TEC Technik

Hochschule Reutlingen
Reutlingen University

Hochschule Reutlingen

Reutlingen University

– Studiengang Mechatronik Bachelor –

Bachelor–Thesis

Entwicklung eines autonomen Systems zur Bilderkennung mithilfe Neuronaler Netze auf dedizierter Hardware

Manuel Barkey
Pestalozzistraße 29
72762 Reutlingen

Matrikelnummer : 762537

Betreuer:	Eberhard Binder
Zweitbetreuer:	Christian Höfert
Abgabedatum:	TT.MM.JJJJ

Inhaltsverzeichnis

Kapitel 1

Eeinleitung

Die Einleitung soll dazu dienen den Leser auf ...

- in Welchem Themengebiet beewegt sich die Arbeit
- was ist die prinzipielle Aufgabenstellung
- wie war die prinzipielle Vorgehensweise
- welche Ergebnisse wurden erzielt
- welche weitere Enwicklung ist noch möglich

Kapitel 2

Grundlagen

Computer könne in kürzester Zeit die komplexesten Berechnungen durchführen, wenn es jedoch um eine für den Menschen so simple Aufgabe wie das erkennen und zuordnen eines Gegenstandes auf einem Bild geht, ist diese Aufgabe praktisch unmöglich für einen Computer mit herkömmlicher programmierung. Da jedoch sowohl die Entwicklung von Deep Learing Algorithmen als auch die Rechenleistung in den letzen Jahren rasante Fortschritte gemacht haben, ist die bis vor kurzem noch unmöglich geglaubte Aufgabe zum alltäglichen ... im Bereich des Maschinellen Sehens geworden.

Um Objekte auf Bildern erkennen und lokalisieren zu können werden Künstliche Neuronale Netze zusammen mit Techniken der Bildverarbeitung angewandt.

Da Neuronale Netze oder Konkret hier Deep Learining eine Unterart des machinellen Lernens ist wird im ersten Teil der Einleiten kurz die wichtigsten Grundlagen des Machine Learnings eingegangen. Anschließend wird die allgemeine Funktionsweise Künstlicher Neuronaler Netze beschrieben, welche als Grundlage zum Verst ändnis von Convolutional Neural Networks, welche in der Bilderkennung angewand werden, dienen.

2.1 Machine Learning

Machine Learining ist ein Teilgebiet der Computer Wissenschaften, das sich mit selbstlernenden Algorithmen befasst. Dabei sollen Programme komplexe Zusammenhänge in einer menge von daten erkennen ohne expliziert dafür programmiert worden zu sein.
Das unterscheidet das Vorgehen stark zur herkömmlichen Programmierung, bei dem der Entwickler bestimmte Regeln in einem Algorithmus definiert und dass Programm dann auf bestimmte Eingaben die richtigen Ausgaben liefert.
Beim Machine Learning erhällt das Programm neben den Eingeben auch die zugehörigen Ausgaben und soll daraus dann die Regeln ableiten.

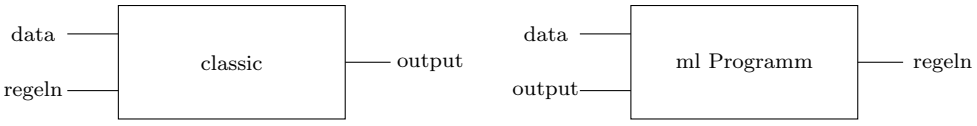

Abbildung 2.1: vgl classisches program und ml

Dieser Prozess wird als lernen bezeichnet und ist vergleichbar mit einer mathematischen Funktion, die numerisch an die richtigen Ausgabewerte angenähert wird.
Die Funktion bzw das Machine Lerning Model wird dabei zunächst mit zufälligen Parameter Werten initialisiert. Anschließend werden ihr die Input Daten übergeben und das Ergebnis dieser Berechnung mit den tatsächlichen Antworten verglichen. Bei Abweichung werden die Parameter der Funk-

tion in die richtige richtung angepasst. Dieser Prozess wird iterativ für eine vielzahl an Input Daten durchgeführt, sodass ein Model geschaffen wird, welches möglichst genaue vorhersagen/Schätzungen für neue Inputs abgiebt kann.

Dieses Vorgehen bezeichnet man auch als *Supervised Learining*. Daneben gibt es auch noch das *Unsupervised Learining* Dabei erhält das Model neben den Input Daten keine Labels. Ziel ist es in den Daten Muster erkennen und zu gruppieren, was dem Labeln entspricht.
Zu den Haupaugaben beim Machine Learning gehören das Sammeln und aufbereiten von Daten sowie das auswählen des geeignetesten Models für die jeweilige Problemstellung.

Die Anwendugen in denen Machine Learning Algorithmen zu finden sind, sind breit gefächert. Websiten wie Facebook, Amazon generieren (Kauf-)Vorschläge Bild und Spracherkennung, Industrie, Robotik, Selbstfahrende Autos. Auch die Erzeugung von Bild und Ton mittels generativer Netze hat eine starke entwicklung erlebt.

2.1.1 Training

Ein Beispeil für einen Machine Learning Algorithmus ist die Lineare Regression. Dise soll wie in [1] beschrieben für einen Eingabevektor x einen skalaren Ausgabewer y liefern, indem x mit einem Vektor aus parametern auch *weights* genannt multipliziert wird.

$$y = w^T x \tag{2.1}$$

Mit hilfe einer Fehlerfunktion wird nun der Abstand der Schätung mit dem tatsächlichen Wert berechnet.

$$E = \sum (y - \hat{y})^2 \tag{2.2}$$

mit $\hat{y}$ für den tatseächlichen Wert.
um den fehler zu minimieren müssen nun die parameter in w angepasst werden.
Da die fehlerfunktion quadratisch ist und von den parametern w abhängt kann durch bilden des gradienten der fehlerfunktion für alle w bestimmt werden wie dieser angepasst werden muss:

$$w \leftarrow w - \eta \frac{\partial E}{\partial w} \tag{2.3}$$

mit:

$$\frac{\partial E}{\partial w} = 2 \sum (w^T x - \hat{y}) x \tag{2.4}$$

Die Schritte (2.1), (2.2) und (3.10) werden dann sooft durchgeführt, bis der Fehler genügend minimiert wurde.
Dabei können entweder alle Input Daten auf einmal verwendet werden *Batch gradient Descent*, nur ein Teil *Mini Batch* oder nur ein zufallig ausgewähltes *Stochastic Gradient Descent*.
Hierbei gilt es η nicht zu groß zu wählen, um das globale Minimum der Fehlerfunktion nicht zu verpassen und auch nicht zu klein da das training sonst zu lange dauern würde.
Neben der Linearen Regression werden häufig auch Modelle benötigt, deren Ausgabe eine kategorsche entscheidung trifft. Beispeile hierfür sind Logistische Regression, oder Suppor Vector Machine.

2.1.2 Validierung und Overfitting

Um zu verhindern, dass ein Model die Trainings Daten nur 'auswendig' lernt, anstatt zu generalisieren, wird während des Trainings eine Validierung durchgeführt. Dafür wird das Datenset in einen Trainings und in einen Test Datensatz aufgeteilt, wobei nur der Trainings Datensatz für die Berechnung der Anpassung der Parameter Werte verwendet wird. Mit dem Test Datensatz kann dann überprüft werden wie gut das Model die Daten generalisiert. Geht der Fehler der Trainings

Daten gegen 0 wohin gegen der Test daten Fehler relativ groß bleibt siehe spricht man von Overfitting. Dies tritt z.B. auf wenn man zu wenige Trainingsdaten hat oder das Model zu viele variable Parameter.
Beim Underfitting hat das Model nicht genügend Parameter um die Daten anzugleichen. Beispiel mit einer Geraden and Datenpunkte mit Verlauf höhren Grad anpassen.
Um Overfitting zu vermeiden können entweder mehr Trainingsdaten, oder Regularisierung verwendet werden. Regularisierung ist eine Technitk, die dafür sorgt, dass das model sich nicht nur den daten anpasst, sondern auch versucht die parameter/gewichte dabei möglichst klein zu halten. Dafür wird der zu minimierenden Loss Function als weiterer Therm eine aufsummierung aller quadrieten Gewichte hinzugefügt. [2]

$$J(w) = E + \lambda \sum w^2 \tag{2.5}$$

2.2 Künstliche Neuronale Netze

Künstliche Neuronale Netze sind eine Form des Machine Learning, bei der das Modell, inspiriert vom menschlichen Gehirn, aus mehreren miteinander verbundenen Einheiten zusammengesetzt ist. Die Einheiten auch Neuronen genannt sind dabei in Schichten angeordnet. Die Eingabeschicht erhällt die zu berechnenden Inputs. Ihr folgen eine oder mehrere versteckte Schichten, welche die Inputs bis zu letzten Schicht, welche den gewünschten Output liefern soll, duchreichen.
Die berechnungen an einem einzelnen Neuron sind dabei nicht sehr komplex und gleichen der berechnung der in 3.1 beschrieben Logistischen regresseion. Wie in BILD dargestellt, werden die mit w gewichteten Inputs x aufsummiert und an die Aktivierungsfunktion $g(z)$ übergeben.

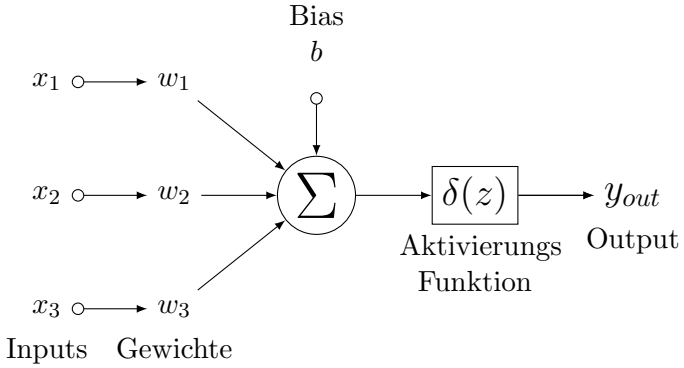

Abbildung 2.2: Einzelnes Perzeptron

hier bild von neuron
Wie in bei der Logistische Regression 3.1 wird zunächst der Fehler mittels Loss Funktion berchnet und anschließend nach dem Gradienten verfahren die Anpassungen für die Parameter/Gewichte w bestimmt.

2.2.1 Mehrschichtiges Netz

Um komplexere Zusammenhänge in Daten zu lernen ist eine verschaltung der Neuronen in Netzwerkartiger Struktur ref BILD notwendig. Für die Anzahl der Hidden Schichten und Neuronen in diesen gibt es keine definierten Vorgeaben. Lediglich die Zahl der In- und Output Neuronen ist vorgegeben.
Die berechnungen von einer Schicht zu nächsten entspricht der Matrixmultiplikatin der Werte an der aktuellen Schcht mit den Gewichten der Verbindungen w zur nächsten Schicht.

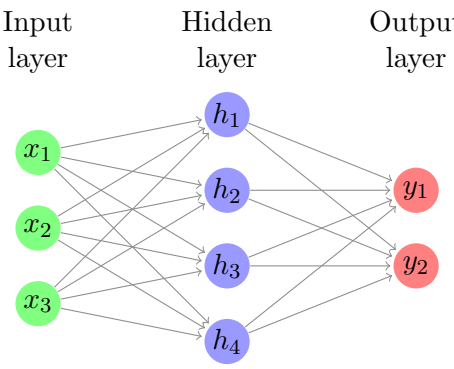

Abbildung 2.3: Neural Network Struktur

$$\begin{pmatrix} x_0 & x_1 & x_2 \end{pmatrix} \cdot \begin{pmatrix} w_{0,1} & x_{0,2} \\ w_{1,1} & x_{1,2} \\ w_{2,1} & x_{2,2} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \tag{2.6}$$

oder:

$$y = \delta(w_0 + w^T x) \tag{2.7}$$

wobei die Aktivierungsfunktion $\delta(z)$ dazu dient den Wert auf einen bestimmten Bereich zu skalieren.

2.2.2 Aktivierungs- und Lossfunktionen

Gängige Aktivierungsfunktion sind:
Sigmoid: erzeugt nicht-linearität und skaliert zw 0 und 1.

$$\delta(z) = \frac{1}{1 + e^{-z}} \tag{2.8}$$

ReLu: setz die negativen Werte zu null (warum von vorteil).

$$\delta(z) = max(0, z) \tag{2.9}$$

Softmax: oft in der letzten Schicht verwendet (bei kategrischer Classification), welche eine Wahrscheinlichkeitsverteilung über allen Output Neuronen, die sich zu 1 aufsummeiren lässt, bildet.

$$\delta(z) = \frac{e^z}{\sum e^x} \tag{2.10}$$

Die Ergebnisse der Aktivierungsfunktionen einer Schich dienen dann als Inputvektor für die nächste Schicht. An der Output Schicht wird nun der Fehler mittels Loss Funktion berechnet. Neben der in 3.1 für Lineare Regression verwendeten *Mean Squre Error* funktion wird für Klassifikationsprobleme häufig die *Cross Enropy Fehlerfunktion* verwendet.

$$E = -\sum \hat{y} log(y) \tag{2.11}$$

Mit $\hat{y}$ für den tatsächlichen Wert (1 oder 0) und y als den geschätzen Wert (zwischen 0 und 1). Durch den Logarithmus wird der Loss besonders gross, für Werte die fälschlicher Weise 0 oder richtung 0 geschätzt wurden.

2.2.3 Backpropagation

Um nach dem Gradienten Verfahren die Werte für die Anpassungen aller Gewichte w zu bestimmen, müssen die Ableitungen über die Kettenregel bestimmt werden. Dabei werden zunächst die Partiellen Ableitungen nach den Gewichtien der letzen Schicht bestimmt. Mit diesen die für die der voletzen, usw bis zur vordersten schicht.

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w} \tag{2.12}$$

Um damit die Gewichte anzupassen:

$$w \leftarrow w - \eta\frac{\partial E}{\partial w} \tag{2.13}$$

Wie bei der in 3.1 beschriebenen Linearen Regression, werden nun auch beim Training des Neuronalen Netzes die Schritte

1. *Forward Propagation* für eine Anzahl N an Inputs
2. Berechnung der *Loss Funktion*
3. *Backpropagation* und Anpassung der Gewichte

sooft wiederholt, bis der Fehler ausreichend minimiert wurde.

2.2.4 Machine Learning Frameworks

2.3 Deep Learning und Computer Vision

Neben den in 3.2 beschrieben *Feed Forwar Netzen* gibt es eine vielzahl and erweiterungen, die jedoch alle das Grundprinzip verwenden.

deep wegen mehr als 1 hidden schicht

Arten von NNs:

- CNN: für bilder, durch faltung erkennt features und ist translatorisch invariant: **Image Classification**
- Recurrecnt Neural Networks/LSTM: durch feedback in network: für Audio und zeitl. anwendungen
- Autoencoder:
- ...

hier geht es um in bla beschriebene cnns

2.3.1 Convolutional Neural Networks

Auch hier wieder biologische inspiration: Auge Rezeptives Feld
math faltung

Conv layer:
Faltung an cnn erklärt: input image als (h,w,c) tensor wird mit filter/kernel gefaltet. daraus erhält man feature map zusammen mit pool layer:

pool layer erklärt

ergibt grundstruktur von cnn

weitere layer wie dropout

2.3.2 Transfer Learining

erklären, dass features von einfach bis immer komplexer werdende muster enthalten, die im bild zu finden sind.
filter können zufällig initialisiert und gelernt werden, oder von vortrainierten netzen wieder verwendet werden. (transfer learining oder fine tuning) da die deatures (besonders in den vorderen layern) immer ähnlich sind und das neu lernen zeitaufwändig und oft sogar ungenauer ist.
je nach ähnlichkeit des eig datensetz zu dem auf das netz urspr trainiert wurde:
scratch, fine tuning, feature extractor

2.3.3 Competitions mit Imagenet und co + cnn winner

zuerst competition erklären
dann chronologische gewinner netzt + besonderheit

2.3.4 Objekt erkennung

Unterschied deutlich machen: klassifikator kann nur ges bild auswertuen und wahscheinlichk angeben welche klasse darauf. keine lokalisierng und keine mult obj

3 Arten der Bilderkennung: Klassifizierung, Objekt Erkennung (für mult + box), Segmentation (jeden pixel)
dafür obj erkennung notw:

Single Shot Detectoren

Two Stage Detectoren

2.4 Hardware

allg zu hardware für deeplearning. Das besser auf gpu als cpu. weitere: tpu, fpga, vpu, wie zb ncs2.

2.4.1 Neural Compute Stick 2

technischen spezifikationen

2.4.2 AI on the edge

was bedeutet dies. cloud unabhängig und ohne groß rechner. bsp anwendungen.

Kapitel 3

Grundlagen

Computer könne in kürzester Zeit die komplexesten Berechnungen durchführen, wenn es jedoch um eine für den Menschen so simple Aufgabe wie das erkennen und zuordnen eines Gegenstandes auf einem Bild geht, ist diese Aufgabe praktisch unmöglich für einen Computer mit herkömmlicher programmierung. Da jedoch sowohl die Entwicklung von Deep Learing Algorithmen als auch die Rechenleistung in den letzen Jahren rasante Fortschritte gemacht haben, ist die bis vor kurzem noch unmöglich geglaubte Aufgabe zum alltäglichen ... im Bereich des Maschinellen Sehens geworden.

Um Objekte auf Bildern erkennen und lokalisieren zu können werden Künstliche Neuronale Netze in CNN Architektur zusammen mit Techniken der Bildverarbeitung angewandt.

Daher werden im folgenden kurz die Begrifflichkeiten Machine Learnin und AI geklärt, und anschließend die Funktions weise von Neuronalen Netzen allgemein sowie Convolutional Neural Networks im Speziellen für die Bilderkennung.
Der Letzte Teil der Grundlagen geht auf die für Deep Learning verwendete Hardwaretypen und anforderungen ein.

3.1 Machine Learning und AI

Machine Learining ist ein Teilgebiet der Computer Wissenschaften, das sich mit selbstlernenden Algorithmen befasst. Dabei sollen Programme komplexe Zusammenhänge in einer menge von daten erkennen ohne expliziert dafür programmiert worden zu sein.
Das Unterscheidet den Algorithmus sehr von einem herkömmlichen Computer Programm, bei dem der Entwickler bestimmte Regeln definiert, sodass das Programm auf bestimmte Eingaben die richtigen Ausgaben liefert.
Beim Machine Learning erhällt das Programm neben den Eingeben auch die zugehörigen Ausgaben und soll daraus dann die Regeln so ableiten, das es zukünftig für ähnliche Input Daten die richtigen Outputs bestimmen kann.

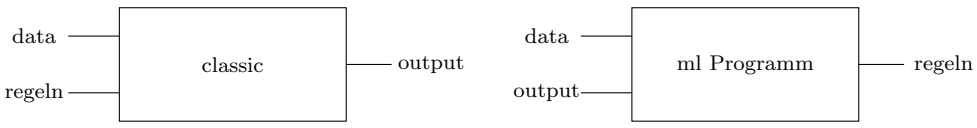

Abbildung 3.1: vgl classisches program und ml

Das Ableiten der Regeln wird als Lernen bezeichnet und ist mit einer numerischen Annäherung einer math Funktion an die In- Output beziehungen der Daten zu verglischen.
Die Funktion oder Modell wird dabei zunächst mit zufälligen Parameterwerten initialisiert. Anschließend wird iterativ mit den Input Daten ein Ergebnis berechnet und mit dem tatsächli-

chen/richtigen Ausgabe Werten verglichen. Bei Abweichung werden die Parameter des Modells in die Richtige richtung angepasst.
Da dem Modell während des Trainings die zu den Inputs gehörigen Outputs bekannt sind, wird von *Supervised Learining* gesprochen.
Dieses wird für Regressionsprobleme (Lineare Regression) sowie Klassifikationsprobleme (Logistische Regression, SVM) angewandt.
Sind die Outputs nicht, bakannt spricht man von *Unsupervised Lerning*. Hierbei soll das Model Mustern in den Daten Erkennen und gruppieren, was zum automatisierten Labeln von Daten eingesetzt wird oder die Rollen eines Datenaufbereitungsschritt für ein anderes ML Modell übernimmt.
Eine Dritte Form des Machine Leaerinigs stellt das Reinforcement Learing dar, bei der ...
Die Anwendugsgebiete für Machine Learning Algorithmen sind sehr vielfältig. Ob Websiten wie Facebook, Amazon generieren Bild und Spracherkennung, Industrie, Robotik, autonom-fahrende Autos.

3.1.1 Machine Learning Frameworks

Die Algorithmen müssen nicht jedesmal neu implementiert werden. Für die gängigen verfahren gibt es Frameworks, welche die Implementierung enthalten und über APIs verwendet werden können (Bsp Tensorflo und Keras)

3.2 Künstliche Neuronale Netze

Künstliche Neuronale Netze sind eine Form des Machine Learning, bei der das Modell, inspiriert vom menschlichen Gehirn, aus mehreren miteinander verbundenen Einheiten zusammengesetzt ist. Die Einheiten auch Neuronen genannt sind dabei in Schichten angeordnet. Die Eingabeschicht erhällt die zu berechnenden Inputs. Ihr folgen eine oder mehrere versteckte Schichten, welche die Inputs bis zu letzten Schicht, welche den gewünschten Output liefern soll, duchreichen.
Die berechnungen an einem einzelnen Neuron sind dabei nicht sehr komplex und gleichen der berechnung der in 3.1 beschrieben Logistischen regresseion. Wie in BILD dargestellt, werden die mit w gewichteten Inputs x aufsummiert und an die Aktivierungsfunktion
$g(z)$ übergeben.

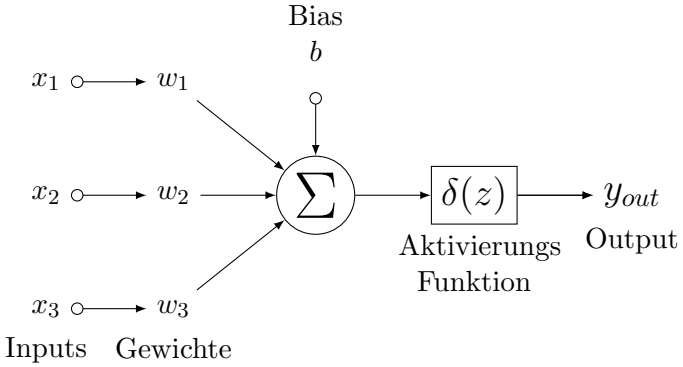

Abbildung 3.2: Einzelnes Perzeptron

hier bild von neuron
Wie in bei der Logistische Regression 3.1 wird zunächst der Fehler mittels Loss Funktion berchnet und anschließend nach dem Gradienten verfahren die Anpassungen für die Parameter/Gewichte w bestimmt.

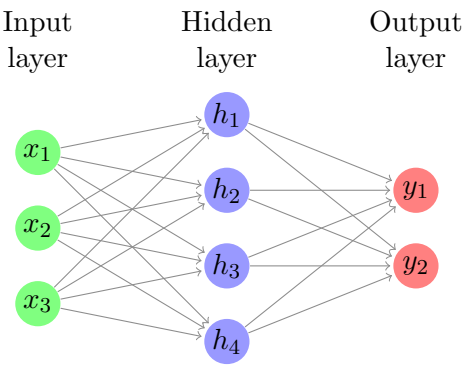

Abbildung 3.3: Neural Network Struktur

3.2.1 Mehrschichtiges Netz

Um komplexere Zusammenhänge in Daten zu lernen ist eine verschaltung der Neuronen in Netzwerkartiger Struktur ref BILD notwendig. Für die Anzahl der Hidden Schichten und Neuronen in diesen gibt es keine definierten Vorgeaben. Lediglich die Zahl der In- und Output Neuronen ist vorgegeben.
Die berechnungen von einer Schicht zu nächsten entspricht der Matrixmultiplikatin der Werte an der aktuellen Schcht mit den Gewichten der Verbindungen w zur nächsten Schicht.

$$\begin{pmatrix} x_0 & x_1 & x_2 \end{pmatrix} \cdot \begin{pmatrix} w_{0,1} & x_{0,2} \\ w_{1,1} & x_{1,2} \\ w_{2,1} & x_{2,2} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \tag{3.1}$$

oder:

$$y = \delta(w_0 + w^T x) \tag{3.2}$$

wobei die Aktivierungsfunktion $\delta(z)$ dazu dient den Wert auf einen bestimmten Bereich zu skalieren.

3.2.2 Aktivierungs- und Lossfunktionen

Gängige Aktivierungsfunktion sind:
Sigmoid: erzeugt nicht-linearität und skaliert zw 0 und 1.

$$\delta(z) = \frac{1}{1 + e^{-z}} \tag{3.3}$$

ReLu: setz die negativen Werte zu null (warum von vorteil).

$$\delta(z) = max(0, z) \tag{3.4}$$

Softmax: oft in der letzten Schicht verwendet (bei kategrischer Classification), welche eine Wahrscheinlichkeitsverteilung über allen Output Neuronen, die sich zu 1 aufsummeiren lässt, bildet.

$$\delta(z) = \frac{e^z}{\sum e^x} \tag{3.5}$$

Die Ergebnisse der Aktivierungsfunktionen einer Schich dienen dann als Inputvektor für die nächste Schicht. An der Output Schicht wird nun der Fehler mittels Loss Funktion berechnet. Neben der in 3.1 für Lineare Regression verwendeten *Mean Squre Error* funktion wird für Klassifikationsprobleme häufig die *Cross Enropy Fehlerfunktion* verwendet.

$$E = -\sum \hat{y} log(y) \tag{3.6}$$

Mit $\hat{y}$ für den tatsächlichen Wert (1 oder 0) und y als den geschätzen Wert (zwischen 0 und 1). Durch den Logarithmus wird der Loss besonders gross, für Werte die fälschlicher Weise 0 oder richtung 0 geschätzt wurden.

3.2.3 Backpropagation

Um nach dem Gradienten Verfahren die Werte für die Anpassungen aller Gewichte w zu bestimmen, müssen die Ableitungen über die Kettenregel bestimmt werden. Dabei werden zunächst die Partiellen Ableitungen nach den Gewichtien der letzen Schicht bestimmt. Mit diesen die für die der voletzen, usw bis zur vordersten schicht.

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w} \tag{3.7}$$

Um damit die Gewichte anzupassen:

$$w \leftarrow w - \eta \frac{\partial E}{\partial w} \tag{3.8}$$

Wie bei der in 3.1 beschriebenen Linearen Regression, werden nun auch beim Training des Neuronalen Netzes die Schritte

1. *Forward Propagation* für eine Anzahl N an Inputs
2. Berechnung der *Loss Funktion*
3. *Backpropagation* und Anpassung der Gewichte

sooft wiederholt, bis der Fehler ausreichend minimiert wurde.

3.2.4 Training

mit $\hat{y}$ für den tatseächlichen Wert.
um den fehler zu minimieren müssen nun die parameter in w angepasst werden.
Da die fehlerfunktion quadratisch ist und von den parametern w abhängt kann durch bilden des gradienten der fehlerfunktion für alle w bestimmt werden wie dieser angepasst werden muss:

$$w \leftarrow w - \eta \frac{\partial E}{\partial w} \tag{3.9}$$

mit:

$$\frac{\partial E}{\partial w} = 2 \sum (w^T x - \hat{y}) x \tag{3.10}$$

Die Schritte (2.1), (2.2) und (3.10) werden dann sooft durchgeführt, bis der Fehler genügend minimiert wurde.
Dabei können entweder alle Input Daten auf einmal verwendet werden *Batch gradient Descent*, nur ein Teil *Mini Batch* oder nur ein zufallig ausgewähltes *Stochastic Gradient Descent*.
Hierbei gilt es η nicht zu groß zu wählen, um das globale Minimum der Fehlerfunktion nicht zu verpassen und auch nicht zu klein da das training sonst zu lange dauern würde.

3.2.5 Validierung und Overfitting

Um zu verhindern, dass ein Model die Trainings Daten nur 'auswendig' lernt, anstatt zu generalisieren, wird während des Trainings eine Validierung durchgeführt. Dafür wird das Datenset in einen Trainings und in einen Test Datensatz aufgeteilt, wobei nur der Trainings Datensatz für die Berechnung der Anpassung der Parameter Werte verwendet wird. Mit dem Test Datensatz kann dann überprüft werden wie gut das Model die Daten generalisiert. Geht der Fehler der Trainings Daten gegen 0 wohin gegen der Test daten Fehler relativ groß bleibt siehe spricht man von Overfitting. Dies tritt z.B. auf wenn man zu wenige Trainingsdaten hat oder das Model zu viele variable Parameter.
Beim Underfitting hat das Model nicht genügend Parameter um die Daten anzugleichen. Beispiel mit einer Geraden and Datenpunkte mit Verlauf höhren Grad anpassen.
Um Overfitting zu vermeiden können entweder mehr Trainingsdaten, oder Regularisierung verwendet werden. Regularisierung ist eine Technitk, die dafür sorgt, dass das model sich nicht nur den daten anpasst, sondern auch versucht die parameter/gewichte dabei möglichst klein zu halten. Dafür wird der zu minimierenden Loss Function als weiterer Therm eine aufsummierung aller quadrieten Gewichte hinzugefügt. [2]

$$J(w) = E + \lambda \sum w^2 \tag{3.11}$$

3.3 Deep Learning und Computer Vision

Neben den in 3.2 beschrieben *Feed Forwar Netzen* gibt es eine vielzahl and erweiterungen, die jedoch alle das Grundprinzip verwenden.

deep wegen mehr als 1 hidden schicht

Arten von NNs:

- CNN: für bilder, durch faltung erkennt features und ist translatorisch invariant: **Image Classification**
- Recurrecnt Neural Networks/LSTM: durch feedback in network: für Audio und zeitl. anwendungen
- Autoencoder:
- ...

hier geht es um in bla beschriebene cnns

3.3.1 Convolutional Neural Networks

Auch hier wieder biologische inspiration: Auge Rezeptives Feld
math faltung

Conv layer:
Faltung an cnn erklärt: input image als (h,w,c) tensor wird mit filter/kernel gefaltet. daraus erhält man feature map zusammen mit pool layer:
pool layer erklärt

ergibt grundstruktur von cnn

weitere layer wie dropout

3.3.2 Transfer Learining

erklären, dass features von einfach bis immer komplexer werdende muster enthalten, die im bild zu finden sind.
filter können zufällig initialisiert und gelernt werden, oder von vortrainierten netzen wieder verwendet werden. (transfer learining oder fine tuning) da die deatures (besonders in den vorderen layern) immer ähnlich sind und das neu lernen zeitaufwändig und oft sogar ungenauer ist.
je nach ähnlichkeit des eig datensetz zu dem auf das netz urspr trainiert wurde:
scratch, fine tuning, feature extractor

3.3.3 Competitions mit Imagenet und co + cnn winner

zuerst competition erklären
dann chronologische gewinner netzt + besonderheit

3.3.4 Objekt erkennung

Unterschied deutlich machen: klassifikator kann nur ges bild auswertuen und wahscheinlichk angeben welche klasse darauf. keine lokalisierng und keine mult obj

3 Arten der Bilderkennung: Klassifizierung, Objekt Erkennung (für mult + box), Segmentation (jeden pixel)
dafür obj erkennung notw:

Single Shot Detectoren

Two Stage Detectoren

3.4 Hardware

allg zu hardware für deeplearning. Das besser auf gpu als cpu. weitere: tpu, fpga, vpu, wie zb ncs2.

3.4.1 Neural Compute Stick 2

technischen spezifikationen

3.4.2 AI on the edge

was bedeutet dies. cloud unabhängig und ohne groß rechner. bsp anwendungen.

Kapitel 4

Anforderungen und Analyse

4.1 Ziel der Arbeit

End to end Prozess von Datensatz beschaffung, über training eines geeigneten Neuronalen Netzes bis hin zu implementierung der Applikation, die auf einem Raspberry Pi läuft und die Inferenz auf dem NCS2 ausführt.

Es sollen in Wild Tiere erkannt werden, die in Deutschland heimisch sind.
Das system soll autonom laufen und den Nutzer informieren (und das erkannte bild senden) sobald etwas erkannt wurde.
Im Optimalfall soll es mithilfe Infrarot kamera auch im Dunkeln Tiere erkennen, (da Nachts mehr tiere zu sehen sein werden)
Verwender werden soll: für Inferenz der in 3.4 beschriebene Neural Compute Stick 2, und für die Stuerung der (Einptaininen Computer) RaspberryPi2.
Um auch im Dunklen oder bei nacht Tiere erkennen zu können soll eine Kamera ohne Infrarot Filter verwendet werden. (evtl noch auf realsense eingehen)
Die Kommunikation zwischen Raspberry und Pc soll über eine server/client tcp Verbindung erfolgen. Die Applikattio soll mitteilen wenn etwas erkannt wurde und das bild zusenden. Ausserdem soll das aktuelle frame abgefragt werden können sowie einstellungen bezüglich infrarot leds vorgenommen werden können.

4.2 Related Work

hier:

- Gibt einen Überblick über verwandte Arbeiten im Gebiet
- Strukturiert und Gruppiert diese Arbeiten sinnvoll
- Deckt möglichst alle relevanten Arbeiten ab
- Erklärt kurz deren Inhalt und was sie von anderen Arbeiten (vorallem der Eigenen!) abheben
- Positioniert die eigene Arbeit im Gebiet

Kapitel 5

Realisierung Objekt Erkennung

5.1 Dataset

Da es sich um supervised Learning handelt müssen trainigs daten gelabelt werden.
für validierung und test muss datensatz zu 80, 10, 10 in test/train/validation aufgeteilt werden.
wie in 3.1 beschieben dient das validierungs set zur überwachung wärend des trainings für overfitting.
Mit dem Test set kann nach dem training die Inferenz also das ausfüren des traininierten models getestet werden.
Für die Objekterkennung muss der Datensatz wie in 3.3 beschrieben neben den gelabelten bildern auch die X- und Y-Koordinaten der Bounding Boxen, welche das objekt auf dem Bild umramt, enthalte. das Objekt befindet enthalten.

5.1.1 Datenbeschaffung

Da das Erstellen eines eingangs erwähnten Datensets von Hand sehr müsam ist wird meistens auf Quellen zurückgegriffen, die schon gelabelte Daten zu bestimmten Klassen zur verfügung stellen.
Neben den in 3.3.3 vorgestellten Seiten bietet OpenImages einen vielzahl an Klassen an, darunter auch Unter der Kategorie Säugetiere eine Auswahl an Wild Tier, welche im folgenden verwendet wurde.
Mit einem Open source Tool [**?**] konnten eine teilmenge aus dem gesammten Open Images datensatzes herunter geladen werden.
Die Label Files haben das anotierungs format `class,xmin,ymin,xmax,ymax` welches wie in 5.1.3 beschrieben wird, noch in ein für tensorflow vertändliches format gebracht werden musste.
Die Verteilung der Klassen im Datensatz war nicht ausbalanciert, wie in **??** zu sehen ist.
Das kann zur folge haben das.

Allg wie viele samples sollte man haben.

Augmentierung im folgenen Teil beschieben.

5.1.2 Augmentierung

was ist Augmentierung
wie wurde es angewand
bsp bilder

5.1.3 TF Record Files

was es ist
wie es erstellu wurde

Wie in 5.1.1 erwähnt verwendet tensorflow ein bestimmtes format für das datenset, sog Protocol buffer tf record files, dateien im Binary format die sowohl die Bilder als auch die Labels enthalten. das sind Protocol buffer welche die daten serealisieren.
evtl hier besp ausschnitt von aufbau eines proto elements.

Um nun die von OpenImages herunter geladenen Bilder und Label Files in das TFRecords Format zu bringen waren mehrere Schritte nötig.
OI - VOC - csv - tf.records

5.2 Training

5.2.1 TF obj det api

Für das Training wurde das Framework Tensorflow verwendet, welces eine Api für Objekterkennung bietet.
welche pretrained modell gibt es und welche kamen in frage (für ncs2) Die Tensorflow Objekt Detection Api bietet eine vielzahle an vortrainierten Modellen, dabei wurden die meiseten auf den COCO datensatz trainiert.

speed/acc trade off

daneben war bei der bei der Ausahl die kompatibilität zu Open Vino zu berücksichtigen:
liste von kompatiblen modellen.

trainiert wurde auf:

- ssd mobile net und inception
- faster rcnn (inception und resnet)
- frcnn

(in eval ergebniss dann etwa so: ssd zu schlechte performence und für appl keine realtime nitwendig, frcnn zu langsam, faster rcnn gute mitte)

5.3 Parameter Optimierung

einstellungen im Config File
tensorflow graph oder plot zeigen
loss erklären (mit formel und für train und eval)

Kapitel 6

Evaluierung

6.1 Evaluierungs Metriken

hier zb confusion matrix, IOU, mAP erklären.

6.2 Ergebnisse

hier tabellarisch ergebnisse der verschiedenen modell und datensetze miteinander verglaichen und evtl auch plots zeigen.
nach verschiedenen kriterien wie zb einfuss augmentierung, einfluss graustufen/rgb, ...

6.3 Infrarot Bilder

Da die Infrarot Kamera Graustufen bildre liefert, stellt sich nun die Frage welche auswirkung dies auf Training und Inferenz hat.

Muss neu auf Graustufen Bilder trainiert werden?

Wenn ja wie?

- input shape des Netzes ändern
- 3 mal selben input drauf geben.

Wenn nein:
gelernte features der farben werden nicht verwendet.
wäre das netzt dann nicht genauer, wenn es nur auf den einen Kanal traininert wird, also nur features die es später tatsächlich verwendet und sich so mehr auf formen und nur eine art von intensität (anstatt der 3 rgb) generali sieren kann?
oder ist da kein merklicher nachteil?
hätte dies auch einen Einfluss auf die Inferenz Zeit wenn nur ein channel verwendet wird??

Kapitel 7

Entwicklung der Anwendung

Hier wird die Inferenz also die implementierun der Application im Open Vino Toolkit beschrieben.
Workflow von OpenVino beschreiben.

7.1 Software

7.1.1 Inferenz

Inference Engine von Oopne Vono
allg aufbau und ablauf: plugin network synch/async ...
Convertierung des exportierten tensorflow graphen

7.1.2 Client Server Verbindung

7.2 Hardware

7.2.1 Raspberry

hier kurz raspi beschreiben und aufbau mit powerbank und gsm modul internt stick.
berechnung stromverbrauch mit ncs2 + cam + ir leds

7.2.2 Realsense bzw RPi NoIR Cam

hier tests mit realsense und raspi cam beschreiben

Kapitel 8

Test und Validierung

Kapitel 9

Zusammenfassung und Ausblick

Die Zusammenfassung bildet mit der Einleitung den Rahmen der Arbeit. Sie greift zu Beginn die Aufgabenstellung auf und beschreibt dann die wesentlichen Punkte des Lösungsweges und die erzielten Ergebnisse kurz und knapp, so dass diese in kürzester Zeit erfasst werden können.
Anschließend werden noch kurz offene Punkte, Verbesserungen oder Weiterentwicklungen diskutiert.
Insgesamt sollten Zusammenfassung und Ausblick anderthalb Seiten nicht überschreiten. In der Regel ist eine Seite ausreichend.

Literaturverzeichnis

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Adaptive Computation and Machine Learning, The MIT Press.

[2] A. Géron, *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, first edition ed. OCLC: ocn953432302.

Anhang A

Beispiel für ein Kaptel im Anhang

A.1 Bsp für ein Abschnitt im Anhang