Projekt Meeresblick

Malte Neysters, Finn Brüggemann, Nico Fuhrberg, Niklas Stockfisch

21. Dezember 2024

1 Vorläufig Malte

Ich habe ausprobiert was man mit der Deep Java Library (kurz djl) machen kann in dieser Woche, djl ist eine Schnittstelle die es einem erlaubt verschiedene Machine Learning Frameworks (mxnet, PyTorch, ONNX RUNTIME und TensorFlow) zu benutzen. Ich habe zuerst mit den verschiedenen Frameworks die Object Detection tests ausgeführt die schon mitgeliefert werden um mir deren Ausgaben anzuschauen und konnte dabei keinen großen Unterschied merken. Darauf habe ich mir die verschiedenen vor trainierten Modelle angeschaut die jeweils Unterstützt werden und festgestellt das PyTorch, TensorFlow und mxnet alle 3 Faster RCNN unterstützen und ONNX RUNTIME PyTorch modelle unterstützt, somit habe ich dann nochmal mitgelieferte Tests für Faster RCNN ausgeführt und fand dabei die Nutzung von TensorFlow am besten und weitere Recherche von Andreas ergab auch, dass ONNX sowie mxnet bei kleineren Objekten probleme haben. Da wir am Ende lieber direkt mit dem Framework arbeiten wollen habe ich mich entschlossen dann zusammen mit Niklas weiter mit TensorFlow zu arbeiten. Wir sind dann diesen beiden Tutorials gefolgt (https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/install.html und https://www.youtube.com/watch?v=yqkISICHH-U) um zu lernen wir man TensorFlow Keras mit seinem eigenen Datensatz trainiert, zum Testen haben wir hierfür zuerst mit labelImg einen eigenen Datensatz erstellt mit je 20 bildern für Treibholz und Müll. Wir hatten hier natürlich die Arbeit aufgeteilt und jeder von uns beiden hat je 20 Bilder gelabelt mit den entsprechenden Boundary Boxen, sobald wir die korrespondierenden xml dateien austauschen wollten haben wir festgestellt in den xml Dateien werden absolute Dateipfade zu dem dazugehörigen Bild angegeben, somit mussten wir zuerst diese alle bearbeiten damit sie auch auf unseren jeweiligen Rechnern gingen. Das nächste Problem trat auf als wir damit unsere record Dateien generieren wollten, da zwar alle Bilder das Dateikürzel .jpg hatten, aber nicht alle .jpg waren hierfür gab es zum glück ein einfaches skript das wir online gefunden haben das schaut welche Dateien welche .jpg sind, nachdem wir die Richtigen Dateien rausgefiltert haben konnten wir also unsere record dateien erzeugen. Zuvor hatten wir

natürlich unsere Bilder in einen Test - und Trainingsdatensatz aufgeteilt, nun konnten wir uns also endlich ein Modell aussuchen und mit hilfe von unseren Datensätzen schauen wie gut diese Funktionieren. Wir hatten uns zunächst für faster_rcnn_resnet50_v1_1024x1024_coco17_tpu-8 und ssd_mobilenet_v2_320x320_coco17_tpu-8 entschieden, da wir diese auch bei anderen Frameworks schon genutzt hatten, nach anfänglichen Problemen mit der Einrichtung der pipelinfe.config Datei, da wir Dateipfade zunächst mit \und absolute angegeben hatten, die Lösung dazu war es stattdessen als relative Dateipfade mit / genutzt haben und diese als Strings angegeben haben, da windows sonst Probleme hat diese zu erkennen. Bei beiden Modellen hatten wir nicht viele True Positives, vermutlich aufgrund unseres kleinen Datensatzes, aber wir hatten kein einziges False Positive. Auch aufgrund des kleinen Datensatzes hatten wir viele False Negatives natürlich. Also wurde als nächstes Überlegt mit hilfe der Edge Detection von Lucas zunächste alles mögliche aus dem Bild herauszufiltern wie das Meer damit man dann weniger Bildfläche hat auf der das Modell suchen muss.

2 Finn

- Information zu OD mit Yolo - Test/Kompilierung von Yolov3 unter Windows sehr aufwendig, da VS BuildTools 2021 (englisch) benötigt wurden und es dauerte, eh der Fehler gefunden war -Yolo9000 Darknetversion von Yolo mit 9000 verschiedenen Klassen, entstanden auf der CVPR 2017 (dadurch relativ alt) https://github.com/philipperemy/yolo-9000

- Versuch Taco (Trash annotation dataset) mithilfe von Yolo zu nutzen - Taco Gitrepo kaum gepflegt - Vergleich der Datasets unter https://github.com/AgaMiko/waste-datasets-review - leider sehr wenig Dokumentation verfügbar und geringe Pflege - Classification/Segmentation nicht sinnvoll, benötigen Detection - Taco bspw kommt mit deutlich veralteter Version von MaskRCNN, die nicht mehr mit aktuellem Python läuft - Versuch TACO mit neuem MaskRCNN scheiterte auch

- Litter nur bezahlt nutzbar, keine Demo -Openlittermap zwar vielversprechend, aber erneut kaum Angaben zum Backbone, sondern nur Appdownload - TACO bboxes an sich optimal, allerdings weiterhin Work in Progress und keinerlei Informationen, wann fertig

- Gedanke TACO per roboflow in Yoloformat umzuwandeln - Upload zwar möglich, aber da Unterteilung in 15 Unterpakete überschreiben sich die Annotationen einzelner Pakete gegenseitig - Ansonsten kein OnlineAnbieter gefunden, der Datasets zwischen Formaten konvertiert - eine Menge von offlinetools verfügbar, allerdings funktionieren alle nur für genau spezifizierte Formate, Tacoformat sticht heraus durch Unterordner (Bspw. https://github.com/ISSResearch/Dataset-Converters) sollte eig Yolo to Coco uvm. unterstützen) -yolov5 besitzt umfängliches Githubrepo mit Beispielen und Anleitungen zum Training - Annotation eines kleinen Datasets aus 20Bildern und Testtraining von weights verlief problemlos,

allerdings kaum Erkennungen durch zu geringe Datenmenge

-Coco Download in verschiedenen Formaten über roboflow möglich, auch richtiges Format für yolov5 - besitzt wohl am meisten gelabelte Bilder und Klassen - leider nach Durchsicht kaum Label im Bereich Müll, allerdings viele Bilder von Alltagsgegenständen, die auf Strandbildern herausgefiltert werden sollten - mögliche Erstellung von Subdatenset

-Annotation von 52 Bildern mit Handtüchern am Strand (erstaunlich wenig originalmaterial zu finden, viel Werbefotos) Download von Bing und Google XML-Annotation mit labelimg

- Erstellung eines kleinen Pythontools zur Markierung zweier Punkte auf geladenem Bild und späteren Angabe der Länge in Real, um durch das Ergebnis Objektgrößen zu berechnen

2.1 Kantenerkennung/Meereslinie

3 Nico Fuhrberg

3.1 Farbsegmentierung

Auf der Suche nach geeigneten Verfahren, den am Strand befindlichen Müll vom Rest des Bildes zu segmentieren, sollte auch ein Augenmerk auf die Verwendung von Farbschwellwerten gelegt werden. So würde die Segmentierung auf pixelweise Operationen zurückgeführt werden, welche durch ihre Einfachheit und dem entsprechend eher geringen Rechenaufwand betrachtenswert wären, wobei allerdings im Vorfeld bereits aufgrund dieser Tatsachen auch von tendenziell ungenauen Ergebnissen auszugehen war.

Die einfachste Idee war zunächst, einen Farbbereich für den Müll festzulegen, sodass innerhalb diesem befindliche Pixel als Müll betrachtet werden. Allerdings war diese Idee aufgrund der zu großen möglichen Varianz des auf den Bildern befindlichen Mülls ebenso schnell verworfen.

Eher fiel auf, dass Bereiche des Bildes, welche keinen Müll enthalten, eher durch einzelne Farben dominiert werden. So entwickelte sich der Ansatz, Farbbereiche für die Bereiche, in denen sich gerade kein Müll befindet, festzulegen und diese so herauszufiltern, dass am Ende idealerweise nur noch der Müll übrig ist. Naheliegend wäre so etwa die Definition eines Farbbereichs für den Sand sowie auch einen für Wasser und Himmel.

OpenCV stellt mit der *threshold*-Funktion eingebaute Mechanismen dar, um eine solche Farbsegmentierung zu realisieren. Hierbei lassen sich nun in verschiedenen Farbmodellen, testweise zunächst hartkodierte Farbschwellwerte definieren, sodass nach innerhalb dieser Räume befindlicher Farbwerte segmentiert

wird.

Allerdings wurde auch hierbei schon erkennbar, dass es etwa einige Überschneidungen der Farbbereiche des Sandes sowie des Mülls geben kann, sodass auch diese Segmentierung nicht eindeutig vorgenommen werden konnte. Weiter problematisch ist die Tatsache, dass diese Schwellwerte für jedes Eingabebild gefunden werden müssten. Zwar wären Normalisierungsoperationen, etwa über den Weißabgleich u.Ä. denkbar, allerdings ergibt sich hier durch Licht, die Aufnahmeverhältnisse der jeweiligen Kameras sowie auch die verschiedenen möglichen Untergrundarten des Strandes weiter eine zu große Varianz an möglichen Farbschwellwertsräumen, nach welchen segmentiert werden müssen könnte. Des Weiteren bliebe die Bestimmung der Anzahl der benötigten Farbschwellwertsräume problematisch, da etwa auch nicht als Müll zu erkennende Grünflächen, Felsen o.Ä. zu berücksichtigen wären. Besonders wäre hierbei weiter zu beachten, dass jeder weitere Farbschwellwertsraum potentiell weitere Schnittmengen mit dem Müll liefern mag und so die Genauigkeit der Segmentierung beeinträchtige.

Entsprechend haben wir die Verwendung einfacher Farbschwellwerte als einen unzureichenden Entwurf erachtet. Da uns zu diesem Zeitpunkt die Ansätze über Kantenerkennung bessere Ergebnisse, insbesondere bezüglich der Robustheit, lieferten, zogen wir diese den Farbschwellwertansätzen vor.

Als Ausblick wären noch weitere farbschwellwertsbasierte Verfahren denkbar, um die Segmentierung des Mülls vorzunehmen. Zwei von ihnen würden eine eher strikte Segmentierung vorsehen, um so tendenziell eher Kerne des Mülls auf dem Bild zu erhalten, welche mit relativ großer Sicherheit richtig segmentiert sind. Einerseits ließen sich durch diese Segmentierung nun Kerne für eine Wasserscheidentransformation auf dem Bild liefern. Andererseits wäre ein Verfahren denkbar, welches über die Dichte der segmentierten Pixel arbeitet - so würden nicht die einzelnen Pixel segmentiert werden, sondern die Bereiche, in denen viele in Frage kommende Pixel befindlich sind.

3.2 Otsu-Segmentierung

Otsus Binarisierung stellt ein nichtparametrisches Schwellwertverfahren zur Bildsegmentierung dar [1]. Sie funktioniert auf Graustufenbildern und sucht dabei den Schwellwert so, dass die Varianz innerhalb der zwei so resultierenden Klassen minimiert wird. Anzumerken ist, dass das Verfahren entsprechend vor allem auf Bildern gut funktioniert, deren zu segmentierenden Klassen sich im Helligkeitshistogramm möglichst deutlich in zwei durch ein möglichst signifikantes Tal getrennte Klassen aufteilen lassen.

In OpenCV ist ein Algorithmus, welcher die Otsu-Binarisierung realisiert, durch die Angabe des Flags *THRESH_OTSU* in der *threshold*-Funktion implementiert und konnte so benutzt werden.

Zwar ließ sich die Binarisierung nach Otsu zur Mülldetektion schnell auf-

grund der bereits angesprochenen großen Variabilität der möglichen Eingabesituationen als tendenziell ungeeignet einstufen; allerdings fiel in der Frage nach der Segmentierung des Hintergrunds auf, dass Himmel und Wasser eine Tendenz zu einer erhöhten Helligkeit im Vergleich zum Rest der Bilder hatten.

Tatsächlich ließen sich einige Versuche durchführen, in welchen das Segmentierungsverfahren relativ brauchbar nach dem Hintergrund segmentiert hat. Weiter ließ sich durch das Weichzeichnen des Eingabebildes als Vorverarbeitungsschritt eine Beeinflussung der Ergebnisse durch lokale Konstraste im Bild unterbinden, welche die Ergebnisse weiter verbesserte.

Allerdings gab es zeitgleich auch andere Testbilder, deren Otsu-Segmentierung keinerlei brauchbare Ergebnisse lieferte.

Letztendlich stellte sich die Robustheit des Verfahrens für unseren Anwendungsfall als eher unausreichend heraus - wie bereits zu erwarten, eignet es sich eher für kontrollierte Szenarien, in denen durch kontrollierte Beleuchtung und wohldefinierte mögliche Szenarien die Segmentierung einfach per Helligkeit vorgenommen werden kann. Durch die variablen möglichen Nutzereingaben sind solche Garantien in unserem Szenario hingegen nicht gegeben.

3.3 Labeln von Bildern zur Objekterkennung

Zum Training des neuronalen Netzes zur Identifikation von Störfaktoren bei der Müllerkennung habe ich ebenfalls das Labeln eines Satzes von Bildern übernommen. Der Fokus der Bilder lag auf dem Lernen der Erkennung von Strandliegen. Das Labeln wurde wie bei den anderen auch mit labelimg gemacht und insgesamt wurden 61 Bilder von mir gelabelt.

Literatur

[1] Nobuyuki Otsu. „A Threshold Selection Method from Gray-Level Histograms“. In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), S. 62–66. DOI: 10.1109/TSMC.1979.4310076.