Dies ist ein Beispiel für ein Todo, und sollte vor Abgabe gelöscht werden ;)

Hochschule für Technik Rapperswil HSR

MRU: Sensors, Actors and Communication

Studienarbeit

Yolo auf Finger

im Studiengang Industrial Technologies

eingereicht von: Heinz Hofmann <hhofmann@hsr.ch>

eingereicht am: 9. Februar 2017

Betreuer/Betreuerin: Herr Prof. Dr. G. Schuster
Frau T. Mendez

Inhaltsverzeichnis

1 Abstract **4**

1.1 Aufgabenstellung 4

1.2 Vorgehen 4

1.3 Fazit 4

2 Resultate **6**

2.1 Testvoraussetzungen 6

2.2 Analyse 6

2.2.1 Distanz 6

2.2.2 Intersection Over Union IOU 10

3 Daten-Pipeline **12**

3.1 Bilder aufnehmen 12

3.2 Fingerdetektion 13

3.3 CSV generieren 14

3.4 Python-Objekt generieren 14

3.4.1 Label-Tensor 15

3.4.2 List of Label-Tensors 15

3.5 Daten in Neuronales Netzwerk einlesen 15

Literatur **17**

Abbildungsverzeichnis

1 Bedeutung der normierten Distanzwerte in der realen Welt . . 7

2 Prediction knapp besser als Distanz=0.02 8

3 Prediction knapp schlechter als Distanz=0.02 8

4 Komplette Wahrscheinlichkeits-Dichte-Funktion der Distanz (Grenze: Dist=0.02) 9

5 Wahrscheinlichkeits-Dichtefunktion der Distanz. Ausreisser nicht miteingerechnet (Grenze: Dist=0.02) 9
6 Prediction knapp besser als IOU=0.4 11
7 Prediction knapp schlechter als IOU=0.4 11
8 Wahrscheinlichkeits-Dichte-Funktion der IOU (Grenze: IOU=0.4) 12
9 Resultate Hough-Transformation 13
10 Label-Tensor 16

1 Abstract

1.1 Aufgabenstellung

Das Ziel dieser Projektarbeit war es, herauszufinden, ob Yolo geeignet wäre, die Fingerspitzen einer Hand in einem Bild zu klassifizieren und genau zu detektieren. Die Vorgaben, was die Genauigkeit betreffen lagen bei 0.1mm. Yolo ist eine Möglichkeit, um mittels Deep-Learning Objekte in einem Bild zu klassifizieren und gleichzeitig deren genaue Position zu detektieren. Daher auch der Ausdruck Yolo (You only look once). Yolo wurde als Konzept gewählt, weil es in diesem Bereich dem aktuellen Stand der Technik entspricht. Gerade die Geschwindigkeit dieses Netzwerks wurde als extrem hoch angepriesen (bis zu 45fps). Diese Geschwindigkeit ist für die letztendliche Anwendung von hoher Wichtigkeit, weil es sich schlussendlich um eine Echtzeitanwendung handeln soll.

1.2 Vorgehen

Mithilfe der Apparatur und Software von Tabea Méndez wurden zuerst Daten generiert. Um diesen Aufwand klein zu halten, wurden nur Daten vom rechten Zeigefinger generiert. Gleichzeitig wurde in Tensorflow die Architektur von Yolo nach gebaut. Dies wäre nur begrenzt nötig gewesen, da fertige Architekturen in Keras oder Darknet online zur Verfügung stehen würden. Um aber einen Lerneffekt im erstellen von Neuronalen Netzwerken zu erzielen, wurde trotzdem alles von Grund auf selber aufgebaut. Rund um die Kernarchitektur von Yolo wurde das Datenhandling, die Kostenfunktionen aber auch sämtliche Validierungen und Tests zweimal erstellt. Einmal für das Pretraining der Kerngewichte auf dem ImageNet Klassifizierungsdatenset und einmal für das ëchte"Training auf den selber generierten Daten. Sobald dies alles aufgebaut und lauffähig war, wurde noch so viel wie möglich experimentiert, um herauszufinden, mit welchen Änderungen und Einstellungen das Lernresultat noch optimiert werden könnte.

1.3 Fazit

Das Pretraining und auch das Training hatten seine Tücken, weil das originale Yolo-Netzwerk extrem gross ist, und entsprechend nahezu den ganzen RAM-Speicher einer GPU benötigte, wodurch nur noch begrenzt Platz für Daten übrigblieb. Diese Probleme konnten einigermassen umgangen werden,

hatten jedoch zur Folge, dass die Bilder von 1280x960 auf 448x448 verkleinert werden mussten, um das Netzwerk zum laufen zu bringen. Dies hatte zur Folge, dass ein Pixel bereits bis zu 1,5mm entsprechen konnte. (Dies sollte Yolo theoretisch nicht daran hindern genauere Aussagen über die Position des Fingerspitzen zu machen.) Trotzdem wurde mit rund 84% der Predictions nur eine Genauigkeit von 15mm erreicht, was in etwa 10 Pixeln entsprach. Mit diesem Ergebnis wurde zwar das Ziel der Aufgabenstellung (0.1mm) um Faktor 150 verpasst, allerdings in 84% der Fälle Predictions gemacht, welche aus subjektiver menschlicher Sicht "gutßind. Dies ist ein einigermaßen erstaunliches Resultat, wenn man bedenkt, dass man zum Trainieren nur rund 18'000 Bilder verwendet hatte. Es ist anzunehmen, dass mit einer Verbesserung der Datengewinnung und entsprechend viel mehr Daten in naher Zukunft mit diesem oder einem ähnlichen Konzept eine Genauigkeit von bis zu 1mm erreicht werden können sollte.

2 Resultate

2.1 Testvoraussetzungen

Das Netzwerk wurde auf 1'200'000 Bildern des ImageNet-1000-class-Datasets vortrainiert. Danach wurde es auf rund 13'900 Bildern aus dem Testaufbau [1] trainiert. Der Test wiederum wurde auf rund 1'500 Bildern ebenfalls aus dem Testaufbau [1] getestet. Diese Testbilder waren dem Algorithmus während des Lernprozesses nicht zugänglich und haben entsprechend keinen Einfluss auf den Lernprozess genommen. Ausserdem wurden diese Bilder so gewählt, dass Sie nicht gleichzeitig aufgenommen wurden. Dies verhindert, dass fast identische Bilder im Training und im Test vorkommen.

2.2 Analyse

Um die Genauigkeit der Predictions unseres Neuronalen Netzwerkes möglichst genau beschreiben zu können wurden die zwei Werte Distanz und IOU gewählt. Obwohl die beiden Werte korrelieren sagt jeder für sich nicht die volle Wahrheit über die Genauigkeit der Vorhersagen aus. Die Distanz ist für die geplante Anwendung der wesentlichere Wert, weil diese Informationen über den Standort des Fingers im Bild preisgibt. Die IOU ist mit der Distanz klar korreliert, denn ist die Distanz zu gross, ist die IOU schnell gleich Null. Sobald die Boundingbox der Prediction und die Boundingbox des Labels sich beginnen zu überlappen sagt die IOU etwas über die korrekte Vorhersage von Breite und Höhe der Boundingbox aus. Auch darüber ob die Box am richtigen Ort liegt, können aufgrund der IOU vage Annahmen getroffen werden. Aber wie gesagt, die Distanz ist dafür der sicherere Wert.

2.2.1 Distanz

Die Distanz beschreibt die normierte Differenz zwischen dem Zentrumspunkt des Labels und dem Zentrumspunkt der Vorhersage. Sämtliche Distanzen wurden so normiert, dass die Höhe des Bildes und auch die Breite gleich eins sind. Die maximale Distanz zwischen zwei Punkten ist also die Diagonale über ein Bild, welche entsprechend sqrt(2) ist. Was diese Normierten Distanzen in der realen Welt bedeuten ist auf Abbildung 1 erklärt. Zum Vergleich, ein Menschlicher Zeigefinger ist zwischen 10 und 20 mm breit. Eine normierte Distanz von 0.02 entspricht auf unserem Versuchsaufbau somit ziemlich genau der Breite eines menschlichen Fingers.

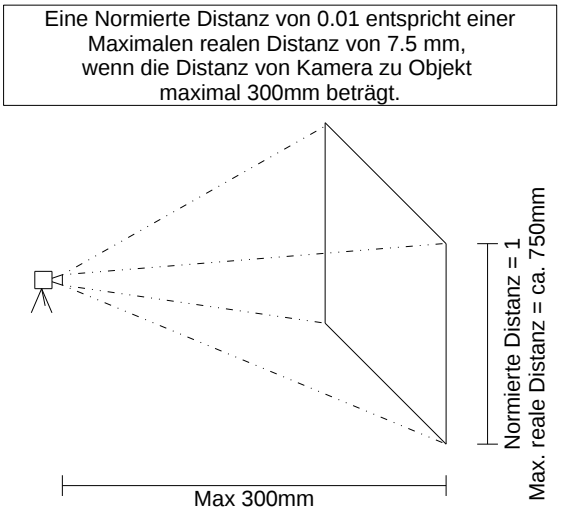

Abbildung 1: Bedeutung der normierten Distanzwerte in der realen Welt

Um die Resultate in gut und schlecht einteilen zu können wurde ein Threshold von 0.02 definiert. Die Definition dieses Thresholds wurde gemacht, indem Bilder zusammen mit der entsprechenden Distanz analysiert wurden. Der Wert 0.02 entspricht somit derjenigen Distanz, welche gerade noch knapp annehmbar ist, um einen Finger als detektiert gelten zu lassen. Um ein Gefühl für diese Distanzen zu kriegen lohnt es sich die Abbildungen 2 & 3 anzusehen, welche Bilder zeigen, die eine Distanz nahe dieses Thresholds aufweisen.

Um die Verteilung der Distanzen gut verstehen zu können, ist in Abbildung 4 eine Wahrscheinlichkeitsdichte der Distanzen im Testset zu sehen. Diese Dichtefunktion wurde erst nach der Bestimmung des Thresholds erzeugt und zeigt, dass rund 84% der Distanzen kürzer sind als 0.02 und somit die entsprechenden Finger ërfolgreichërkannt wurden.

Erstaunlich ist auch, dass die Distanzen, welche grösser als 0.25 sind in der Wahrscheinlichkeitsdichte in kleinen Bündeln vorkommen. Dies lässt darauf schliessen, dass die Trainingsdaten nicht komplett Bias-Frei sind. Nach kurzer Kontrolle konnte tatsächlich festgestellt werden, dass z.B. bei einer Distanz von ca. 0.4 immer ein bestimmter Punkt des Hintergrundes vorhergesagt wurde, welcher tatsächlich ganz selten in den Labels als Finger markiert wurde.

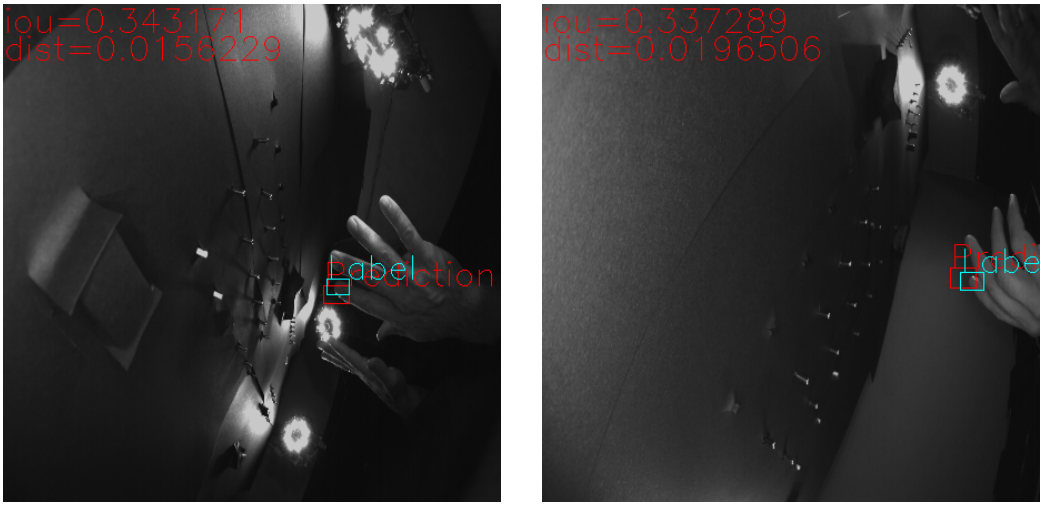

Abbildung 2: Prediction knapp besser als Distanz=0.02

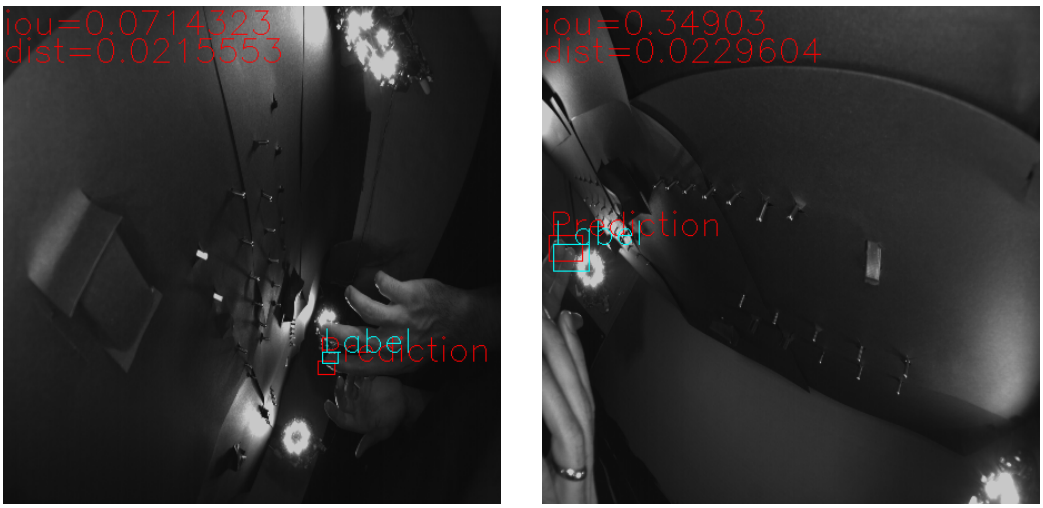

Abbildung 3: Prediction knapp schlechter als Distanz=0.02

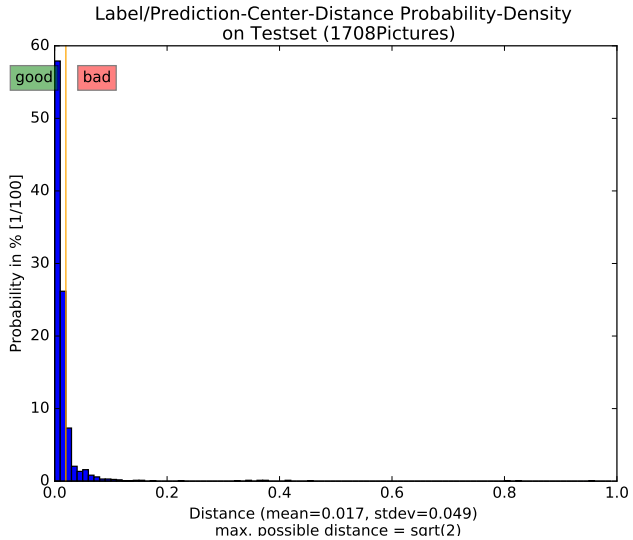

Abbildung 4: Komplette Wahrscheinlichkeits-Dichte-Funktion der Distanz (Grenze: Dist=0.02)

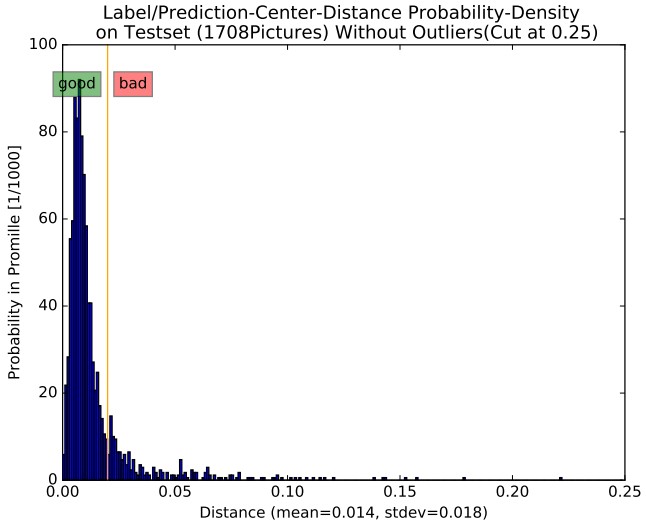

Abbildung 5: Wahrscheinlichkeits-Dichtefunktion der Distanz. Ausreisser nicht miteingerechnet (Grenze: Dist=0.02)

Um die Statistik nicht von Ausreissern, welche aufgrund von falschen Labels entstanden sind verfälschen zu lassen, wurde wie in Abbildung 5 noch eine zweite Wahrscheinlichkeitsdichte-Funktion erstellt. Spannend: Der Mittelwert ist sofort um einen Drittel kleiner als zuvor.

2.2.2 Intersection Over Union IOU

Die IOU beschreibt die Überlappung der vorhergesagten Boundingbox und der Boundingbox des Labels. Daher sagt die IOU einerseits etwas über die korrekte Grösse der Boundingbox, sowie deren korrekte Lage aus. Um wieder etwas über gut und schlecht aussagen zu können, wurde wieder ein Threshold definiert (0.4). Da durch die IOU wie erwähnt mehrere Faktoren beschrieben werden, ist die Grenze verschwommener. So gibt es nach menschlicher Ansicht hervorragende Vorhersagen, welche eine IOU von 0.3 haben und wiederum mässige Vorhersagen mit einer IOU von nahezu 0.4. Um ein Gefühl für diesen Threshold zu kriegen lohnt es sich die Abbildungen 6 & 7 zu berücksichtigen. So fiel die Entscheidung den Threshold konservativ zu wählen, sodass nur Werte als gut erachtet werden könne, welche auch gut sind.

Auch für die IOU gibt es zur Übersicht eine Wahrscheinlichkeitsdichte die in Abbildung 8 betrachtet werden kann. Aus dieser Grafik kann gelesen werden, dass rund 6% der Vorhersagen klar falsch sind, weil die IOU nur null ist, wenn sich die beiden Boundingboxen nicht berühren. Entsprechend kann gesagt werden, dass rund 94% der vorhersagen zumindest sehr grob richtig sind, weil sich bei diesen 94% die Boundingboxen von Label und Prediction zumindest ein ganz kleines bisschen überlappen.

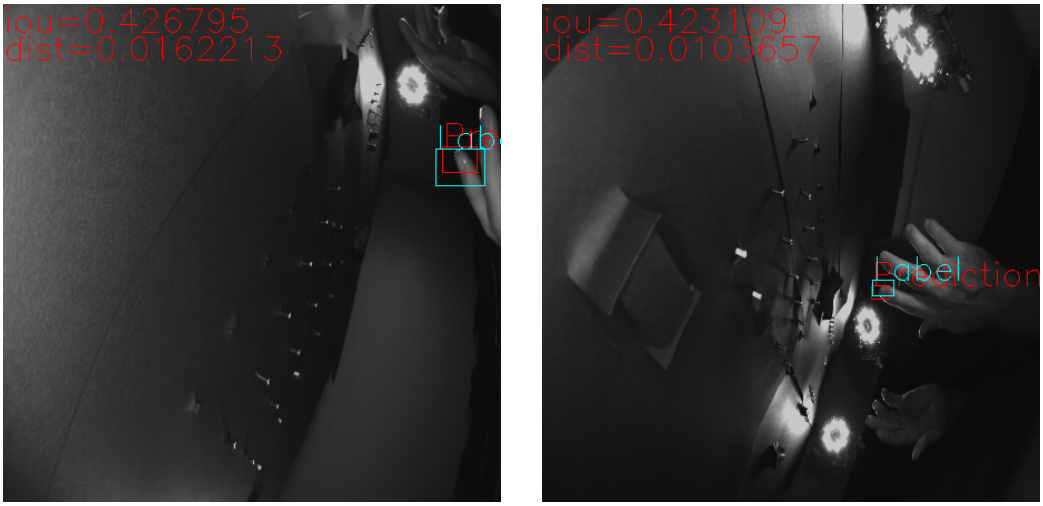

Abbildung 6: Prediction knapp besser als IOU=0.4

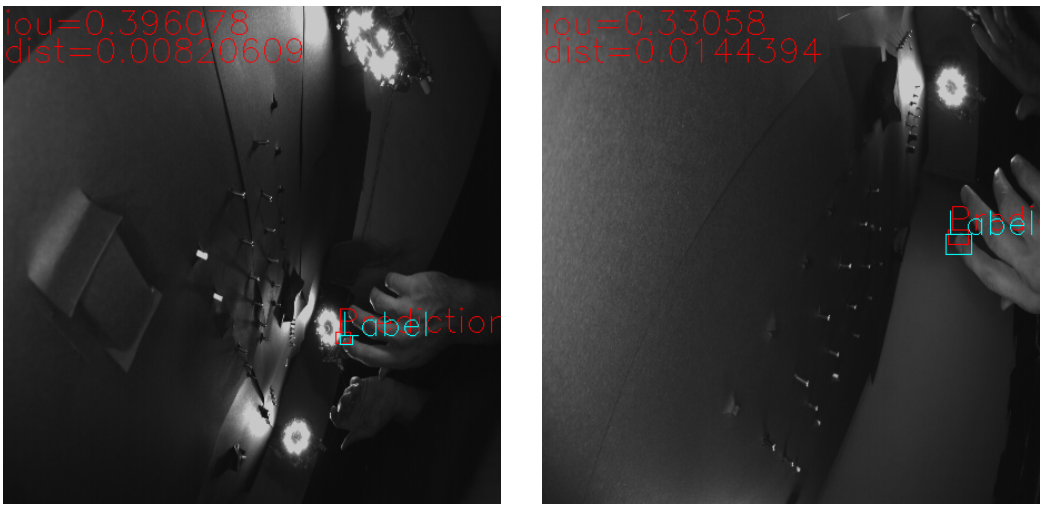

Abbildung 7: Prediction knapp schlechter als IOU=0.4

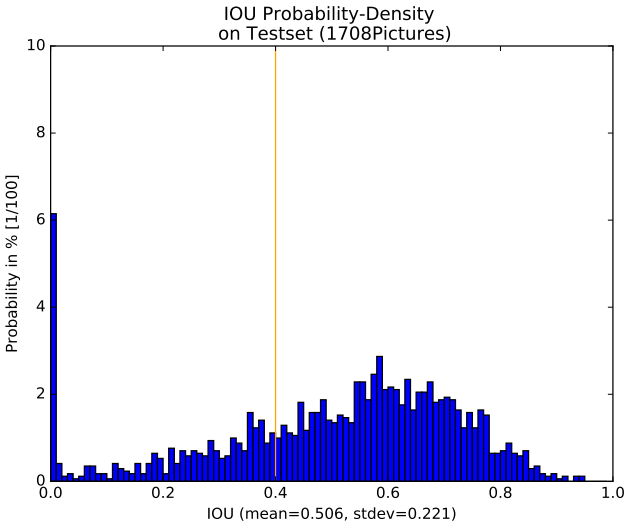

Abbildung 8: Wahrscheinlichkeits-Dichte-Funktion der IOU (Grenze: IOU=0.4)

3 Daten-Pipeline

3.1 Bilder aufnehmen

Die Aufnahme der Bilder geschah unverändert mit Apparatur und C++ Code von Tabea Méndez welche aus Ihrer Masterarbeit [1] entstand. Das Ergebnis waren jeweils 8 Bilder aus einer Situation. Eine Situation bestand aus 4 Kameras, wobei jede Kamera jeweils ein schwarzweiß-Bild mit UV-Beleuchtung und ein schwarzweiß-Bild mit normaler weisser Beleuchtung gemacht hatte. Wegen schlechter Erfahrungen mit Restlicht wurde der Aufbau mit schwarzem Papier abgedeckt. Diese schlechten Erfahrungen wurden gemacht, weil zu diesem Zeitpunkt zum Labeling der Daten noch keine Zeitinformation verwendet, also die Finger noch nicht von Bild zu Bild getrackt wurden. Pro Durchgang konnten maximal 6000 Situationen aufgenommen werden, bevor der Arbeitsspeicher des dafür verwendeten Computers an seine Grenzen kam.

Eine Verbesserung könnte hier erreicht werden, wenn man das Programm in 2 verschiedene Threads aufteilen würde. Dabei wäre ein Thread für das Aufnehmen der Daten und der andere für das abspeichern derselben zuständig. So könnte “zeitlich unbegrenzt“ Daten aufgenommen werden. Dies würde aber nur nötig, falls tatsächlich in Zukunft mit einem Roboter Daten aufgenommen würden.

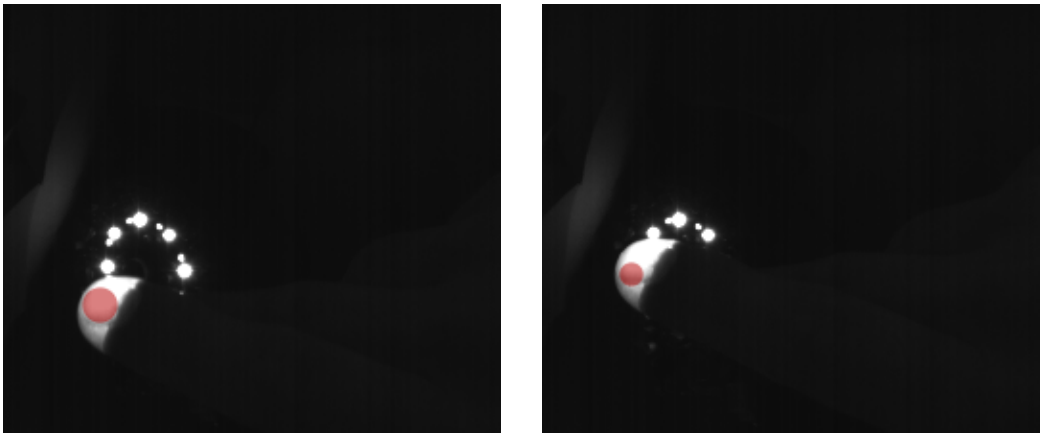

Abbildung 9: Resultate Hough-Transformation

3.2 Fingerdetektion

Zur Fingerdetektion wurde der Matlab-Fingerdetektor aus der Masterarbeit von Tabea Méndez [1] verwendet. Um Zeit bei der Datenaufnahme einzusparen wurde Zeit und Rauminformation nicht miteinbezogen. Dies brachte einige neue Probleme mit sich. So wurden auch mit Restlicht beleuchtete Punkte im Hintergrund oder LED's, als Finger erkannt. Dieses Problem konnte aber weitgehend behoben werden, indem in den Matlab-Fingerdetektor noch einige Filter eingebaut wurden.

1. Überspringen von Bildern, welche eine gewisse Helligkeit überschreiten. Dies sortiert Bilder aus, welche eine grosse Hintergrundhelligkeit und dadurch auch viele fehlerhaft erkannten Fingerspitzen enthält aus.

2. Aussortieren von erkannten Punkten, die zu gross sind, als dass Sie ein Fingerspitz sein könnten. Dieser Punkt ist teilweise redundant mit dem ersten Punkt, da so grosse Punkte nur im Hintergrund bei einer extrem grossen Helligkeit auftreten können.

3. Aussortieren von erkannten Punkten, welche zu klein sind, als dass Sie eine Fingerspitze sein könnten. Damit werden die meisten LED-Punkte entfernt.

4. Von den übrigen Punkten wird dann nur noch der Grösste behalten. Diesem wird somit das Label "rechter Zeigefingerspitz" verliehen.

Mit diesen Filtern konnte ein hoher Prozentsatz der rechten Zeigefinger korrekt detektiert werden. Auf den Finger selber bezogen war die Genauigkeit

leider jedoch relativ schlecht. Dies aus dem einfachen Grund, dass die Detektionspunkte nicht immer genau in der Mitte des Fingers zu liegen kamen. Weiter waren auch die Boundingboxen, welche sich aus dem Resultat der Hough-Transformation [1] berechnen liessen nicht sehr genau. Wie man in der Abbildung 9 sehen kann, können sich diese innerhalb des Fingers auch bei sehr ähnlichen Bildern stark unterscheiden.

3.3 CSV generieren

Das Fingerspitzentracking wurde mit Matlab gemacht und die entsprechenden Labels als .mat-File abgespeichert. Das Deeplearning hingegen wurde mit Tensorflow und entsprechend mit Python angegangen. Leider war es nicht möglich mit Python direkt .mat-Files zu öffnen. Aus diesem Grund wurde ein kleines Matlab-Skript erstellt, welches die Labels als CSV abspeichert. Im Zuge dieses Skripts wurden ausserdem die Daten in Test und in Trainingsdaten aufgeteilt und je einem seperaten CSV abgespeichert. In diesem CSV gehört jedem Bild eine Zeile. Pro Zeile bzw. Bild werden folgende Punkte beschrieben:

1. Eindeutiger Bildname, mit welchem das Bild aus dem Directory geladen werden kann.
2. X-Koordinaten im Range [0:1280]
3. Y-Koordinaten im Range [0:960]
4. Durchmesser des Resultats der Hough-Transformation
5. Wahrscheinlichkeit, dass ein rechter Zeigefinger in diesem Bild ist. (Entweder 1 oder 0, je nach dem, ob ein Finger erkannt wurde.)

3.4 Python-Objekt generieren

Um die Daten einfach im Trainingsprozess aufrufen zu können, wurde eigens eine kleine Python-Klasse geschrieben. Die Daten müssen allerdings noch vor deren Verwendung im Training durch eine Funktion dieser Klasse vorverarbeitet werden. Die Gründe für die Vorverarbeitung sind:

1. Die Bilder wurden bisher Kameraweise bearbeitet. Dies bedeutet, die Bilder heissen bei verschiedenen Kameras genau gleich. Mit der Vorverarbeitung werden alle Bilder an einem gemeinsamen Ort gespeichert.

Ausserdem erhält jedes Bild einen neuen Namen / eine neue Nummerierung, was sie eineindeutig bezeichnen lässt.

2. Um im Training einfach mit den Label-Daten umgehen zu können und um Rechenaufwand während dem Training zu sparen wurden in der Vorverarbeitung die Labels zu demjenigen Tensor zusammengefügt, welcher in Abbildung 10 zu sehen ist.

3. Die Distanzen X und Y sowie die Höhe und Breite der Boundingbox mussten noch normalisiert werden, damit beim Training einfacher gerechnet werden kann.

3.4.1 Label-Tensor

Die Labels pro Bild sind in einem Tensor angeordnet. (Siehe Abbildung 10) Diese Anordnung wurde stark am Output-Tensor wie er im Yolo-Paper [2] erscheint angelehnt. Dabei wird das Bild in ein 7x7 Raster aufgeteilt. Für jedes Element dieses Gitternetzes....

Hier weiterschreiben

1. x = Die Distanz des Zentrums der Fingerspitze zum linken Rand der Gitterzelle. Diese Distanz ist normiert auf den Bereich [0:1]. Ist der Zeigefinger nicht in dieser Gitterzelle, ist die Variable x = 0.

2. asdfasd

3.4.2 List of Label-Tensors

3.5 Daten in Neuronales Netzwerk einlesen

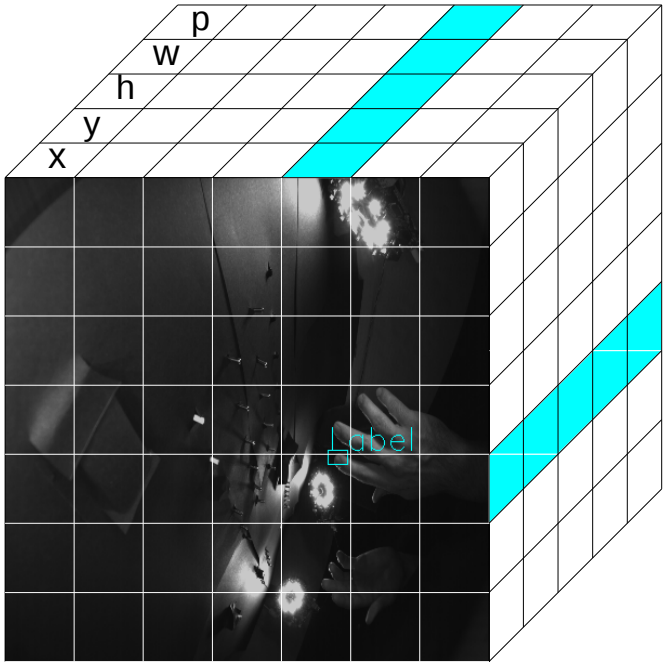

Abbildung 10: Label-Tensor

Literatur

[1] Tabea Méndez. Fingerspitzen-Tracking im 3D-Raum. Master's thesis, HSR, June 2017.

[2] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.