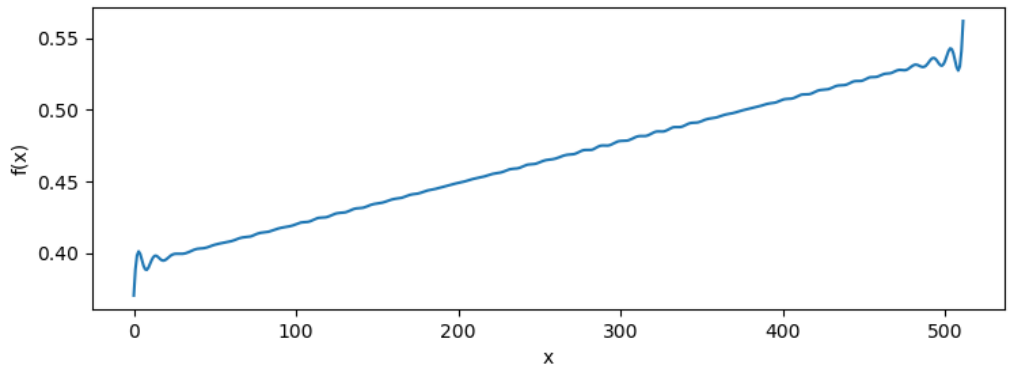

Abbildung 7.15: Gewichtsfunktion $f(x)$ nach Training mit Gaußkurven ($\sigma = 5$)

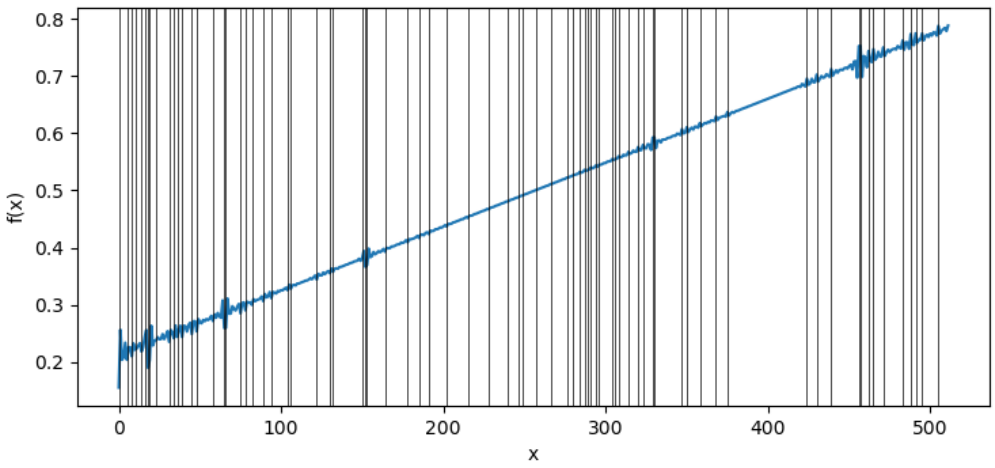

Abbildung 7.16: Gewichtsfunktion $f(x)$ nach Training mit Gaußkurven ($\sigma = 1$) und Markierung der Einträge des Testsets

7.2 Positionsschätzung des Rosettenzentrums

Zunächst muss das gesuchte Feature lokalisiert werden. In dieser Arbeit ist das das Rosettenzentrum der Pflanze. Die Eingaben sind also die in Kapitel 6 auf Seite 30 beschriebenen RGB Bilder. Die gewünschte Ausgabe ist eine zweidimensionale Wahrscheinlichkeitsdichtefunktion, deren Schwerpunkt genau an den Koordinaten des Rosettenzentrums liegen soll.

7.2.1 Vorüberlegungen zur Lokalisierung des Rosettenzentrums

Zur Findung von Features bieten sich Faltungsschichten an, da die verschiedenen Filter auf bestimmte Merkmale trainiert werden können. Doch zur Konstruktion dieser Faltungsschichten muss zunächst die Frage gestellt werden, was gelernt werden soll. Bei der Betrachtung der Eingabebilder fällt auf, dass das Rosettenzentrum immer von ähnlichen Merkmalen umgeben ist. In Abbildung 7.17 sind drei Bereiche markiert, die solche Merkmale darstellen könnten. Markiert sind ein Blattanfang (blau), ein Stängel (gelb) und eine Blattkante (violett). Diese sind in Abbildung 7.18 auf der nächsten Seite noch einmal vergrößert dargestellt.

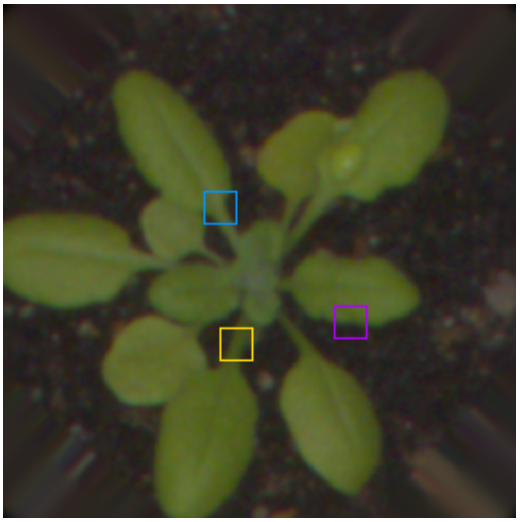

Abbildung 7.17: Eingabebild mit markierten Merkmalen

Hierbei gilt es zu beachten, dass es sich lediglich um eine Vorüberlegung handelt, welche Merkmale das Netz lernen könnte. Die wirklich erlernten Filter lassen sich nur schwer bis gar nicht vorhersagen. Für die Überlegungen, wie ein solches Netz zu konstruieren wäre, reicht allerdings die Betrachtung weniger Beispielmerkmale.

Bei den drei ausgewählten Bereichen fällt auf, dass die Größe 17×17 gerade so ausreicht, um die charakteristischen Merkmale zu beinhalten. Dass bei einer Blattkante (violett) nur eine Kante zu erkennen ist, bei einem Stängel (gelb) aber zwei, ist zum Beispiel ein charakteristisches Merkmal. Diese Beobachtung lässt schon vermuten, dass

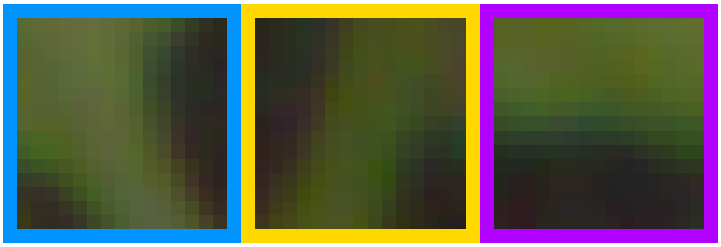

Abbildung 7.18: Merkmale

die erste Faltungsschicht, deren Filter direkt auf den Eingabebildern arbeiten, Filter mit einer Größe von mindestens 17×17 beinhalten sollte.

Die nächste Frage, die sich stellt, ist, wie viele Filter die erste Schicht enthalten sollte. Diese Frage ist ebenfalls nur sehr schwer zu beantworten. Aus der Tatsache, dass ein Filter nicht ein Feature und das selbe Feature, welches aber um einen gewissen Winkel rotiert ist, erkennen kann, ergibt sich schon die Tendenz dazu, viele Filter zu nehmen. Damit können dann viele Rotationen des selben Filters abgedeckt werden.

Die erste Schicht eines solchen Netzes würde also die zuvor beschriebenen Merkmale finden. Eine weitere Faltungsschicht müsste diese dann in einen Zusammenhang bringen. Auch hier muss wieder eine hohe Anzahl von Filtern in Betracht gezogen werden, um der hohen Anzahl verschiedener Kombinationen gerecht zu werden. Zusätzlich müsste eine letzte Faltungsschicht die Filterdimension auf 1 reduzieren, um nur eine einzelne Heatmap auszugeben.

7.2.2 Entwicklung der Faltungsnetze

Anders als bei den Netzen, die die gefundene Position auf Koordinaten abbilden (s. Abschnitt 7.1), lassen sich diese Faltungsnetze durch ihre Komplexität nur langsam trainieren. So benötigt ein Netz wie in Abbildung 7.19 auf der nächsten Seite links mit $1,6$ Millionen Parametern, das mit 2108 Bildern trainiert wird, auf dem hier verwendeten System circa eine Minute für eine Epoche, also eine Iteration über alle Trainingsdaten. Ein Training über 2000 Epochen würde also bereits $1,4$ Tage dauern. Üblich sind allerdings deutlich mehr Bilder und Epochen. Durch die zeitliche Begrenzung dieser Arbeit sind die Möglichkeiten der Variation also beschränkt, weshalb nur einige wenige Netzarchitekturen getestet werden konnten. Es kann nicht garantiert werden, dass andere Netzarchitekturen keine besseren Ergebnisse erzielen.

Abbildung 7.19 auf der nächsten Seite zeigt zwei getestete Architekturen, die die Vorüberlegungen des vorherigen Kapitels berücksichtigen. Für jedes Eingabebild wurde als Ground Truth ein Ausgabebild der Größe 46×46 erzeugt, dessen Pixelwerte eine zweidimensionale Gaußverteilung beschreiben. Das σ wurde hierfür ad hoc auf $\sigma = 12$ gesetzt. Diese hat das Maximum an der auf die geringere Größe angepasste Position, an der sich das Rosettenzentrum im Eingabebild befindet. So wurden die Ein- und Ausgabepaare erstellt, die das Trainings- bzw. Testset bilden.

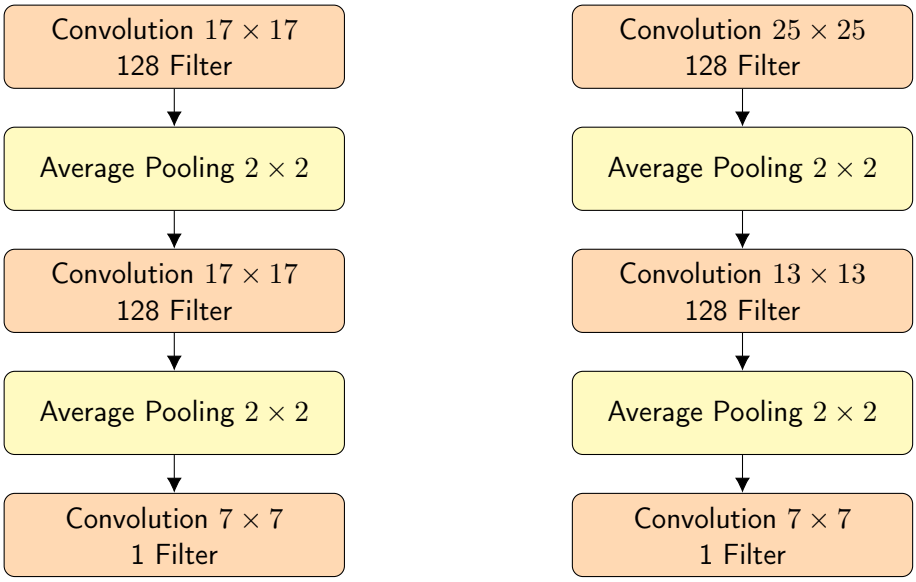

Abbildung 7.19: Netzarchitektur `ff_net1` (links) und `ff_net2` (rechts)

Nach Beendigung des Trainings nach 1000 Epochen wurden beide Netze untersucht. In Abbildung 7.20 auf der nächsten Seite sind die von den Netzen gelernten Filter der ersten Schicht zu sehen. Dort ist zu erkennen, dass bereits einige Filter Kanten oder Formen abbilden. Der Großteil zeigt jedoch lediglich Rauschen. Außerdem ist zu erkennen, dass die Anzahl der Filter, die kein Rauschen zeigen, bei `ff_net1` höher ist (ca. 29), als bei `ff_net2` (ca. 18). Diese Beobachtung lässt vermuten, dass das Training nicht ausreichend genug war, um komplexe Filter zu erlernen. Da schon die erste Schicht überwiegend aus Rauschen besteht, werden diese Netze nicht anwendbar sein, um das Rosettenzentrum zu lokalisieren. Aus diesem Grund wurde im Folgenden ein Ansatz des Transfer Learning verfolgt.

7.2.3 Verwendung eines vortrainierten Netzes

Da die Ausbildung einfacher *Low Level* Filter mit den Vorhandenen Trainingsdaten nicht oder nur schwer möglich ist, wurde hier der Ansatz verfolgt, die erste Faltungsschicht eines bereits trainierten Netzes zu verwenden. Die Wahl fiel dabei auf das oft als `VGG16` betitelte Faltungsnetz der Visual Geometry Group der University of Oxford, die 2014 den ersten Platz bei der ILSVRC [1] belegten [SZ14].

Die erste Schicht des `VGG16` besteht aus 64 Filtern der Größe 3×3. Der Rand des Eingabebilds wird vor der Faltung allerdings mit Nullen aufgefüllt, weswegen die Ausgabe der Schicht die Größe der Eingabe behält. Die geringe Größe der Filter sorgt dafür, dass nur einfachste Merkmale abgebildet werden. Die Kombination dieser zu komplexeren Merkmalen sollen die nachfolgende Schichten übernehmen. Da die Gewichte dieser

[1] ImageNet Large Scale Visual Recognition Challenge [Onle]

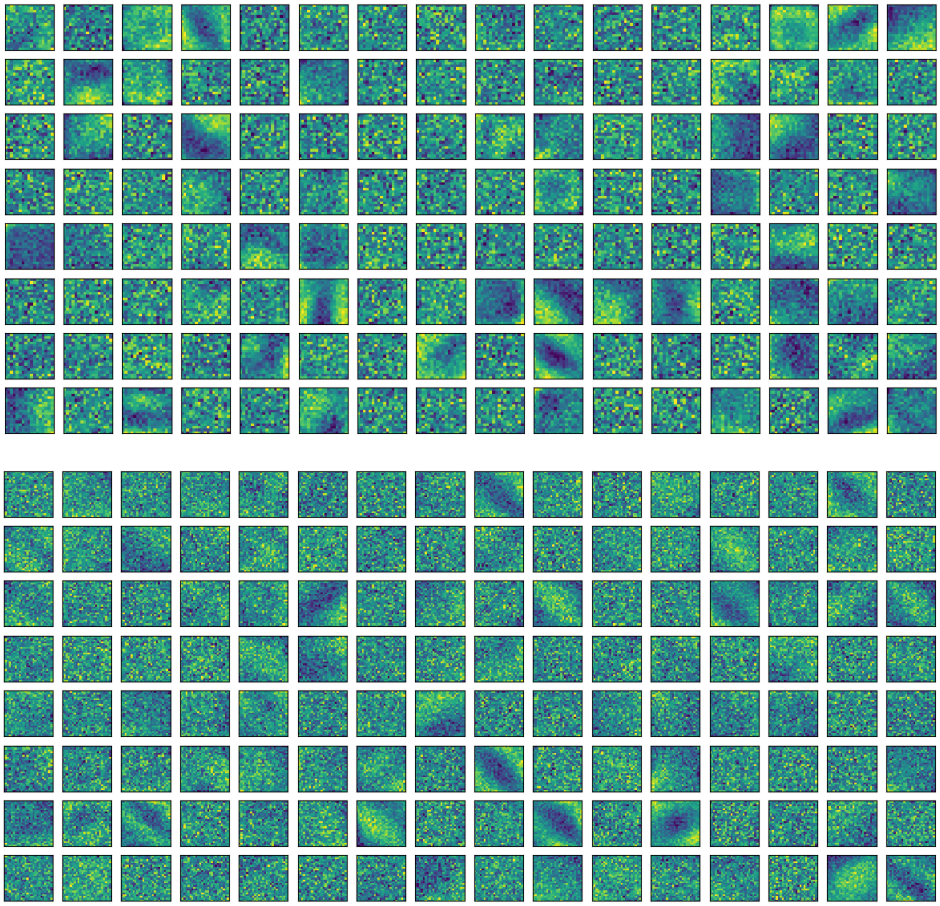

Abbildung 7.20: Gelernte Filter der ersten Schicht von `ff_net1` (oben) und `ff_net2` (unten)

Filter bereits trainiert sind, wurde die erste Schicht für das Training *eingefroren*. Das bedeutet, dass die Gewichte der Schicht durch das Training nicht verändert werden können. Alle anderen Schichten können trainiert werden. Abbildung 7.21 auf der nächsten Seite zeigt die Architektur des erstellten Netzes.

Abbildung 7.22 auf Seite 50 zeigt an einigen Eingabebeispielen, wie die Ausgaben des trainierten Netzes aussehen. Zu sehen sind Heatmaps, deren Schwerpunkt dem Rosettenzentrum der Pflanze auf dem Eingabebild entsprechen soll. Zu beachten ist hierbei, dass durch die Faltungsschichten ein Rand von 24 Pixeln abgeschnitten wird. Die wesentliche Sturktur der Pflanze ist ebenfalls zu erkennen. `ff_pretrained` könnte also in der Lage sein das Rosettenzentrum der Pflanzen in Form einer zweidimensionalen Funktion abzubilden, die einer Wahrscheinlichkeitsdichtefunktion nahe kommt. Zu beachten ist hierbei allerdings, dass die Integrale unter den Ausgaben im Allgemeinen nicht auf 1 normiert sind, wie in Abschnitt 7.1 vorausgesetzt war. Die Fehlerfunktion

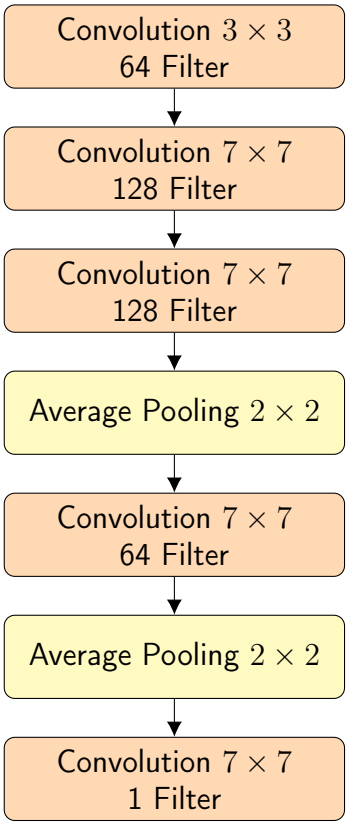

Abbildung 7.21: Netzarchitektur `ff_pretrained`

des Netzes erfährt durch das Training noch keine Sättigung und würde vermutlich noch weiter fallen. Das ist in Abbildung 7.23 auf der nächsten Seite zu erkennen. Das Netz könnte durch weiteres Training also noch verbessert werden.

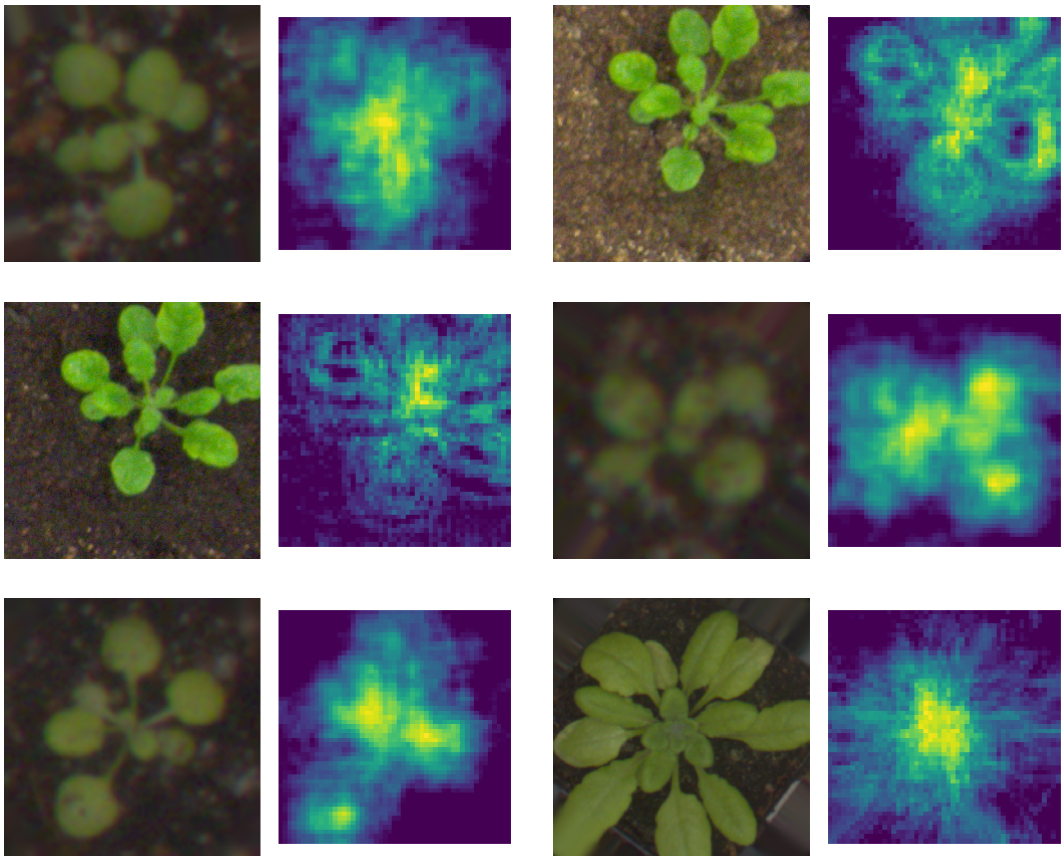

Abbildung 7.22: Ein- und Ausgabe des trainierten `ff_pretrained`

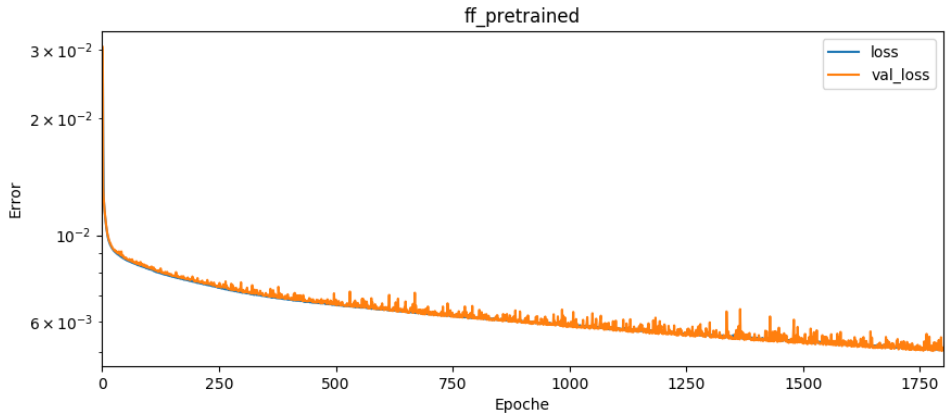

Abbildung 7.23: Error Funktion von `ff_pretrained`

8 Ergebnisse

Um die Erkenntnisse aus Kapitel 7 anzuwenden, müssen die beiden Teilaufgaben nun miteinander verknüpft werden. Dafür muss ein Netz erstellt werden, das ein Netz für die Feature Findung und eins für die Umrechnung der Lokalisierung in Koordinaten miteinander verbindet. Das Ergebnis wäre ein Netz, das ein RGB Bild als Eingabe erhält und zwei Koordinaten als Ausgabe bereitstellt. Dieses Kapitel beschreibt ein solches Netz und stellt die Ergebnisse vor.

8.1 Umrechnung der Koordinate im Zweidimensionalen

Bisher wurde die Umrechnung der Koordinate nur eindimensional betrachtet. Für die tatsächliche Verwendung wird allerdings die Umrechnung einer zweidimensionalen Gaußkurve in zwei Koordinaten benötigt. Hierbei gilt es allerdings zu beachten, dass die Integrale unter den Eingabekurven nun nicht mehr auf 1 normiert sind, da sie nicht künstlich erstellt, sondern vom Feature Findungs Netz geschätzt werden. Durch eine Normierungsschicht zwischen den beiden Teilnetzen lässt sich dieses Problem lösen, allerdings wäre das wieder ein direkter Eingriff ist die Funktion des Netzes. Möchte man erreichen, dass das Netz die Normierung selbst lernt, ist die Verwendung einer dritten Ausgabekoordinate z, die den Wert des Integrals angibt, eine Möglichkeit. Die Pixelkoordinaten x und y müssen dann für das Training mit z multipliziert werden. Da ein solches Netz in der Nutzphase allerdings 3 Koordinaten ausgibt, müssen die Ergebnisse noch interpretiert werden. Durch folgende Rücktransformation lässt sich die gesuchte Position (x', y') aus dem Ausgabetriplett (x, y, z) berechnen:

$$\begin{aligned} x' &= \frac{x}{z} \\ y' &= \frac{y}{z} \end{aligned} \tag{8.1}$$

Da die Verwendung einer vom Netz gelernten Normierung allerdings nicht schnell und genau gelernt werden kann, sondern wieder einer genaueren Betrachtung bedarf, wurde für diese Arbeit die Normierung von einer separaten Schicht vorgenommen.

Das Umrechnungsnetz besteht also aus einer Fully Connected Schicht mit 2 Ausgabeneuronen. Da sich diese Netze sehr schnell trainieren lassen und die Erzeugung von künstlichen Trainingsdaten ebenfalls von geringem Aufwand ist, wurde das Umrechnungsnetz mit deutlich mehr Trainingsdaten vortrainiert, als nötig sind, um alle Möglichkeiten abzudecken. Um die zuvor in Abschnitt 7.1 beobachteten Probleme mit Oszillationen in der Gewichtsfunktion zu vermeiden, wurde das σ der Gaußfunktion für jede Eingabe zufällig bestimmt. Dabei galt $0 < \sigma \leq 8$.

8.2 Kombination der Teilnetze

Keras unterstützt das Zusammenbauen neuer Netze aus bereits trainierten Netzen. Ebenfalls lassen sich aus fertigen Modellen einzelne Schichten oder Gewichte entnehmen und woanders weiter verwenden. So ist auch die Kombination der beschriebenen Teilnetze in Keras unproblematisch, wie das folgende Beispiel zeigt. Dafür werden einfach vorhandene Netze über die Funktion `load_model` geladen und anschließend miteinander verknüpft.

```
model1 = keras.models.load_model('model1.f5')
model2 = keras.models.load_model('model2.f5')
model = Sequential()
model.add(model1)
model.add(model2)
model.compile(loss='mse', optimizer='sgd')
```

Voraussetzung hierfür ist, dass das Format der Ausgabe eines Netzes oder einer Schicht mit dem der Eingabe der nachfolgenden Schicht übereinstimmt. Die zuvor beschriebenen Netze für die Findung des Features und die Umrechnung der Lokalisierung in Koordinaten sind so konstruiert worden, dass die Aus- und Eingaben kompatibel sind. Die Normierungsschicht, die dazwischen geschaltet werden muss, verändert die das Format der Daten nicht. Somit lässt sich das endgültige Netz `final_net` problemlos zusammenbauen.

Abbildung 8.1 auf der nächsten Seite zeigt zufällig gewählte Bilder aus einem durch Drehung generierten Datensatz, der nicht Teil des Trainings war. Die orangen Kreuze in den Bildern markieren die Schätzung für die Position des Rosettenzentrums vom Netz. Die weißen Kreuze markieren die Ground Truth Position. Augenscheinlich liegen diese Schätzungen nah an der tatsächlichen vorliegenden Position. Eine Verarbeitung des gesamten Datensatzes ergibt eine mittlere Abweichung von 8.46 Pixeln. Dieser doch recht hohe Wert kommt durch Ausreißer zustande (in Abbildung 8.1 unten rechts), bei denen die Schätzung weit vom Rosettenzentrum abweicht. Der größte Ausreißer hatte eine Abweichung von 38.68 Pixeln. Dabei ist zu beachten, dass die vorliegenden Ground Truth Daten von Hand festgelegt wurden, das Rosettenzentrum unter Umständen aber nicht exakt bestimmbar ist.

Abbildung 8.1: Von `final_net` geschätzte Rosettenzentren mit Ground Truth

9 Fazit und Ausblick

Die Fragestellung dieser Arbeit, inwiefern künstliche neuronale Netze das Regressionsproblem der Positionsschätzung bearbeiten können, wurde anhand von Rosettenzentren untersucht. Dafür wurde das Problem in zwei Teilprobleme unterteilt.

Das Lokalisieren des gesuchten Features wurde mittels Faltungsschichten realisiert, wobei auffiel, dass im zeitlich stark beschränkten Rahmen dieser Arbeit kein ausreichendes Training für die Ausbildung von nutzbaren Filtern möglich war. Deshalb wurde ein Ansatz verfolgt, der auf eine vortrainierte erste Schicht setzt, was in der Literatur häufig als Transfer Learning bezeichnet wird.

Das Umrechnen der Lokalisierungsinformation in Koordinaten, welches durch eine einzelne Fully Connected Schicht realisiert werden sollte, wurde durch eine Regularisierung der Trainingsdaten praktikabel. Eine detaillierte Analyse des Lernverhaltens mittels der Fouriertransformation war hier ein wichtiger Aspekt.

Das Netz, das aus der Kombination der beiden Teilnetze entstand, erreicht eine durchschnittliche Abweichung der Schätzung von der tatsächlich vorliegenden Position von 8.46 Pixeln. Auf Bildern der Größe 256×256 entspricht das einer Abweichung von 3.30%. Da allerdings Ausreißer auftreten, deren Schätzung unbrauchbar ist, ist das Netz im jetzigen Zustand nicht anwendbar. Anhand des Fehlerverlaufs beim Training des Netzes, das die Lokalisierung vornimmt, ist allerdings zu erkennen, dass dieses Training noch gewinnbringend fortgeführt werden kann. Eine Erweiterung der Trainingsdaten um weitere Bilder, sowie ein länger andauerndes Training, könnten die Güte der Lokalisierung und somit auch die Vorhersagen des resultierenden Netzes deutlich verbessern.

Viele in Keras vorhandene und durch den Programmierer einstellbare Parameter sind in dieser Arbeit nicht betrachtet worden. Diese können jedoch interessante Auswirkungen auf die Trainings- und Vorhersageperformance neuronaler Netze haben. Beispielhaft wäre hier der gewählte Optimierer für den Backpropagation Algorithmus zu nennen. Weiterführende Experimente könnten somit das Training beschleunigen. Dies würde Tests mit verschiedenen Netzwerkarchitekturen vereinfachen.

Dass neuronale Netze herkömmliche Algorithmen bei Klassifikationsproblemen schlagen, ist mittlerweile bekannt. Diese Arbeit hat gezeigt, dass auch Regressionsprobleme durch neuronale Netze bearbeitet werden können, wobei dies durch rein datengetriebenes Training bisher nur mit noch unzureichender Genauigkeit möglich ist. Die Trennung des translationsinvarianten und -varianten Teils des Netzes kann hier einen bedeutenden Geschwindigkeitsvorteil beim Training bewirken. Durch die künstliche Generierung der Trainingsdaten für den translationsvarianten Teil ist außerdem ein Training auf ein weites Spektrum an Möglichkeiten durchführbar.

10 Abbildungsverzeichnis

11 Literatur

[Gir+13] Ross B. Girshick u. a. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* abs/1311.2524 (2013). URL: http://arxiv.org/abs/1311.2524.

[Onlh] *Wikipedia: Schätzung*. [Online; Stand 10. August 2017]. URL: https://de.wikipedia.org/wiki/Sch%C3%A4tzung.

[GMT15] Mario Valerio Giuffrida, Massimo Minervini und Sotirios Tsaftaris. "Learning to Count Leaves in Rosette Plants". In: *Proceedings of the Computer Vision Problems in Plant Phenotyping (CVPPP)*. Hrsg. von H. Scharr S. A. Tsaftaris und T. Pridmore. BMVA Press, 2015, S. 1.1–1.13. ISBN: 1-901725-55-3. DOI: 10.5244/C.29.CVPPP.1. URL: https://dx.doi.org/10.5244/C.29.CVPPP.1.

[Aks+15] Eren Erdal Aksoy u. a. "Modeling leaf growth of rosette plants using infrared stereo image sequences". In: *Computers and Electronics in Agriculture* 110 (2015), S. 78 –90. ISSN: 0168-1699. DOI: http://dx.doi.org/10.1016/j.compag.2014.10.020. URL: http://www.sciencedirect.com/science/article/pii/S0168169914002816.

[Onlb] *Amazon Lex*. [Online; Stand 30. August 2017]. URL: https://aws.amazon.com/de/lex/.

[SKP15] Florian Schroff, Dmitry Kalenichenko und James Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering". In: *CoRR* abs/1503.03832 (2015). URL: http://arxiv.org/abs/1503.03832.

[SLD16] Evan Shelhamer, Jonathan Long und Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *CoRR* abs/1605.06211 (2016). URL: http://arxiv.org/abs/1605.06211.

[KH15] Yuanzhi Ke und M. Hagiwara. "A natural language processing neural network comprehending English". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. 2015, S. 1–7. DOI: 10.1109/IJCNN.2015.7280492.

[MP43] Warren S. McCulloch und Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), S. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: http://dx.doi.org/10.1007/BF02478259.

[Roj93] Raúl Rojas. *Theorie der neuronalen Netze - Eine systematische Einführung*. Springer-Verlag Berlin Heidelberg New York, 1993. ISBN: 3-540-56353-9.

[GBB11] Xavier Glorot, Antoine Bordes und Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*. Apr. 2011. URL: http://jmlr.csail.mit.edu/proceedings/papers/v15/glorot11a/glorot11a.pdf.

[GW08] Rafael C. Gonzales und Richard E. Woods. *Digital Image Processing*. Pearson Education, Inc., 2008. ISBN: 0-13-505267-X.

[Onld] *Faltungsmatrix*. [Online; Stand 22. August 2017]. URL: https://de.wikipedia.org/wiki/Faltungsmatrix.

[Aph] Aphex34. *Typical CNN*. [Online; Stand 26. August 2017]. URL: https://de.wikipedia.org/wiki/Datei:Typical_cnn.png.

[KSH12] Alex Krizhevsky, Ilya Sutskever und Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Hrsg. von F. Pereira u. a. Curran Associates, Inc., 2012, S. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[Lee+11] Honglak Lee u. a. "Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks". In: *Commun. ACM* 54.10 (Okt. 2011), S. 95–103. ISSN: 0001-0782. DOI: 10.1145/2001269.2001295. URL: http://doi.acm.org/10.1145/2001269.2001295.

[Jä93] Bernd Jähne. *Digitale Bildverarbeitung*. Springer-Verlag Berlin Heidelberg New York, 1993. ISBN: 3-540-56926-X.

[Wea89] H. Joseph Weaver. *Theory of discrete and continuous fourier analysis*. John Wiley & Sons, Inc., 1989. ISBN: 0-471-62872-7.

[Onla] [Online; Stand 16. August 2017]. URL: https://de.wikipedia.org/wiki/Spektrale_Leistungsdichte.

[Cho+15] François Chollet u. a. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[Onlf] *Keras Documentation*. [Online; Stand 31. Juli 2017]. URL: https://keras.io/.

[Onlg] *TensorFlow Documentation*. [Online; Stand 01. August 2017]. URL: https://www.tensorflow.org/.

[Onlc] *CNTK Documentation*. [Online; Stand 01. August 2017]. URL: https://docs.microsoft.com/en-us/cognitive-toolkit/.

[Min+15] Massimo Minervini u. a. "Finely-grained annotated datasets for image-based plant phenotyping". In: *Pattern recognition letters* 81 (2015), 80–89. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2015.10.013. URL: http://juser.fz-juelich.de/record/276445.

[Onle] *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*. [Online; Stand 27. August 2017]. URL: `http://www.image-net.org/challenges/LSVRC/`.

[SZ14] K. Simonyan und A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014).