HTWK Leipzig
Fakultät für Mathematik, Informatik und Naturwissenschaften
Institut für Informatik

Masterarbeit

Objekterkennung und Zuordnung im Roboterfußball (SPL) - Tore

Leipzig, 22. Mai 2017

vorgelegt von
Frank, Büchel
Matrikelnummer: 64766
Studiengang Master Informatik

Betreuende Professorin: Prof. Dr. rer. nat. Sybille Schwarz
Zweitgutachter: M. Sc. Thomas Reinhardt

Abstract

In der *Standard Plattform League (SPL)*, treten Teams aus autonom agierenden humanoiden Robotern im Fußball gegeneinander an. Insbesondere für die Orientierung sind Torpfosten ein wichtiges Objekt auf dem Spielfeld. Durch Algorithmen der Bildverarbeitung kann ein Bild auf Merkmale eines Pfostens untersucht werden. Bedingt durch die Merkmalähnlichkeit mit anderen Bildobjekten, können die Pfosten nicht eindeutig erkannt werden. Eine Zuordnung der erkannten Objekte in die Klasse Tor stellt eine Herausforderung dar. Für diese Aufgabe wurde ein künstliches neuronales Netz als Klassifikator konstruiert. Durch manuelle Kennzeichnung der Pfosten werden positive Beispiele für Klassifikationsaufgaben erzeugt. Die Resultate der Lernaufgaben zeigen, dass eine Zuordnung der hypothetischen Tor-Objekte mit einer Trefferquote von bis zu 60 % und Präzision von 90 % möglich ist.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

> *By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup.* [14]

Dies ist die offizielle Mission des RoboCup Verbandes.

RoboCup hat zum Ziel, auf ansprechende, zugleich herausfordernde Weise, eine Unterstützungsplattform für Robotik und künstliche Intelligenz zu bieten [14].

Um die Mission zu erfüllen, setzt der Verband immer wieder neue Meilensteine. In Abhängigkeit der zugrunde liegenden Roboter-Plattform definiert der Verband Umweltbedingungen, die während eines Spielablaufs zu bewältigen sind. Dabei ist deren stetige Erweiterung als Annäherung an die Zielsituation zu verstehen, also den offiziellen *FIFA* Regeln zu genügen. Mit jeder Änderung entstehen neue Herausforderungen. Entstandene Lösungen müssen überdacht, sogar teils verworfen werden. Ideen, Lösungen und Ansätze der unterschiedlichen Disziplinen des *RoboCup - Robot Soccer* Bereiches ermöglichen eine enorme Vielfalt an Strategien der Informationsverarbeitung unterschiedlichster Quellen.

Ebenso stellt sich die Standard Plattform League (*SPL*) immer schwierigeren Aufgaben. Der Verband schreibt in dieser Liga die zu verwendende Robotertechnologie vor. Alle zu einem Turnier registrierten Teams nutzen die gleiche, humanoide *NAO*-Roboterplattform. Jede technische Modifikation an der Plattform ist untersagt. Diese Arbeit steht im Kontext der *SPL*, nutzt die Resourcen des *NAO-H25* unter den Liga-Vorgaben des Verbandes.

Motivation

Um den Spielablauf zu gewährleisten, müssen alle in der Spielumwelt definierten Markierungen und Objekte bekannt sein. Dazu gehören Feld, Begrenzungslinien, Ball, befreundete sowie feindliche Roboter und Tore.

Nach wie vor liegt die Konzentration auf flüssigen Bewegungen der Roboter auf dem Spielfeld sowie die Erkennung des Balls. Da die Roboter autonom agieren sollen, müssen sie beispielsweise in der Lage sein, nach einem Sturz selbständig aufzustehen. Alle Entscheidungen im Spielverlauf, müssen vom Roboter selbständig getroffen werden. In letzten Arbeiten des Nao-Teams der HTWK Leipzig entstand ein Modul zum Aufbau einer Spielstrategie, welche die Verfolgung von Zielen steuern und den Ablauf verbessern soll. Die Erkennung der Tore auf dem Spielfeld ist nach wie vor eine Herausforderung. Primär bieten sie die Möglichkeit zur Orientierung der Roboter. Sekundär sind sie das Ziel für den Ball.

Tatsache ist, dass ein Erreichen des gesetzten Ziels, eine Partie zu gewinnen, im Wesentlichen von einer anderen Partei abhängig ist. Vorteile gegenüber dem gegnerischen Teams sind folglich notwendig. Ablauf der Bewegungen, Informationsqualität über Umwelt und Spielablauf sowie die Lokalisierung eines Roboters können maßgeblich zum Sieg beitragen. Bildgebung ist für die Plattform wichtigste Informationsquelle zur Untersuchung der Umwelt. Im besten Fall sollte eine Informationsgewinnung zu jedem verfügbaren Bild erfolgen. Daher ist eine effiziente, robuste und schnelle Abarbeitung der Prozessschritte obligatorisch und muss Echtzeitkriterien genügen.

Diese Arbeit setzt sich zur Aufgabe, die Registrierung der Tor-Objekte maßgeblich zu verbessern. Dabei werden Torpfosten registriert, um als zugesicherte Information in die Lokalisierung einzufließen. Ein Algorithmus zur Pfostenerkennung schlägt Bildpositionen vor, die für ein oder mehrere Torpfosten sprechen. Neben der Erkennung ist die Zuordnung gefundener Objekthypothesen, also Vorschläge zu Torpfosten, in die Tor- bzw. Ablehnungsklasse die Hauptaufgabe. *Convolutional Neuronal Networks* stellen die Basis der Klassifikatoren. Durch Bildmarkierungen werden der Klassifizierungsaufgabe Grundtatsachen zur Verfü-

gung gestellt. Jedem Vorschlag soll auf Basis gelernter Tatsachen durch den Klassifikator eine Wahrscheinlichkeit zugeordnet werden, um eine Tor-Hypothese zu bestätigen oder abzulehnen.

2. Grundlagen

Für das Verständnis der folgenden Kapitel werden dem Leser zunächst Grundlagen vorgestellt. Dies betrifft eine Erläuterung einiger Elemente im Umfeld der *SPL*, im Speziellen die Tore. Abschnitt 2.2 bietet einen kleinen Einblick in Definition und Techniken der digitalen Bildverarbeitung. Anschließend werden die Themen *Maschinelles Lernen* und *Neuronale Netze* in den Abschnitten 2.3 und 2.5 umfangreicher ausgeführt.

2.1. Standard Platform League - SPL

Die *SPL* verlangt einheitliche Bedingungen für alle teilnehmenden Teams. Dadurch wird die technische Herausforderung minimiert, sowie Vorteile durch darin begründete Differenzen vermieden. Durch Vorgabe der Roboterplattform nutzen alle Teams gleiche Sensoriken, Bildgebungsverfahren und Hardware.

Auch das Spielfeld und am Spielgeschehen involvierte Objekte sind definiert. Zum Zeitpunkt der Erstellung dieser Arbeit wurde das Spielumfeld erneut deklariert. Im Wesentlichen wurde das Spielfeld, vormals ein einfacher grüner Teppichbelag, gegen einen Kunstoffrasen ersetzt. Dadurch wurden die bisherigen Bewegungsabläufe gestört. Eine Überarbeitung war notwendig, um sich der neuen Gegebenheit anzupassen. Tabelle 2.1 zeigt die wichtigsten Maße auf, die in dieser Arbeit zu Diskussionen herangezogen werden [1].

Objekt	Maße	Farbe
Pfosten	800 mm x 100 mm	Weiß, Grau oder Schwarz
Querlatte	100 mm x 150 mm	Weiß, Grau oder Schwarz

Tabelle 2.1.: Eine Übersicht der wichtigsten Maße, die in in dieser Arbeit Verwendung finden.

2.2. Digitale Bildverarbeitung

Ein Bild im digitalen Sinne ist ein Abbildung

$$B : cos \rightarrow col \tag{2.1}$$

mit der diskreten endlichen Menge aller Positionen $pos \subset \mathbb{N}$ und der diskreten, endlichen Menge aller Farben $col \subset \mathbb{N}$. Ein Bild ist meist als Rechteckgitter $\{0, ..., m_0\} \times ... \times \{0, ..., m_n\}$ definiert. Die Verarbeitung legt den Fokus auf die Untersuchung digitaler Bilder hinsichtlich bestimmter Merkmale. Merkmale können beispielsweise Flächen von bestimmter Intensität sein, aber auch Intensitätsunterschiede im Vergleich von benachbarten Bildpunkten.

2.2.1. Scanlines

Bei der Untersuchung digitaler Abbildungen werden bestimmte Positionen $p \in pos$ auf Intensitäten untersucht. Ein gängiges Verfahren für eine solche Abtastung sind vertikal oder horizontal positionierte Scanlines. Sie repräsentieren einen lokalen Teilausschnitt des Bildes. Oft genügt eine sequentielle oder auch Schrittweiten gesteuerte [1] Abtastung dieser Teilinformationen, um Merkmale festzustellen. Auf diese Weise können bestimmte Regionen ausgemacht werden, die im Interesse der Verarbeitungsaufgabe sind (engl.: *Region of Interest, ROI*). Eine solche Eingrenzung legt in der Regel den Fokus für einen späteren Verarbeitungsschritt fest.

[1] Beim Abarbeiten einer Struktur mit Index, z.B. einer Liste, wird statt jedem Element mit einer Schrittweite n nur jedes n-te Element betrachtet

2.2.2. Faltung - Konvolution

Eine Faltung $(f * g)(x)$ ist eine Gewichtung einer Funktion f durch eine Funktion g, wobei der Funktionswert $f(\tau)$ durch $g(x - \tau)$ gewichtet wird.

Auf $\mathbb{R}^n$ ist die eine Faltung definiert zu:

$$(f * g)(x) = \int_{\mathbb{R}^n} f(\tau)g(x - \tau)\mathrm{d}\tau \tag{2.2}$$

Im diskreten Fall eines Bildes ist das Integral auf einem Bildbereich durch folgende Summe darstellbar:

$$\sum_{p \in P, P \subseteq pos} B(p) \tag{2.3}$$

Nach 2.1 und 2.2 entspricht eine Faltung zweier Bilder B, B_σ an einer Position p:

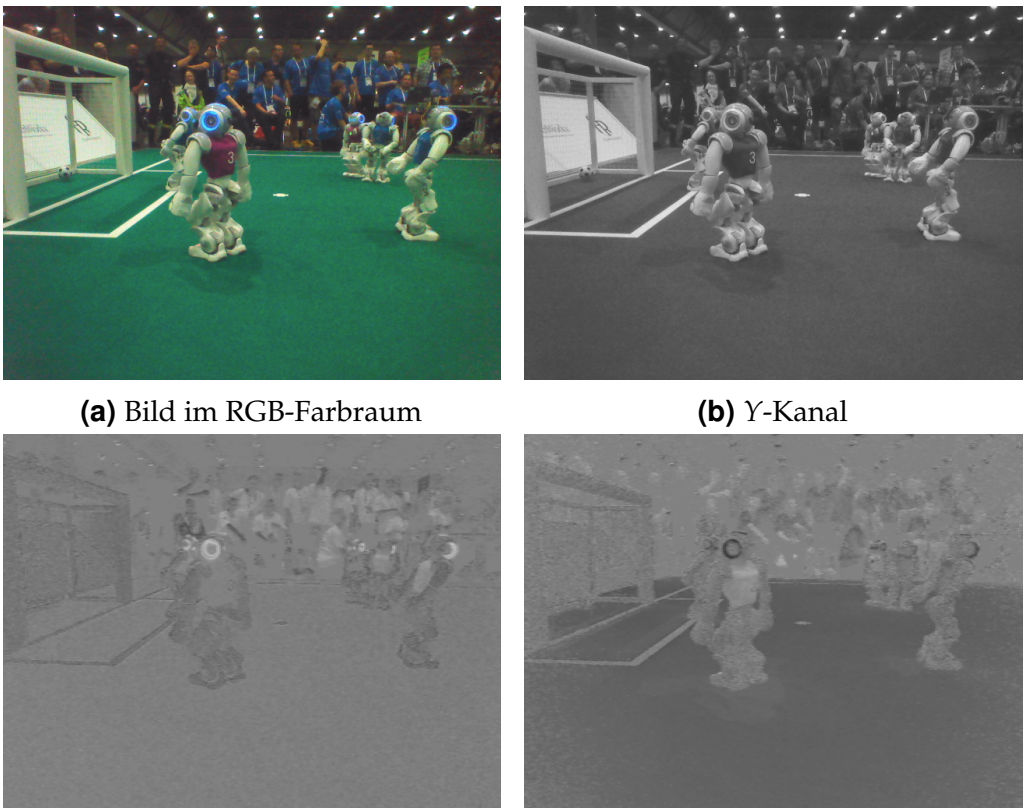
$$(B * B_\sigma)(p) = \sum_{p'} B(p')B_\sigma(p'), \quad p - \sigma/2 < p' \leq p + \sigma/2 \tag{2.4}$$

B' nennt man auch Faltungskern und σ die zugehörige Kerngröße. Ausgewählte Filter haben dabei eine besondere Funktion, wie zum Beispiel Weichzeichnung oder die eines Kantendetektors.

2.2.3. Farbmodell

Die Kamera eines *NAO* liefert Bilder im *YCrCb Helligkeits-Farbigkeits-Modell*. Es orientiert sich am natürlichen Sehsinn und der Farbwahrnehmung des Menschen, der Helligkeitsunterschiede besser erkennen kann als kleine Unterschiede im Farbton, Unterschiede im Farbton jedoch besser als in der Farbsättigung.

Das Modell umfasst die drei Kanäle *Y*, *Cb* und *Cr*. Durch den *Y*-Kanal wird die Helligkeit eines Bildpunktes in Graustufen wiedergegeben. Die Chrominanz-Kanäle *Cb* und *Cr* codieren ein Farbigkeitsmaß. *Cb* codiert die Abweichung von Grau hin zur Farbigkeit von blau und gelb, *Cb* entsprechend von grau zu rot und



(a) Bild im RGB-Farbraum **(b)** *Y*-Kanal

(c) *Cb*-Kanal **(d)** *Cr*-Kanal

Abbildung 2.1.: Die drei Kanäle des *YCbCr*-Farbmodells des Bildes (a), dem *Y*-Kanal (b) sowie den Kanälen *Cb* (d) und *Cr* (d)

türkis. Die Farben blau und gelb stehen sich wie die Farben rot und türkis komplementär gegenüber, schließen sich also gegenseitig aus. Abbildung 2.1 zeigt eine Beispielabbildung in den drei Kanälen des Farbmodells. Es ist deutlich zu erkennen, wie die Farbe blau im *Cb*-Kanal durch helle Graustufenintensitäten repräsentiert wird. Im Gegensatz dazu erscheinen die gelben Elemente dunkel. Analog wird im *Cr*-Kanal das grüne Spielfeld durch niedrigere Intensitäten repräsentiert als die roten Trikots der *NAOs*.

2.3. Machine Learning

Diese Disziplin beschäftigt sich mit der computergestützten Modellierung und Realisierung von Lernproblemen [6]. Häufigste Anwendungsgebiete sind adaptive Systeme, also solche die einer Selbstregulierung unterliegen, sowie jene zur Extraktion von Wissen aus großen Datenbeständen. Maschinelles Lernen ist daher ein Nachbargebiet des *Data Mining*. Dabei gibt es verschiedene Ansätze den Lernvorgang zu gestalten [11].

> *Lernen ist jeder Vorgang der ein System in die Lage versetzt, bei der zukünftigen Bearbeitung der selben oder einer ähnlichen Aufgabe diese besser zu erledigen.*

Weiterhin definiert Michalski (Entnommen [6]):

> *Lernen ist das Konstruieren oder Verändern der Repräsentation von Erfahrungen.*

Diese Definitionen beschreiben maschinelles Lernen: Ein Algorithmus repräsentiert durch seine Formulierung und Beschreibung Wissen; durch adaptive Anpassung seiner Parameter in Folge des Sammelns von Erfahrungen ist der Algorithmus in der Lage, seine Aufgaben besser zu erledigen. Die adaptive Anpassung kann durch drei Lernmethoden initiiert werden:

Supervised Learning Die Methode des überwachten Lernens beschreibt das Lernen aus einer Vorgabe von Beispielen. Dem Lernalgorithmus wird eine Menge von Objekten als Beispiel des funktionalen Zusammenhangs vorgegeben. Ein Exempel dazu ist die Menge von manuell als *Spam* und *Nicht-Spam* klassifizierten Emails. Durch die Vorgabe von Beispielobjekten, und damit deren Merkmale, ist ein Lernalgorithmus in der Lage die entsprechenden Klassen zu differenzieren. Auf ähnliche Weise kann durch Vorgabe diskreter Werte, z.B. aus einer Messung, eine zugrundeliegende Funktion approximiert werden (Abs. 2.4). Die Klassifizierung von Objekten und Annäherung an Funktionen sind die Hauptziele dieser Lernmethode. Algorithmische Vertreter sind neuronale Netze, Support Vektor Maschinen (*SVM*) und Entscheidungsbäume. Bei weiterem Interesse zu

letztgenannten Vertretern sei auf [6] verwiesen.

Unsupervised Learning Unüberwachtes Lernen verzichtet auf die Vorgabe von Beispielen. Durch diese deskriptive Methode werden unklassifizierte Daten auf markante Strukturen untersucht, die zu einer Klassifikation beitragen. Dabei sind die Strukturen meist lokal, d.h. sie lassen sich nicht in allen Objekten wiederfinden. Auf diese Weise werden Daten und Objekte auf unbekannte Zusammenhänge untersucht, weshalb hiesige Methoden im Gebiet Data Mining verwendet werden. Beispiele für diese Verfahren sind Clusteranalyse und Sub-Gruppen-Entdeckung.

Reinforcement Learning Beim Verstärkungslernen wird eine Art Lohn- bzw. Strafsystem verwendet, um den Lernprozess zu unterstützen. Hier liegt die Problematik der Auffindung einer optimalen Handlungsweise zugrunde. Speziell in der autonomen Robotik werden Methoden dieser Art untersucht, um sie zu befähigen, selbständig mit der realen Welt zu interagieren.

Im Rahmen dieser Arbeit soll aus Gründen der Komplexität des Themengebiets eine Beschränkung auf die Methode des überwachten Lernens genügen.

2.4. Supervised Learning - Überwachtes Lernen

Das Ziel dieser Lernmethode ist das Erlernen von Zusammenhängen aus einer vorher zusammengetragenen Menge an Daten. Die eingangs erwähnte Vorgabe von Beispielen des funktionalen Zusammenhangs kann man wie folgt genauer beschreiben. Beim *Funktionenlernen* geht man von einer unbekannten Funktion f aus, die jedem Beispielelement e aus der Beispielmenge E einen Funktionswert $y = f(e)$ zuordnet. Nun ist eine Hypothese, genauer eine Funktion h, gesucht, die für jedes $e \in E$ die Funktion f möglichst gut approximiert.

$$h(e) \approx f(e), e \in E \tag{2.5}$$

Viel mehr noch fordert man von h ein möglichst gut approximiertes Ergebnis für Eingaben, deren Zielwert noch unbekannt ist. Da man häufig daran interes-

siert ist, mit der Hypothese h Elemente aus der Eingabemenge X einer Klasse zuzuordnen, wird y auch allgemein Klassifikator genannt. So kann das *Funktionenlernen* wie folgt definiert werden.

Für eine Stichprobenmenge E aus der Instanzmenge X sei D die Wahrscheinlichkeitsverteilung auf X, und Y die Menge der möglichen Zielwerte. Aus der Menge aller möglichen Hypothesen H über X finde ein $h \in H$, sodass für $(x, y) \in X \times Y$ gilt $y = f(x)$ und $h(x) \approx f(x)$. Das heißt der Fehler $err_D(h, f)$ gemäß der Verteilung D gezogenen Instanzen aus X sei möglichst gering.

Als Gütemaß der Funktion h dient der *Trainingsfehler* (auch Verlustfunktion oder engl. Loss). Die Bezeichnung liegt nahe, da die Beispielmenge E auch als Trainingsmenge bezeichnet wird.

$$err_E(h) = \frac{1}{|E|} \sum_{(x,y) \in E} err(h(x), y) \tag{2.6}$$

Die Zielmenge Y kann hier von numerischer oder diskreter Natur sein. Im Fall der numerischen Ausprägung nennt man die Lernaufgabe *Regression*. In diesem Fall ergibt sich der Quadratfehler

$$err(h(x), y) = (h(x) - y)^2 \tag{2.7}$$

als Verlustfunktion. *Begriffslernen* nennt man die Lernaufgabe im diskreten Fall, wenn Y demnach von binärer Gestalt ist. Als *Klassifikation* bezeichnet man eine Lernaufgabe, wenn Y durch mehrere, wenige Werte bestimmt ist. Auch hier ist Y eine diskrete Zielmenge und die Aufgabe kann auf das Begriffslernen zurückgeführt werden. Es ergibt sich für den diskreten Fall die Fehlerfunktion

$$err(h(x), y) = \begin{cases} 0 & \text{falls } h(x) = y \\ 1 & \text{sonst} \end{cases} \tag{2.8}$$

Der Trainingsfehler liefert also ein Gütemaß über die Approximation der Hypothese an die Zielmenge Y, also wie gut h mit der unbekannten Funktion f

übereinstimmt. Jedoch ist nicht nur das Gütemaß bezüglich der Beispielmenge $E \subset X$ von Interesse. Interessant ist der Fehler der Hypothese bezüglich noch unbekannter Instanzen aus X. Schließlich geht man davon aus, dass unter h ein $x \in X$ mit einer ähnlichen Häufigkeit gewählt wird, wie ein Element e aus der Beispielinstanz E. So soll die Hypothese ein Ergebnis von ähnlicher Qualität liefern. Es gilt also die Annahme, dass die Wahrscheinlichkeitsverteilung D eine Auswahlwahrscheinlichkeit von Elementen aus X sowohl für Beispiele als auch zukünftige Instanzen angibt. Es ergibt sich somit der wahre Fehler

$$err_D = err(h(x), y), x \in E_D \tag{2.9}$$

Kann ein Lernverfahren nicht sicherstellen, dass die Hypothese eine Approximation von f nicht ausschließlich für Elemente aus E liefert, tritt eine Überanpassung ein. Das heißt die Annäherung an die Zielmenge Y für alle Beispielvorgaben ist von hoher Güte, weist also einen geringen Trainingsfehler auf. Jedoch wird das Ergebnis von f für zukünftige Instanzen mit einem wahren Fehler, auch Generalisierungsfehler, verfehlt. Dieses Problem kann durch Verwendung einer Validierungsmenge aufgezeigt werden (siehe Abschnitt 2.5.9).

2.5. Neurale Netze - NN

Obwohl die Untersuchungen zu Neuronalen Netzen auf Mitte des 20 Jahrhunderts zurück geht, erfahren neuronale Netze einen neuen Trend. Dieses Konstrukt erfreut sich der Nähe durch Analogien in unterschiedlichen Forschungsdisziplinen, wie beispielsweise der Lernpsychologie und der Spracherwerbsforschung [15]. Mit dem Grundstein der Untersuchung wurde die Fähigkeit zur Realisierung einfacher logischer Funktionen, wie beispielsweise *XOR* [2] festgehalten. Dabei wurde das Neuron in seiner einfachsten Form mit der *Propagierungsfunktion* und der *Aktivierungsfunktion* ϕ als Basis-Datenverarbeitungselement definiert. Eingabesignale werden der Propagierungsfunktion zugeführt, die eine Aktivierung des Neurons erzeugt. [8] Wenige

[2] Exklusiv oder

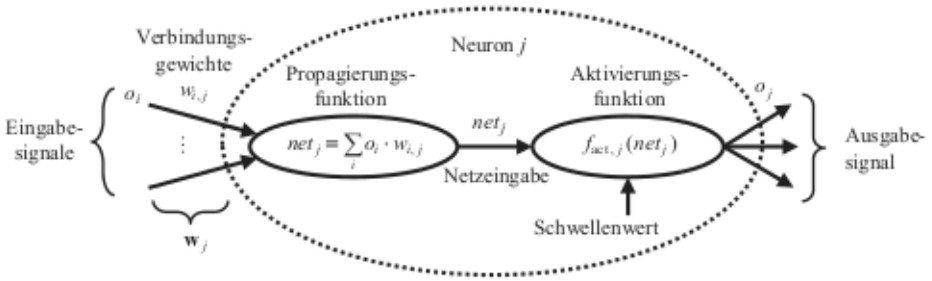


Abbildung 2.2.: Aufbau eines Neurons

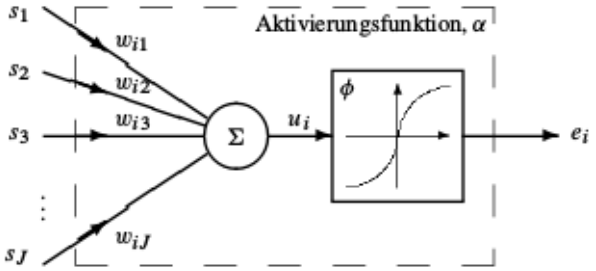


Abbildung 2.3.: Schematische Darstellung der Aktivierungsfunktion eines Neurons. Eingänge (bzw. Stimuli) s_j, Übertragungsgewichte w_j des Neurons i, Summation Σ, Potential $u_i = \sum_{j=1}^{J} w_j s_j$, statische Nicht-Linearität ϕ, Erregung $e_i = \phi(u)$

Jahre später wurde durch *Hebb* in der Psychologie die These aufgestellt, das Aktivierung oder Hemmung von Synapsen durch das Produkt von Pre- und Post-Synaptischer Aktivität berechenbar ist. So erfährt das Modell des Neurons eine biologisch, psychologische Interpretation.

2.5.1. Das Neuron

Grundlegend wird ein Neuron beschrieben durch Eingänge mit Übertragungsgewichten und einer Aktivierungsfunktion. Abbildung 2.2 [8] zeigt den schematischen Aufbau eines Neurons, Abbildung 2.3 [6] geht schematisch auf die Aktivierungsfunktion ein.

Das Neuron erfährt durch das Eingangssignal und den korrespondierenden

Übertragungsgewichten ein Potential. Die Funktion $\phi(u)$ aktiviert das Neuron in Abhängigkeit des vorhandenen Potentials, auch Erregung genannt. Die Übertragungsgewichte $w_{i,j}$ bestimmen das Potential im Wesentlichen, somit auch die Erregung. Um eine bestimmte Aktivierung eines Neurons zu erzeugen, können die Übertragungsgewichte dementsprechend parametrisiert werden. Ausgehend von einem Neuron ergibt sich die Erregung:

$$e := \phi\left(\sum_j w_j \, s_j\right) = \phi\left(\mathbf{ws}\right) \tag{2.10}$$

2.5.2. Netzarchitektur

Um komplexere, nichtlineare Optimierungsprobleme zu bewältigen, werden Neuronen in Netzwerken verknüpft. Die so entstehenden Netzwerke unterscheiden sich neben der Neuronenanzahl durch die Richtung der Signalanbindung. Ein Neuronales Netz besteht aus einer Menge von Knoten, den Neuronen, und einer Menge gerichteter Kanten. Die gerichteten Kanten des Netzgraphen deuten in Propagierungsrichtung der Erregungssignale und liefern eine Ordnung der Neuronenaktivierung [3]. Weiter entsprechen die gewichteten Kanten den Übertragungsgewichten der Stimuli zur Potentialregulierung. Erregungen können auch als Stimuli zurückgeführt werden, wodurch sogenannte Rekurrenz-Netze entstehen. Diese finden in dieser Untersuchung jedoch keine Anwendung und werden daher nicht näher betrachtet.

Multi Layer Perzeptron - MLP

Ein Neuronales Netz wird in seiner Architektur durch Schichten (engl.: *Layer*) beschrieben und deshalb auch *Multi Layer Perzeptron* genannt. Die einfachste Ausprägung eines Neuronalen Netzes besteht aus zwei Schichten. Eingangssignale werden von der Eingabeschicht direkt an die Ausgangsschicht propagiert. Die Eingabeschicht dient demnach lediglich als Verteilerschicht. Erweiterte Netze besitzen weitere Layer zwischen Ein- und Ausgabeschicht. Da die Erregun-

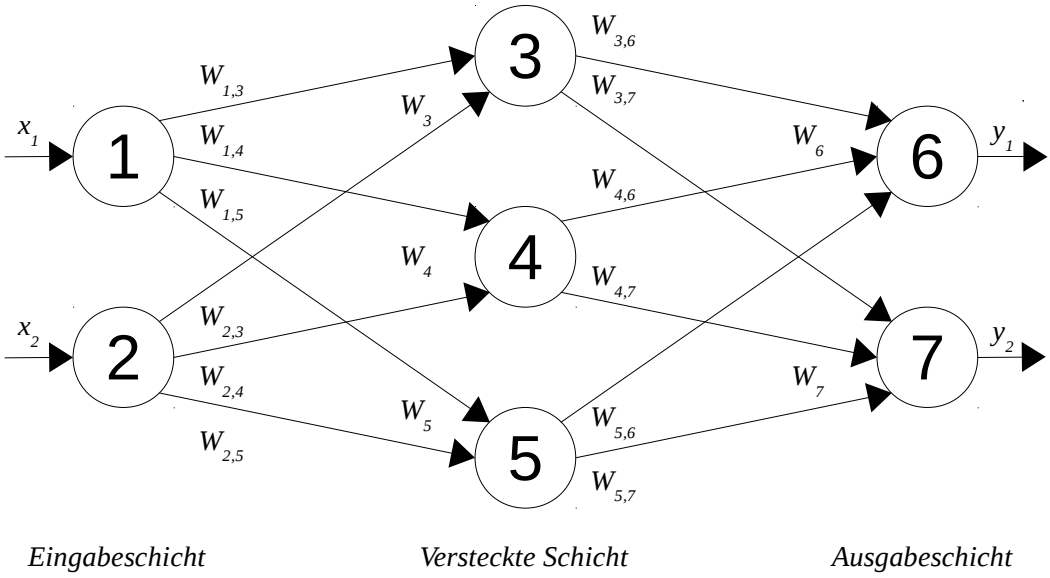


Abbildung 2.4.: Multi Layer Perceptron mit den zwei Eingabe-Neuronen, drei Neuronen in der versteckten Schicht und zwei Ausgabe- Neuronen

gen von Neuronen eines solchen Layers nach außen nicht sichtbar, sondern gewissermaßen versteckt sind, ergibt sich die Bezeichung *versteckte Schicht* (engl. *Hidden Layer*). Zur genauen Architekturbezeichnung wird jeder Layer durch die Anzahl der Neuronen angegeben. Sind alle Neuronen eines Layers mit jedem Neuron der Folgeschicht durch eine Kante verbunden, so nennt man diese Verbindungsart *Fully Connected*. Zur Veranschaulichung siehe Abbildung 2.4. Im Zusammenhang mit *Convolutional Neural Networks*, kurz *CNN* werden *MLP* als letzte Stufe verwendet (Abs. 2.5.3).

Neuronen können für binäre Klassifikationsaufgaben verwendet werden. Verschaltet in Netzen werden sie in der Praxis für Aufgaben wie Klassifikation in K-Klassen und Funktionsapproximation verwendet (Abs. 2.5.4).

2.5.3. Convolutional Neuronal Networks (CNN)

Die Hauptaufgabe dieses Netzwerktyps liegt in der Faltung von Eingangssignalen. Daher werden sie in der Verarbeitung und Klassifikation von Reizen mit räumlichen und merkmalsbasiertem Kontext verwendet. Sie dienen Lernverfahren in der Signalverarbeitung, Sprachverarbeitung und vor allem in der Bildverarbeitung als Basis (Abs.3). Wegen der Optimierung auf die Erfassung von Signaleigenschaften, sind sie Gegenstand dieser Arbeit. Da die Eingangsdaten durchaus von hoher Dimension bestimmt sind [3], werden Neuronen dieses Layers im Gegensatz zum *MLP* nicht *Fully Connected* verbunden. Die räumliche Eigenschaft von Bilddaten kann in Faltungsschichten ausgenutzt werden, um die Anzahl der Neuronenverbindungen deutlich zu reduzieren. Den im Folgenden genannten Schichten werden unterschiedliche Aufgaben zu teil und finden oft wiederholte Anwendung [3]. Durch Aufteilung in funktionale Blöcke und deren beliebig tiefe Anordnung, verleihen den auf *CNN* aufbauenden Lernaufgaben daher auch den Begriff *Deep Learning*.

Convolution Layer

Die Faltung ist in der Bildverarbeitung ein bewährtes Mittel, um räumliche Eigenschaften zu detektieren. Gewichte dieser Schichten werden als Filter oder Kern bezeichnet und verleihen diesem Typ Layer den Namen. Merkmale eines Bildbereiches sind nicht zwangsläufig auf einen beliebigen anderen Bildbereich übertragbar. So kann eine Objektkante auf einem Teil eines Bildes detektiert werden, in einem anderen nicht. Jedoch ist das Auftreten dieser Eigenschaft in anderen Bereichen nicht generell ausgeschlossen. Aufgrund dieser Tatsache sind Neuronen dieser Ebene für einen dedizierten räumlichen Bereich des Eingangsreizes zuständig. Es genügt, räumliche Teile des Bildes als Reizsignal an zuständige Neuronen zu leiten, anstatt alle oder ein Einzelnes zu stimulieren. Ein erlernter Faltungskern kann damit trotz separatem räumlichen Zuständigkeitsbereich jedem Neuron als Potentialparameter dienen. So ist außerdem gesichert, dass ein erlerntes visuelles Konzept, repräsentiert durch einen Faltungs-

[3] man verdeutliche sich die Anzahl von $8.44935 \cdot 10^{11}$ an notwendigen Verbindungen für ein *RGB*-Bild in 640×480 Bildpunkten Auflösung

kern, auf jedem Bereich des Bildes geprüft werden kann. Außerdem ergibt sich für die Faltungsoperation die Möglichkeit zur Parallelisierung. Die Neuronen dieser Schicht dienen also dem Erlernen verschiedener visueller Konzepte. In der Regel wird für diese Schicht eine Vielzahl von Faltungskernen von einheitlicher Kerndimension konfiguriert, um so mehrere Merkmale eines Bildes zu erlernen.

Striding, Subsampling und Pooling

Unter den Faltungsschichten und der daran gebundenen lokalen Vernetzung der Neuronen geht der globale Kontext von Karteneigenschaften verloren. Um dieser Tatsache Rechnung zu tragen, dienen die folgenden Schichten zur Reduktion und Aggregation der Merkmalskartierung.

Subsampling Dieser Layer reduziert die Dimension der Eingangsreize aus einer vorausgegangen Schicht auf eine feste kleinere Größe. So wird nur jedes $p-$te Reizsignal durch das Netz propagiert. Eine durch Convolution entstandene Merkmalskarte wird durch Skalierung reduziert.

Stride Hier wird eine Faltung auf definierten Raster punkten mit einem Versatz von s durchgeführt. Auch diese Technik führt zur Reduktion einer Merkmalskarte. Subsampling und anschließende Faltung liefern das gleiche Resultat wie Anwendung von Stride für $p = s$.

Pooling Hier wird neben der Reduktion auch eine Aggregation erreicht. Üblicherweise werden die Ausprägungen *Max-Pooling* und *Average-Pooling* verwendet. Der Pooling-Faktor k (auch Kernel der Dimension $k \times k$ genannt) erwirkt für die Eingabeelemente $x_1, ..., x_n$ und einem Stride s (dtsch.: Schritt) im eindimensionalen Fall eine Ausgabe $y_1, .., y_m$. Max-Pooling wie folgt definiert [3]:

$$m = \left\lceil \frac{n}{s} \right\rceil \tag{2.11}$$

$$y_i = max(x_{s(i-1)-v}),\ i = 1, ..., m,\ v \in 0, ..., k-1 \tag{2.12}$$

Die Variante *Average-Pooling* ist definiert durch:

$$y_i = \frac{1}{k} \sum_{v \in 0,\ldots,k} \left(x_{s(i-1)-v} \right) \tag{2.13}$$

Abbildung 2.5 zeigt das Ergebnis der Anwendung von Max- und Average-Pooling auf eine Eingabe. Es gibt auch neue Ansätze für diese Schicht, z.B. Max-Pooling-Position, MPP (siehe Kapitel 3, Abs.3.3).

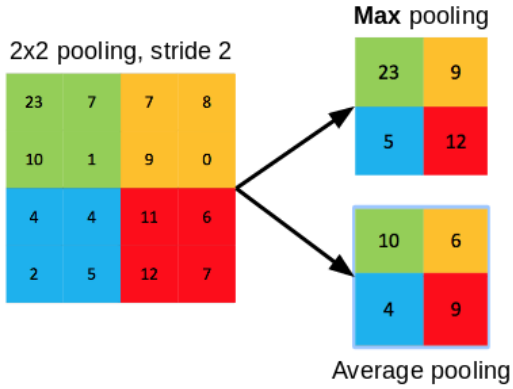


Abbildung 2.5.: Beispiel für eine Anwendung von max-pooling und avg-pooling (r.) auf eine Eingabe(l.)

Architektur

In State of the Art CNN werden Faltungsschicht und Pooling-Layer alternierend angeordnet. Die Faltungsschicht wird auch als Filter-Bank bezeichnet [9]. Durch die häufig tiefe Verkettung dieser Schichten findet man diese Netzwerke auch unter der Bezeichnung Deep Neural Network *DNN*.

In der Literatur ist die Architekturangabe nach Schema *[NcF][-NcF]*[-Nf]** zu finden.

NcF Benennt einen convolutional layer mit N Kernen und einem Faltungskern der Dimension $F \times F$

Nf Einen Fully Connected Layer mit N Neuronen.

2.5.4. Lernaufgaben neuronaler Netze

Wie bereits zum Abschluss des Abschnitts 2.5.2 kurz erwähnt, werden neuronale Netze in der Praxis für Lernaufgaben wie Klassifikation und Approximation von Funktionen oder Operatoren verwendet. Eine weitere Lernaufgabe ist die Datenrepräsentation durch *Merkmalskarten* (engl.: Feature Maps). Die Klassifikations- und Approximationsaufgaben werden in den folgenden Abschnitten näher erläutert. Eine Erläuterung zu Netzwerken für die Erzeugung von Merkmalskarten erfolgt in Abschnitt 2.5.3.

Klassifikation

Logische und semilineare Neuronen sind lineare Klassifikatoren. Für logische Neuronen werden schaltende, diskrete Aktivierungsfunktionen verwendet.

$$\phi(u) = \begin{cases} 0 & u < 0 \\ 1 & u \geq 0 \end{cases} \qquad (2.14) \qquad \phi(u) = \begin{cases} -1 & u < 0 \\ 1 & u \geq 0 \end{cases} \qquad (2.15)$$

Semilineare Neuronen sind durch kontinuierliche Abbildungen $\phi : \mathbb{R} \to \mathbb{R}$ mit einer Kennlinie definiert. Sie sind in der Lage, eine Menge von Stimuli in zwei Klassen zu unterteilen. Mit einer logischen Aktivierungsfunktion, z.B. jene aus Gleichung 2.14, übernimmt das Potential $u = \sum_i w_i s_i$ eine Diskriminanzfunktion und ist linear abhängig vom Eingabevektor $\mathbf{s} = (s_1, ..., s_n)^T$. Setzt man $s_0 = 1$ ergibt sich durch den Schwellwert $-w_0$ die Hyperebene

$$y = \sum_{i=1}^{n} w_i s_i - w_0 = \mathbf{w}^T \mathbf{s} = 0 \qquad (2.16)$$

welche den Raum der Eingabestimuli in zwei Klassen unterteilt [6]. Das Neuron

erfährt also Erregungen, sodass

$$y(\mathbf{s}) = \begin{cases} 0 & \mathbf{s} \notin k \\ 1 & \mathbf{s} \in k \end{cases} \tag{2.17}$$

Um dieses Konzept auf K Klassen zu erweitern, kann für jede Klasse eine separate Diskriminanzfunktion

$$y_k(\mathbf{s}) = \mathbf{w}_k^T \mathbf{s} \tag{2.18}$$

verwendet werden, sodass die Klasse i getrennt ist von Klasse j durch die Hyperebene y mit:

$$y = (\mathbf{w}_j - \mathbf{w}_i)^T \mathbf{s} - (\mathbf{w}_{j0} - \mathbf{w}_{i0}) \tag{2.19}$$

Klassen, die nicht durch lineare Hyperebenen trennbar sind, können auf diese Weise nicht bestimmt werden. Der zugrundeliegende Raum der Eingangsvektoren kann jedoch durch Transformation auf einen anderen Merkmalsraum transformiert werden [6].

$$L(\mathbf{s}) = (\varphi_1(\mathbf{s}), ..., \varphi_m(\mathbf{s}))^T \tag{2.20}$$

So entsteht die verallgemeinerte Diskriminanzfunktion:

$$y_k(\mathbf{s}) = \sum_{j=1}^{m} \mathbf{w}_{k,j} \varphi_j(\mathbf{s}) + \mathbf{w}_{k0} = \sum_{j=0}^{m} \mathbf{w}_{k,j} \varphi_j(\mathbf{s}) = \mathbf{w}_k^T \, L \tag{2.21}$$

Die Verallgemeinerung beruht auf dem Theorem zur Separierbarkeit von Datensätzen. Demnach nimmt die Wahrscheinlichkeit einen Raum in Klassen zu unterteilen mit der Dimension des Raumes zu. Dazu sei nach Görz et al. [6] auf besagtes Theorem von Cover [2] aus dem Jahre 1965 verwiesen. Eine Klassifizierung in K Klassen kann durch ein neuronales Netz mit K Neuronen in der Ausgabeschicht umgesetzt werden. Abschnitt 2.5.2 erläutert einige Architektur-

ansätze neuronaler Netzwerke.

2.5.4.1. Approximation

Diese Aufgabe erfordert eine stetige Transformation eines Erregungsvektors in einen Anderen. Im Falle diskreter Erregungsvektoren approximiert man Funktionen, im kontinuierlichen Fall Operatoren. Gleichung 2.21 stellt bereits ein Beispiel für eine allgemeine Funktionsapproximation dar. In der Regel werden dazu mehrschichtige Perzeptronen (engl.: *Multi Layer Perceptron - MLP*) verwendet (Aufbau neuronaler Netze bzw. *MLP* siehe Abschnitt 2.5.2). Nach dem *Kolmogorov-Arnold representation Theorem* [17], auch Superpositionstheorem genannt, kann eine multivariante, kontinuierliche Funktion f dargestellt werden durch die Summe

$$f(\mathbf{x}) = f(x_1, ..., x_n) = \sum_{i=0}^{2n} \Phi_i \left(\sum_{j=1}^{m} \phi_{i,j}(\mathbf{x}) \right) \tag{2.22}$$

Ein *MLP* mit n Eingangsstimuli, $2n + 1$ semilinearer Neuronen in der mittleren Schicht und m Neuronen in der Ausgabeschicht ist dann eine Anwendung dieses Theorems für eine Funktion $f : [0, 1]^n \rightarrow \mathbb{R}^m$. Mit einem geeigneten Lernverfahren, wie beispielsweise *Backpropagation*, (Abschnitt 2.5.5) kann f entsprechend approximiert werden.

2.5.5. Überwachtes Lernen mit neuronalen Netzen

Im Kontext des maschinellen Lernen ist mit dem Finden optimaler w_i für bekannte Erregungen $y_i \in E$ die Lernaufgabe definiert. Diese kann bereits durch ein einzelnes Neuron erfolgen. Der Einfachheit halber wird der Lernprozess an einem Neuron aufgezeigt.

Mit 2.10 ist die Funktion ϕ eine Hypothese, welche die unbekannte Funktion f approximiert. Eine geeignete Fehlerfunktion ist beispielsweise der Quadratfehler

$$J = (\phi(\mathbf{s}) - y)^2 \tag{2.23}$$

welcher Auskunft gibt, inwieweit durch Beispiel-Stimuli ein Potential erzeugt wird, das die erwartete Erregung hervorruft. Das Finden der optimalen Parameter durch Anpassung unterliegt einigen Fallstricken.

2.5.6. Gradientenabstieg

Das Gradientenabstiegsverfahren (engl. *Gradient Descent*) ist eine Methode zur Realisierung eines Lernverfahrens. Man kann es wie folgt zusammenfassen [6]:

Die Parameter der Hypothese, also die Gewichte des Neurons, sind dann optimal, wenn J minimal ist. Betrachtet man den Anstieg von J, dann zeigt sich das Optimum, bedingt durch die gewählte Fehlerfunktion, im lokalen Minimum, hier am Scheitelpunkt der Parabel. Konsequenterweise wird es also mit Richtungswahl zum Scheitelpunkt erreicht.

$$\nabla w_i = -\lambda \frac{\partial J}{\partial w_i} \tag{2.24}$$

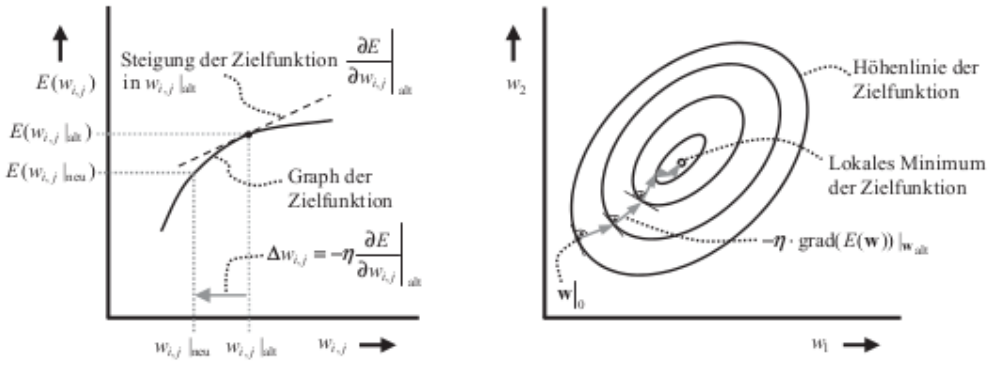
Das Update der Gewichte ergibt sich zu

$$w_i = w_i - \nabla w_i \tag{2.25}$$

Für ein kompakteres Ergebnis bezüglich der Ableitung des Quadratfehlers dient der Vorfaktor $1/2$. Dann ergibt sich der Gewichtegradient zu:

$$\nabla w_i = -\lambda \frac{1}{2} \frac{\partial J}{\partial w_i} = -\lambda \frac{1}{2} \frac{\partial}{\partial w_i} (\phi(\mathbf{s}) - e_i)^2 = -\lambda (\phi(\mathbf{s}) - e_i) w_i \tag{2.26}$$

Durch die Gestalt von 2.25 zeigt sich die Herangehensweise der Approximation. Eine schrittweise Wiederholung und Rückführung der Abweichung führt zu einem abnehmenden Fehler und wird auch *Regression* genannt. Die Anzahl der Iterationen oder die Definition eines maximal zulässigen Fehlers bestimmt die



(a) Gradientenabstieg für einen Wichtungsparameter

(b) Gradientenabstieg für zwei Wichtungsparameter

Abbildung 2.6.: Prinzip des Gradientenabstiegs

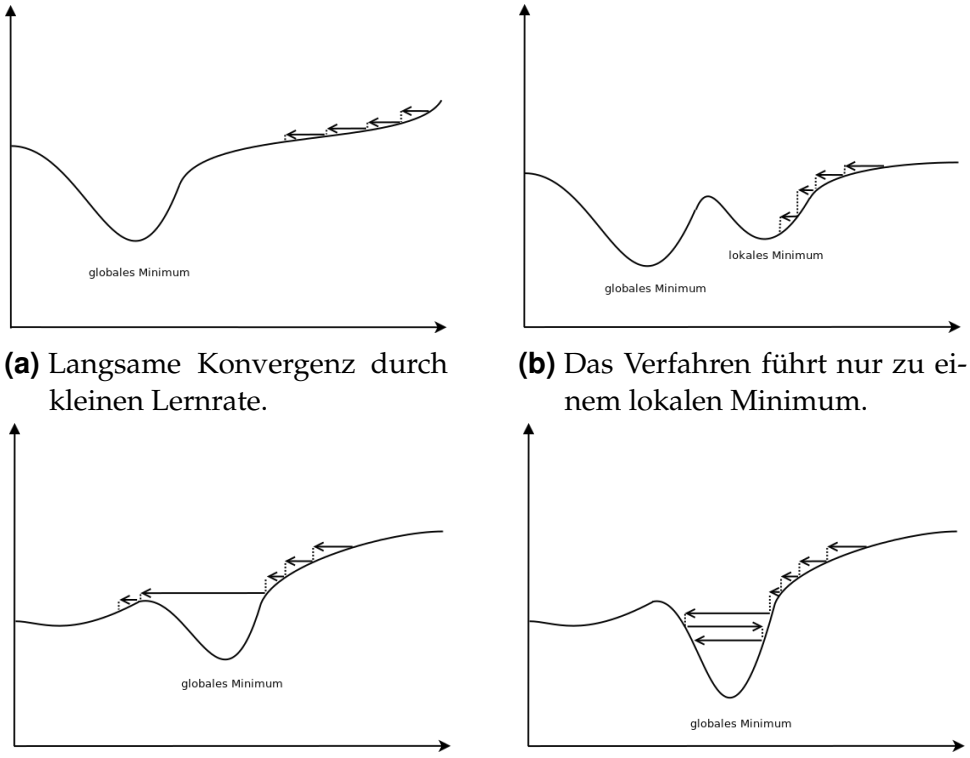
Güte der Annäherung an die unbekannte Funktion f. Abbildung 2.6 (entnommen aus [8]) veranschaulicht das Verfahren des Gradientenabstiegs.

Um dieses Lernverfahren auf ein komplettes Netzwerk von Neuronen zu übertragen, ist die Rückführung des Gradienten durch das gesamte Netz notwendig. Das populäre *Back Propagation* Verfahren beruht auf dem *Gradienten Abstiegsverfahren* und ist recht einfach umzusetzen. Grob umschrieben setzt es auf die Anwendung der Kettenregel der Differenzialrechnung, um die Gradienten durch das Netzwerk rückwärts zu propagieren.

2.5.7. Lernrate

Die Verwendung einer Rate, bzw. Schrittweite λ, kann bei geeigneter Wahl zu einer schnelleren Konvergenz des Lernverfahrens führen. Insbesondere dann, wenn das Terminieren des Trainings durch fixe Iteration gegeben ist, kann die Schrittweite zu einem besseren Ergebnis führen.

In Abschnitt 2.5.6 wurde sie bereits in die Gleichungen 2.24 und 2.26 einbezogen. Sie reguliert also nachträglich die tatsächlich verwendete Abstiegsschrittweite und beeinflusst die Aktualisierung der Neuronengewichte. Bewegt man sich zu einem Iterationsschritt auf einem flachen Plateau, ist der Gradient unter Umständen sehr klein. Diesbezüglich kann die Schrittweite zu einer schnelleren



(a) Langsame Konvergenz durch kleinen Lernrate.

(b) Das Verfahren führt nur zu einem lokalen Minimum.

(c) Überspringen des globalen Minimums.

(d) Oszilieren der Fehlerfunktion verlangsamt das Verfahren.

Abbildung 2.7.: Probleme durch setzen der Schrittweite mit der Lernrate

Konvergenz des Verfahrens beitragen. Ist jedoch das Gegenteil der Fall, kann mit dem Update der Neuronengewichte durch einen großen An- bzw. Abstieg an einem Punkt der Kostenfunktion eine Oszilation entstehen (Abb.2.7d). Die Konvergenz des Verfahrens wird verlangsamt, im schlimmsten Fall gar verhindert. In diesem Fall kann der Schritt in die richtige Richtung, jedoch mit kleineren Schritten, zu Konvergenz führen. Abbildung 2.7 stellt die Probleme des Verfahrens Gradientenabstieg dar. Die Beobachtung des Trainingsfehlers im Verlauf kann eine eventuelle Modifikation der Lernrate bei einer Trainingswiederholung aufzeigen. Im besten Fall kann eine dynamische Regulierung durch das Lernverfahren auf dessen Konvergenz einwirken.

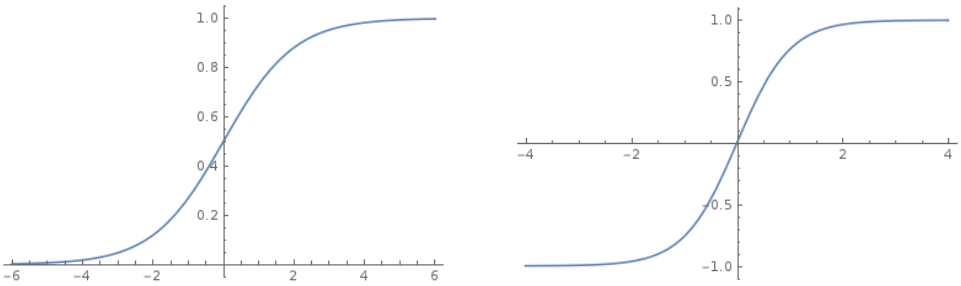


Abbildung 2.8.: Graph der sigmoiden Aktivierungsfunktionen ϕ_{sig} links und ϕ_{tanh} rechts

2.5.8. Aktivierungsfunktionen

Die Wahl der Aktivierungsfunktion hat direkte Einwirkung auf die Lernaufgabe. Da w_j stets proportional zur Amplitude von ϕ ist, somit auch proportional zu ϕ', ist entsprechend die Aktualisierung der Gewichte betroffen. Im Falle kontinuierlicher Aktivierungsfunktionen bedient man sich aus denen, die einen s-förmigen Verlauf aufweisen, sogenannte Sigmoide Funktionen. Die Ableitungen Sigmoider Funktionen wie der Tangens Hyperbolicus *tanhyp* oder *sigmoid* führen auf sich selbst zurück und äußern sich entsprechend durch geringfügige Aktualisierung der Gewichte bei großen Eingangsamplituden im Fall ϕ_{sig} sowie auch bei kleinen Amplituden im Fall ϕ_{tanh}. Der Wertebereich von ϕ_{tanh} ist gegenüber ϕ_{sig} größer. Schließlich liefert die logistische Funktion für negative Argumente kleine positive Funktionswerte. Der Lernprozess wird verlangsamt, die Neuronen sind gesättigt. Eine Methode dem entgegenzuwirken, ist die Normierung des Eingangssignals auf einen Wertebereich, welcher dem der Sigmoide gerecht ist. Abbildung 2.8 präsentiert die Graphen dieser Funktionen.

$$\phi_{sig}(x) = \frac{1}{1+e^{-x}} \qquad \phi_{tanh}(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$
$$\phi'_{sig}(x) = \phi(1-\phi_{sig}) \qquad \phi'_{tanh}(x) = 1-\phi_{tanh}(x)^2$$

Die Rampenfunktion *ReLU* 2.27 bedient sich dem biologischen Vorbild der Neuronenaktivierung. Deren Verwendung führte in Untersuchungen [12, 5] zu bes-

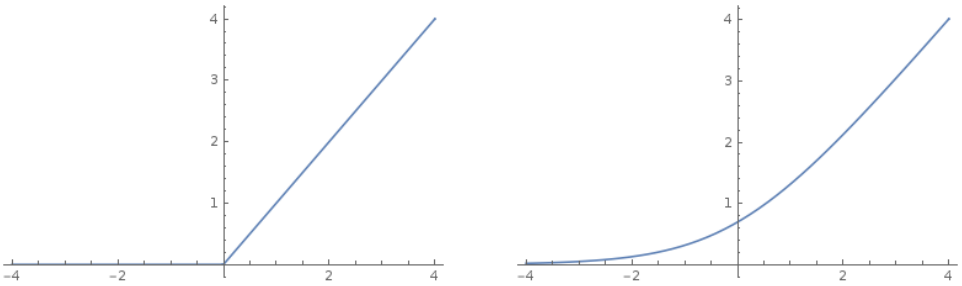


Abbildung 2.9.: Graph der sigmoiden Aktivierungsfunktionen ϕ_{ReLU} links und ϕ_{sp} rechts

seren Resultaten als deren sigmoiden Pendants und wird daher in der Praxis durch die weichere approximierten Variante *softplus* 2.29 oft verwendet. Graphen der Funktionen 2.27 und 2.29 finden sich in Abbildung 2.9.

$$\phi_{ReLU}(x) = max(0, x) \tag{2.27}$$

$$\phi_{sp}(x) = log(1 + e^x) \tag{2.28}$$

$$\phi'_{sp}(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}} \tag{2.29}$$

2.5.9. Modellselektion

Zur Erfüllung einer Lernaufgabe mittels neuronalen Netzen stehen verschiedene Möglichkeiten zur Verfügung, sich dem gewünschten Ergebnis zu nähern. Mit beliebiger Initialisierung der Netz-Gewichte werden durch Wiederholung des Lerverfahrens mit selber Netzarchitektur verschiedene Gewichte berechnet. Voneinander verschiedene Netze werden erzeugt, demnach unterschiedliche Klassifikatormodelle mit entsprechender Güte. Ebenso ist durch Modifikation der Netzarchitektur die Menge der Hypothesen zu einer Lernaufgabe erweiterbar. Die Methode der Modellsektion weitet das Lernproblem über das Erlernen optimaler Parameter aus auf den Raum möglicher Modelle [8]. Es entsteht das Problem, gefundene Modelle zu vergleichen, denn auch das Verhalten des Klassifikators unter trainingsfremden Stimuli muss bewertet werden. Natürlich haben die Trainingsdaten einen wesentlichen Einfluss auf die resultierenden Modelle [8].

Rauschen durch Messfehler Ist die Datengrundlage für den Trainingsprozess verrauscht, beispielsweise durch Messfehler oder unvollständige Daten, so besteht die Möglichkeit, dass sich das Netz entsprechend anpasst. Die Merkmalsvektoren werden falsch klassifiziert, wodurch das erlernte Klassifikatormodell an Güte verliert.

Überanpassung - *Overfitting* Sind im Verhältnis des Dimensionsraums zu wenig Trainingsdaten vorhanden, so ist das Resultat unter Umständen eine sehr gute Annäherung an die Zielvorgabe, da die Trainingsdaten exakt interpoliert werden. Jedoch geht die Generalisierungsleistung für den unbekannten Definitionsbereich verloren. Für neue Eingangsvektoren wird also eine unpräzise oder gar falsche Ausgabe erzeugt. In diesem Fall ist das Lernergebnis hinsichtlich der Daten überangepasst. Abbildung 2.10 zeigt die Anpassung eines Modells an die Trainingsdaten.

Um den Generalisierungsfehler zu bestimmen, wird eine *Kreuzvalidierung* durchgeführt. Dazu wird die Menge der Trainingsdaten geteilt. Ein Teil wird zur Bestimmung der Gewichte als Vorgabe verwendet, der andere Teil zur Bestimmung des Fehlers. Der Trainingsfehler als Gütemaß im Lernprozess zeigt die Anpassung an das gewünschte Zielergebnis auf, jedoch lediglich bezüglich der interpolierten Trainingsdaten. Dazu ist der Validierungsfehler ein Maß für den Lernerfolg abseits der Trainingsvorgabe. Eine Aufteilung der Trainingsdaten kann durch anteiliges, zufälliges Ziehen aus der Grundmenge erfolgen, der Validierungsteil sollte jedoch nicht zu klein gewählt werden. Als Abbruchkriterium des Lernvorgangs ist der minimale Validierungsfehler deshalb eine bessere Wahl als der Trainingsfehler.

Eine weitere Variante berücksichtigt kleine Mengen an Trainingsdaten. Wenn eine separate Verifikationsmenge aufgrund des Datenmangels nicht von Vorteil ist, kann die Methode *K-fold Cross Validation* zu einem validierten Ergebnis führen. Dazu erfolgt die Aufteilung des Datensatzes in K Chargen (*Batches*). Das Training wird K-fach wiederholt und zur Verifikation ein jeweils anderer Datenblock verwendet. Als Verifikationsfehler ergibt sich das Mittel aller Einzelergebnisse. In extremster Form kann zu einem Trainingsvorgang genau ein Erregungsvektor ausgelassen und zur Verifikation verwendet werden.

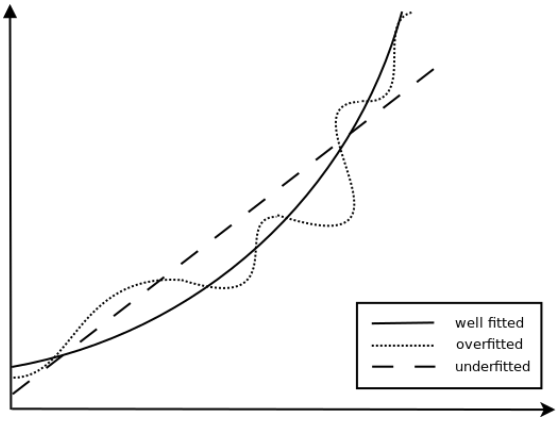


Abbildung 2.10.: Beispiel für Über- und Unteranpassung

3. Stand der Forschung

Die Klassifikationsaufgabe ist in vielen Disziplinen anzutreffen. Zur Lösung der Aufgabe in der Bildverarbeitung werden entsprechend *Convolutional Neural Networks* zur Klassifikation verwendet. In Studie 3.2 wird das CNN als Präprozessor zur Erzeugung von Merkmalskarten verwendet, um die eigentliche Klassifikationsaufgabe mit einem *MLP* durchzuführen. Es wird deutlich, dass die Architekturen der Klassifikatornetze stark variieren.

3.1. Klassifikation von Malaria infizierten Blutzellen

Zur *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)* wurde eine Studie zur CNN basierten Bildanalyse für Malaria Diagnose vorgestellt. [10] Malaria ist eine weltweit verbreitete Krankheit und nur unter Aufwand und Expertise zu diagnostizieren. Die Diagnose einer Malaria-Infektion geschieht durch visuelle mikroskopische Untersuchung einer Blutabstrichprobe. Im Speziellen werden die roten Blutkörperchen auf Parasitenbefall untersucht. Da für diese Untersuchungen qualifizierte Techniker benötigt werden und die Resultate auch abhängig sind von deren Erfahrung, sind Falschdiagnosen nicht auszuschließen. Das Klassifizierungsproblem wird durch die Studie zum einen mit Hilfe eines *CNN* gelöst und zum anderen mit einer vormals vorgeschlagenen Stütz-Vektor-Methode verglichen. Der Lernaufgabe standen 20.578 Aufnahmen einzelner infizierter und gesunder Blutzellen mit gleichem Anteil zur Verfügung. Davon wurden 90 % der Bilder für den Lernvorgang, 10 % für die Validierung verwendet. Alle Bilder wurden in der Größe normalisiert

Measure	CNN model	Transfer Learning
Accuracy	97.37%	91.99%
Sensitivity	96.99%	89.00%
Specificity	97.75%	94.98%
Precision	97.73%	95.12%
F1 Score	97.36%	90.24%
Matthews correlation coefficient	94.75%	85.25%

Abbildung 3.1.: Die statistischen Kennzahlen aus dem Vergleich CNN und SVN als Klassifikatoren

auf 44×44 Pixel mit drei Farbkanälen. Das ausgewählte Netz besteht aus den 4 Blöcken *32c5-32c5, 64c5-64c3, 128c5x64-R-256c4, 256f-256f-2f*. Feinheiten im Netz treten durch das Beschreibungsschema in den Hintergrund, daher lohnt ein Blick auf die Architekturabbildung der Arbeit. Die ersten zwei Filterbänke verwenden nach jedem Faltungs-Layer die Aktivierungsfunktion ReLU, die dritte Filterbank nur zwischen den Faltungsschichten. Für den 4 Block ist *sigmoid* gewählt, die *softmax* Funktion liefert das Endresultat, die Wahrscheinlichkeit einer Infektion.

Die Autoren stellen fest, dass die Klasssifikationsaufgabe durch ein CNN deutlich an Genauigkeit, Präzision und Spezifität (richtig positiv Rate) gegenüber der ursprünglich vorgeschlagenen Methode gewinnt. Abbildung 3.1, entnommen aus [10] stellt die Kennzahlen der Auswertung gegenüber. Das verwendete Netzwerk stellt den derzeit besten Klassifikator dieser Aufgabe im Vergleich mit Ansätzen anderer Publikationen.

3.2. Scene Detection

Die Autoren J. Stamford & B. Peach forschen im Feld *Ambient Assisted Living - AAL* (*dtsch. Umfeld Assistiertes Leben*). Anwendungen sollen Menschen bei der Interaktion mit ihrem Umfeld unterstützen. Dazu stellen sie in ihrer Arbeit [16] eine Lösung zur automatischen Szenenerkennung vor. Ihre Motivation ist es, Menschen im hohen Alter oder mit visueller Beeinträchtigung einen Mechanismus zu bieten, der eine Szene klassifizieren kann. Einen Nutzen sehen sie im Innen- sowie Außenbereich. Sie verwenden ein *CNN* zur Reduktion der Bilddimension und Merkmalskartierung. Zusätzlich wird ein *Multi Layer Perceptron*

- *MLP* mit den extrahierten Merkmalskartierung des *CNN* trainiert. Das *MLP* dient als Klassifikator zur Einordnung der auf den Bildern erkenntlichen Szenen. Ziel ist es, anhand neun markanter Merkmal eine abgebildete Szene möglichst genau zu klassifizieren.

Stamford und Peach konnten zeigen, dass eine Dimensionsreduktion verwendeter Bilder von 1920×1080 auf 29 Pixel als gute Grundlage für die Lernaufgabe des Klassifikators dient. Ein *40c3-20c3-1c2-1c2* Netz mit *3-3-2-2* Max-Pooling in jeder Faltungsbank, dient hier der Dimensionsreduktion. Der *MLP*-Klassifikator, also ein *Fully Connected*-Block, hat drei versteckte Schichten zu 500 , 200 , 50 Neuronen und 9 Neuronen im der Ausgabe-Layer. Für das Training wählten die Autoren 50 % der Daten, die andere Hälfte zum Test. Zur Auswertung der Performance wurde die Fläche unter der Kurve gewählt (*engl.: Area under the Curve (AUC)*). Hier wird die *Falsch-Positiv-Rate* und Trefferquote gegenüber gestellt und die Fläche unter der resultierenden Kurve als Gütemaß veranschlagt. Es wurde zum Training ein *AUC* von 0,96 und beim Test ein *AUC* von 0,97 erreicht, ein bemerkenswertes Ergebnis.

3.3. Klassifizierung von Straßenschildern mit Max-Pooling-Position

Die Klassifizierung von Straßenschildern *Traffic Sign Recognition - TSR* ist Herausforderung für eine Reihe von Studien. Zur *2016 International Conference on Machine Learning and Cybernetics* stellten die Autoren der Studie [13] eine neue Variante des *Poolings* vor. Da diese zum *Max-Pooling* neben dem Feature von maximaler Intensität auch die Position des Features innerhalb der Karte berücksichtigt, bezeichneten sie die Methode als *Max-Pooling-Position (MPP)*. Die Position des Features von maximaler Intensität wird binär codiert und fließt so in das anschließende *MLP*, die *Fully Connected*-Bank des Neuronalen Netzes ein. So sollen zur Klassifizierung die visuellen Attribute als zusätzliches Merkmal berücksichtigt werden. Den Autoren steht mit *German Traffic Sign Recognition Benchmark - GTSRB* eine Datengrundlage zum Benchmarking zur Verfügung. Sie umfasst 13340 Bilder mit einer Unterteilung in 26 Klassen. Zusätzlich sind

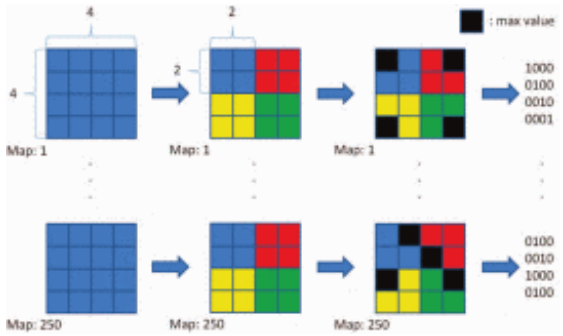


Abbildung 3.2.: Beispiel zur vorgestellten Pooling-Methode

die visuellen Attribute (z.B. Kreis, Dreieck, Symmetrie, Pfeil, Farben etc.) kategorisiert. Dem Lernvorgang standen 70 % der Daten zur Verfügung, zur Validierung die restlichen 30 %. Neben einer hohen mittleren Genauigkeit von 95,53 % in der Klassifikation konnten auch die visuellen Attribute der jeweiligen Klasse ermittelt werden.

4. Methode

Zur Klassifizierung der algorithmisch gefundenen Eigenschaften wurde der Fokus dieser Arbeit auf das Prinzip des überwachten Lernens gelegt. Für die Erzeugung einer Grundwahrheit (engl.: *Ground Truth*) für das Training eines Klassifikators wurden Torpfosten auf Bildern gekennzeichnet. Die Bilder stammten aus dem Log-Pool des Nao-Teams. Sie wurden während des *Robocups* 2016 gewonnen. Von beiden *NAO*-Kameras wurden Abbildungen der unteren Kamera bevorzugt, da auf deren Aufzeichnungen der Schmelzpunkt Spielfeld und Pfosten häufiger zu sehen ist. Es folgt eine Beschreibung der Vorgehensweise der Untersuchungsumsetzung.

Selektieren der Aufnahmen Das Programm *ImageSelectionTool* aus dem Werkzeugkasten des Nao-Teams dient der Selektion von Bilddaten. Es wurden alle Bilder ausgewählt, auf denen ein unverdeckter, durch einen Menschen identifizierbarer, Torpfosten zu erkennen ist. Ebenso wurde darauf geachtet, das die Bilder nicht von deutlicher Bewegungsunschärfe geprägt sind. Natürlich sind sie während eines normalen Spielablaufes nicht auszuschließen, jedoch stellen sie eher die Ausnahme.

Markieren von Torkomponenten Das Hauswerkzeug *LabelingTool* ermöglichte das Setzen unterschiedlich definierter Markierungen. Geometrische Figuren wie Kreise und Rechtecke dienen der Flächenmarkierung bzw. Eingrenzung von Objekten, Linien dem Objektverlauf oder der Ausrichtung. Torpfosten wurden durch Linien markiert, deren Bildpunkte zu Beginn des Pfostens auf dem Spielfeld und der oberen Kante der Querlatte gesetzt sind. Der Verlauf mittig zur Längsachse wurde angestrebt. Aus praktikablen Gründen wurde auch die Querlatte markiert, hier sitzen die Bildpunkte der Markierungslinie jeweils an den äußeren Kanten. Auch hier ist der

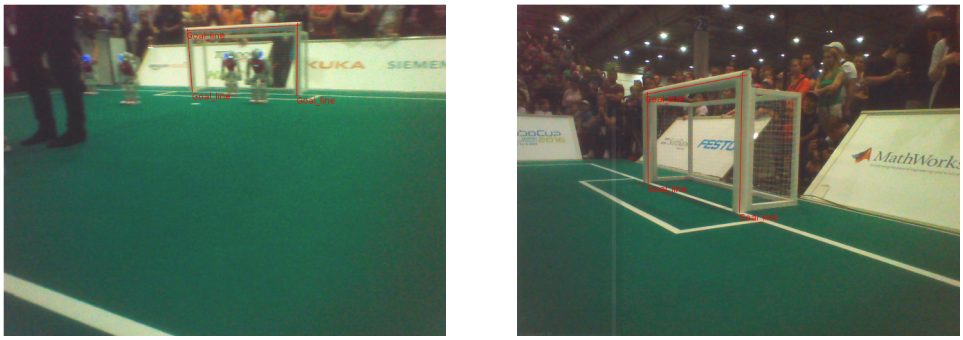


Abbildung 4.1.: Markierte Torpfosten

Verlauf mittig zur Längsachse. Dabei schneiden die Pfosten-Markierungen stets die der Querlatte. Abbildung 4.1 zeigt die Markierung der Torpfosten in zwei Beispielbildern. Das Werkzeug speichert jedes Label mit Bezug auf Bildname, Nutzername, Objektname, Label-Name und zugehöriger Punktmenge der Markierungsfigur.

Trainings- und Testdaten Die markierten Bilder wurden unterteilt in Trainings- und Testsets. Testdaten werden dem Lernvorgang nicht zur Verfügung gestellt, sondern dienen der Evaluierung des Klassifikators.

Die folgenden Schritte wurden in erwähnter Reihenfolge angewandt und durch verschiedene Parameter variiert. Als grundlegender Parameter für alle Komponenten sei die *Patch-Größe* im Voraus erwähnt. Der Parameter definiert, unter einem Begriff zusammengefasst, den zu extrahierenden Bildbereich (Abs. 5.2) und schließlich die Dimension eines Merkmalsvektors.

Extrahieren der Samples Zur Versorgung des Klassifikators werden Untersuchungsproben *(engl.: Sample)* aus der Bildmenge extrahiert. Das Programm `sampleextractor` erzeugt durch die Parameter Label-Name, Bildmenge sowie Sample-Größe für eine Objektklasse zwei Mengen von Samples, unterteilt in positive und negative Proben. Die Entscheidung basiert auf der Ergebnismenge des zugrunde liegenden Algorithmus (siehe Abs. 5.1) und der Grundwahrheit durch Bildmarkierungen. Alle gefundenen hypothetischen Pfosten, die nicht durch eine Markierung abgedeckt sind, entsprechen nicht der Grundwahrheit. Sie repräsentieren keine Torpfosten und werden entsprechend als Negativmuster registriert. Die Stichproben-

entnahme erfolgt sowohl für das Trainingsset als auch für das Testset. Alle Samples werden ihrer Grundwahrheit entsprechend in einer *lmdb* Datenbank gespeichert, ein einfacher eingebetteter *Key-Value-Store* mit niedrigen Zugriffszeiten. Für eine visuelle Inspektion werden die Proben durch das Programm `lmdbsamplevisualizer` visualisiert und als *PNG*-Dateien gespeichert.

Trainieren des Klassifikators Zur Definition eines Klassifikator-Netzes, sowie Algorithmen und Funktionen des Lernverfahrens wurde das Deep Learning Framework *Caffe* [7] gewählt. Durch eine strukturierte Beschreibung der Architektur und Funktionen, wie etwa Aktivierungsfunktionen der einzelnen Schichten oder sog. *Solver* des Lernverfahrens (z.B. *Backprop* und Spezialisierungen), können Ideen und Aufgaben schnell und unkompliziert umgesetzt werden. Zu jedem Trainingsdurchlauf erzeugt das Framework eine Momentaufnahme (engl.: Snapshot). Jeder so resultierende Zustand der aktualisierten Neuronen-Gewichte einer Iteration kann prinzipiell für eine Klassifizierung verwendet werden.

Evaluierung Eine Evaluierung erfolgt durch ein konkretes Modul. Es speist das trainierte Netz sequentiell mit den Proben des Testsets und führt Protokoll. Für jedes zu trainierende Objekt wird das Klassenzuordnungsresultat des Klassifikatornetzes gespeichert. Dabei werden Kennzahlen wie *Hit-Rate* und *Precision* bestimmt, um den Trainingserfolg des Klassifikators aufzuzeigen. Durch einen Distanzparameter können Stichproben, deren Markierung die Abstandvorgabe überschreiten, verworfen werden. Damit ist eine Bewertung des Klassifikators bezüglich der Objektentfernung möglich. Die Resultate dieses Schrittes dienen Kapitel 6 als Diskussionsgrundlage.

Zu jeder Initialisierung eines Lernvorgangs wird ein Ordner erzeugt, der neben Konfigurationsparametern den kompletten Programmcode enthält. Durch einen Zeitstempel im Ordnernamen kann ein Lernvorgang eindeutig referenziert und mehrfach wiederholt werden. Programmcode, Kompilat, Stichproben, Parameter des Lernverfahrens und Resultate des Trainings sind reproduzierbar gespeichert.

5. Umsetzung

Im Folgenden werden die Techniken zur Umsetzung aufgezeigt. Die in 2 erläuterten Grundlagen fließen an dieser Stelle ein.

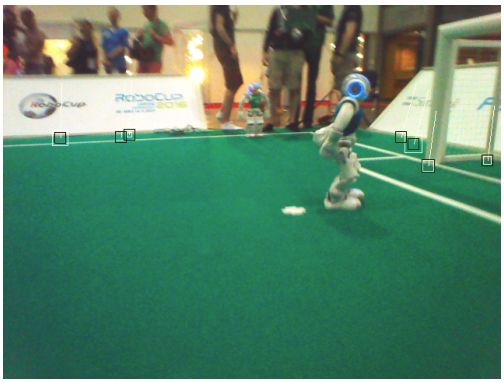
5.1. Tor Hypothesen Generator I

Mit dem Team-Report 2016 des Teams *Berlin United - Nao Team Humbold* wurde bereits ein Ansatz für den folgend beschriebenen Algorithmus zur Torerkennung durch *Edgel Detection* geliefert.

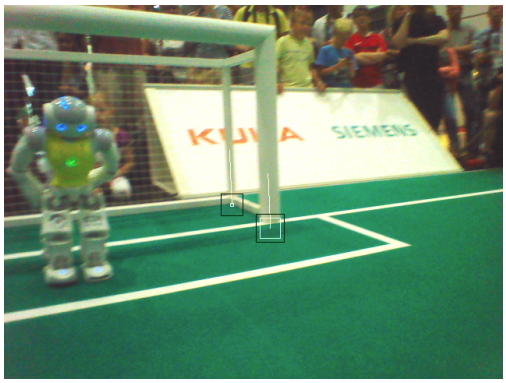
Die Klasse `GoalHypothesesGenerator` bedient sich des Vorwissens der Detektionskomponenten zur Feld- (`FieldDetector`) und Linienerkennung (`LineDetector`). Feld- und Linienerkennung liefern Parameter zur Helligund Farbigkeit von grün und weiß im *YCbCr* Raum. Durch die Feldlinienerkennung existiert eine Punktmenge, welche die äußere Kante des Feldes repräsentiert. Parallel dazu werden für eine Abtastung der Umgebung oberhalb und unterhalb der Feldgrenze Scanlinien angesetzt. In sequentieller Abtastung mit Schrittweite wird für jede Scanline das Gradientenfeld ermittelt und durch eine approximierte Gauß-Faltung das Rauschen minimiert. Erfolgt bei anschließender Untersuchung der Gradienten ein signifikanter An- bzw. Abstieg im Feld, wird der Bildpunkt als *Edgel* betrachtet. Auf diese Weise ergibt sich für jede betrachtete Abtastlinie eine Edgel-Menge, die potentielle Außenkanten eines Objektes darstellen. In einer festgelegten horizontalen Umgebung erfolgt eine vertikale Suche nach Kantenhäufungen auf den Gradientenfeldern. Falls eine Häufung vertikal benachbarter Kantenpunkte gefunden wird, liegt der Verdacht nahe, ein zur Feldlinie nahezu senkrecht ausgerichtetes Objekt registrieren zu

können. Dabei werden jene Punkte festgehalten, zu denen Gradienten eine minimale Differenz aufweisen. Finden sich mindestens k vertikal benachbarte *Edgels*, so werden sie als Außenkante eines hypothetischen Torpfostens betrachtet und entsprechend als `GoalEdge` zusammengefasst. Für jede Pfostenkante wird mit Hilfe der gefundenen *Edgels* die passende Geradengleichung erstellt. Die `GoalEdge`-Objekte wurden von einer Richtung ausgehend ermittelt und nach Abschluss der Kantensuche bezüglich der horizontalen Feldlinienposition sortiert. Somit kann man ein `GoalEdge`-Objekt als linke bzw. rechte Pfostenkante einordnen.. Eine Analyse der Bildpunkte zwischen einer hypothetisch linken und rechten Kante soll eine Pfosten-Hypothese bestärken. Für einen parametrisierten Betrag werden die Bildpunkte mittig zu den Kantenlinien absteigend in Richtung Spielfeld auf Differenzen hinsichtlich der Farbwerte grün und weiß durch Stichproben analysiert. Dabei ist das Antreffen grüner Punkte ein Abbruchkriterium, da die Annahme gilt, die Linie am Kreuzungspunkt von Feld und Pfosten gefunden zu haben. In diesem Fall ist die Hypothese bestätigt. Die Mitte der Kreuzlinie wird als Basispunkt eines Pfostens und die zugehörige, ermittelte Breite registriert. Abschließend wird die Varianz der Hypothesenbildpunkte, also des Kantenzwischenraums, errechnet. Diese dient bei Sortierung der Ordnungsrelation. Aus der Ergebnismenge werden, der Varianz nach aufsteigend, die n besten `GoalPost` Objekte gewählt, sodass für jede Abbildung höchstens n Hypothesen (`ObjectHypothesis`) resultieren. Die Begrenzung wurde zum Training auf höchstens $n = 6$ Pfostenhypothesen festgelegt.

Abbildung 5.1 zeigt Resultate des Erkennungsalgorithmus (im Folgenden Verlauf auch als *THG* bezeichnet). Jede quadratische Markierung enthält den ausgemachten Punkt einer Pfostenhypothese mittig zentriert. Ein Seite des weißen Quadrates entspricht der gefundenen Breite des Pfostens. Die Linie entspricht der ermittelten Hypothesengerade, und ist vom Basispunkt aufwärts eingezeichnet. In Abbildung 5.1a wird der Umfang der Hypothesenmenge deutlich. Mit Abbildung 5.2 ist ein Negativbeispiel der vom Algorithmus ermittelten Hypothesen gezeigt. Es bleibt anzunehmen, dass durch Sortierung und Beschränkung der Pfostenhypothesen, das korrekte Objekt verworfen wurde.



(a) Potentielle Torpfosten



(b) Gutes Resultat

Abbildung 5.1.: Resultate des Tor-Hypothese-Generators. Quadrate markieren den gefundenen Kreuzpunkt aus Spielfeld und Pfosten, die Linie zeigt die ermittelte Gerade auf

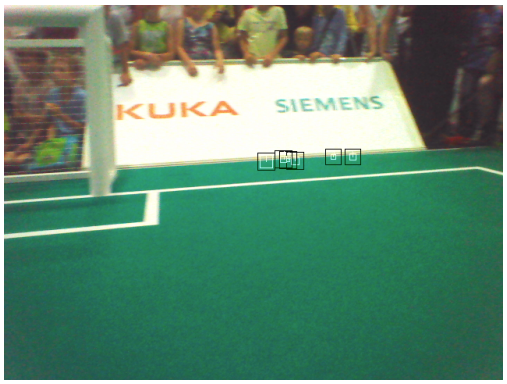


Abbildung 5.2.: Negatives Resultat des Tor-Hypothesen-Generators

5.2. Feature Extraction

Um ein registriertes Objekt für ein *CNN* zu parametrisieren, ist die Abbildung des Objektes auf die Eingabestruktur des *CNN*-Input-Layers zu transformieren. Dazu werden verschiedene Möglichkeiten diskutiert und in Abschnitt 6 die damit erzielten Resultate analysiert.

Die unter 5.1 erläuterten Resultate dienen der Extraktion als Rahmenparameter, um den betreffenden Bildausschnitt zu beschreiben. Das Ziel ist es, eine Bildpunktteilmenge der Abbildungsfläche einer Pfosten-Hypothese als repräsentativen Merkmalsvektor zu definieren. Im Folgenden werden die Resultate einer solchen Transformation verwendeten Begriff als *Samples* oder *Patches* bezeichnet, um von dem Hypothesenbegriff aus Kapitel 2,Abs. 2.3 etwas zu differenzieren.

5.2.1. Feature Extractor 1

Der folgend beschriebene Algorithmus FE_1 verfolgt das Ziel, einen Merkmalsvektor mit Begrenzung auf den Bildbereich des Pfostens zu erzeugen. Als Eingrenzung der Abbildungsfläche dient Breite der Objekthypothese O. Der Merkmalsvektor des *CNN* Eingabe-Patches p_n der Dimension $n \times n$ ist von quadratischer Form, weshalb bezüglich Breite O_w eine Skalierung nötig ist, um das Abbildungsareal auf das Patchfeld zu transformieren. Es ergibt sich für ein Patch p_n der Skalierfaktor $s = \frac{O_w}{p_n}$.

Der unter Abschnitt 5.1 diskutierte Algorithmus liefert, wie bereits beschrieben, eine ermittelte Gerade, welche die Ausrichtung der Pfosten-Hypothese beschreibt. Ausgehend vom Bildpunkt der Hypothese, also dem Ausgangspunkt des Pfostens vom Feld, erfolgt eine Abtastung der Umgebung parallel zur beschreibenden Gerade. Abbildung 5.3 veranschaulicht das Vorgehen der Transformation. Der Pfeil über die Längsachse des Torpfostens veranschaulicht die vom *Tor Hypothesen Generator* errechnete Pfostengerade. Abbildung 5.5 zeigt zwei Proben, die vom Algorithmus extrahiert wurden, in Vergrößerung.

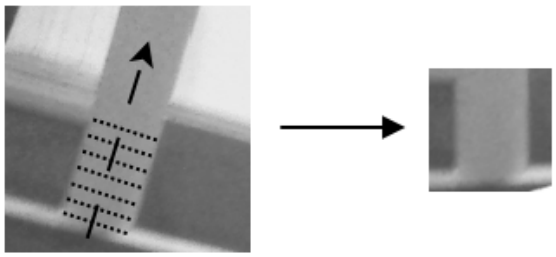


Abbildung 5.3.: Beispieltransformation mit Abtastung der Objekthypothese über die ermittelte Gerade

Abbildung 5.4.: Beispielsamples nach FE_1

Abbildung 5.5.: Vergrößerte p_{18} Patches nach FE_1 mit erweitertem Abtastradius.

Abbildung 5.6.: Beispielsamples nach FE_2

5.2.2. Feature Extractor 2

Eine Alternative zu FE_1 lockert die Bindung des Merkmalsvektors an das Feature. Hier werden Merkmale zusammengetragen, die neben dem Feature selbst das Bildumfeld des Hypothesenpunktes betreffen. So ist nicht nur der Bildbereich des Pfostens im Fokus, sondern auch andere Eigenschaften wie Linien, Spielfeld und Netz sind enthalten.

In Abhängigkeit der ermittelten Objektbreite wird der umgebende Bildbereich abgetastet. Hierzu wird nicht die Objektgerade verfolgt, sondern der Bildbereich um den Objektpunkt herum. Da hier keine Ausrichtung der Bildpunkte wie bei FE_1 stattfindet, können diese Stichproben für eine Vergrößerung der Trainingsdatenmenge rotiert und gespiegelt werden. Abbildung 5.6 zeigt zwei extrahierte Patches nach FE_2.

5.2.3. Feature Extractor 3

Eine dritte Variante FE_3 nutzt das Modul `ProjectionUtil.h` des Nao-Frameworks. Es realisiert diverse Funktionen zur Projektion eines Bildpunktes auf die Spielfeldfläche. Mit dem bekannten Öffnungswinkel der Nao-Kamera kann für jeden Bildpunkt der Blickwinkel bestimmt werden. Zusätzlich addieren sich Roll- und Nick-Winkel bezüglich der Längsachse des *NAOs*. Ebenfalls ist mit den Maßen des Naos die Höhe bekannt. Unter der Voraussetzung, dass der Roboter senkrecht auf dem Feld steht, lässt sich die Entfernung für einen Bildpunkt trigonometrisch berechnen. Wenn für eine Objekt in 1 m Entfernung zum Nao die Pixelbreite bezüglich des Objektradius bekannt ist, kann für einen

Abbildung 5.7.: Beispielsamples nach FE_3

Bildpunkt unter Hinzunahme der errechneten Distanz der Objektradius in Pixeln berechnet werden.

Diese Berechnungen werden in FE_3 für einen Hypothesenpunkt statt der ermittelten Pfostenbreite verwendet und in Abhängigkeit des erwarteten Objektradius entsprechend das Umfeld abgetastet. Dabei ist der Bezugspunkt zentral im Patch-Feld angesetzt. Auch in diesem Fall können zusätzliche Patches durch Rotation und Spiegelung generierte werden.

5.3. Spezifikation des Neuronalen Netzes

Das zur Untersuchung verwendete *CNN* hat die Gestalt *8c3-10c3-32f-1f*. Abbildung 5.8 zeigt eine schematische Darstellung des Netzes. Für jede Abbildung erzeugt FB_1 demnach acht *Feature-Maps*, FB_2 zehn. Nach den Standardeinstellungen werden zur Faltung keine Randpixel hinzugefügt. Durch den Pooling-Layer mit $k = s = 2$ werden die Feature-Maps aus FB_1 auf ein Viertel ihrer Dimension reduziert. Alle Layer verwenden die Aktivierungsfunktion *ReLU*, mit Ausnahme der Schicht *1f*. Zur Konfiguration wurden folgend beschriebene Funktion verwendet.

Solver Eine Funktion zur Aktualisierung der Neuronen-Gewichte wird auch als *Solver* bezeichnet. Im Abschnitt 2.5.6 wurde eine Methode beschrieben.

Mit Adagrad [3] wird in dieser Untersuchung eine Erweiterung der Energieminimierungsfunktion als Verbesserung der Gewichtsteaktualisierung (Gleichung 2.25) verwendet. Dieser *Solver* normiert den aktuell betrachteten, individuellen Gradienten einer Schicht L innerhalb eines Zeitfensters.

$$(W_{t+1})_i = (W_t)_i \, v_t \tag{5.1}$$

$$v_t = -\lambda \frac{(\nabla_W L)_i}{\sqrt{\sum_{t'}^{t} (\nabla_W L_{t'})_i^2}} \tag{5.2}$$

Er kompensiert damit die eventuellen Auswirkungen der gesetzten Lernrate λ. Dabei wird das Zeitfenster über einen Akkumulator realisiert, um den Speicherverbrauch zu berücksichtigen.

Xavier Initialisierung Für eine Initialisierung der Faltungskerne wurde die normalisierte Xavier Initialisierung gewählt. In einer vergleichenden Studie [4] wurde durch gleichnamigen Autor die Auswirkung der Gewichteinitialisierung nach verschiedenen Methoden untersucht. Demnach versprechen die Autoren mit einer zufälligen Initialisierung der Gewichte eines Layers l und zugehöriger Dimension n des Eingangsvektoren nach der Verteilung 5.3 beste Voraussetzung für das Lernverfahren.

$$w_{i,j}^{l} \in \left[-\frac{\sqrt{6}}{\sqrt{n_l + n_{l+1}}}, \frac{\sqrt{6}}{\sqrt{n_l + n_{l+1}}} \right] \tag{5.3}$$

Sie kann in *caffe* als Standardinitialisierung gewählt werden, so auch in dieser Untersuchung.

Softmax Die Exponentialfunktion

$$\phi_{softmax}(\mathbf{s})_j = \frac{e^{s_j}}{\sum_{k=1}^{K} e^{s_k}} \tag{5.4}$$

wird als logistische Aktivierungsfunktion verwendet und normalisiert den Eingangsvektor $\mathbf{s} \in \mathbb{R}^K$ auf das Intervall $[0, 1]$ [21]. Im diskutierten *CNN* findet sie im Layer *FC2* Einsatz und liefert damit eine probabilistische Aussage zur Klassenzugehörigkeit eines Eingabevektors.

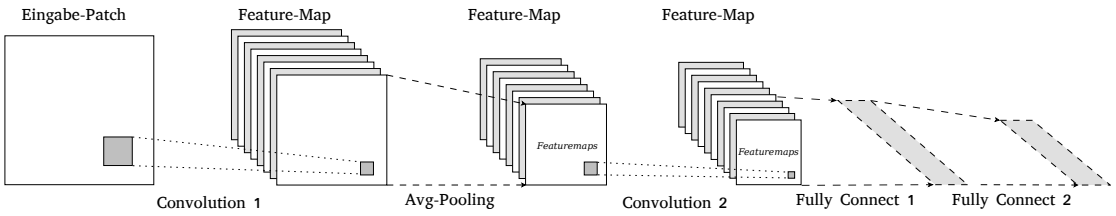


Abbildung 5.8.: Schematische Darstellung der CNN-Architektur

5.4. Training

Für jeden Lernvorgang wird eine Umwelt deklariert, die zu verwendende Bilddaten, zugehörige Labels, Parameter zum Lernverfahren und das nach Abschnitt 5.3 spezifizierte *CNN* definiert. Umweltparameter und resultierende Erzeugnisse aus Umwelt sowie Lernverfahren werden dabei in einer Dateistruktur zusammengefasst, um eine Wiederholung oder Rekonstruktion des Vorgangs zu vereinfachen.

CNN Es werden jeweils zwei Netze erzeugt, die der Spezifikation 5.3 entsprechen. Das Trainings-Netz wird ausschließlich zum Lernvorgang verwendet und hat die beiden Ausgangsschichten *Accuracy* sowie *Softmax*. *Accuracy* erzeugt eine Genauigkeitsangabe bezüglich der Zielbestimmung. Das Testnetz verzichtet stattdessen auf die *Accuracy* Ausgabe und bietet für eine Evaluierung nur den Ausgang *Softmax* zur Einordnung der Klassenzugehörigkeit. Bis auf die benannten unterschiedlichen Ausgänge sind alle Layer identisch.

Dimension der Strichproben Zur Extraktion der Stichproben p_n ist die Dimension $n \times n$ der quadratischen *Samples* vorgegeben. Entsprechend ist auch die Dimension des Eingangsvektors vom *CNN* festgelegt und damit die Komplexität des Netzes bestimmt. Es wurden p_{10}, p_{12}, p_{15} und p_{18} als Stichproben gewählt.

Bilddaten und Labels Zwei Parameterdateien verweisen durch Pfadangaben jeweils auf Bilddateien. Sie bestimmen die Daten für das Lernverfahren und Daten zur Evaluierung. Mit der Angabe eines Label-Namens wird der Markierungstyp bekannt gegeben, nach dem die extrahierten Samples in positive und negative Stichproben aufgeteilt werden (siehe Kap. 4, Abs.

Extrahieren der Samples). Aus den Trainings und Evaluierungsdaten werden nur jene berücksichtigt, für die mindestens eine Markierung existiert. Tabelle 5.1 zeigt die Aufteilung der Bilddaten der Lernaufgabe. Im späteren Untersuchungsverlauf wurde eine Umverteilung vorgenommen, um auf den Generalisierungsfehler einzugehen. Die Umverteilung und das Markierungsverhältnis zeigt Tabelle 5.2.

	Training	Test
markiert	3997	1906
unmarkiert	6159	3751

Tabelle 5.1.: Verteilung der Bilddaten für die Lernaufgabe

	Training	Test
markiert	5384	1088
unmarkiert	8040	2880

Tabelle 5.2.: Verteilung der Bilddaten nach Erweiterung der Trainingdaten

Parameter zum Lernverfahren Das Lernverfahren terminiert nach einer festen Zahl I von Iterationen. Nach N Schritten wird eine Kreuzvalidierung durchgeführt (Abs. 2.5.9). Sie ist aber kein Kriterium im Lernvorgang. R ist die festgelegte Lernrate (Abs. 2.5.7)

Mit der Initialisierung eines Trainings werden zunächst die Dateistruktur der Umwelt und die Netze erzeugt. Weiterhin werden aus den deklarierten Bilddatenmengen Samples extrahiert und gespeichert. Zur Inspektion erzeugt das Skript zusätzlich eine zusammenhängende Bilddatei aus den Stichproben im *PNG*-Format. Die Lernvoraufgabe ist nach der Initialisierung explizit auszuführen und wird vom Framework *Caffe* übernommen. Während des Trainings speichern *Snapshots* den aktuellen Zustand der gelernten Filter. Im Falle eines Abbruchs kann der Trainingsprozess bei erneuter Ausführung im entsprechenden Zustand fortfahren. Konsolenausgaben zeigen zu jeder Iteration den aktuellen Trainings- sowie Validierungsfehler auf (siehe 2.5.9). Am Ende des Lernvorgangs, also nach I Iterationen, liegt ein *Snapshot* mit den gelernten Filterkernen der Klassifizierungsaufgabe vor.

	Tor	Keine Klasse
richtig	r_p	r_n
falsch	f_p	f_n

Tabelle 5.3.: Zuordnungstabelle in richtig positive t_p, richtig negative t_n und falsch positive f_p bzw. falsch negative Zuweisung f_n

Trainings- und Validierungsfehler Die zuvor erwähnten Protokollausgaben vom Framework *caffe* wurden zu jedem Durchlauf in einer Datei festgehalten. Ausgaben des Trainings- sowie Validierungsfehlers jeder Iteration wurden extrahiert und aufbereitet. Sie dienen im Kapitel 6 der visuellen Auswertung des Lernvorgangs.

5.5. Evaluierung

Ein separates Evaluierungsprogramm verwendet das während der Initialisierung erzeugte Testnetz als Klassifikator auf der Evaluierungsbildmenge. Jeder *Snapshot* des Lernvorgangs entspricht einem Zustand der Lernaufgabe. Die Filterkerne des Testnetzes werden zu Beginn des Evaluierungsprozesses mit einem *Snapshot* initialisiert. Nach Klassifizierung der Validierungsbildmenge wird eine Ausgabedatei mit statistischen Kennzahlen erzeugt. Mit den durch die Markierungen geschaffenen Tatsachen in Form positiver und negativer Stichproben (siehe Kap. 4 Extrahieren der Samples), wird der Klassifikator beurteilt. Resultate der Klassifizierung werden in richtige und falsche positive bzw. negative Klassenzuweisungen kategorisiert (siehe Tab. 5.3). Dabei geschieht eine solche Einordnung stets anhand eines Schwellwerts bezüglich der von der vom Netz gelieferten Wahrscheinlichkeit zur Klassenzugehörigkeit.

Die *Trefferquote* P_H (Sensitivität, Empfindlichkeit, engl. Hitrate) zeigt im Test den Anteil der korrekt klassifizierten Objekte relativ zur Menge aller Objekte der selben Klasse auf. Somit gibt sie die Wahrscheinlichkeit an, mit der ein Objekt einer zugehörigen Klasse korrekt zugeordnet wird.

$$P_H = P(positive\ Klassifizierung | korrekte\ Zuordung) = \frac{r_p}{r_p + f_n} \tag{5.5}$$

Die Kennzahl *Präzision* P_P (Relewanz, Wirksamkeit, Genauigkeit, engl. Precision) repräsentiert die Wahrscheinlichkeit, mit der ein Objekt bei einer positiven Klassifizierung der Klasse korrekt zugeordnet wird.

$$P_P = P(korrekte\ Zuordnung | positive\ Klassifizierung) = \frac{r_p}{r_p + f_p} \tag{5.6}$$

Das Evaluationmodul protokolliert zu jeder Klassifikation eines Testelementes die Wahrscheinlichkeit zur Klassenzugehörigkeit. Durch einen Mindestschwellwert von 75 % beginnt das Protokoll, und hält in Schritten von 0,25 % jedes Klassifizierungsergebnis fest. Für jeden Schwellwert werden, nach obigem Schema, korrekte oder inkorrekte Zuordnung der Beispielelemente gezählt und daraus anschließend die bedingten Wahrscheinlichkeiten *Hitrate* und *Precision* berechnet. Liefert das Netz für ein Testdatum eine Wahrscheinlichkeit der Klassenzugehörigkeit von über 75 %, wird die entsprechende Bewertung vollzogen. Im anderen Fall ist das Ergebnis irrelevant.

Im Kapitel 6 werden die Ergebnisse der vorgestellten Algorithmen und die aufbauende Lernaufgabe unter diesen Kennzahlen analysiert. Außerdem werden Trainingsfehler sowie Validierungsfehler des Lernverfahrens für jede Hypothese der Lernaufgabe diskutiert.

Eine Überprüfung, ob die Anwendung der Klassifikatoren tatsächlich den Echtzeitanforderungen genügen konnte aufgrund technischer Probleme nicht in diese Arbeit einfließen. Eine Auswertung dazu steht entsprechend aus und wird an anderer Stelle diskutiert. Um die erwähnten Module und Werkzeuge besser einordnen zu können, bietet Abbildung 5.9 einen groben Überblick über die Architektur der verwendeten Programme. Das Modul `Vision` steuert alle Komponenten der Bildverarbeitung und ist Teil der Firmware. Dort sind die zu verwendenden Algorithmen deklariert, sowie der *GoalDetector2* instantiiert. Nachdem ein neuronales Netz mit dem Framework *caffe* trainiert wurde, wird der entsprechende Snapshot mit dem Firmware-Kompilat auf den NAO übertragen. Bei einem Start wird das Netz erzeugt und die Neuronengewichte durch `GoalDetector2` geladen. Einmal Initiiert, wird der komplette Verarbeitungsprozess für jedes Bild durch die Klasse `Vision` abgearbeitet.

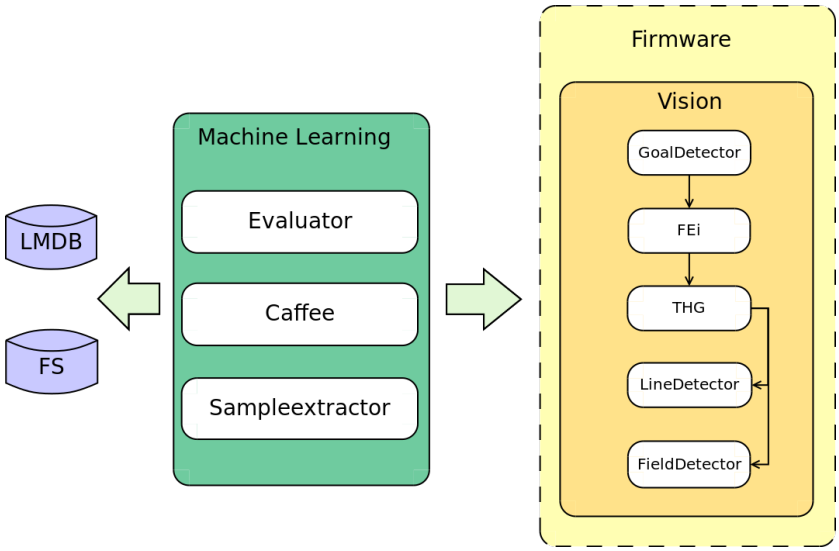


Abbildung 5.9.: Architektur der Machine Learning Komponenten und Firmware

6. Auswertung

Das Lernverfahren wurde mit allen in 5.2 beschriebenen Verfahren zur Merkmalsextrahierung durchgeführt. Für eine Differenzierung wurden die verschiedenen Ansätze leicht modifiziert. Jeder Extraktor wurde, zur Erweiterung der Abtastfläche um den Punkt einer Objekthypothese, mit einem Skalierungsfaktor versehen. Aufgrund der Skalierung sollen die Stichproben geringfügig variieren und damit auf den Lernvorgang und die Ergebnisse der Klassifikatoren einwirken. Die Resultate schlagen sich in Lernkurven und Validierung während des Lernvorgangs nieder. Auch die fixierte Lernrate wurde verkleinert, um die Auswirkung auf die Fehlerminimierung im Lernvorgang, somit auf die Lernkurve der Aufgabe zu beobachten. Durch die Funktionen der Bibliothek `ProjectionUtil` konnte ein Vergleich der Trefferquoten und Genauigkeiten bezüglich unterschiedlicher Objektdistanzen durchgeführt werden. Zusätzlich wurden die zugrundeliegenden Trainings- und Testdaten umverteilt, um dem Lernvorgang eine größere Beispeilmenge zur Verfügung zu stellen. Es wurden den Testdaten insgesamt XXX neue Bilder hinzugefügt, sogleich XXX Bilder entnommen und dem Lernvorgängen als Beipiel zugeführt. Tabelle 6.1 zeigt das Verhältnis der von *THG* gelieferten Hypothesen und tatsächlichen Übereinstimmungen mit Markierungen. In Tabelle 6.2 ist das Verhältnis der Datengrundlage nach einer Umverteilung aufgezeigt.

Es folgt die Auswertung der Lernaufgaben nach den beschriebenen Kriterien.

	Hypothesen	Markierungen		unregistriert	
Training	13245	3997	(30,2 %)	k.A.	
Test	8430	1906	(22,6 %)	353	(18,25 %)
Test 3 m				151	(7,92 %)
Test 5 m				213	(11,18 %)

Tabelle 6.1.: Gegenübertellung Tor-Hypothesen vs. Label-Treffer

	Hypothesen	Markierungen		unregistriert	
Training	18864	5384	(28,5 %)	k.A.	
Test	4122	1088	(26,4 %)	196	(17,70 %)
Test 3 m				110	(10,11 %)
Test 5 m				136	(12,50 %)

Tabelle 6.2.: Gegenübertellung Tor-Hypothesen vs. Label-Treffer der Trainings bzw. Testdaten

6.1. Auswertung des Lernverfahrens

Zur Untersuchung der Lernaufgabe wurde der unter 5.1 geschilderte Algorithmus zur Erzeugung potentieller Tor-Hypothesen verwendet. Mit den unter 5.2 geschilderten Algorithmen wurden aus den vom THG gelieferten Vorschlägen die Merkmalsvektoren extrahiert. Ebenso wurde die fixiert Lernrate reduziert, um die Minimierung der Verlustfunktion durch *Backprop* (2.5.5) zu verbessern.

Im Folgenden werden zunächst die Ergebnisse des Lernvorgangs diskutiert. Die Betrachtung erfolgt für jede Variante der generierten Stichproben, die dem Lernverfahren als Datengrundlage dienten.

Der in 5.2.1 beschriebene Algorithmus konzentriert sich gänzlich auf die Merkmale eines Pfostens und vernachlässigt dessen Umgebung. Generell ist der Punkt der Tor-Hypothese in FE_1 als unterster Bezugspunkt festgelegt. Von diesem ausgehend wird der errechneten Gerade strikt gefolgt, also vom Spielfeld aufsteigend. Als vertikales Limit ist dabei die Höhe des quadratischen Patches angesetzt. In der Horizontalen wird die durch den HG gefundene hypothetische Breite des Pfostens gewählt. Somit wird im Verlauf die Nachbarschaft beidseitig vom jeweiligen Punkt der Gerade aus untersucht. Das Verhältnis $\frac{h_w}{p_i}$ bestimmt

dabei die abzutastende Umgebung. Auf diese Weise sollen die Bildmerkmale der Hypothese für die Erzeugung einer Stichprobe vom Umfeld störungsfrei abgetastet werden.

Abbildung A.1 zeigt mit A.1a den Verlauf der Fehlerminimierung und A.1b den Fehler bezüglich der Testdaten während des Lernvorgangs. Es ist ersichtlich, dass mit der Größe des Merkmalsraums, bestimmt durch die Dimension der Stichproben, der Fehler im Verlauf der Iterationen besser minimiert wird. Für die erzeugten Stichproben von geringer Größe p_{10}, p_{12} hingegen bleibt die Lernkurve A.1a weitestgehend konstant. Die Neuronen scheinen gesättigt. Jedoch fällt gerade für die Patches p_{15}, p_{18} auf, dass der Validierungsfehler mit Fortschreiten des Trainings stetig steigt. Dies lässt Überanpassung der Gewichte auf die Lerndaten schließen. Mit der Größe des zugrundeliegenden Merkmalsraums nimmt der Generalisierungsfehler ab. Um der Stabilität der Lernkurve unter Verwendung kleiner Merkmalsräume entgegen zu wirken, wurde die fix gesetzte Lernrate mit $\lambda = 8 \cdot 10^{-5}$ auf ein Drittel reduziert. Die zugehörige Lernkurve in Abbildung A.2a zeigt jedoch keine ersichtliche Verbesserung.

Mit FE_2 und der vom Pfosten unabhängigen Abtastung des Umfeldes war eine automatisierte Erweiterung der Trainingsbilder durch Rotation und Spiegelung sinnvoll. Dem Lernverfahren standen damit erheblich mehr Daten zur Verfügung. Somit deckt der Definitionsbereich der Lern-Hypothese mit der künstlichen Vergrößerung auch vormals unbekannte Merkmale und Invarianzen ab. Dies zeigt sich in der Validierung durch einen deutlich verringerten Anstieg des Fehlers bezüglich der Testdaten. Eine leichte Tendenz zur Überanpassung der Netz-Gewichte ist erkennbar, jedoch weitaus weniger deutlich gegenüber FE_1. Auch die Lernkurve ist für jede Komplexität der Merkmalsvektoren ähnlich im Verlauf. Die horizontale Einschränkung des Abtastbereiches von FE_2 ist wieder an die vom Algorithmus ermittelte Breite der Pfosten-Hypothese bemessen. Eine Skalierung der Abtastung um den Faktor $s = 1.5$ erweitert den Merkmalsvektor in das Umfeld des Pfostens. Damit wird im Merkmalsraum eine Varianz erzeugt, die sich, wie ein Vergleich der *HR*-Diagramme 6.3 und 6.4 mit den Diagrammen 6.7 und 6.8 zeigt, jedoch nur kaum deutlich macht. Der Einfluss ist auch im Vergleich der FE_1 Ergebnisse zu beobachten.

Mit FE_3 wird der ermittelte Bezugspunkt jeder Hypothese in das Zentrum der zu extrahierenden Stichprobe gesetzt (Abbildung A.6). Wiederholte Messungen weisen für groß dimensionierte Merkmalsräume stets einen deutlichen Generalisierungsfehler gegenüber den Testdaten auf.

Erhöhung des Skalierungsfaktors

Der Skalierungsfaktor wurde erhöht zu $s = 1.5$, um das erweiterte Umfeld der Pfostenhypothese einzubeziehen. Es zeigt sich unter FE_1 eine Verbesserung in der Validierung für p_{18}, bei p_{15} jedoch nicht. Eine Auswirkung auf FE_2 ist nicht ersichtlich, immerhin schneidet er in Tests jeweils mit einem guten Ergebnis ab. Auch unter FE_3 zeigen sich keine markanten Unterschiede bezüglich der Testdaten.

6.2. Auswertung der Performance

Es folgen *HR*-Diagramme (*Hitrate vs. Precision*), um die Leistung der Klassifikatoren zu visualisieren. Dabei werden die Klassifikatoren unter Berücksichtigung der Merkmalsvektoren gegenübergestellt.

Durch die semilineare Aktivierungsfunktion *softmax* am Netzausgang, wird die Wahrschinlichkeit der Klassenzugehörigkeit geliefert. Da dies keine binäre Aussage ist, muss ein definierter Schwellwert als Unterstützung dienen. Natürlich wäre der höchst mögliche Wahrscheinlichkeitswert von 99,9 % wünschenswert. Die Diagramme unterstützen die Suche. Sie zeigen das Verhältnis von Trefferquote und Präzision. Berücksichtigt man die Tatsache, dass mit zunehmendem Schwellwert entsprechend die Empfindlichkeit in der Klassenzuordnung abnimmt, die Genauigkeit jedoch steigt, sind die besten Klassifikationsergebnisse in der oberen rechten Diagrammhälfte zu suchen. Dort verspricht der Klassifikator hohe Genauigkeit und Präzision.

Der Klassifikator mit Merkmalsextraktionsstrategie FE_1 schneidet nach allen Lernaufgaben am schlechtesten ab. Es ist anzunehmen, dass sich hier die deut-

lich geringere Anzahl von Beispieldaten zeigt. Im Gegensatz dazu warten FE_2 und FE_3 mit bessern Kennzahlen auf. Zunächst einmal ist festzustellen, das mit Zunahme der Patchgrößen entsprechend die Trefferquote aller Klassifikatoren steigt. Eben dazu nimmt jedoch die Genauigkeit stetig ab. Ein Ausnahme stellt FE_1 mit p_{10} dar, dort ist zunächst eine Zunahme der Genauigkeit gegenüber der Trefferquote zu beobachten, diese fällt jedoch wieder ab. Die Klassifikatoren mit Merkmalsvektoren nach FE_2 weisen eine deutlich höhere Wirksamkeit auf, im Mittel ca. 8 % gegenüber FE_3. Ebenfalls auffällig ist die Stabilisierung von FE_3 hinsichtlich Trefferquote und Präzision für groß dimensionierte Merkmalsvektoren (vergleiche Abb. 6.3, 6.4). Das gleiche Verhalten ist beim Klassifikator unter FE_1 zu beobachten, wenn auch mit schlechter Zuordnungsleistung. Und es ist ein markantes Merkmal. Denn wie in Abschnitt 5.5 erläutert, werden die Zuordnungsresultate des Netzes erst ab einem Schwellwert von mindestens 75 %, also im Prognoseintervall 75 % bis 100 % gezählt. Wenn der Klassifikator also ab dem Schwellwert im Verhältnis Trefferquote zu Genauigkeit im Prognoseintervall stabil ist, führt das Verlangen nach einer strengeren, qualitativ hochwertigeren Klassifikatorprognose nicht zwangsläufig zu einer genaueren Klassifikation. Insbesondere zeigt sich das stabile Verhalten des Netzes mit Merkmalen nach FE_3 und p_{15} Patches in Abbildung 6.3. Im Gegensatz dazu verhalten sich Klassifikatoren unter Strategie FE_2. Mit einem vergleichweise niedrig angesetzten Schwellwert garantieren sie eine hohe Sicherheit. Die Wahscheinlichkeit sinkt, dass falsche Hypothesen versehentlich als positiv zugeordnet wurden.

Erhöhung des Skalierungsfaktors

Auffällig ist, dass die *HR*-Verhältniskurve unter dem erweiterten Abtastradius zur Merkmalsextraktion weniger stark abfällt. Insgesamt verschiebt sich das Ergebnis positiv hinsichtlich der Präzision. Weniger Auffällig ist der Gewinn an Treffsicherheit, wenn auch vorhanden. Hier knüpft das stabile Verhalten von FE_3 an, nimmt leicht zu und gewährleistet mehr Sicherheit, dass ein Tor erkannt wurde, wenn es als solches klassifiziert wurde. In diesem Fall zeigt sich für FE_3 jedoch eine stärkere Garantie mit Patches p_{12}. Die Präzision liegt nach Abbildung 6.6 bei 86 %.

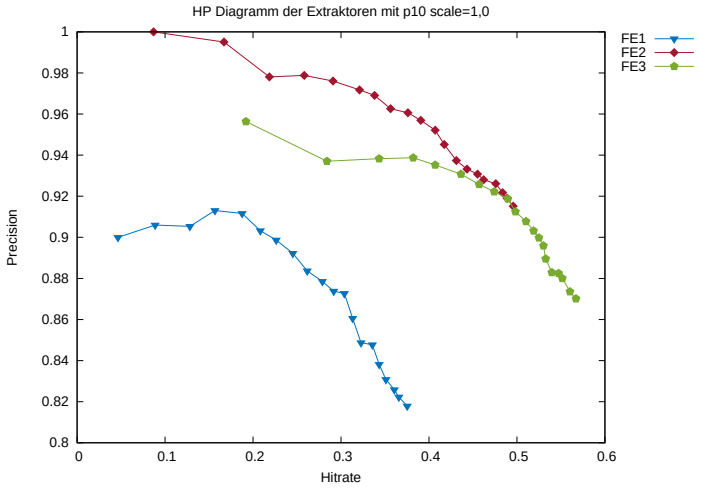


Abbildung 6.1.: Validierungsergebnis der Klassifikatoren mit p_{10}-Merkmalsvektoren

6.2.1. Differenzierung nach Objektdistanz

Als zusätzliche Bewertungsgrundlage wird, wie bereits erwähnt, das Verhalten der trainierten Klassifikatoren unter verschiedenen Distanzen zur Pfostenhypothese untersucht. Mit der Distanz korreliert entsprechend die Größe des abgebildeten Objektes, zu dem eine Hypothese vorliegt. Da die von der *NAO* Plattform gelieferten Nick- und Rollwinkel, bedingt durch Rauschen, teils ungenau sind, sind die Berechnungen betroffen. Dennoch wurden zur Untersuchung die Beschränkungen auf 5 m und 3 m gewählt. Alle Hypothesen über dieser Grenze werden zur Evaluation verworfen. Wieder wurden alle FE_i nach Standardeinstellungen und mit $s = 1.5$ evaluiert.

Die Abbildungen 6.1 bis 6.4 mit den Abbildungen A.8 bis A.23 dienen dem Vergleich. Es zeigt sich, dass mit verringerter Distanz zum Pfostenvorschlag, bei in etwa gleichbleibender Trefferquote die Wirksamkeit sinkt. Der Klassifikator nach FE_3 bleibt für die Patches p_{15} und p_{18} weiterhin stabil, eine gute Eigenschaft. Den Abbildungen zufolge lässt sich schließen, dass mit abnehmender Distanz zum Objekt die Klassifizierung bezüglich schwächelt und falsche Hypothesen als der Klasse Pfosten zuordnet. Das gilt für alle Klassifikatoren. Dies könnte dem Umstand geschuldet sein, dass der Lernaufgabe weniger Beispiele

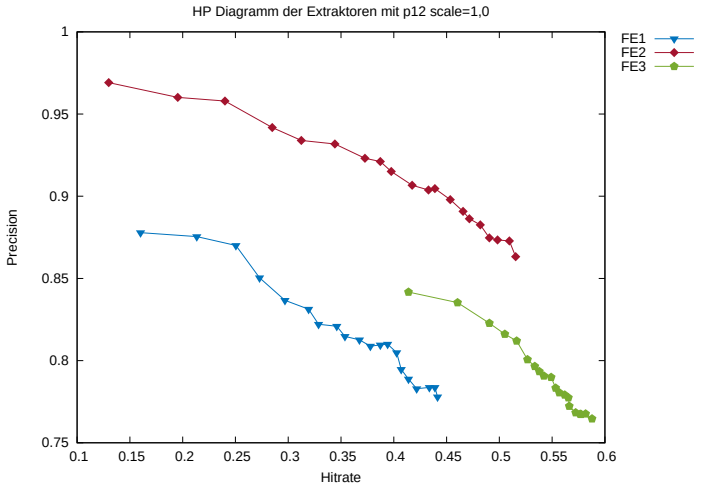


Abbildung 6.2.: Validierungsergebnis der Klassifikatoren mit p_{12}-Merkmalsvektoren

zur Verfügung standen.

6.2.2. Umverteilung der Trainingsmenge

Für eine endgültige Modellselektion und einen unbefriedigenden Generalisierungsfehler, folgt nun die Auswertung der Klassifikatoren nach einer Umverteilung der Datengrundlage. Die Art der Umverteilung wurde in Abschnitt 5.4 beschrieben. Sie schafft mehr Beispiel-, jedoch weniger Validierungsdaten, weswegen die Vorherige und nun folgende Untersuchungen nicht direkt miteinander verglichen werden können. Ein Vergleich der Abbildungen 6.1 bis 6.8 mit A.27 bis A.34 zeigt bei leichtem Rückgang der Trefferquote einen Gewinn an Genauigkeit. Dass also weniger Hypothesen fälschlich der Klasse Pfosten zugewiesen werden, ist bereits mit dem geringeren Generalisierungsfehler erklärt. Die Zunahme der Abweisung richtiger Hypothesen schlägt sich in der Hitrate nieder. Dieses Phänomen wird mit dem geänderten Testdatensatz korrelieren.

Der Vollständigkeit halber wurde auch für diese Trainingsmenge eine Untersuchung zur Leistung der Klassifikatoren hiinsichtlich der Objektgröße unternommen. Dazu sei auf die Abbildungen A.35 bis A.42 verwiesen. Eine erneute Ana-

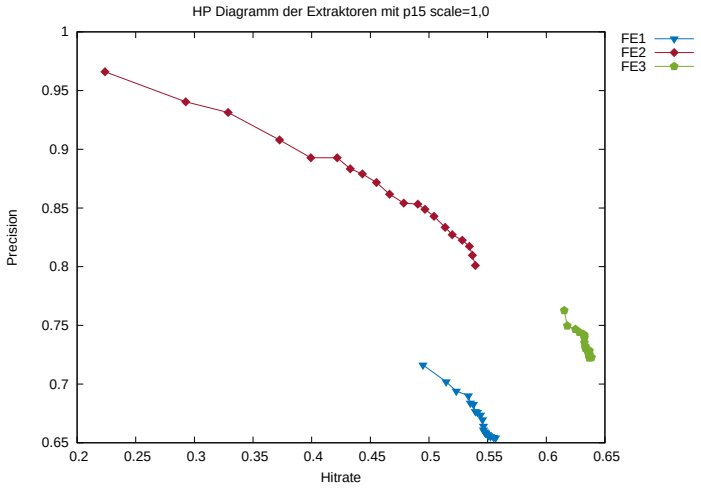


Abbildung 6.3.: Validierungsergebnis der Klassifikatoren mit p_{15}-Merkmalsvektoren

lyse des Klassifikatoverhaltens hinsichtlich der Distanz zum Objektvorschlag, soll aufzeigen, ob die Klassifikationsleistung mit der Menge an Beispieldaten zunimmt. Zunächst ist eine positive Veränderung in der Wirksamkeit zu beobachten. Tatsächlich ist jedoch bezüglich der Distanz keine Veränderung gegenüber dem ersten Untersuchungsdurchlauf auszumachen. Die Annahme über eine ungenügende Menge an Beispieldaten konnte nicht bestätigt werden. Vielmehr scheinen die Merkmale von Objekten in geringer Distanz durch die Extraktoren weniger gut wiedergegeben zu werden. Hier besteht Optimierungspotential.

6.3. Zusammenfassung

Es zeigt sich, dass die Klassifikatoren durchaus eine hohe Präzision aufweisen können. Um dem Weltmodell qualitativ hochwertige Informationen zur Verfügung zu stellen, ist ein Klassifikator mit hoher Präzision zu wählen. Da FE_2 stets wenig falsche Klassifizierungen aufweist, ist er die erste Wahl. Im Falle der Patchgröße p_{15} kann er bereits mit einer Genauigkeit von über 90 % aufwarten. Dies bereits ab einer prognostizierten Wahrscheinlichkeit von 75 %. Mit Mermalsvektoren der Form p_10 ist die Genauigkeit gar noch höher, mit wenig

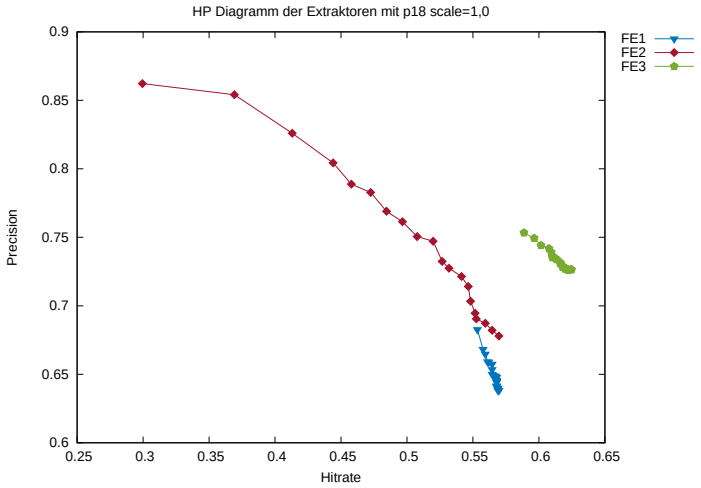


Abbildung 6.4.: Validierungsergebnis der Klassifikatoren mit p_{18}-Merkmalsvektoren

Kompromiss bezüglich der Empfindlichkeit. Um eine höhere Trefferquote zu erzielen, mit Verzicht auf etwas Präzision zeigt auch FE_3 mit Patches p_{10} eine gute Performance.

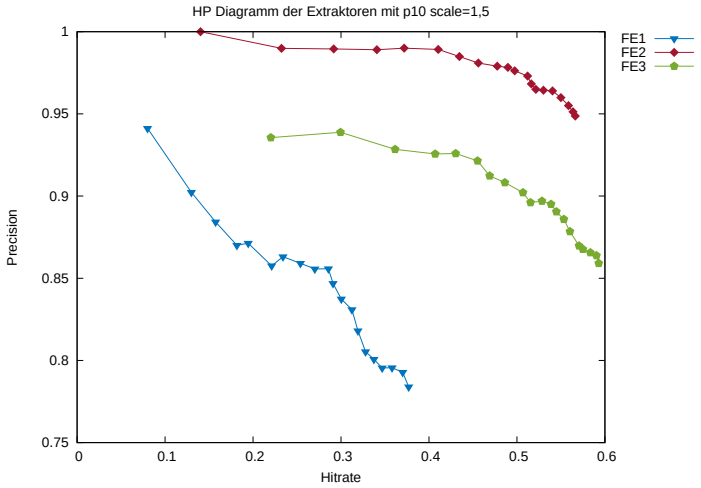


Abbildung 6.5.: Validierungsergebnis der Klassifikatoren mit p_{10}-Merkmalsvektoren unter erweitertem Abtastradius

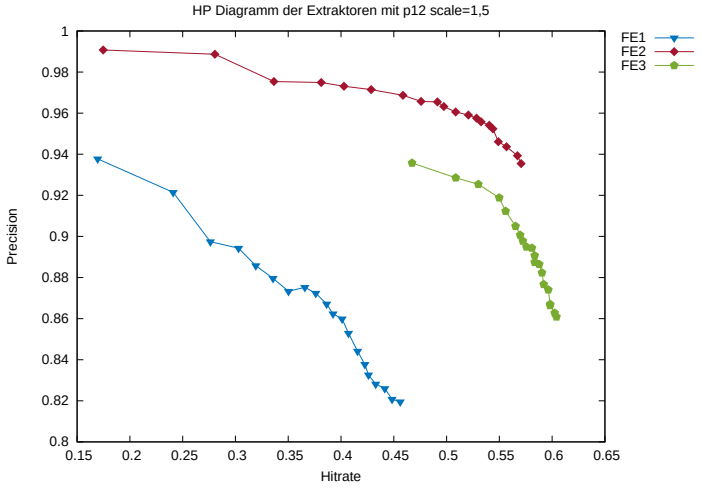


Abbildung 6.6.: Validierungsergebnis der Klassifikatoren mit p_{12}-Merkmalsvektoren unter erweitertem Abtastradius

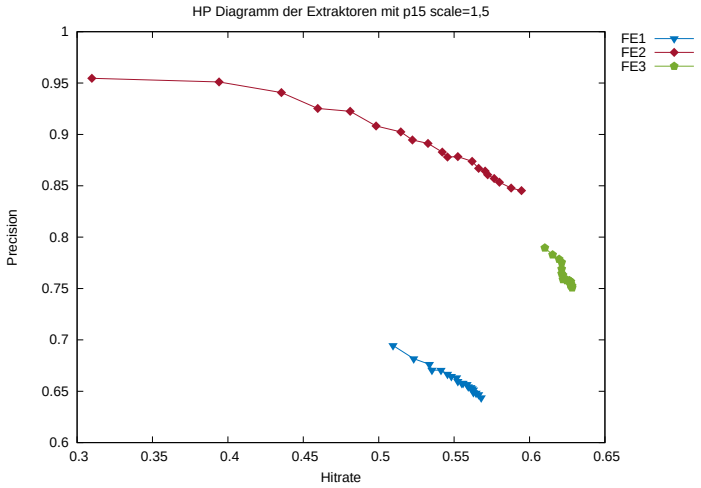


Abbildung 6.7.: Validierungsergebnis der Klassifikatoren mit p_{15}-Merkmalsvektoren unter erweitertem Abtastradius

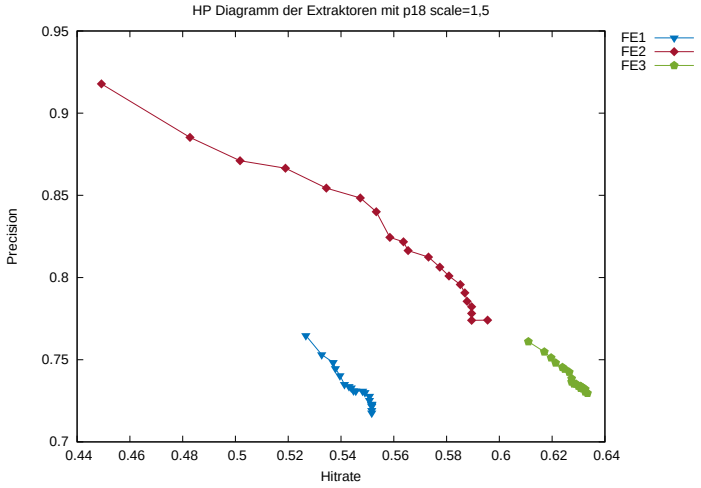


Abbildung 6.8.: Validierungsergebnis der Klassifikatoren mit p_{18}-Merkmalsvektoren unter erweitertem Abtastradius

7. Fazit

Auf der Suche nach einem geeigneten Klassifikator wurden verschiedene Methoden zur Extraktion von Bildmerkmalen vorgestellt. Die Ergebnisse zeigen, dass eine Annäherung an das Ziel, Torerkennung im Roboterfußball, möglich ist.

Es konnte gezeigt werden, dass die Methoden des maschinellen Lernens gute Möglichkeiten bieten, Resultate der Bildverarbeitung durch Klassifizierung zu ergänzen. Speziell wurde in dieser Untersuchung eine Eignung der Methode des überwachten Lernens diskutiert. Der Algorithmus *THG* zur Registrierung der Torpfosten liefert einem trainierten Klassifikator gute Zuarbeit. Nach den Testergebnissen liefert er zunächst Hypothesen für mehr als 80 % der Pfostenmarkierungen aus, die zur Überwachung der Lernaufgabe bereitgestellt wurden. Der Umfang ermittelter Hypothesen, übertrifft die Anzahl bereitgestellter Markierungen um den Faktor 3,76 (s. Tab. 6).

Mit Hilfe eines *Convolutional Neural Network* konnte ein neuronales Netz trainiert werden, das mit einer Trefferquote von 52 % einen abgebildeten Torpfosten als solchen klassifiziert. Gesetz dem Fall einer positiven Klassifizierung, besteht nach der ermittelten Wirksamkeit des Klassifikators eine Wahrscheinlichkeit von 95 % aufwärts, dass es sich tatsächlich um einen Pfosten handelt. In Anbetracht der riesigen Menge an Vorschlägen zu potentiellen Torpfonsten durch den Erkennungsalgorithmus, ist das *CNN* bezüglich der Selektierung ein Zugewinn.

Wie sich zeigt, ist es wichtig, zur Evaluation und Modellselektion die Testmenge möglichst beizubehalten. Im besten Falle sollte das Verhältnis angepasst werden. So ist eine problemfreie, vergleichende Untersuchung trainierter Klassifikatoren gewährleistet.

Aussicht

Im Laufe der zukünftigen Turniere vergrößert sich der Bilderpool von Spielsituationen. Dieser Bestand ist eine gute Voraussetzung dafür, bestehende, sowie neue Probleme als Lernaufgaben zu spezifizieren. Klassifikatoren können durch mehr Beispiele ihre Genauigkeit steigern. Damit stehen dem Weltmodell der Strategie und der Lokalisierung zugesicherte Informationen bereit, die einen Spielverlauf prägen können.

Glossar

Caffe Framework für das Definieren, Trainieren und Integrieren Neuronaler Netze [7]

Data Mining Systematische Anwendung statistischer Methoden auf große Datenbestände

Edgel Kunstwort aus Edge (*Kante*) und Pixel

Fully Connected Alle Neuronen einer Schicht werden jeweils mit allen Neuronen der Folgeschicht verknüpft

Key-Value-Store Ein Paar bestehend aus einem Schlüssel und einem zugehörigen Wert. Durch Angabe des Schlüssels kann der Wert referenziert werden. Diese Abbildung ist die einfachste Möglichkeit der Persistierung

Label Markierung bzw. Kennzeichnung

RoboCup Internationales Turnier in dem autonom agierende Roboter in verschiedenen Disziplinen Spiele oder besondere Aufgaben erledigen.

Snapshot Ein Schnappschuss speichert den Zustand eines Prozesses zum Zeitpunkt der Betrachtung

Spam Eine elektronische Mail, die vom Empfänger als unverlangt eingestuft wird [22]

Schrittweite Beim Abarbeiten einer Struktur mit Index, z.B. einer Liste, wird statt jedem Element mit einer Schrittweite n nur jedes n-te Element betrachtet

Abkürzungsverzeichniss

CNN Convolutional Neuronal Network

FB Filterbank

FIFA Fédération Internationale de Football Association (deutsch Internationaler Verband des Association Football) [18]

LMDB Lightning Memory-Mapped Database

NAO [19]

NN Neuronales Netz (*engl. Neural Network*)

PNG Portable Network Graphics [20]

ReLU Rectified Linear Unit

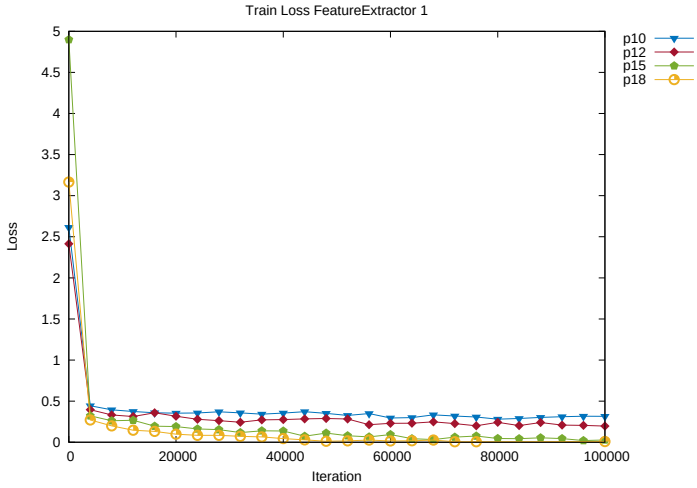
ROI Region Of Interest

SPL Standard Plattform League

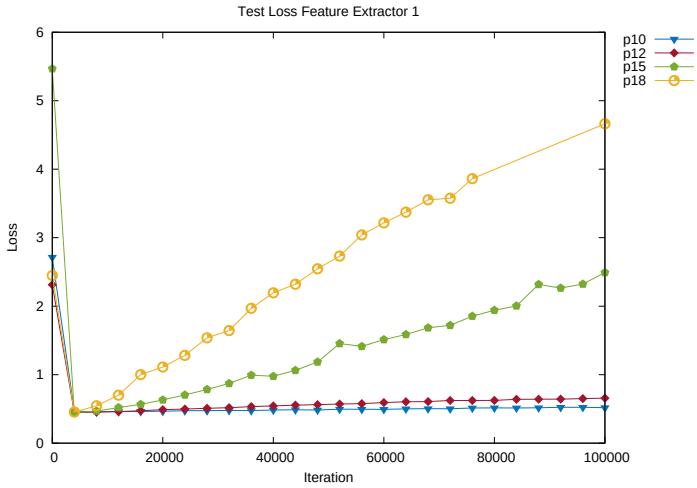
Anhang A.

Referenzierte Abbildungen

A.1. Abbildungen: Auswertung des Lernverfahrens

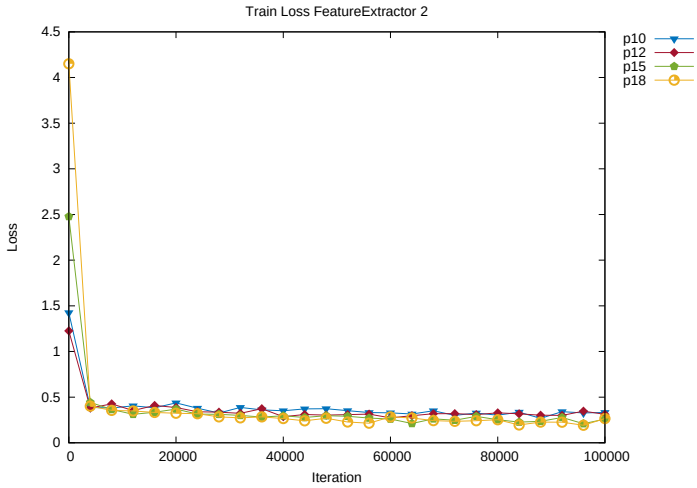


(a) Lernkurve

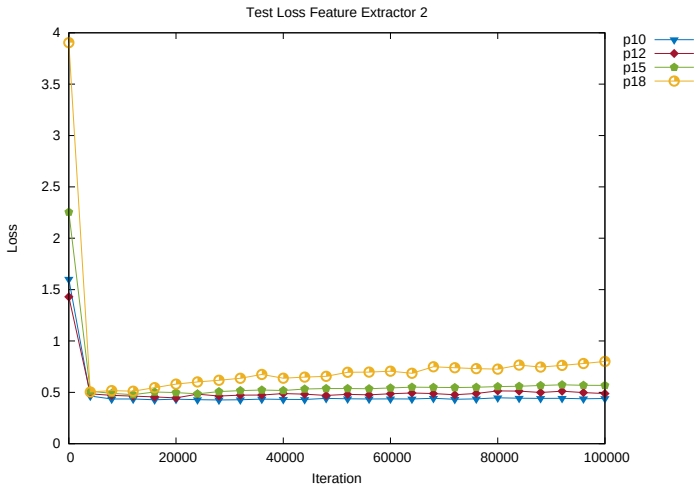


(b) Generalisierungsfehler

Abbildung A.1.: Visualisierung der Lernaufgabe durch FE_1

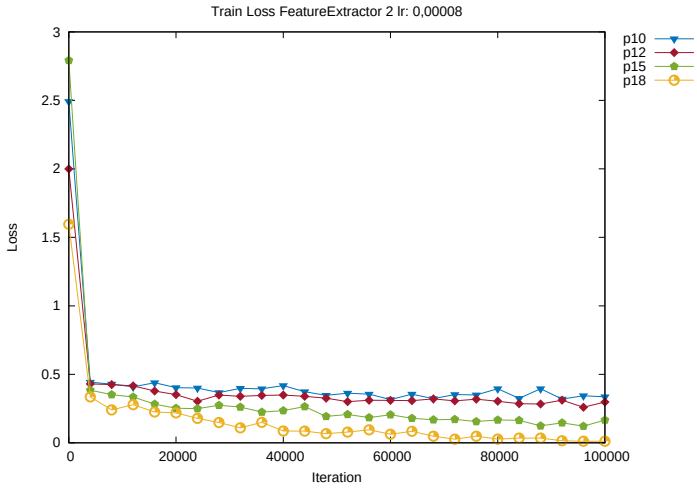


(a) Lernkurve

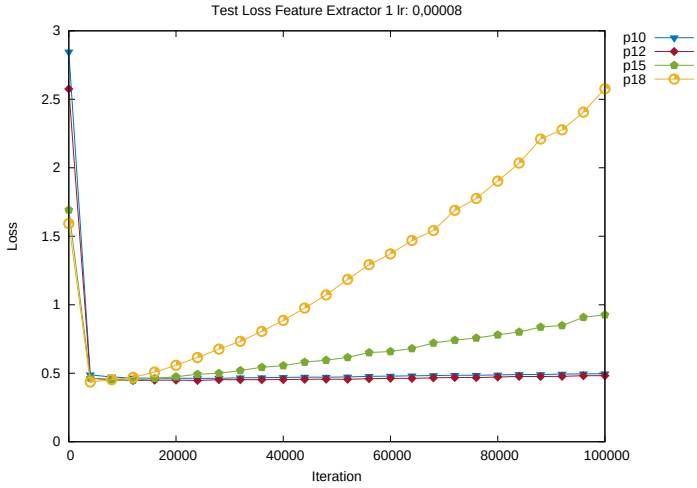


(b) Generalisierungsfehler

Abbildung A.4.: Visualisierung der Lernaufgabe durch FE_2

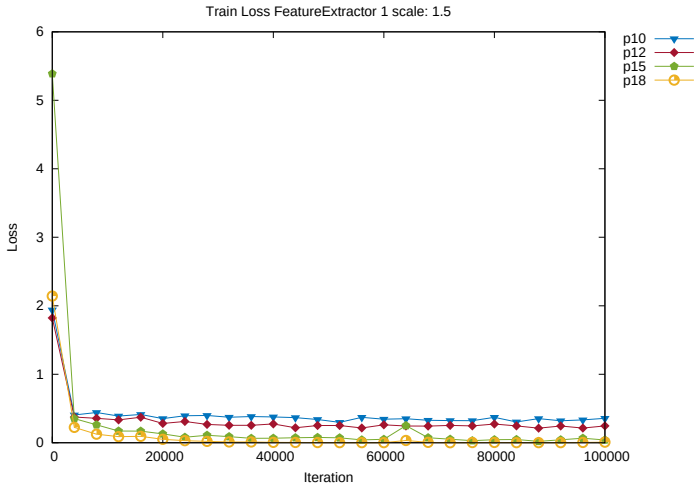


(a) Lernkurve

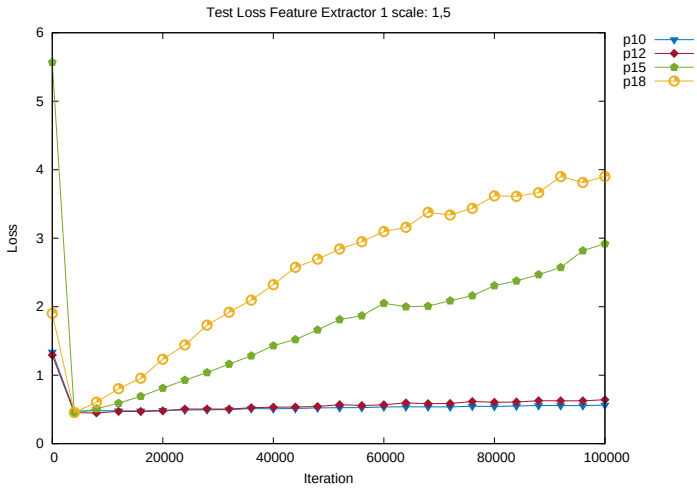


(b) Generalisierungsfehler

Abbildung A.2.: Visualisierung der Lernaufgabe durch FE_1 unter Lernrate $\lambda = 8 \cdot 10^{-5}$

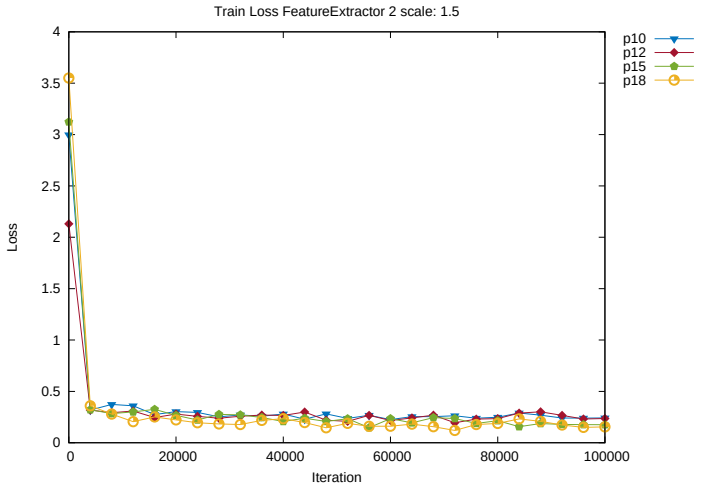


(a) Lernkurve

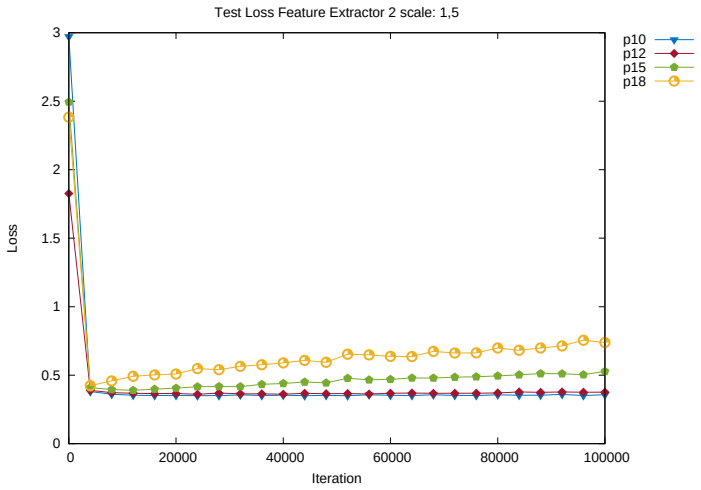


(b) Generalisierungsfehler

Abbildung A.3.: Visualisierung der Lernaufgabe durch FE_1 mit vergrößertem Abtastradius

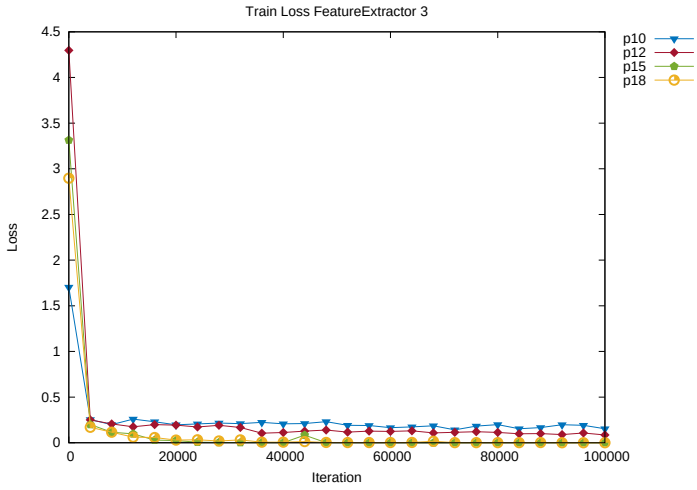


(a) Lernkurve

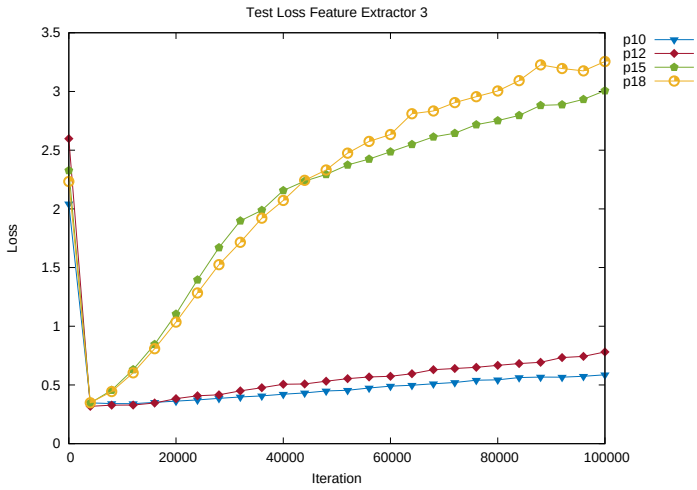


(b) Generalisierungsfehler

Abbildung A.5.: Visualisierung der Lernaufgabe durch FE_2 mit vergrößertem Abtastradius

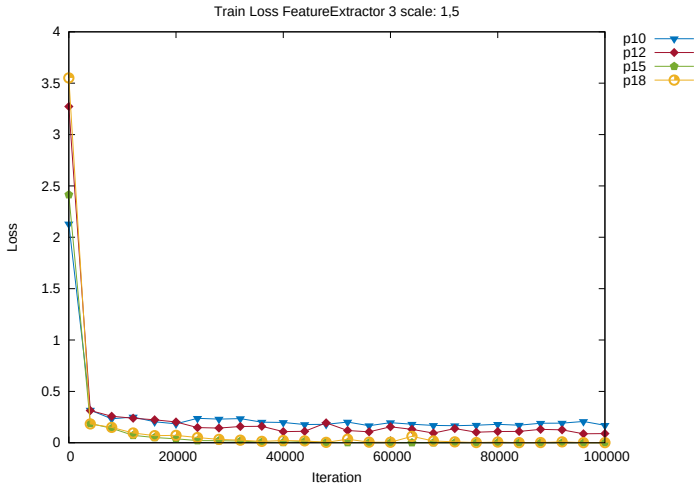


(a) Lernkurve

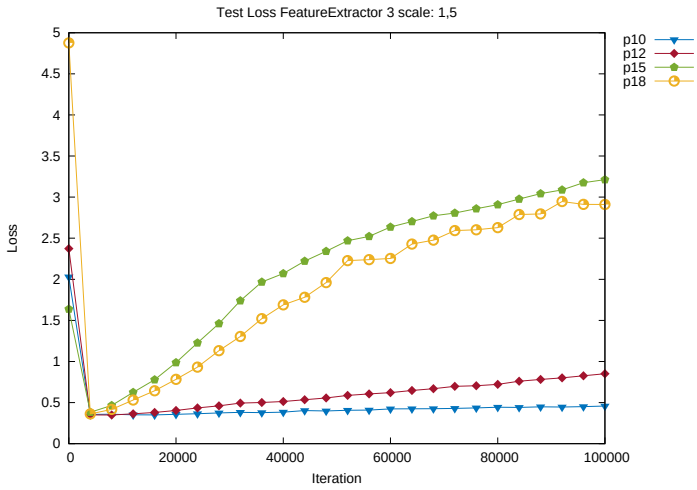


(b) Generalisierungsfehler

Abbildung A.6.: Visualisierung der Lernaufgabe durch FE_3 mit zentriertem Objektpunkt



(a) Lernkurve



(b) Generalisierungsfehler

Abbildung A.7.: Visualisierung der Lernaufgabe durch FE_3 mit zentriertem Objektpunkt und vergrößertem Abtastradius

A.2. Abbildungen: Auswertung nach Distanz

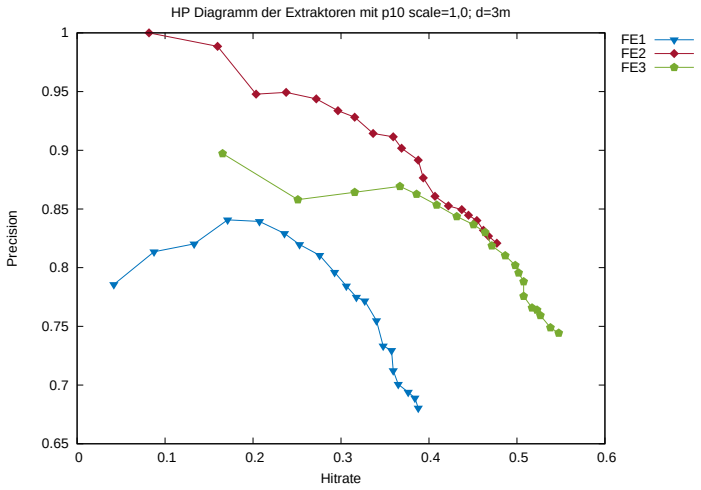


Abbildung A.8.: Performance p_{10}, Distanz 3 m

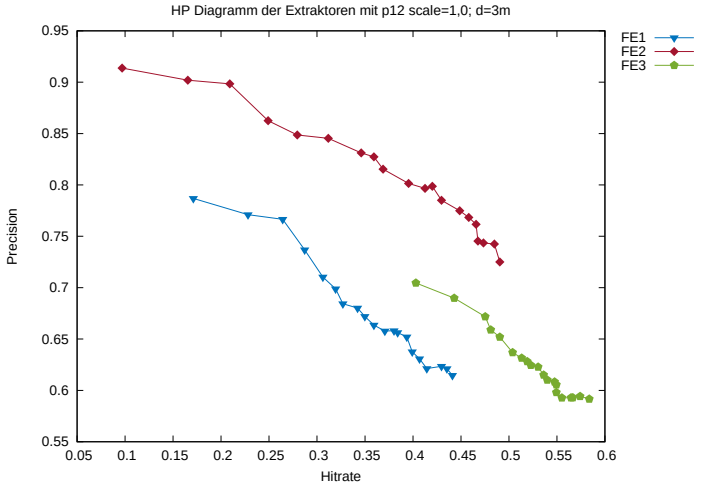


Abbildung A.9.: Performance p_{12}, Distanz 3 m

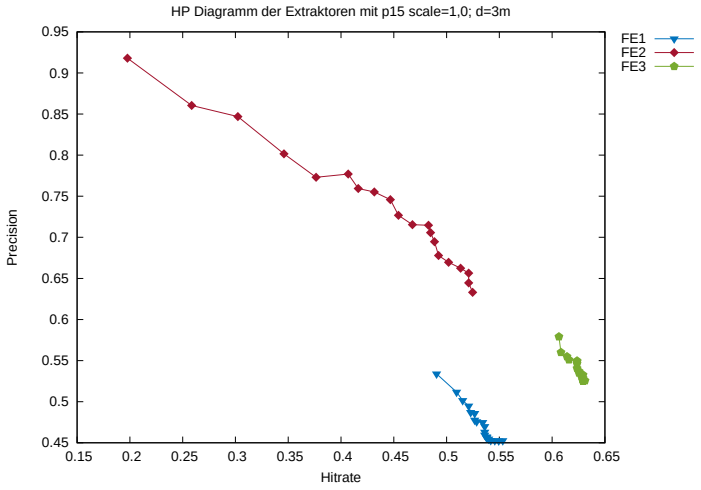


Abbildung A.10.: Performance p_{15}, Distanz 3 m

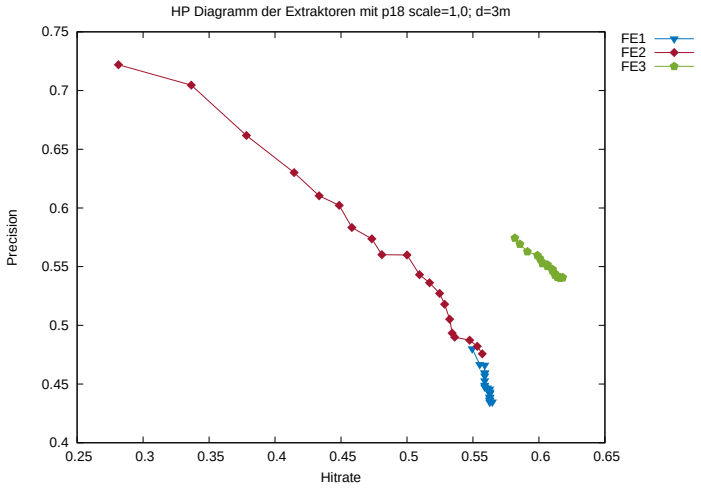


Abbildung A.11.: Performance p_{18}, Distanz 3 m

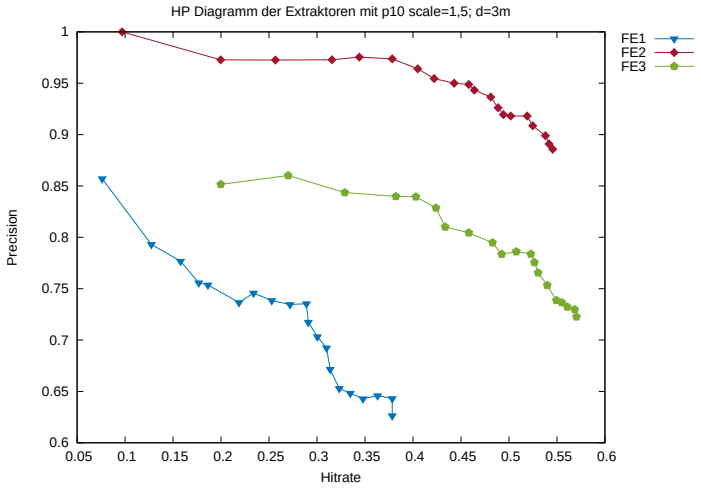


Abbildung A.12.: Performance p_{10}, Distanz 3 m, erweiterter Abtastradius

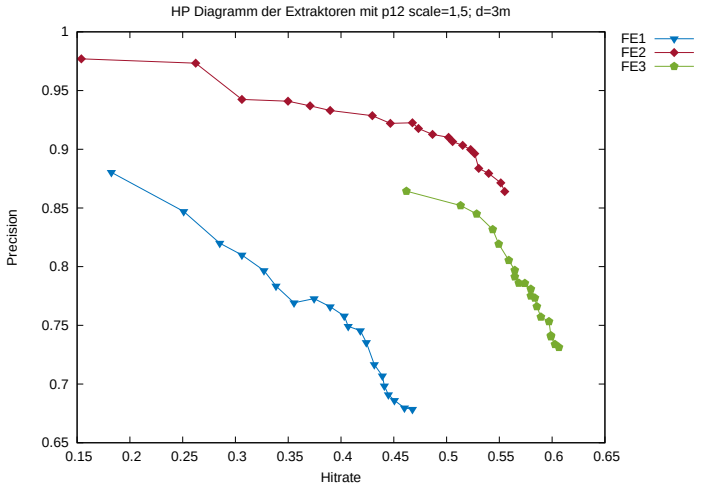


Abbildung A.13.: Performance p_{12}, Distanz 3 m, erweiterter Abtastradius

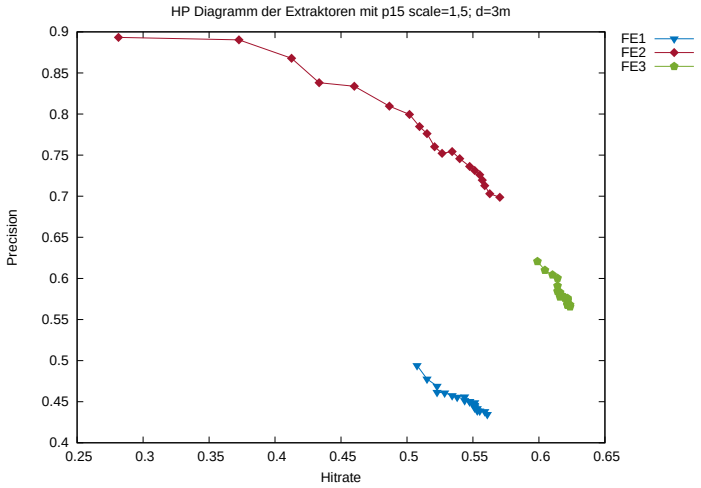


Abbildung A.14.: Performance p_{15}, Distanz 3 m, erweiterter Abtastradius

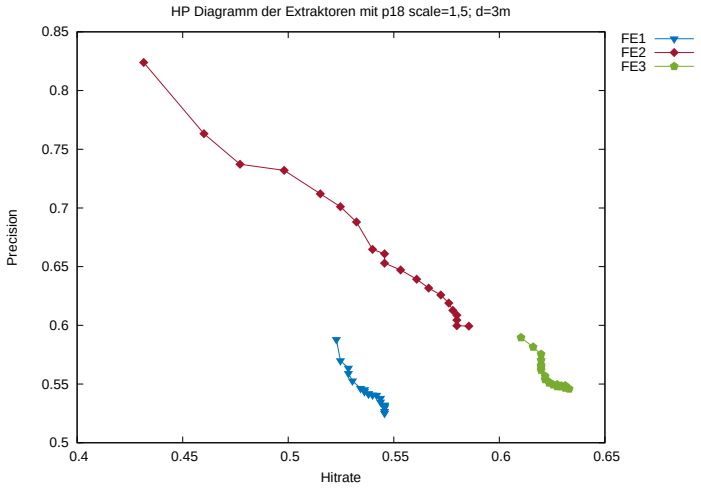


Abbildung A.15.: Performance p_{18}, Distanz 3 m, erweiterter Abtastradius

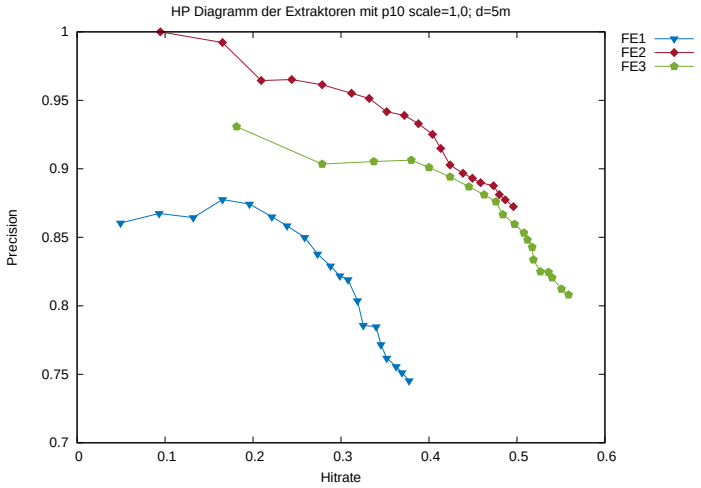


Abbildung A.16.: Performance p_{10}, Distanz 5 m

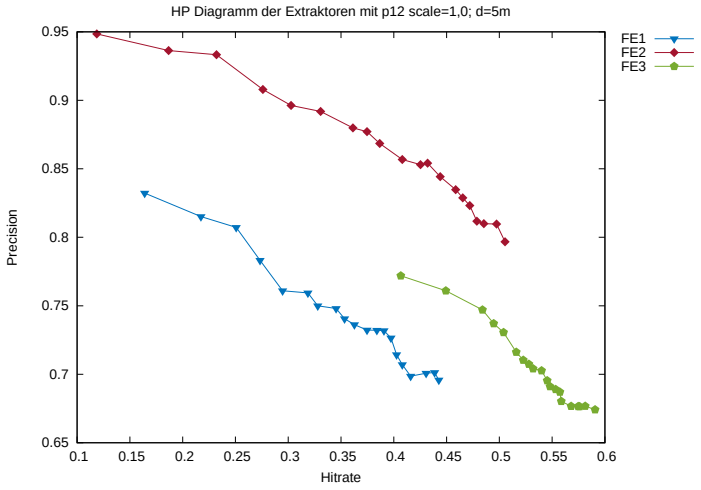


Abbildung A.17.: Performance p_{12}, Distanz 5 m

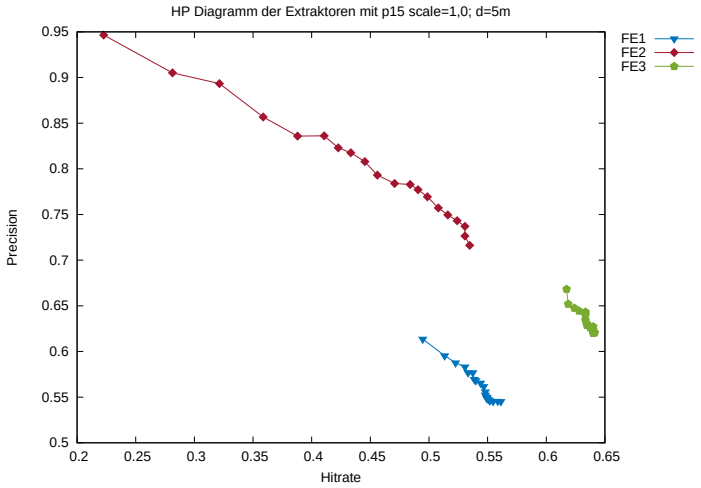


Abbildung A.18.: Performance p_{15}, Distanz 5 m

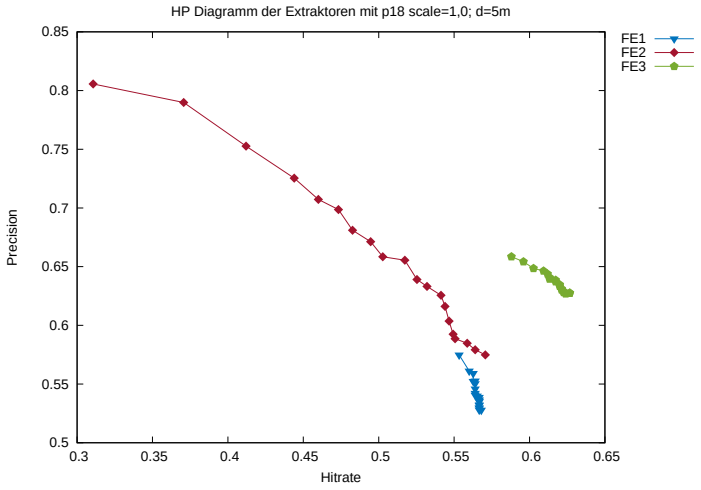


Abbildung A.19.: Performance p_{18}, Distanz 5 m

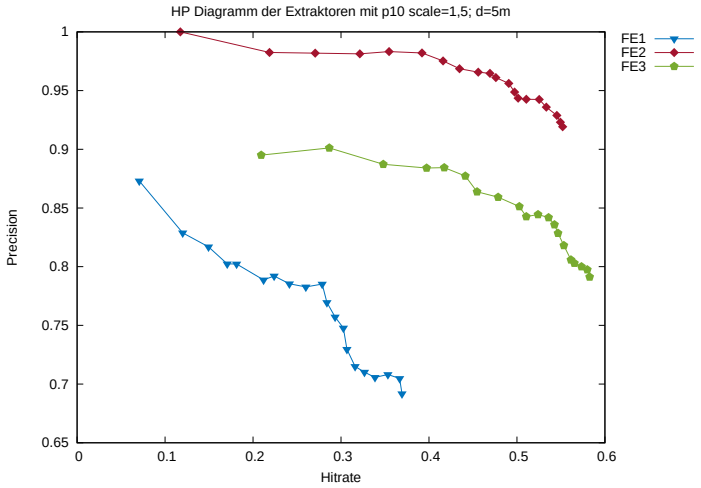


Abbildung A.20.: Performance p_{10}, Distanz 5 m, erweiterter Abtastradius

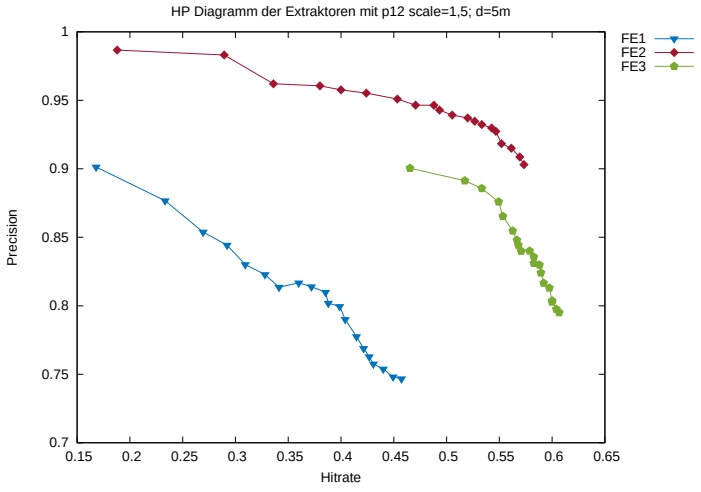


Abbildung A.21.: Performance p_{12}, Distanz 5 m, erweiterter Abtastradius

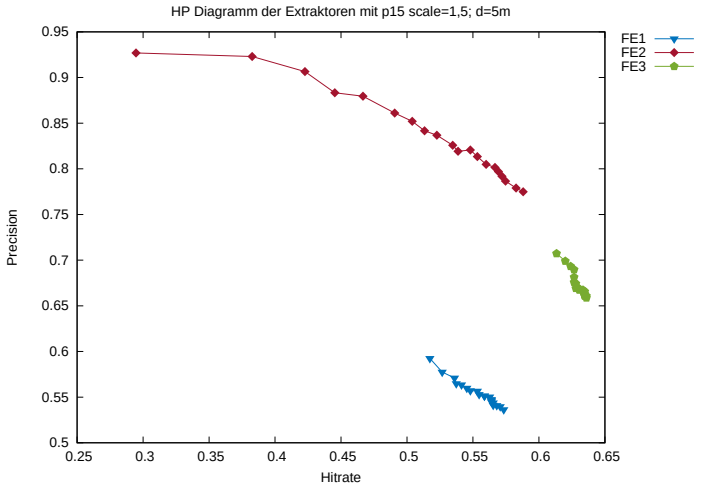


Abbildung A.22.: Performance p_{15}, Distanz 5 m, erweiterter Abtastradius

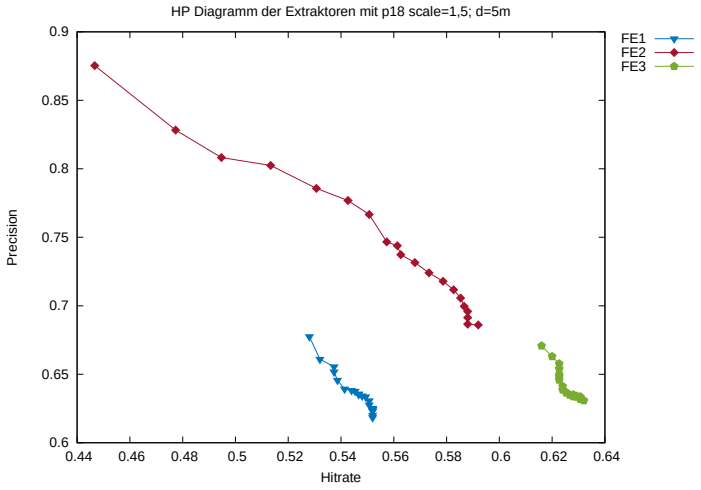


Abbildung A.23.: Performance p_{18}, Distanz 5 m, erweiterter Abtastradius

A.3. Abbildungen: Umverteilung der Trainingsdaten

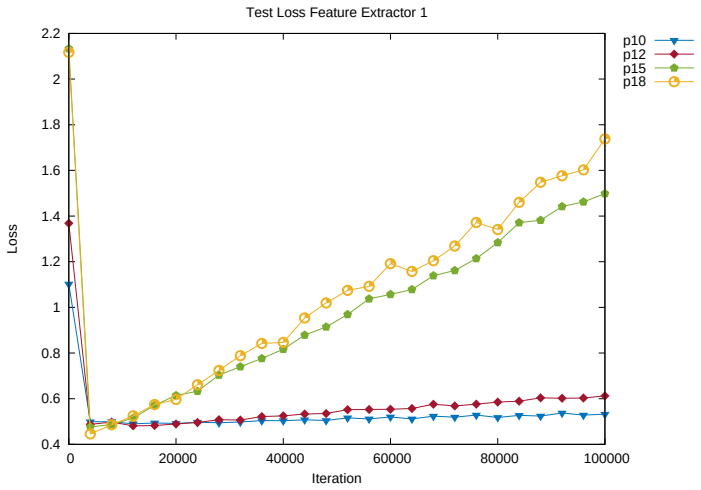


Abbildung A.24.: Testergebnis FE_1 nach Umverteilung der Trainingsdaten

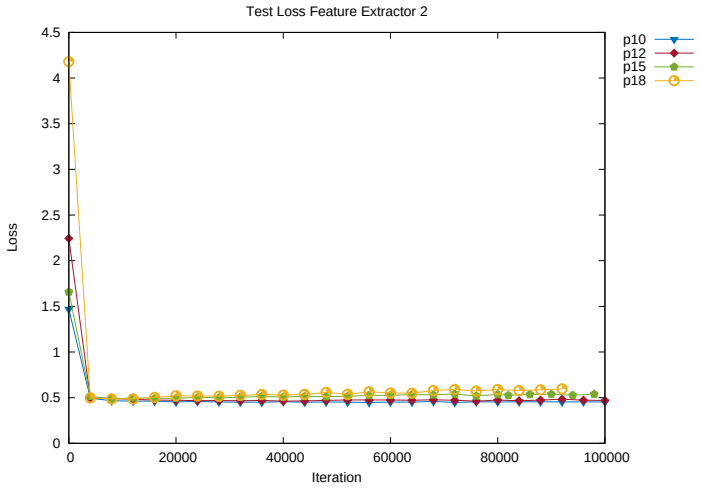


Abbildung A.25.: Testergebnis FE_2 nach Umverteilung der Trainingsdaten

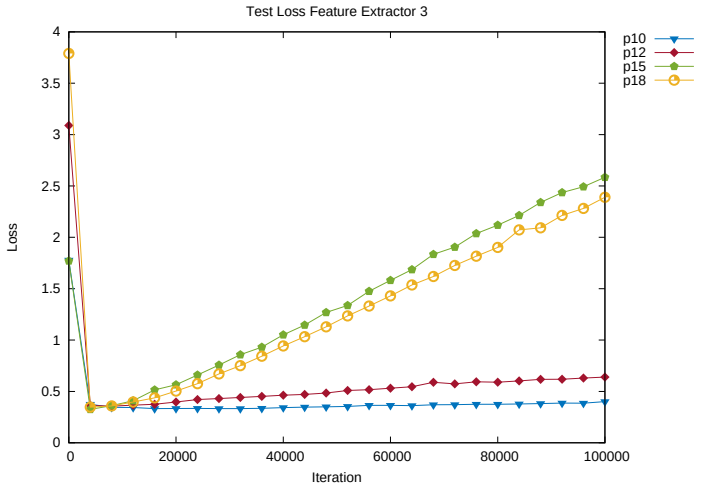


Abbildung A.26.: Testergebnis FE_3 nach Umverteilung der Trainingsdaten

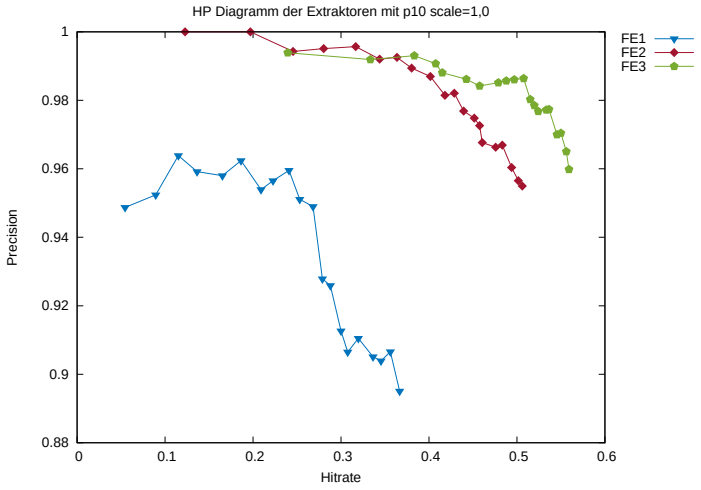


Abbildung A.27.: Performance FE_i, p_{10} nach Umverteilung der Trainingsdaten

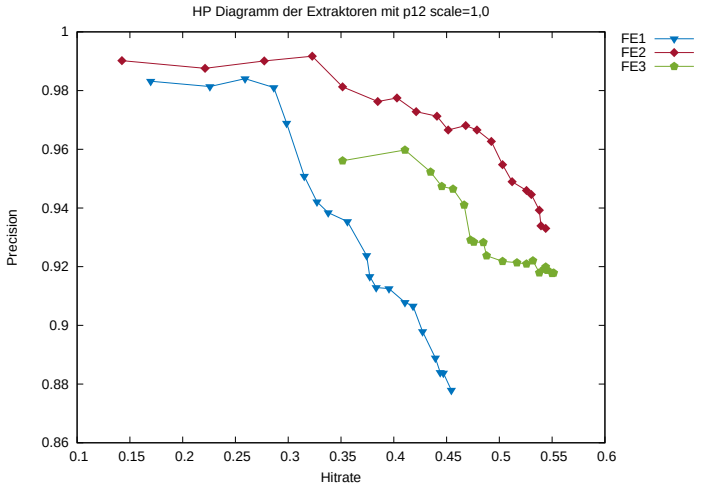


Abbildung A.28.: Performance FE_i, p_{12} nach Umverteilung der Trainingsdaten

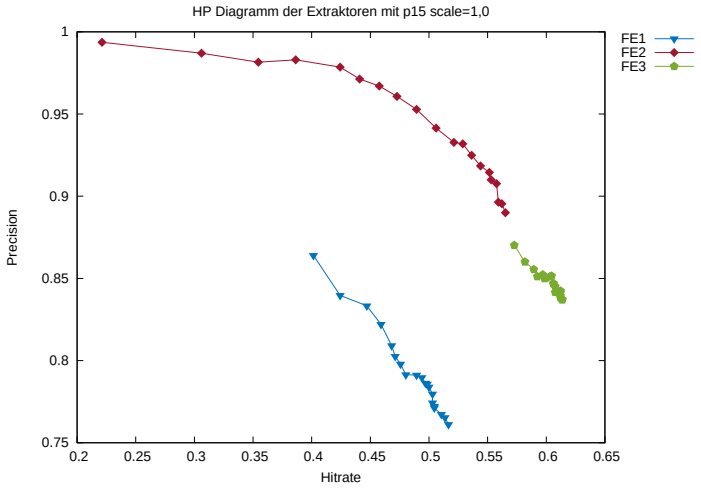


Abbildung A.29.: Performance FE_i, p_{15} nach Umverteilung der Trainingsdaten

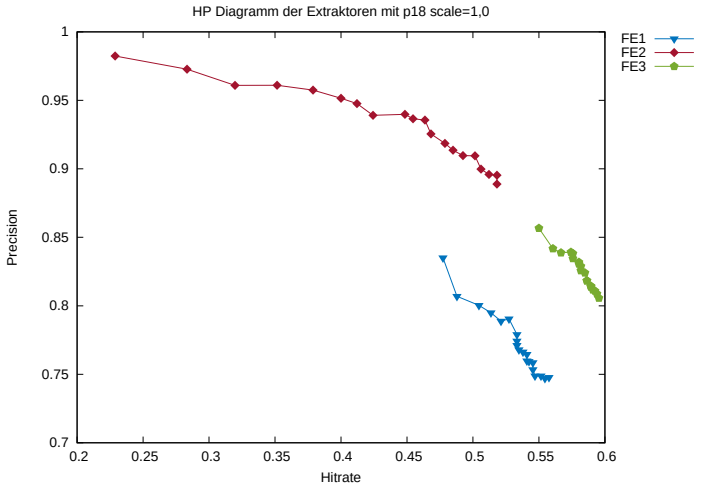


Abbildung A.30.: Performance FE_i, p_{18} nach Umverteilung der Trainingsdaten

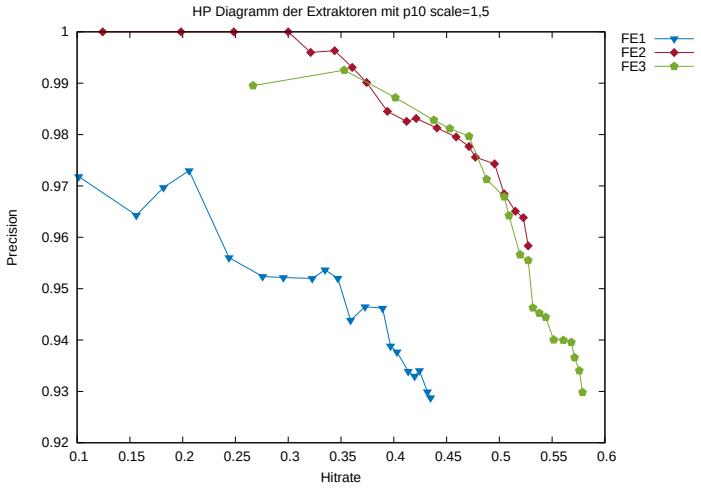


Abbildung A.31.: Performance FE_i, p_{15}, $s = 1,5$ nach Umverteilung der Trainingsdaten

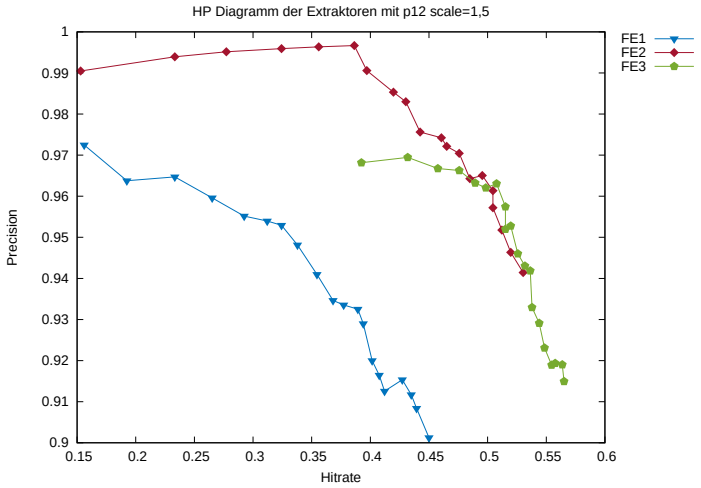


Abbildung A.32.: Performance $FE_i, p_{18}, s = 1,5$ nach Umverteilung der Trainingsdaten

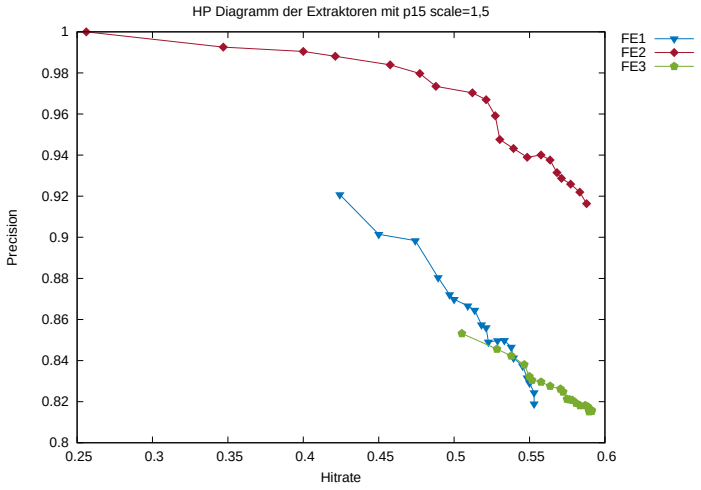


Abbildung A.33.: Performance $FE_i, p_{15}, s = 1,5$ nach Umverteilung der Trainingsdaten

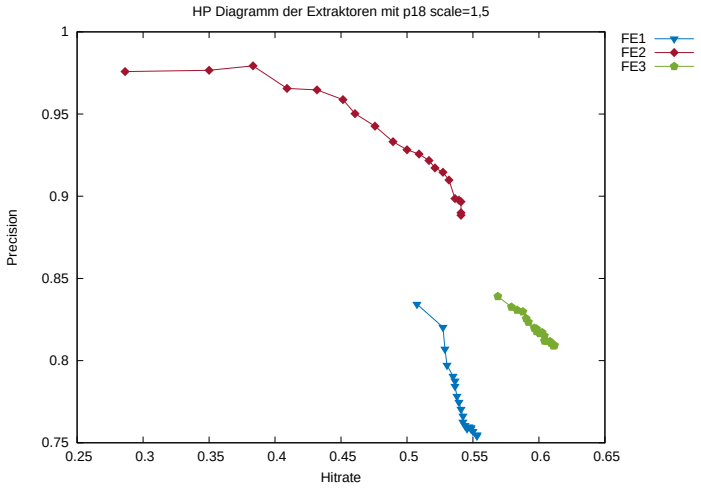


Abbildung A.34.: Performance FE_i, p_{18}, $s = 1,5$ nach Umverteilung der Trainingsdaten

Umverteilte Trainingsdaten, Auswertung nach Distanz

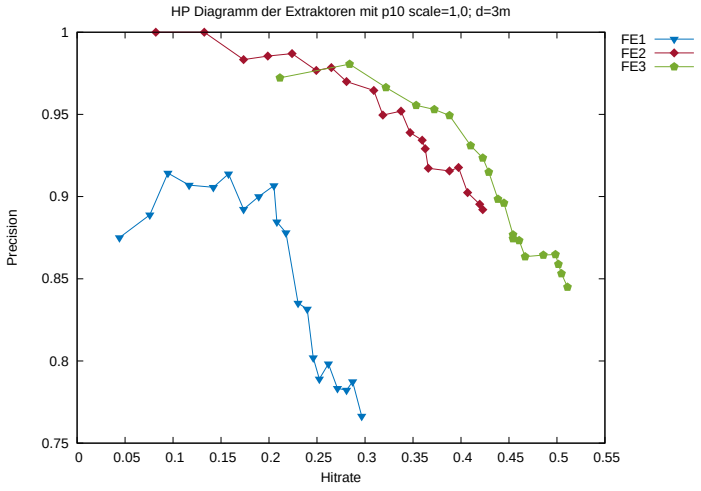


Abbildung A.35.: Performance p_{10}, Distanz 3 m nach Umverteilung der Trainingsdaten

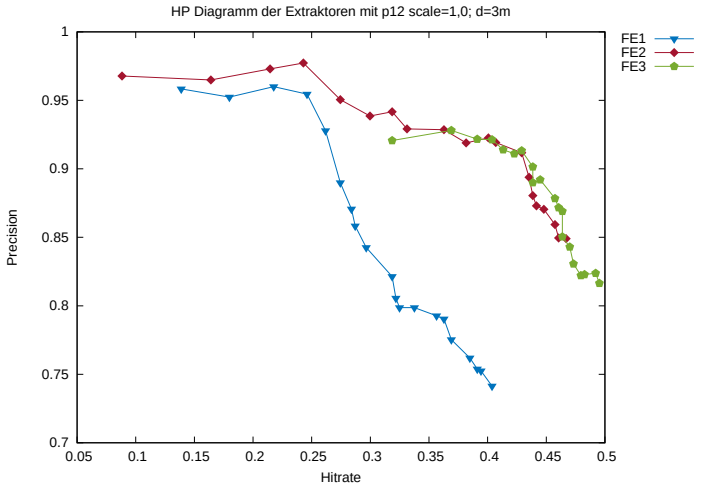


Abbildung A.36.: Performance p_{12}, Distanz 3 m nach Umverteilung der Trainingsdaten

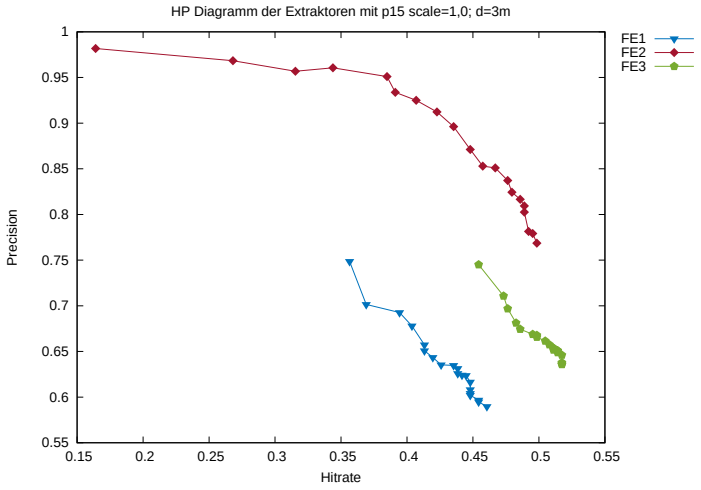


Abbildung A.37.: Performance p_{15}, Distanz 3 m nach Umverteilung der Trainingsdaten

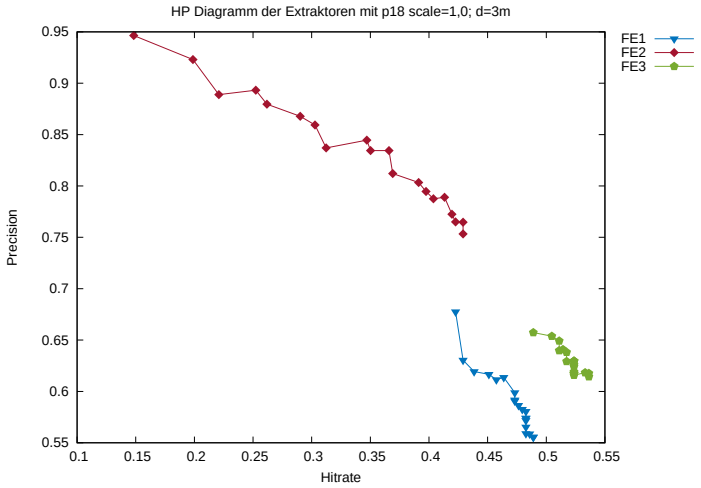


Abbildung A.38.: Performance p_{18}, Distanz 3 m nach Umverteilung der Trainingsdaten

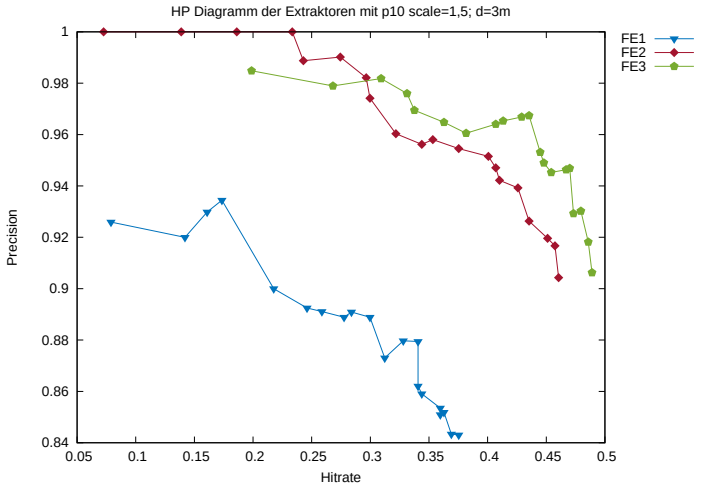


Abbildung A.39.: Performance p_{10}, Distanz 3 m, erweiterter Abtastradius nach Umverteilung der Trainingsdaten

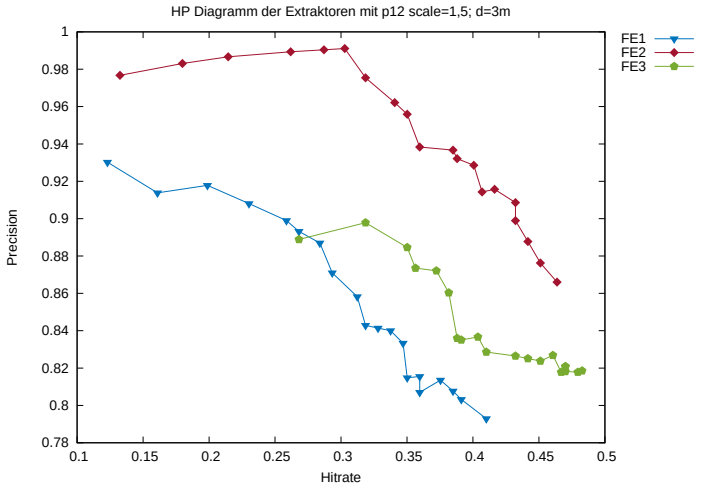


Abbildung A.40.: Performance p_{12}, Distanz 3 m, erweiterter Abtastradius nach Umverteilung der Trainingsdaten

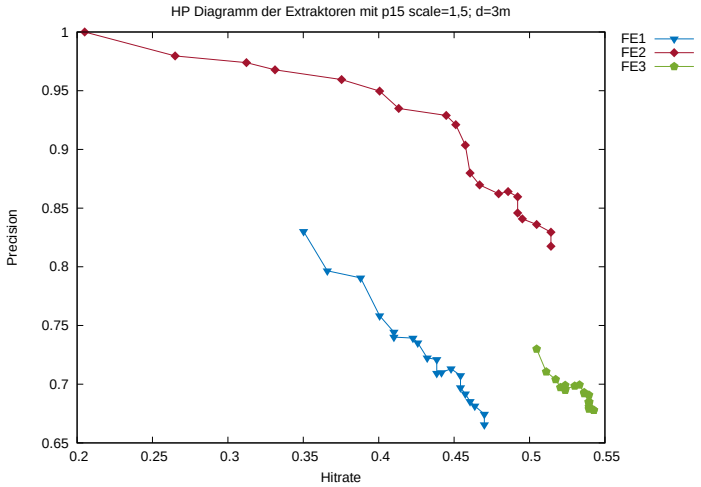


Abbildung A.41.: Performance p_{15}, Distanz 3 m, erweiterter Abtastradius nach Umverteilung der Trainingsdaten

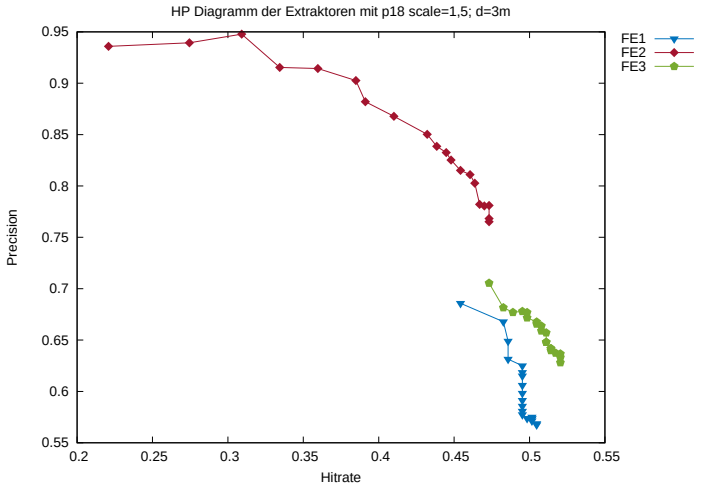


Abbildung A.42.: Performance p_{18}, Distanz 3 m, erweiterter Abtastradius nach Umverteilung der Trainingsdaten

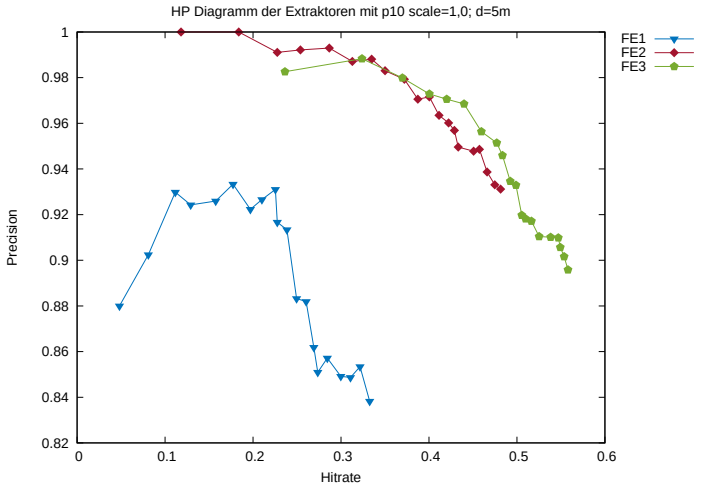


Abbildung A.43.: Performance p_{10}, Distanz 5 m nach Umverteilung der Trainingsdaten

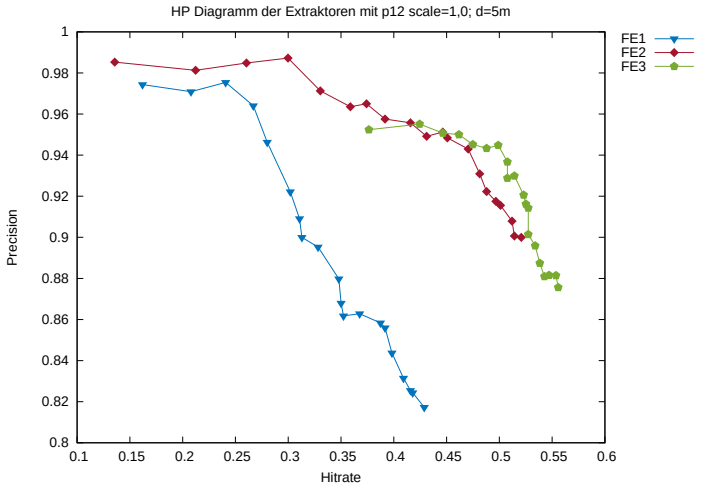


Abbildung A.44.: Performance p_{12}, Distanz 5 m nach Umverteilung der Trainingsdaten

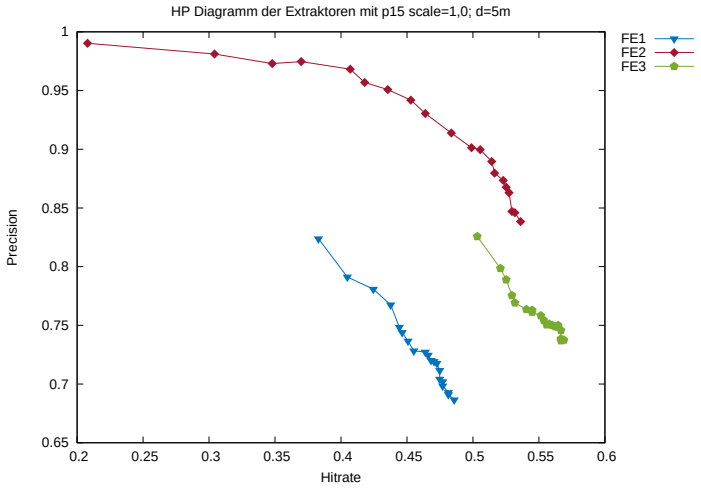


Abbildung A.45.: Performance p_{15}, Distanz 5 m nach Umverteilung der Trainingsdaten

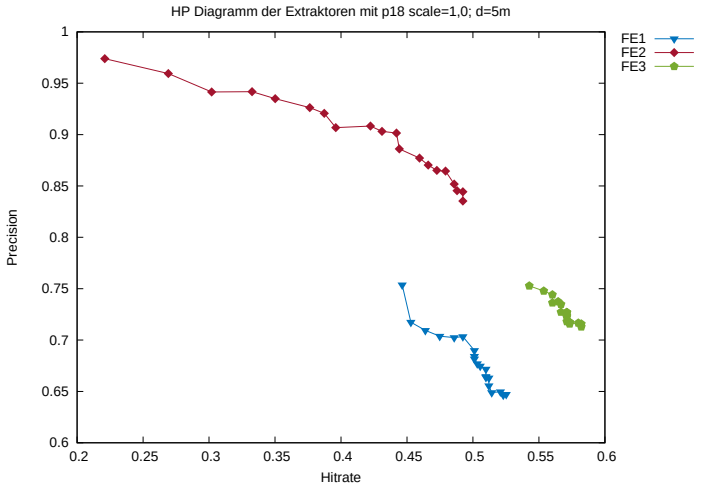


Abbildung A.46.: Performance p_{18}, Distanz 5 m nach Umverteilung der Trainingsdaten

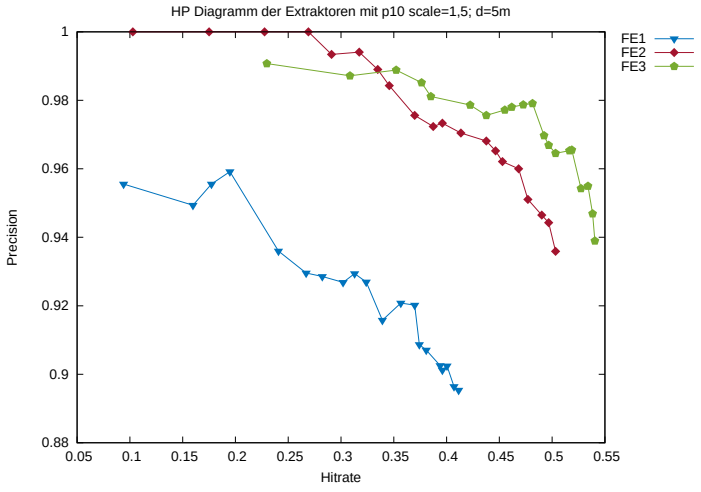


Abbildung A.47.: Performance p_{10}, Distanz 5 m, erweiterter Abtastradius nach Umverteilung der Trainingsdaten

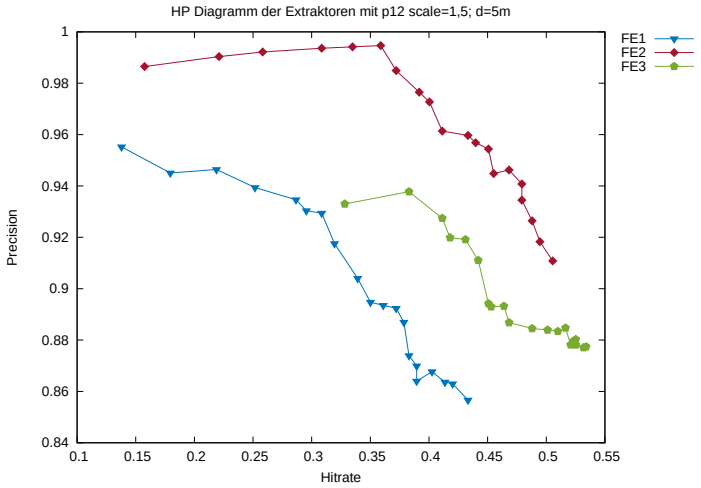


Abbildung A.48.: Performance p_{12}, Distanz 5 m, erweiterter Abtastradius nach Umverteilung der Trainingsdaten

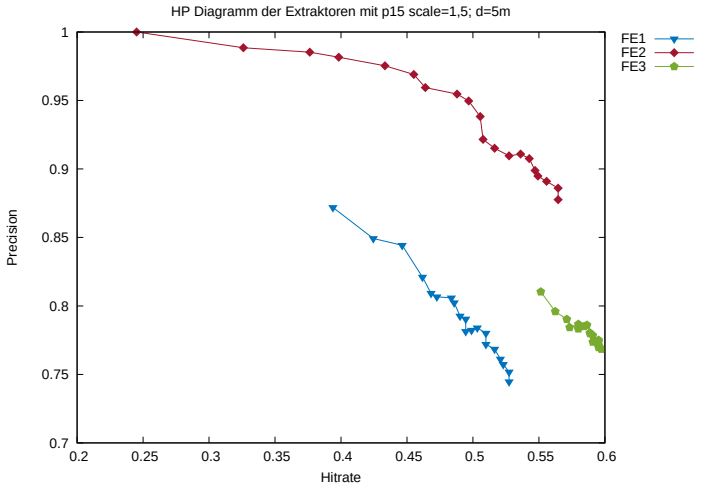


Abbildung A.49.: Performance p_{15}, Distanz 5 m, erweiterter Abtastradius nach Umverteilung der Trainingsdaten

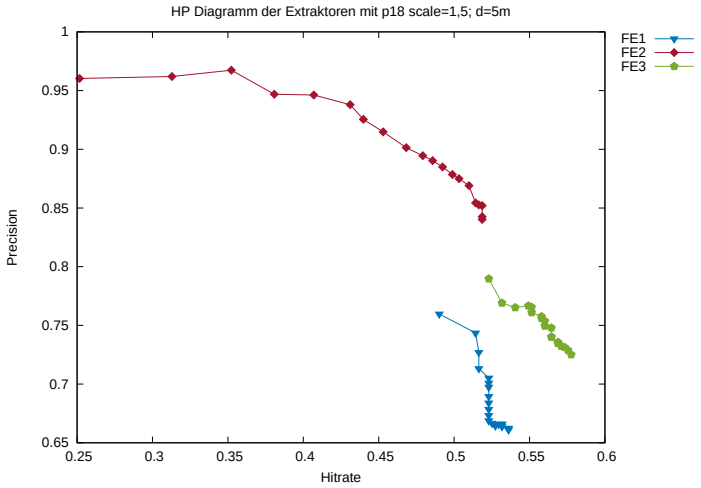


Abbildung A.50.: Performance p_{18}, Distanz 5 m, erweiterter Abtastradius nach Umverteilung der Trainingsdaten

Literaturverzeichnis

[1] RoboCup Technical Committee. Robocup standard platform league (nao) rule book, 2016. [Online; Stand 23. April 2017]. 4

[2] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14(3):326–334, June 1965. 19

[3] Philipp Fischer. *Convolutional networks to relate images*. PhD thesis, Freiburg im Breisgau,, 2016. 13, 15, 16, 41

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. 42

[5] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey J. Gordon and David B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011. 24

[6] Ute Schmid Günther Görz, Josef Schneeberger. Handbuch der künstlichen intelligenz. *eBook-Paket OWV Informatik 2013*, page 665, 2013. 8, 9, 12, 18, 19, 21

[7] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Con-

volutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 34, 61

[8] Andreas. Kroll. *Computational Intelligence : Probleme, Methoden und technische Anwendungen*. De Gruyter Studium. De Gruyter Oldenbourg, Berlin ;Boston, 2., überarbeitete aufl. edition, 2016. 11, 12, 22, 25

[9] Yann LeCun, Koray Kavukvuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *Proc. International Symposium on Circuits and Systems (ISCAS'10)*. IEEE, 2010. 17

[10] Z. Liang, A. Powell, I. Ersoy, M. Poostchi, K. Silamut, K. Palaniappan, P. Guo, M. A. Hossain, A. Sameer, R. J. Maude, J. X. Huang, S. Jaeger, and G. Thoma. Cnn-based image analysis for malaria diagnosis. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 493–496, Dec 2016. 28, 29

[11] Ryszard Stanislaw Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine learning : an artificial intelligence approach*. Symbolic computation/Artificial intelligence. Springer-Verlag Berlin and Heidelberg GmbH & Co. K :, Berlin, softcover reprint of the original 1st ed. 1983. edition, 2013. Paperback. Book. 8

[12] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814. Omnipress, 2010. 24

[13] R. Q. Qian, Y. Yue, F. Coenen, and B. L. Zhang. Traffic sign recognition using visual attribute learning and convolutional neural network. In *2016 International Conference on Machine Learning and Cybernetics (ICMLC)*, volume 1, pages 386–391, July 2016. 30

[14] robocup.org. Objective, 2017. [Online; Stand 27. April 2017]. 1

[15] Robert S. Siegler. *Entwicklungspsychologie im Kindes- und Jugendalter*. Spektrum Akad. Verl., Heidelberg ; München, [nachdr.] edition, 2008. Hier auch später erschienene, unveränderte Nachdrucke. 11

[16] J. Stamford and B. Peach. Scene detection using convolutional neural networks. In *2nd IET International Conference on Technologies for Active and Assisted Living (TechAAL 2016)*, pages 1–6, Oct 2016. 29

[17] Wikipedia. Kolmogorov–arnold representation theorem — wikipedia, the free encyclopedia, 2016. [Online; accessed 4-May-2017]. 20

[18] Wikipedia. Fifa — wikipedia, die freie enzyklopädie, 2017. [Online; Stand 27. April 2017]. 62

[19] Wikipedia. Nao (roboter) — wikipedia, die freie enzyklopädie, 2017. [Online; Stand 21. April 2017]. 62

[20] Wikipedia. Portable network graphics — wikipedia, the free encyclopedia, 2017. [Online; accessed 13-May-2017]. 62

[21] Wikipedia. Softmax function — wikipedia, the free encyclopedia, 2017. [Online; accessed 7-May-2017]. 42

[22] Wikipedia. Spam — wikipedia, die freie enzyklopädie, 2017. [Online; Stand 27. April 2017]. 61

Erklärung

„Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann“.

Ort Datum Unterschrift