

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

1 Einleitung	3
2 Grundlagen	5
2.1 Machine Learning	5
2.2 Künstliche Neuronale Netze	5
2.3 Deep Learning und Computer Vision	8
2.4 Hardware	10
3 Anforderungen und Analyse	11
3.1 Ziel der Arbeit	11
3.2 Related Work	11
4 Realisierung Objekt Erkennung	13
4.1 Dataset	13
4.2 Training	14
4.3 Parameter Optimierung	15
5 Evaluierung	17
5.1 Evaluierungs Metriken	17
5.2 Ergebnisse	17
6 Entwicklung der Anwendung	21
6.1 Aufbau	21
6.2 OpenVino Toolkit	22
6.3 Raspberry Pi Kamera	24
6.4 Server-Client-Connection	24
6.5 Anwendung gesamt	24
7 Test und Validierung	25
8 Zusammenfassung und Ausblick	27
A Beispiel für ein Kapitel im Anhang	31
A.1 Bsp für ein Abschnitt im Anhang	31

Kapitel 1

Einleitung

Im Rahmen der Bachelor Arbeit wurde ein Überwachungssystem zur Wildtiererkennung, entwickelt, welches auf einem Raspberry Pi läuft und den Nutzer bei Erkennung bestimmter Tiere automatisch benachrichtigt, sowie das Bild an einen Server sendet.

Die Erkennung der Tiere erfolgte mithilfe Neuronaler Netze, wodurch es möglich ist die Überwachung gezielt nur auf bestimmte, relevante Tiere anzuwenden und so den Datenverkehr gering zu halten.

Die Inferenz der Neuronalen Netze wurde dabei auf einer separaten Hardware, dem Neural Compute Stick 2 von Intel ausgeführt.

Des weiteren wurde eine Infrarotfähige Kamera verwendet, damit das System auch in der Nacht einsetzbar ist.

Kapitel 2

Grundlagen

Im folgende Kapitel wird zunächst auf die Grundlegenden Techniken des Machine Learings, insbesondere auf die für die Bilderkennung verwendeten Convolutional Neural Networks eingegangen. Anschließend wird es um die verwendete Hardware, den Neural Compute Stick 2 un seine Anwendungen gehen.

2.1 Machine Learning

Beim Machine Lerining, welches ein Teilgebiet der Computerwissenschaften ist, geht es um Algorithmen, die Zusammenhänge in großen Datenmengen erkennen sollen, ohne explizit darauf programmiert worden zu sein.

Eine Form davon ist das *Supervised Learning*, bei der das Programm neben den Input Daten auch die Zugehörigen Ausgaben erhält und daraus dann die Regeln für Zusammenhänge herleiten soll. Dadurch unterscheidet sich das Vorgehen wesentlich zur klassischen Programmierung, bei der die Regeln vorab definiert werden.



Das herleiten der Regeln erfolgt beim Machine Learning dabei in einem iterativen Prozess, welcher als Training bezeichnet wird. Dabei soll eine math. Funktion, welche die Zusammenhänge beschreibt numerisch angenähert werden. Ist der Zusammenhang linear, spricht man von einer Regression, handelt es sich um Kategorische, liegt ein Klassifizierung problem vor.

Weitere Formen neben dem *Supervised Learning* sind das *Unsupervised Learning*, bei der das Programm keine Labels erhält, sondern diese durch Clustering Verfahren selber finden soll, oder das *Reinforcement Learning*, bei dem das Programm mit der Umwelt interagieren soll.

Da hier jedoch ausschließlich mit dem Supervised Learing gearbeitet wurde, werden diese Techniken nicht näher erläutert.

2.2 Künstliche Neuronale Netze

Künstliche Neuronale Netze sind eine Form des Machine Learning, bei der das Modell aus einer Vielzahl an Einheiten besteht, welche einen Input erhalten und dafür einen Wert ausgeben. Inspiriert vom menschlichen Gehirn werden diese Einheiten, auch Neuronen genannt, Netzwerkartig in Schichten miteinander verbunden. Dadurch können auch komplexere Zusammenhänge zwischen In- und Output Daten gelernt werden.

2.2.1 Funktionsweise einzelnes Perzeptron

Die berechnung eines einzelnes Neuron ist entspricht der Logistischen Regression und ist in Abbildung 2.1 dargestellt.

Zunächst werden die mit w_i gewichteten Inputs x_i aufsummiert und anschließend einer Aktivierungsfunktion $\delta(z)$ übergeben.

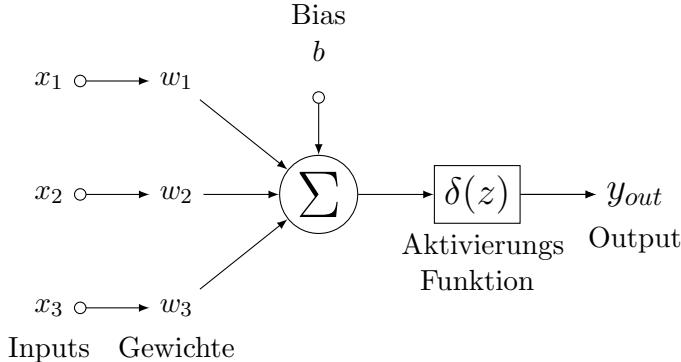


Abbildung 2.1: Einzelnes Perzepron

Die Aktivierungsfunktion soll den Wert auf einen bestimmten Bereich skallieren. Für einen binären klassifikator, also $y = 1$ oder 0 wird dafür die *Sigmoid Funktion* verwendet, welche einen Wert S-Förmig zwischen 0 und 1 skaliert.

Die vorhersage berechnet sich demnach aus

$$z = \sum_i (w_i x_i + b) \quad (2.1)$$

$$y = \frac{1}{1 + e^{-z}} \quad (2.2)$$

Mithilfe einer Loss Funktion $L(y, \hat{y})$ wird nun bestimmt wie sehr sich geschätzte Wert y von dem tatsächlichen Wert \hat{y} unterscheidet.

Für Regressions Probleme wird dafür oft der Absolute oder Quadratische Abstand verwendet. Bei Klassifikationen ist jedoch eine Logarithmische Fehler berechnung effektiver.

Durch den Logarithmus wird der Loss um so größer, je weiter die Schätzung y (0 bis 1) vom tatsächlichen Wert \hat{y} (0 oder 1) abweicht.

$$L = \hat{y} \log(y) + (1 - \hat{y}) \log(1 - y) \quad (2.3)$$

Um nun die Schätzungen \hat{y} beim nächsten mal zu verbessern muss die Lossfunktionen minimiert werden. Dafür wird das Gradienten Verfahren angewandt bei dem die Loss funktion partiell nach den Parametern W_i und b_i abgeleitet wird.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial w} \quad (2.4)$$

Nun können die Parameter W und b angepasst werden.

$$w \leftarrow w - \eta \frac{\partial L}{\partial w} \quad (2.5)$$

wobei die *Learningrate* η die Schrittweite mit der die Anpassung vorgenommen werden soll angibt. Die Schritte (2.1) bis (2.5) werden dann solange wiederholt, bis der Loss genügend minimiert wurde. Dabei können entweder alle Input Daten auf einmal *Batch gradient Descent*, nur ein Teil *Mini Batch* oder nur ein zufällig ausgewähltes *Stochastic Gradient Descent* verwendet werden.

2.2.2 Mehrschichtiges Netz

Da die Möglichkeiten mit einem einzelnen Neuron stark begrenzt sind werden für komplexere Zusammenhänge mehrere Neuronen in Schichten miteinander verbunden. Die Verbindungen sind unterschiedlich stark gewichtet wodurch auf Eingaben in die erste Schicht *Input Layer* an der Letzten Schicht, *Output Layer* die zugehörigen Ausgaben erzeugt werden können.

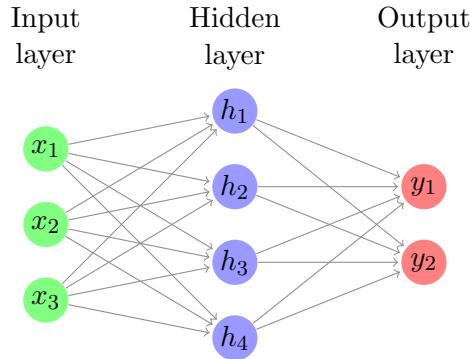


Abbildung 2.2: Neural Network Struktur

Die Berechnung erfolgt gleich wie beim einzelnen Perzeptron für alle Neuronen einer Schicht gleichzeitig mit Matrixmultiplikation

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

Die Ergebnisse der Aktivierungsfunktion in einer Schicht dienen dann als Input für die nächste Schicht. Für *Hidden Layer* wird anstelle der in 2.2.1 beschrieben Sigmoid Funktion ReLU verwendet, da das Skalieren zw 0 und 1 nur für die Ausgabe Schicht

$$\delta(z) = \max(0, z) \quad (2.7)$$

Diese setzt nur die negativen Werte zu 0. Durch die dadurch erhaltenen größeren Gradienten kann das Netz schneller lernen.

Enthält die Output Schicht mehr als ein Neuron, soll also keine binäre sondern Kategorische Klassifizierung erreicht werden, wird anstelle der Sigmoid die *Softmax* Funktion verwendet.

$$\delta(z) = \frac{e^z}{\sum e^x} \quad (2.8)$$

welche eine Wahrscheinlichkeitsverteilung über allen Output Neuronen bildet, die sich zu 1 aufsummieren lässt. Somit ergibt sich für jedes Output Neuron, das für eine mögliche Klasse steht, eine Wahrscheinlichkeit, mit der es sich um diese Klasse handelt.

Um nun für die berechnete Lossfunktion die Anpassungen für die einzelnen Gewichte zu berechnen, werden Schrittweise die Partiellen Ableitungen rückwärts für alle Layer vorgenommen, *Backpropagation* sodass für die Layer $L = 1 \dots N$ die Gradienten $\frac{\partial L}{\partial w_1}$ bis $\frac{\partial L}{\partial w_L}$ zur Anpassung der w Matrizen entsteht.

Neben dem Ansatz des Gradienten Descent für die Optimierungs gibt es noch weitere, effizientere Verfahren wie z.B. Momentum oder Adam.

2.2.3 Validierung und Overfitting

Um zu erkennen, ob das Netz die Trainings Daten nur 'auswendig' lernt, oder die Zusammenhänge in ihnen für neue Daten generalisiert hat, wird während des Trainings eine Validierung durchgeführt. Dafür wird das Datenset in ein Trainings- und in ein Test Set aufgeteilt und nur mit den

Trainingsdaten die Backpropagation zur Parameter Anpassung durchgeführt. Mit dem Test Datensatz wird nur der Loss berechnet und mit ausgegeben. Wenn sich nur der Trainings Loss verringert ?? ist das ein Zeichen für Overfitting, das heißt keine Generalisierung findet statt.

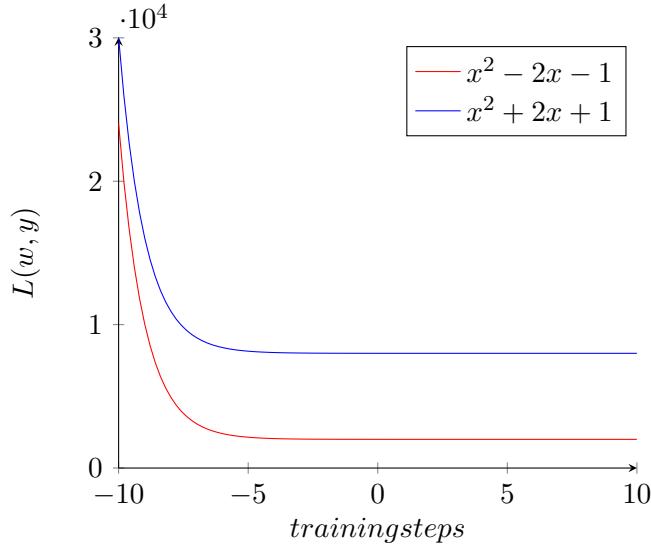


Abbildung 2.3: irgend ein plot

Grund dafür sind entweder zu wenige Trainingsdaten oder das Netz hat zu viele Neuronen, ist also überparametrisiert, und damit zu viele Freiheiten sich den Daten anzupassen. ??

Hat das Modell zu wenige Parameter um die Komplexität der Daten anzugeleichen, Bsp, Gerade an Funktion höheren Grades, spricht man von Underfitting. ??

Um Overfitting zu vermeiden können entweder mehr Trainingsdaten verwendet werden. Stehen diese jedoch nicht zur Verfügung, kann Regularisierung verwendet werden, eine Technik die neben der Minimierung der Loss-Funktion auch versucht die Gewichte auf einem kleinen Wert zu halten. Dafür wird der zu minimierende Loss Function als weiterer Term eine Aufsummierung aller quadrierten Gewichte hinzugefügt. (2.9) [1]

$$J(w) = E + \lambda \sum_i w_i^2 \quad (2.9)$$

Weitere Möglichkeiten sind Daten Augmentation oder Drop Out, auf die in Abschnitt 2.3 genauer eingegangen wird.

2.3 Deep Learning und Computer Vision

Neben den in 2.2 beschriebenen sogenannten *Feed Forward Netzen* gibt es eine Vielzahl an Erweiterungen der Netzwerk-Architektur, für unterschiedliche Anwendungsbereiche, die jedoch alle das beschriebene Grundprinzip verwenden.

Hat ein NN mehr als eine Hidden Schicht, spricht man von Deep Learning.

Arten von NNs:

- CNN: für Bilder, durch Faltung erkennt Features und ist translatorisch invariant: **Image Classification**
- Recurrent Neural Networks/LSTM: durch Feedback in Network: für Audio und zeitl. Anwendungen
- Autoencoder, GANs, etc.

Computer Vision beschäftigt sich mit der Bild erkennung und Objekt Lokalisierung. Anwendungen sind gesichtserkennung oder Autonomes Fahren.

2.3.1 Convolutional Neural Networks

Um Bilder /Inhalte mithilfe Neuronaler Netze zu erkennen, werden die einzelnen Pixelwerte der Bilder als Inputs verwendet und das auf dem Bild zu erkennende Objekt als output verwendet. Bilder werden als Matrizen der Form $height \times width \times colorchannels$ dargestellt.

Da dies für regulere/vollständig verbundene Neuronale Netze eine enorme Anzahl an Parametern und damit einhergehender rechenkost bedeuten würde, werden hier CNNs verwendet, eine Architektur in der Parameter von verschiedenen Neuronen gemeinsam genutzt werden.

Hauptbestandteil von CNNs sind die *Convolutional Layers* welche die mathematische Faltungsoperation zwischen Input Bild und Filter/Kernel durchführen.

Die Filter meist der Form 3×3 oder 5×5 und mit der selben Tiefe wie der Input, werden während des *Forward Pass/bropagation* zeilenweise über das Bild geschoben und an jeder Stelle das Kreuzprodukt berechnet. Jedes Ergebnis dieser Berechnungen ergibt einen Pixel Wert der nächsten Schicht auch Feature Map genannt. ??

Ein weiterer Bestandteil von CNNs sind die Pooling Layer, welche eine bestimmte Anzahl an Pixeln zB 3×3 zu einem Wert zusammenfassen wodurch sich die Parameter Anzahl des Bildes verringert. Das hat den Vorteil, dass

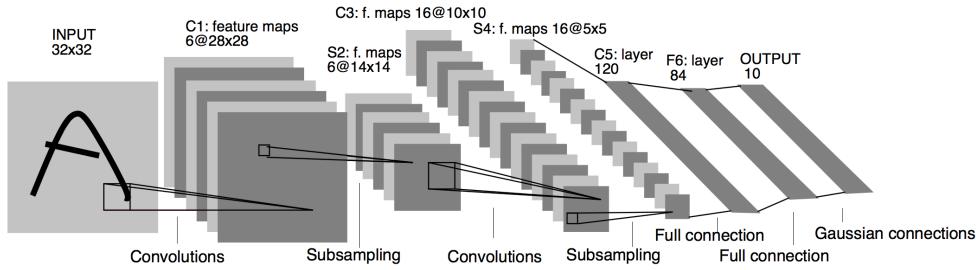


Abbildung 2.4: LeNet-5 cite lecun

Ziel dieser Operation ist es, dass die Filter Maps bestimmte Muster/features die zu einer bestimmten Klasse gehören lernen. Mit diesen Filtern können dann unabhängig wo im Bild befindlich, die Features wie zB horizontale/vertikale Linien, Ecken oder Kreise gefunden werden.

Beispiel:

$$\begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \quad (2.10)$$

erkennt vertikale Linien im Bild.

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 learning oder fine tuning) da die features (besonders in den vorderen layern) immer ähnlich sind und das neu lernen zeitaufwändig und oft sogar ungenauer ist. je nach Ähnlichkeit des eig datenset zu dem auf das netz ursprünglich 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 wahrscheinlichkeit angeben welche klasse darauf. keine lokalisierung 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

2.3.5 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 Tensorflow und Keras)

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

Anforderungen und Analyse

3.1 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 2.4 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.

3.2 Related Work

hier:

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

Kapitel 4

Realisierung Objekt Erkennung

4.1 Dataset

Da es sich um supervised Learning handelt müssen trainings daten gelabelt werden. für validierung und test muss datensatz zu 80, 10, 10 in test/train/validation aufgeteilt werden. wie in 2.1 beschieben dient das validierungs set zur überwachung während des trainings für overfitting.

Mit dem Test set kann nach dem training die Inferenz also das ausführen des traininierten models getestet werden.

Für die Objekterkennung muss der Datensatz wie in 2.3 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.

4.1.1 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 verfüzung stellen. Neben den in 2.3.3 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 4.1.3 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.

4.1.2 Augmentierung

was ist Augmentierung
wie wurde es angewandt
bsp bilder

4.1.3 TF Record Files

was es ist
wie es erstellt wurde

Wie in 4.1.1 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 serialisieren.

Evtl hier besp Ausschnitt von Aufbau eines Proto Elements.

Um nun die von OpenImages heruntergeladenen Bilder und Label Files in das TFRecords Format zu bringen waren mehrere Schritte nötig.

OI - VOC - csv - tf.records

4.2 Training

4.2.1 TF obj det api

Für das Training wurde das Framework Tensorflow verwendet, welches eine API für Objekterkennung bietet.

Welche pretrained Modell gibt es und welche kamen in Frage (für NCS2)? Die Tensorflow Object Detection API bietet eine Vielzahl an vorgebildeten Modellen, dabei wurden die meisten auf den COCO Datensatz trainiert.

Speed/Acc Trade Off

Daneben war bei der bei der Auswahl die Kompatibilität zu OpenVINO zu berücksichtigen: Liste von kompatiblen Modellen.

Trainiert wurde auf:

- SSD MobileNet und Inception
 - keine Region Proposal, dafür vordefinierte Ankerboxen, und CNN mit unterschiedlichen Schichten
- Faster R-CNN (Inception und ResNet)
 - ist ein Two-stage Detector: 1. ROIs mithilfe RPN oder SelSearch finden, darauf dann Classifier anwenden
- FRCNN

(In eval Ergebniss dann etwa so: SSD zu schlechte Performance und für Appl keine Realtime notwendig, FRCNN zu langsam, Faster R-CNN gute Mitte)
Hier ersten Durchlauf (mit Overfitting) darstellen

4.2.2 Regularisierung

Um das Overfitting zu vermeiden gibt es wie in 2.2 beschrieben verschiedene Möglichkeiten.
Untersucht wurde hier

- Augmentierung
- Early Stopping
- ... weitere z.B. L_1 , L_2

4.2.3 Training grayscale

Da die Kamera im Infrarot Modus ein Graustufen Bild mit nur einem Farbchannel liefert, muss dies für die Inferenz berücksichtigt werden.

Es ergeben sich hier mehrere möglichkeiten:

1. Normales (r, g, b) Netz
2. Ein Farbchannel (gr) Netz
3. Drei Farbchannel (gr, gr, gr)

Für 1. und 3. Müssten die bilder der Kamera vor der Inferenz auf 3 Farbchannel ($3 \times \text{grau}$) erweitert werden.

Um das Netz auf einen Channel zu trainieren wurde im config file ...

Um $3 \times \text{gray}$ zu trainieren wurden die Bilder in OpenCV in grau convertiert und wieder als jpgs abgespeichert.

Die Ergebnisse sind in Kapitel 5 dargestellt.

4.3 Parameter Optimierung

einstellungen im Config File

tensorflow graph oder plot zeigen

loss erklären (mit formel und für train und eval)

Kapitel 5

Evaluierung

5.1 Evaluierungs Metriken

mAP

- IoU Intersection over Unit: überlappung pred box mit ground truth box fläche beider boxen zusammen
- Confusion matrix
 - TP: True Positive (richtig auf ja getippt) (TP wird bei IoU \geq threshh. (übl 0.5) festgelegt)
 - TN: True Negative (richtig auf nein getippt)
 - FP: False Positive (fälschlicher ja getippt)
 - FN: False Negative (fälschlicherweise auf nein getippt)
- Precision: $TP/(TP + FP)$: möglichst viele richtige aus allen finden
- Recall: $TP/(TP + FN)$: möglichst **nur** die richtigen (sensitivity)
- AP = Precision-Recall verhältniss, genauer fläche unter kurve der beiden
- mAP für alle klassen gemittelt

Loss

kombination aus binärem classification log loss (cross entropy) für boxen und L1 smooth loss.

5.2 Ergebnisse

5.2.1 Modell Vergleich

hier modelle auf genauigkeit und geschwindig keit vergleichen und für nächste abschnitte eins auswählen

5.2.2 Regularisierung

ergibt dann zB early stoping und aug sind gleich.

daher wird im nächsten abschnitt geproüft wie sich die Modelle für Daten einer anderer Distribution verhalten.

Regularisierung	mAP	Loss
keine	1	2
Early Stopping	1	2
Augmentierung	1	2
weight decay	1	2

Tabelle 5.1: Regularization

5.2.3 Dataset Distributions

Da sowohl trainings als auch eval daten aus dem web bezogen wurden, also der gleichen distribution sind, diese aber nicht unbedingt den realen bedingungen entsprechen, wurde ein weiteres Evaluierungs Set mit Eigenen Aufnahmen erstellt, welche, da in der Umgebung (Wiltierpark in Reutlingen) aufgenommen, der tatsächlichen daten wahrscheinlichkeit eher entspricht.

ergebnisse tabellarisch vergleichen: handy vs orig für die in 5.2.2 beschriebenen Regularisierungs techniken

wenn für Augmentierung besser, heist das Augmentierung ist besser als early stopping für robustheit gegenüber umgebung (zb belichtungen)

Die Regularisierungen Early Stopping und Augmentierung wurden nun noch einmal mit einem eigenen Datenset evaluiert und miteinander verglichen.

Regularisierung	mAP_{orig}	mAP_{handy}	$Loss_{orig}$	$Loss_{handy}$
Early Stopping (100k steps)	0.6715	0.4265	0.6742	0.267
Augmentierung (200k steps)	0.6914	0.4537	0.6738	0.2503

Tabelle 5.2: Regularization

Wie zu erwarten unterscheiden sich die Loss Werte für das Origianel Eval Set wegen Early stopping nicht sehr. Für die Handy Bilder ist der Augmentierungs Loss etwas besser, was auf eine gr robustheit gegenüber Daten aus anderer Distribution zurückzuführen ist.

Die mAP Werte sind sowohl für original als auch für handy datenset bei Augmentierung deutlich besser, da sich wie in 5.1 der mAP erst später seinem Endwert annähert als der Loss.

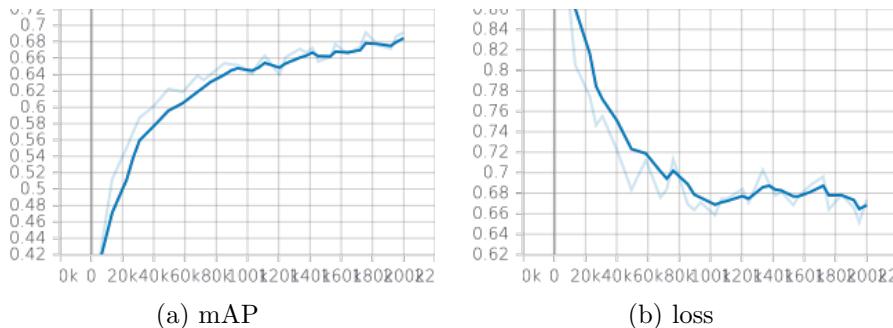


Abbildung 5.1: Loss und mAP für 200000 Steps

5.2.4 Graustufen/Infrarot Bilder

Da, wie die Ergebnisse in 5.2.2 und 5.2.3 gezeigt haben eine Augmentierung (welche Augmentierungen) die erfolgreichste Regularisierungs technik war, wurde für die Graustufen Bilder nur Auf Augmentierte Bilder mit faster Inception trainiert:

hier für die drei in 4.2.3 verwendeten modelle jwls für die in 5.2.3 beschriebenen eval daten sätze vergleichen.

Modell	Dataset	mAP	Loss
rgb	original	0.6556	0.1451
	handy	0.4155	0.2389
gray 1 channel	original	0.5625	0.1716
	handy	0.3226	0.2747
gray 3 channel	original	0.664	0.1653
	handy	0.438	0.2492

Tabelle 5.3: Grayscale

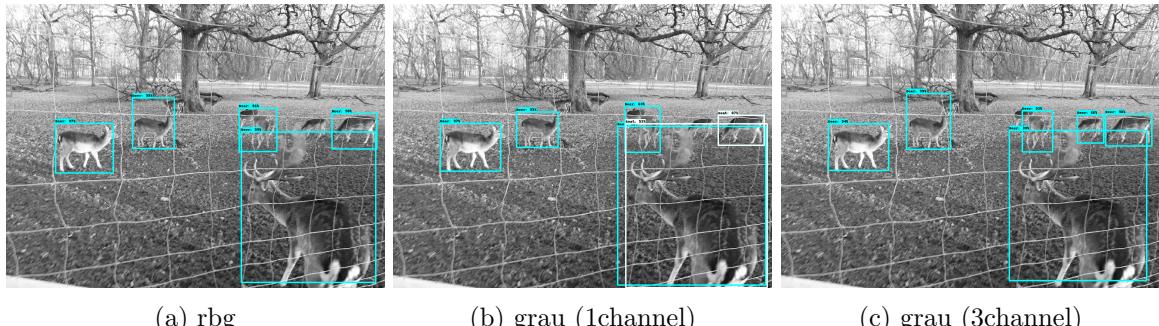


Abbildung 5.2: Vergleich der Inferenz von grau bildern für verschiedene Modelle

Diskussion des Ergebnisses: welchen einfluss haben Form und Farbe bei training, unnötig gelernte features für gray bilder? schnelligkeit?

Da es sich bei den convertierten graustufen bilder ja nicht um echte Graustufen bilder handelt, wurden Test Bilder von Alltags gegenständen mit der Für die Applikation verwendeten RaspiCam im Infrarot Modus aufgenommen. Um diese

(hat der Infrarot Modus mit Wärme, also lebendigkeit des Objektes zu zun??)

Um diese Zu testen wurde ein weiteres Netz auf diese Gegenstände (Gesicht, Hand,) trainiert und im folgenden auf den Datensatz true-ir evaluiert.

Kapitel 6

Entwicklung der Anwendung

In diesem Kapitel wird die Entwicklung der Anwendung als Autonomes Edge System auf dem Raspberry Pi zusammen mit dem Neural Compute Stick 2 und einer geeigneten Kamera beschrieben. Ebenso wird die Integration des trainierten Tensorflow Models in die Applikation sowie die Implementierung der Netzwerk Verbindung zu dem System beschrieben.

6.1 Aufbau

Die Anwendung soll auf dem ein Platinen Computer Raspberry Pi 4 Abbildung ?? laufen, an den die nötigen Komponenten angeschlossen werden. Dazu gehören der Neural Compute Stick 2, zur Ausführung der Inferenz, ein Kamera Modul, mit welchem die Bilder aufgenommen werden, sowie ein WiFi Stick und Powrebank.

Der Neural Compute Stick wird über USB angeschlossen und kann nach installation des OpenVino Toolkits 6.2 verwendet werden.

Bei der Kamere handelt es sich um ein Infrarot Fähiges *RaspberryPi Camera Module* welches zusammen mit zwei Infrarot LEDs montiert wird. 6.3



Abbildung 6.1: Raspberry Pi 4

- Netzwerkverbindzuung:
 - GSM Module
 - WiFi-Stick
- Powrebank/Akku aus Sp. Verbrauch von:
 - NCS2
 - Kamera
 - LEDs
 - WiFi Stick

6.2 OpenVino Toolkit

Die Implementierung der Inferenz des trainierten Models wurde mithilfe des OpenVino Toolkits vorgenommen, eine Anwendung zur Optimierung und Ausführung von CNNs auf Intel Hardware. Es vereinfacht und Optimiert damit die Verbindung zwischen Training des Models und bereitstellen in einer Anwender Applikation, wie in Abbildung 6.2 schematisch dargestellt.

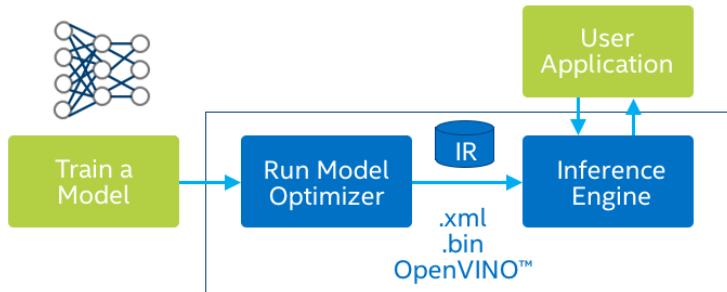


Abbildung 6.2: Workflow: OpenVino Toolkit

Das Toolkit besteht im Wesentlichen aus den zwei Komponenten *Model Optimizer* und *Inference Engine*

Mit dem Model Optimizer können Netze die in den Frameworks TensorFlow, Caffe, MXNet, Kaldi oder ONNX trainiert wurden in die von OpenVino verwendete Intermediate Representation des Modells gebracht werden.

Diese ist ein Framework unspezifisch Dateiformat, welches aus einer .xml Datei für die Struktur/Architektur des Modells und einer .bin Datei für die trainierten gewichten besteht.

Die InferenceEngine ist eine Runtime welche eine API für die Sprachen C++ und Python zur Integration und Nutzung der Inferenz in der Anwendung bereitstellt.

Dafür werden die IR Dateien des Models in ein Hardware spezifisches Plugin geladen. Dieses kann die User Applikation für die Inferenz von Image Classification, ObjectDetection sowie Instance Segmentation Modellen nutzen.

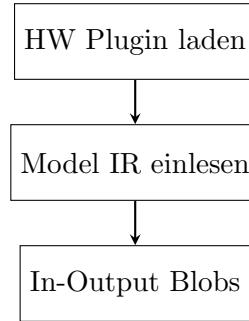
Implementierung

Die Implementierung der Inferenz wurde in Python vorgenommen.

Dafür waren folgende Schritte nötig:

1. HW Plugin laden
2. Model IR einlesen
3. In-Output Blobs allokieren
4. ausführbares Model laden
5. inferenz request abgeben
6. Bild als Array in Input Blob laden
7. Inferenz
8. Output verarbeiten, wieder zu Schritt 6

Blobs sind In-Output Tensoren



```

plugin = IEPlugin(device='MYRIAD')
net = IENetwork(model=model_xml, weights=model_bin)
input_blob = next(iter(net.inputs))
exec_net = plugin.load_network(network=net)
infer_request = exec_net.requests[request_id]
# bild mit opencv als numpy arra laden und von hwc nach nchw umstellen
res = exec_net.infer(inputs={input_blob: image})
# res enthaelt liste mit allen erkannten klassen auf dem Bild
# fuer Objekt Detection zusaetlich noch Bounding Box koordinaten
  
```

Die Inferenz kann entweder Synchron oder Asynchron ausgeführt werden. Der programmatische Ablauf der hier verwendeten asynchronen Inferenz ist im Folgenden als Pseudocode dargestellt.

```

while true do
    capture frame;
    populate Next InferRequest;
    start Next InferRequest; // asynchroner aufruf
    if wait for Current done then
        // wird in eigenem verarbeitet
        display Current;
    end
    swap Current and Next InferRequests;
end
  
```

Algorithm 1: Asynchrone Inferenz

Das Ergebnis eines InferRequest für Object Detection Modelle enthält eine Liste mit allen möglichen erkannten Objekten, jedes davon bestehend aus einem Array mit den Indices:

0. batch index
1. class label
2. Wahrscheinlichkeit
3. x_{min} Box Koordinate
4. y_{min} Box Koordinate
5. x_{max} Box Koordinate
6. y_{max} Box Koordinate

Mit über die Wahrscheinlichkeit ließen sich die Ergebnisse nach einem bestimmte Threshhold ausfiltern.

Die Box Koordinaten wurden in Prozent der Bild- Breit/Höhe angegeben wodurch sie wieder in die Original bild Größe für die Bounding Boxes übertragen werden können.

6.3 Raspberry Pi Kamera

Bei der Kamera handelt es sich um das OV5647 5MP Modul mit regelbarem Infrarotfilter. Zusammen mit zwei Infrarot LEDs von der Firma Quimat Abbildung ??

Wird der Infrarotfilter ausgeschaltet ist es durch die Infrarot LEDs mit 850nm welligen Licht möglich auch bei Dunkelheit Aufnahmen zu machen, die in Graustufen Werten dargestellt werden.

6.4 Server-Client-Connection

6.5 Anwendung gesamt

Da die Inferenz sehr rechenaufwendig ist, sollen die Frames der Kamera nur dann inferiert werden, wenn eine Bewegung stattfindet. Dafür wurde der Inferenz ein Bewegungsmelder vorgeschaltet. Dieser wurde mithilfe der Library OpenCV implementiert, indem zu Beginn des Kamerea Streams ein Referenzbild gespeichert wurde, mit dem die aktuellen Frames verglichen werden. Ist der absolute Abstand der einzelnen Array Elemente/Werte der Bilder größer als ein bestimmter Threshold, wird dies als Bewegung gewertet.

Kapitel 7

Test und Validierung

Kapitel 8

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] 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 Kapitel im Anhang

A.1 Bsp für ein Abschnitt im Anhang