



Praktikum Computer Vision

bei Jun.-Prof. Dr. Benjamin Risse

Smart Camera Operator

Vorgelegt von Gruppe 6 :

Tobias Johanning
421503
t.joha02@wwu.de
M.Sc. Informatik

Markus Konetzny
406462
m.kone06@wwu.de
M.Sc. Informatik

Tabea Preusser
428026
t.preu04@wwu.de
M.Sc. Informatik

Münster, den 05.03.2020

Inhaltsverzeichnis

1. Einleitung	1
2. Entwurfsentscheidungen	3
2.1. Setup	3
2.2. Toolselection	4
2.3. Maschine Learning	5
3. Anwendungsaufbau	6
3.1. Programmaufbau	6
3.2. Video-Pipeline	6
3.3. Zustandsautomat	8
4. Implementierung	10
4.1. Phase 1	10
4.2. Phase 2	10
4.3. Phase 3	12
5. Ergebnisauswertung	16
5.1. Planung	17
5.2. Umsetzung und Zielerreichung	17
5.3. Ausblick	18
6. Schlussfolgerung und Ausblick	19
Anhang A. Videos	20
A.1. Ergebnisse	20
A.2. Siegerehrung	20

1. Einleitung

Motivation

Im Rahmen eines Projektes des Unternehmens Rimondo sollte die Abdeckung der Videoaufnahme von Reitsportturniere maximiert werden. Dazu sollen neben den internationalen auch von möglichst vielen, im besten Falle sogar allen, nationalen Turnieren in NRW Videoaufnahmen erstellt werden können, was später auch auf ganz Deutschland erweitert werden soll. Neben Bereitstellung der Turnieraufnahmen in der Mediathek von Rimondo sind auch Live-Aufnahmen in Zukunft geplant. Alleine 2018 fanden in NRW über 600 nationale und 82 internationale Reitsportturniere¹ statt, von denen nur ein Bruchteil verfilmt wurde. Dabei stellt das grundlegende Problem die hohen Personalkosten der Kameraleute dar, die für stundenlange professionelle Aufnahmen benötigt werden. Während dies für internationale Turniere, wie das *Turnier der Sieger* in Münster, für Fernsehsender finanzierbar ist, ist dies für die kleineren, regionalen Turniere nicht rentabel. Da jedoch die Kundengruppe, welche aus den Turnierteilnehmern und deren Fans besteht, mit über 98.000 Mitgliedern in Reitsportvereinen in NRW als mögliche Kunden, großes Potenzial darstellt, wird eine Lösung benötigt. Aus diesem Grund wurden Projektgruppen von Informatikstudenten der WWU vor die Aufgabe gestellt die Kameraführung für Reitsportturniere zu automatisiert, indem ein *Smart Camera Operator* im Rahmen eines Praktikums erstellt wird. Dies sollte eine Software umfassen, welche in der Lage ist den Kameramann für Turnieraufnahmen im Pferdesport zu ersetzen.

Aufgabenstellung und Zielsetzung

Die Erstellung des *Smart Camera Operators* zur Videoaufnahme von Reitsportturnieren sollte in drei Phasen unterteilt erreicht werden, wobei die Wahl der Methoden und Verfahren freigestellt wurde.

Die erste Phase hatte die Erstellung von Trainingsdaten von Pferden und Reitern zum Ziel, womit die Vorarbeit für die weiteren Phasen geleistet wurde. Dazu hat Rimonod auf einer Online Platform ca. 1300 Frames aus Reitvideos zur Verfügung gestellt, auf denen mithilfe eines Labeling-Tools die Position der Reiter und Pferde eingetragen werden sollte. Jeder Teilnehmer des Praktikums hatte vom 24.10.2019 bis zum 14.11.2019 Zeit in den Bildern mindestens 200 Reiterpaare, genauer 400 Reiter oder Pferde, zu kennzeichnen. Diese Daten

¹Daten der Deutschen Reiterlichen Vereinigung E.V.

1. Einleitung

wurden anschließend von Rimondo ausgewertet und in Form einer Datenbank für die zweite Phase des Projektes weiterverwendet.

Die zweite Phase, vom 19.11.2019 bis 19.12.2019, umfasste das Training eines Detektors anhand der zuvor erstellten Datenbank. Dieser soll in der Lage sein anhand von Reitern und Pferden auch Reiterpaare zu erkennen. Als Grundlage zum Training des Detektors wurde mehrstündiges Videomaterial von Weitwinkel-Kameras von Reitern aus derselben Reithalle zur Verfügung gestellt. Es sollen erste Videos erstellt werden, die anhand der Region of Interest (Roi) aus den Frames der detektierten Reiterpaare erstellt werden, wobei noch kein großer Wert auf flüssiges Tracking gelegt werden musste. Zudem soll dieser Detektor genutzt werden, um weitere Bilddaten schnell zu labeln und ihn mit diesen Daten weiter zu trainieren.

In der dritten Phase soll der endgültige *Smart Camera Operator*, vom 20.12.2019 bis 6.3.2020 erstellt werden. Es wurde dazu weiteres Videomaterial zur Verfügung gestellt, welches mehrere Reithallen und einen Reitplatz abdeckte. Dabei soll die genauere Lokalisierung des Reiterpaars für eine robustere Implementierung des Trackers genutzt werden. Außerdem soll das Tracking der Rois flüssiger werden, wobei beispielsweise Filter sowie Entzerrung des Bildes eingesetzt werden können.

Unser persönliches Gruppenziel ist es einen robusten und vielseitigen Tracker mit Maschine Learning zu erstellen, der in der Lage ist einem einzelnen Reiterpaar zu folgen. Dabei liegt unser Fokus auf Problemen wie Verdeckung und Kreuzen von Reitern in unterschiedlichen Umgebungen, während wir uns weniger mit dem Perfomance Aspekt befassen wollen. Zudem wollen wir im Hinblick auf Benutzerfreundlichkeit das Labeling weiterer Daten und das Konvertieren der Videos mithilfe einer GUI umsetzen.

Aufbau der Arbeit

In Kapitel 2 gehen wir auf die grundlegenden Entwurfsentscheidungen des Projektes ein, wobei die verwendete Hard- und Software sowie der Maschine Learning Ansatz vorgestellt werden. Weiter wird in Kapitel 3 der Anwendungsaufbau anhand der Funktionsweise des Detektors sowie der Benutzeroberfläche beleuchtet. Die Implementierung der einzelnen Phasen des Trackers wird anschließend in Kapitel 4 im Detail erläutert. Die erreichten Ziele und möglichen Verbesserungen mit dafür wichtigen Schwerpunkten werden in Kapitel 5 diskutiert und zum Schluss fasst Kapitel 6 das erfolgte Projekt zusammen.

2. Entwurfsentscheidungen

2.1. Setup

Programmiersprache

Für die Wahl der Programmiersprache haben wir zunächst die Vorkenntnisse aller Gruppenmitglieder in verschiedenen erlernten Sprachen abgeschätzt und betrachtet welche Sprachen einen einfachen Einstieg in das Thema Maschine Learning erlauben. Deshalb fiel unsere Wahl trotz geringerer Geschwindigkeit im Vergleich zu C++ auf Python, da ohne viel Vorwissen schnell ein erster funktionierender Prototyp erstellt werden kann und wir zudem unserer Kenntnisse in dieser Sprache vertiefen können.

Für Python gibt es eine Vielzahl geeigneter Entwicklungsumgebungen wie Eclipse oder Visual Studio, wir haben uns jedoch für die kostenlose Variante von PyCharm entschieden, mit der alle Gruppenmitglieder vertraut waren.

Versionsverwaltung

Um Änderungen an Dateien und Quellcode zu erfassen und sinnvoll zu strukturieren, bietet sich aufgrund von Zusammenarbeit mehrerer Gruppenmitglieder der Einsatz einer Versionsverwaltung an. Die Wahl, mit welcher Versionsverwaltung das Projekt umgesetzt werden sollte, fiel auf Git als verteiltes System, welches wir in Form von Github nutzen. Dies hat den Grund, dass Git von der genutzten Entwicklungsumgebung PyCharm unterstützt wird und alle Mitglieder unserer Gruppe bereits Github durch vorherige Projekte vertraut waren, sodass relativ wenig Einarbeitungszeit erforderlich war.

Hardware

Mit dem Ansatz im Verlauf des Projektes Maschine Learning zu verwenden, wurde recht schnell deutlich wie wichtig eine gute Hardware Ausrüstung ist, um gute Performance beim Training des Models und bei der Detektion zu erreichen. Während die Umsetzung auch ausschließlich mit CPU möglich ist, besticht der Einsatz von GPU mit deutlicher Geschwindigkeit. Um diesen Vorteil zu nutzen, haben wir uns entschlossen für das Training mit dem kostenlosen Cloud-Service von Google Colaboratory in Verbindung mit Google Drive zu arbeiten, der ebenfalls kostenlos GPU Nutzung ermöglicht. Dabei stehen uns 25 GB Ram und je nach Zuweisung eine Tesla T4 GPU mit ca.8 GB oder

2. Entwurfsentscheidungen

eine Tesla K80 GPU mit ca. 12 GB zur Verfügung, was einen 25-fachen Geschwindigkeitsvorteil von GPU gegenüber CPU darstellt. Passend zur Wahl der Programmiersprache arbeitet Google Colab mit Jupyter Notebooks und hat bereits die meisten Bibliotheken installiert, wobei fehlende mit Kommandozeilen Befehlen noch hinzugefügt werden können. Die Einbindung von Github Projekten in Google Colab ist ebenfalls möglich, was wir für die Nutzung der Datenbank und Maschine Learning benötigten.

2.2. Toolselection

Für die Implementierung in Python wurden fachspezifische Bibliotheken eingesetzt, von denen wir die wichtigsten hier aufführen wollen

Pandas Zum lesen, sortieren und aufteilen der Datenbank haben wir die Bibliothek Pandas verwendet, die effizient in der Lage ist diese Daten zu manipulieren, zu filtern und mit fehlenden Werten umzugehen.

Opencv Das bekannteste Computer Vision Framework Opencv besticht mit seiner Vielzahl an Algorithmen, deren Schnelligkeit durch das C++ Backend und Benutzerfreundlichkeit durch den Python Wrapper besticht. Zum Einsatz kommt OpenCv sowohl bei der Extraktion von Frames, beim Filtern als auch durch die Zeichenfunktionen.

Numpy Für wissenschaftliche Berechnungen kam Numpy bei großen Bilddatenmengen, bei Vektorrechnungen und dem Versuch mit Ausgleichskurven zu arbeiten zum Einsatz.

Scipy Die SciPy Library bietet neben Numpy eine breite Masse an verschiedenen mathematischen Werkzeugen und numerischen Algorithmen. Besonders die vordefinierte Funktion für den eindimensionalen Gausfilter half beim Smoothing aller Boundingboxen.

tensorflow Das Framework TensorFlow bietet ein umfangreiches Grundgerüst für Machine-learning mit dem es möglich ist, auf Machine-learning basierende Programme einfach zu erstellen und zu verwenden.

keras Ursprünglich war Keras teil der Tensorflow Core API, jedoch wird es als eigenständige Bibliothek weitergeführt. Durch eine einheitliche Schnittstelle sorgt Keras für einen erheblich einfacheren und benutzerfreundlichen Umgang mit Tensorflow. Die Kombination aus Keras und Tensorflow sorgte für einen einfachen und schnellen Einstieg in die Welt der auf Maschine Learning basierenden Programme.

2. Entwurfsentscheidungen

PyQt Für die Gestaltung der grafischen Nutzeroberfläche haben wir uns entschieden PyQt5 einzusetzen, da uns diese Bibliothek bereits aus der Programmiersprache C++ in ähnlicher Form bekannt war. Zudem besticht diese im Vergleich zu Alternativen wie TKinter durch den Ansatz die GUI vom Backend zu trennen und die Möglichkeit ein modernes Design zu erstellen.

2.3. Maschine Learning

Da die Verwendung von Python im Bereich Maschine Learning populär ist, gibt es viel Frameworks, welche die Klassifikation oder Detektion von Objekten in Bildern unterstützen.

Wir haben uns aus folgenden Gründen für Mask-RCNN¹ entschieden:

- **zuverlässige Detektion:**

Auch unter schwierigen Bedingungen verspricht M-RCNN eine präzise, zuverlässige Detektion. Auch wenn die Geschwindigkeit nicht an Alternativen wie YOLO heranreicht, ist die Qualität der Klassifikation und Detektion immer sehr hoch.

- **leichter Einstieg:**

Auch wenn es subjektiv (noch) nicht so weit verbreitet ist wie YOLO fanden sich zwar wenige - dafür aber sehr gute Quellen zum Einstieg inklusive Tutorials und jupyter notebooks im Git repository der Entwickler.

- **Segmentierung:**

M-RCNN kann Segmentierungsmasken berechnen, mit denen wir die Qualität unseres Models gut beurteilen konnten. Auch wenn wir die Masken letztendlich noch nicht verwendet haben, erschien es uns als sehr vorteilhaft, diesen Trumpf in der Hinterhand zu haben.

- **Interesse:**

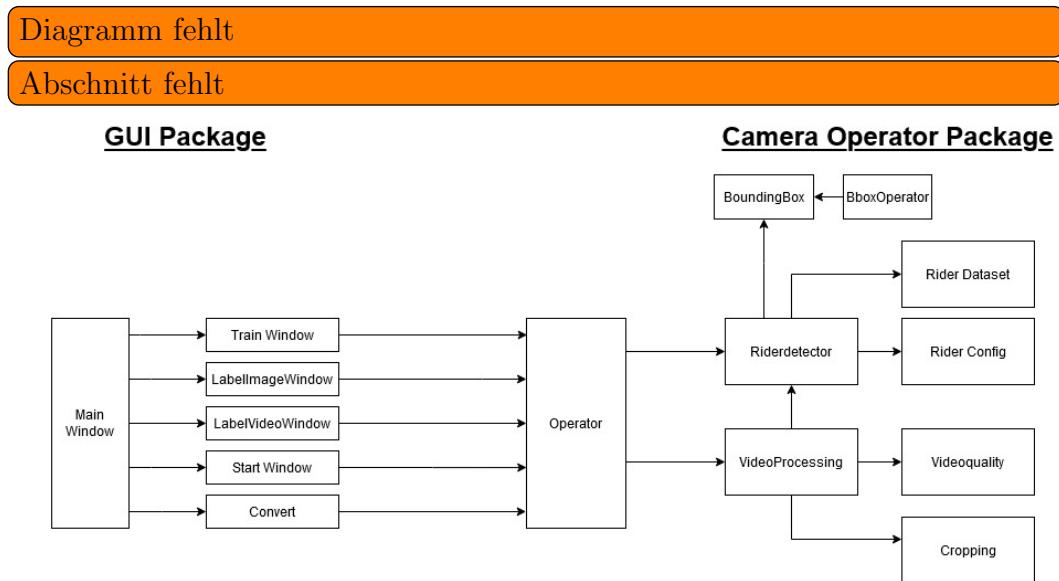
Beim Stöbern haben wir gesehen, dass M-RCNN ein relativ neues Verfahren ist. In dem Wissen, dass die meisten anderen Gruppen sich für YOLO entscheiden würden, sind wir nicht zuletzt auch deshalb den Weg mit M-RCNN gegangen, um zu herauszufinden, wie es sich im Vergleich zu YOLO verhält. Sinn und Ziel des Praktikums ist es ja auch gewesen, Neues auszuprobieren, zu experimentieren und aktuelle Verfahren spwie state-of-the-art-Techniken kennenzulernen.

¹Mask RCNN

3. Anwendungsaufbau

3.1. Programmaufbau

Klassendiagramm



Schnittstelle zwischen Gui und Backend: Operator Gui nutzt das Designpattern des Model View Controller: Controller MainWindow?, Model= Backend, View= Widgets

3.2. Video-Pipeline

Anhand der Ziele von den Phasen zwei haben wir die Vorgehensweise mit einer Video-Pipeline geplant, welche wir dann schrittweise mit sinnvoller Aufgaben-teilung umsetzen konnten. Die erste Version umfasst dabei lediglich die grundlegenden Funktionen, die für das sinnvolle bestimmen der Rois nötige sind. So wird hier pro Frame eines Videos für alle Detektionen ein gemeinsamer Bild-ausschnitt berechnet und diese zu einem Ausgabevideo zusammengefügt. Diese Video-Pipeline konnten wir dann in der dritten Phase um weiter Aspekte erweitern, wobei diesmal der Fokus auf der Qualität des Ausgabevideos lag. Die Detektion konnte auf Frames mit geringerer Auflösung schneller durchgeführt werden, weshalb die Größe der gefundenen Boxen anschließen angepasst werden musste. Anstatt dem vorherigen Ansatz alle Reiter und Pferde zu verfol-

3. Anwendungsaufbau

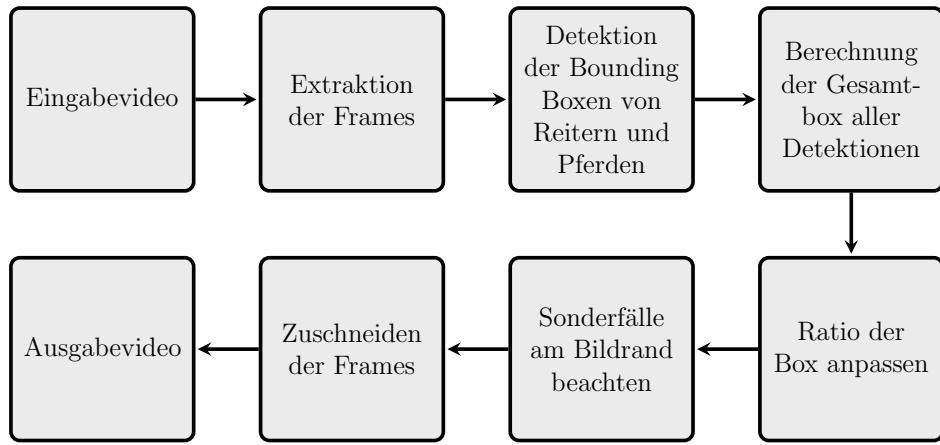


Abb. 1.: Video-Pipeline Phase 2

gen, wird nun ein einzelnes Reiterpaar ausgewählt. Die Qualität des Ausgabevideos wird durch weniger sprunghafte Unterschiede in der Größe der aufeinanderfolgen Rois verbessert, sodass der Bildfluss flüssiger wird. Zusätzlich wird nun auch die Audiodatei des Ursprungsvideos dem Ausgabevideo hinzugefügt.

3. Anwendungsaufbau

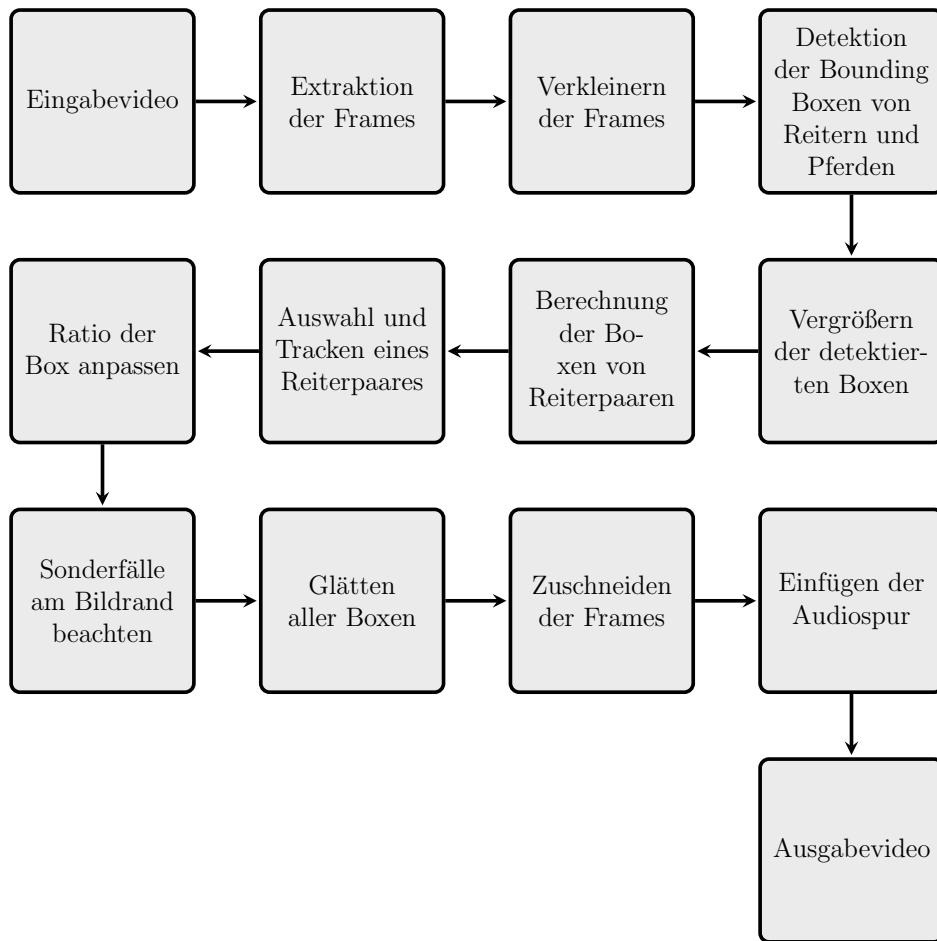


Abb. 2.: Video-Pipeline Phase 3

3.3. Zustandsautomat

Folgender Moore-Automat 3 soll zeigen, wie der Smart Camera Operator idealerweise funktionieren soll. Der Automat startet im *missing* Zustand. Die Transitionen *gefunden* und *nicht gefunden* beziehen sich auf ein Reiterpaar, das im Bild gefunden wurde - oder nicht. In der unteren Hälfte jedes Zustands befinden sich die Aktionen, die durchgeführt werden solange sich der Automat in dem Zustand befindet.

3. Anwendungsaufbau

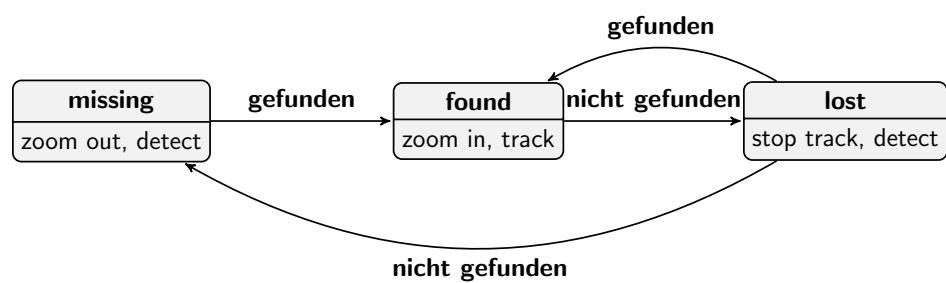


Abb. 3.: Zustandsautomat des Smart Camera Operators

4. Implementierung

4.1. Phase 1

Wie vorgesehen hat jedes Gruppenmitglied die Online-Platform von Rimondo genutzt um auf Bilder aus einer Reithalle 400 Reiter oder Pferde zu markieren. Dazu wurden Bounding Boxen um die erkannten Reiter bzw. Pferde eingezeichnet, die ein passendes Label zur Unterscheidung hinzufügten. Die gesamten erstellten Daten wurden am Ende der Phase von Rimondo in eine Datenbank umgewandelt mit welcher in der nächsten Phase der geplante Detektor entwickelt werden sollte.

4.2. Phase 2

Training des Model

Bevor wir mit dem Training des Detektors angefangen haben, mussten wir uns in die Thematik von Image Detection und in die Verwendung von Mask Rnn einlesen und an diese an einigen Beispielen testen.

Im ersten Schritt haben wir die bereitgestellten Daten aus der Datenbank von Phase 1 aufbereitet, um diese zum Training zu verwenden. Dazu haben wir die Datenbank aufgeteilt, sodass pro Frame eine csv Datei mit allen Labels im Format "image,label,x,y,width,height" existierte. Da jeder Frame mehrfach gelabelt wurde, haben wir mithilfe von Schwellwerten die zusammengehörigen Dopplungen bestimmt und davon den durchschnittlichen Wert abgespeichert. Anschließend haben wir die hohe Auflösung der Frames verringert, um die Trainingszeit zu verringern. Zuletzt haben wir die Daten in einer passenden Ordnerstruktur (4) von "accepted_images" und dem darin liegenden Ordner "annotations" in ein Github Projekt eingebunden, sodass das Training von Google Colab aus erfolgen kann.

Im zweiten Schritt haben wir die benötigten Klassen zum Training erstellt. Dazu wurde die Klasse RiderConfig als Unterklasse der Config Klasse von Mask Rnn erstellt, in der die Parameter individuell angepasst wurden. Wichtig war die Anzahl der zu detektierenden Klassen, die neben Reiter und Pferd auch den Hintergrund

Masken
Reiter
Pferd
Beispiel

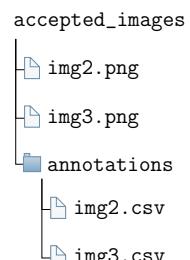


Abb. 4.: Ordnerstruktur
Training

4. Implementierung



Abb. 5.: Segmentierung von Reiter und Pferd nach dem Training der zweiten Phase

umfasst. Zudem wurde die Leistung der verfügbaren GPU angepasst sowie die Trainings- und Validierungsschritte pro Epoche. Weiter wurde die Klasse RiderDataset als Unterklasse der Dataset Klasse von Mask Rcnn erstellt. In dieser wurden die nötigen Funktionen zum Laden des Datensatzes in Trainings und Test Modus, zum Laden der Masken und extrahieren der Boxen anhand von Eckpunkten aus den cvs Dateien überschrieben.

Damit konnten wir im dritten Schritt mit dem Training des Models beginnen konnten, wozu wir aufbauend auf den MS COCO Gewichten Transfer Lernen genutzt haben. Für das Training haben wir den Datensatz in 70% Trainings-, 15 % Test- und 15% Validierungsdaten unterteilt. Die erste Version unseres Detektors haben wir mit 75 Epochen mit 500 Schritten trainiert, bis der Verlust pro Epoche sehr gering wurde. Die damit erreichte Leistung unseres Detektors war für die bereitgestellten Videos ausreichend zuverlässig(siehe 5), sodass wir danach mit diesem die ersten Videos erstellen konnten .

Zahlen fehlen noch

Zur vielfältigen Nutzung des erstellten Detektors, haben wir das Training mit weiteren gelabelten Bilddaten ermöglicht, um den Detektor auch in anderen Hallen zuverlässiger verwenden zu können. Dazu werden die Bilder eines Ordners nacheinander angezeigt, wobei die detektierten Pferde und Reiter mit farbigen Bounding Boxen eingezeichnet werden. Anhand dieser Boxen kann der Nutzer entscheiden, ob die Daten geeignet sind, sodass die akzeptierten Bilder und deren Boxen wie in der Ordnerstruktur von 4 wiederum abgespeichert werden, um den Detektor damit weiter zu trainieren.

Extraktion der Rois

Mithilfe von OpenCV werden alle Frames des Eingabevideos einzeln durchlaufen und der zuvor trainierte Detektor auf jedes einzeln angewandt. Dadurch haben wir pro Frame alle erkannten Reiter und Pferde mittels Bounding Boxen abgespeichert und anhand dieser weiter die Rois berechnet, da unser Fokus aus Zeitgründen noch nicht auf Erkennung von Reiterpaaren lag. Wir verfolgten hier den Ansatz zunächst alle Reiter und Pferde in den Rois abzubilden und erst in der dritten Phase zu selektieren um auf ein einzelnes Paar zu

4. Implementierung

fokussieren. Dieser Modus kann weiterhin gut für Aufwärmphasen der Reitturniere genutzt werden, bei denen mehrere Reiterpaare auf dem Turnierplatz sind. Die Umsetzung ist als austauschbarer Filter konzipiert, der eine Gesamt-Bounding-Box anhand der maximalen und minimalen Eckpunkte aller detektierten Bounding-Boxen bestimmt. Ausgehend von der Gesamt-Bounding-Box müssen für die Erstellung erster Videos noch nötige Sonderfälle beachtet werden. So wird der Zoom unter 480 Pixel verhindert, um eine gute Bildqualität zu ermöglichen. Die längere Seite der Bounding-Box ist ausschlaggebend für die Berechnung der passenden Ratio des Bildausschnittes, sodass dann die optimalen Eckpunkte berechnet werden können. Wichtig ist es noch die Ränder eines Frames zu berücksichtigen und die Ratio entsprechend zu korrigieren. Im letzten Schritt werden die Rois mithilfe der kalkulierten Gesamtbox aus jedem Frame ausgeschnitten. Nachdem alle Frames auf die gleiche Größe gebracht werden, was durch die gleiche Ratio aller Frames keine Verzerrung zur Folge hat, werden die Frames mit OpenCV in ein Ausgabevideo geschrieben.

Reiterpaarerkennung

Aus der Detektion erhalten wir eine Menge aus RoIs in Form von Boundings eines Mensch oder ein Pferd umranden. Diese Menge aus RoIs wird nun gefiltert, da wir nur noch Reiterpaare weiterbetrachten wollen. Ein Reiterpaar ist definiert als ein Mensch, der sehr nah an oder auf einem Pferd befindet.

Da die Bounding Boxen Rechtecke sind, können wir also berechnen, wie groß der Flächeninhalt der überlappenden Rechtecke von Reiter und Pferd ist. Ein Flächeninhalt gleich Null bedeutet dann, dass sie sich überhaupt nicht überlappen. Dadurch ist es nicht nur möglich zu berechnen, ob ein Pferd nah an einem Menschen ist - sondern auch wie nah. Es kann ein Threshold gewählt werden, der die Größe des errechneten Flächeninhalts prüft. Beispielsweise hätte ein Mensch *auf* einem Pferd einen größeren Flächeninhalt als ein Mensch *neben* einem Pferd, was je nach Wunsch also angepasst werden kann.

Wenn eine Mensch-RoI also eine Pferd-RoI überlappt, zeichnen wir um beide eine Reiterpaar-RoI. Damit haben wir einzelne Pferde, Reiter und Fehldetections vom Tracking ausgeschlossen.

4.3. Phase 3

Weiterentwicklung des Detektors

Mit Beginn der dritten Phase wurden weitere Reitvideos zur Verfügung gestellt, welche in verschiedenen Hallen sowie im Außenbereich gefilmt wurden. Der erste Test mit dem Detektor der zweiten Phase verlief nicht sonderlich zufriedenstellend, wodurch wir mehrere Probleme identifizieren konnten. Zum einen wurden nicht alle Reiter und Pferde, die weit entfernt von der Kamera

4. Implementierung



Abb. 6.: Segmentierung von Reiter und Pferd nach dem Training der zweiten Phase

waren, korrekt detektiert. Dieses Ergebnis war aufgrund des zuvor recht einseitigen Trainings nicht verwunderlich, da sich die Hintergründe, Reiter und Pferde unterschieden. Zum anderen wurden auch viele sitzende Personen als Reiter identifiziert und einige Gegenstände wie beispielsweise Pflanzen als fehlerhaft erkannt. Nachdem wir jedoch ungefähr 100 weitere Frames pro neuer Umgebung ausgewählt und damit den Detektor weiter trainiert hatten, wurden ein Großteil der Fehler beseitigt (siehe 6). Durch die Erweiterung des Detektors ist dieser nun auch in der Lage unterschiedlich gekleidete Reiter und Pferde mit abweichende Fellfarben zuverlässiger zu detektieren.

Benutzeroberfläche

Im Hinblick auf die Kundengruppe haben wir das Kennzeichnung weiterer Bilddaten und die Konvertierung eines Videos für autonome Kameraführung in eine GUI eingebunden. Die GUI wurde mit PyQt5 erstellt und das Standardaussehen durch Einbinden eines Stylesheets modifiziert. Mit dieser minimalen Benutzeroberfläche kann der Nutzer nun zunächst auswählen ob weitere Daten gelabelt werden sollen oder ein Video umgewandelt werden soll 7. Bei der Auswahl weiter Bilddaten können sowohl Ordner mit Bildern als auch Videos mithilfe extrahierter Frames verwendet werden. Dazu kann der Nutzer pro

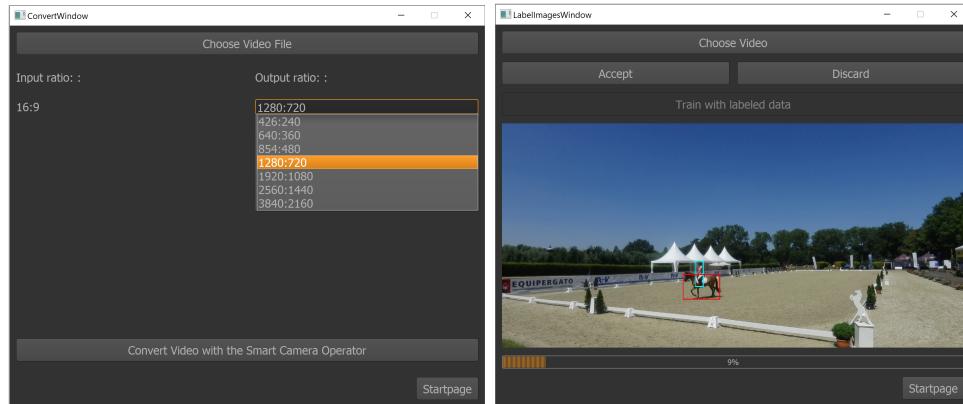


Abb. 7.: Benutzeroberfläche Konvertieren und Labeln eines Videos

4. Implementierung

Bild entscheiden ob die eingezeichnete Detektion gut genug ist diese akzeptieren oder ablehnen und im Anschluss den Detektor damit weiter trainieren. Für die Umwandlung eines Videos wird nach Auswahl der Datei und der Auflösung des Ausgabeformates ein Video unseres Smart Camera Operators erstellt.

Performance

Da der Fokus unserer Gruppe auf der Genauigkeit des Detektors liegt, haben wir nur einige grundlegende Verbesserungen der Performance vorgenommen, welche dem nicht entgegenwirken. Im ersten Schritt wir dazu anhand der Auflösung des Eingabevideos entschieden, ob die Frames vor der Detektion verkleinert werden sollten. Dies war bei den aufgenommenen Videos mit 4k Auflösung sinnvolle, da Mask Rcn auch mit einfacher HD Auflösungen ähnlich gute Ergebnisse liefert. Somit konnte der Zeitaufwand der Detektion von ca. 0.81 auf ca. 0.48 Sekunden pro Detektion verbessert werden. Anschließend müssen die detektierten Boxen wieder entsprechend hochskaliert werden um mit den Ursprungsbildern übereinzustimmen.

Als weitere Verbesserung haben wir den Einsatz von Ausgleichsgeraden und Ausgleichskurven getestet. Als Ansatz haben wir den Gedankengang verfolgt, nicht für jeden Frame Detektionen durchzuführen, sondern nur für jeden n-ten Frame, wodurch die Anwendung beschleunigt würde. In beiden Fällen haben wir mithilfe von "numpy.polyfit" die Berechnungen anhand der letzten 10 Detektionen erstellt. Die Ergebnisse für die Ausgleichsgerade stellten sich zwar performancemäßig als sehr positiv dar, jedoch ging die Videoqualität stark zurück. Das Ausgabevideo wies aufgrund der Ausgleichsgeraden sehr regelmäßige gerade Kameratranslationen auf, die die Gesamtqualität verminderten. Als nächstes nutzten wir statt der Geraden Ausgleichskurven, wobei sich bereits Kurven vom Grad 2 oder 3 ausreichend gut eigneten. Während dadurch die vorherigen Makel im Ausgabevideo gelöst wurden, stellte sich nur sehr geringfügiger bis gar kein Performacevorteil heraus. Aufgrund dessen wurden beide Versuche die Anzahl der Detektionen zu verringern verworfen und aus Zeitgründen nicht weiter verfolgt.

Videoqualität

Der Schwerpunkt der Phase richtet sich auf ein robusteres und flüssigeres Tracking, was wir mit besonderen Fokus auf Ausnahmesituationen und die Qualität der Kameraführung umgesetzt haben.

Tracking

Da der Detektor nicht komplett fehlerlos ist und demnach ein uninteressantes und entferntes Objekt für einen Zeitraum auch mal priorisieren kann, ist es wichtig diese Sprünge zu unterbinden. Ein Reiterpaar bewegt sich im Normalfall nicht innerhalb 2 Frames über den ganzen Platz. Der Abstand des

4. Implementierung

Reiterpaars zu seiner Position auf dem vorherigen Bild ist also relativ gering. Um diese Sprünge zu vermeiden prüfen wir also, ob die Position des getrackten Objektes des aktuellen Bildes stark von der Position des vorherigen getrackten Objektes abweicht. Damit der Wert der vorherigen Position als relativ akkurat gelten kann, nehmen wir, statt nur die Position des vorletzen Bildes, den Median der letzten 25 getrackten Boxen. Sollte nun die Distanz einen Schwellwert überschreiten, wird die neue Box ignoriert und für das aktuelle Bild die zuletzt valide Box verwendet. Damit der smarte Camera-Operator nicht auf einer Position hängen bleibt, ist mitunter ein Sicherungsmechanismus eingebaut, der ihn wieder zurücksetzt und neues Tracking erlaubt.

Sollte das Reiterpaar sich hinter einem Hindernis verstecken oder aus dem Bild hinauslaufen, wird die Kamera an der zuletzt gesichteten Stelle verbleiben bis das Paar wieder sichtbar geworden ist oder ein neues Paar detektiert wird.

Glättung

Um die Folge der RoIs zu glätten, damit sich nach dem Zuschniden ein sanftes Seherlebnis für den Zuschauer bietet, haben wir verschiedene Verfahren ausprobiert.

Das effektivste, effizienteste und gleichzeitig das einfachste Verfahren, das zum Einsatz kam, war ein eindimensionaler Gaußfilter. Wir haben eine Folge aus RoIs, dabei hat jede ROI 4 Attribute - die Koordinaten $(x_1, y_1), (x_2, y_2)$ des Rechtecks. Wir verwenden für alle Koordinaten x_1, y_1, x_2, y_2 aller RoIs einen 1d-Gaußfilter mit $\sigma = 5$, genauer nutzen wir also insgesamt 4 Mal einen Filter über die RoIs für jedes der 4 Attribute.

Ausführung in Verbesserungen?

Eine weitere Möglichkeit wäre es gewesen, die Richtungsänderungen zu erkennen/berechnen und dann einfach die RoIs linear von einer Richtungsänderung zur nächsten wandern zu lassen. Das wäre auch für die Echtzeit-Anwendung nötig gewesen, da hier die Kamera ja rotieren muss bei Richtungsänderung. Die Richtungsänderungen lassen sich mit *optical flow* berechnen. Wir haben aber auch eine für unseren Anwendungsfall effizientere Möglichkeit gefunden, die aber aus Platzgründen und weil sie sowieso nicht implementiert wurde hier nicht weiter thematisiert wird.

5. Ergebnisauswertung

Qualitative Auswertung:

- Diskussion Zielerreichung
- Untersuchung der Bedienung
- Empfehlung noch nicht implementierter Funktionen

Bugs:

- Prototyp
- fehlender Praxistest
- bisher nur Grundfunktionen

Verbesserungen:

- Performance für Live Übertragung
- Model Training erweitern
- Festlegung welchem Reiterpaar situationsbedingt gefolgt werden soll
- Bildentzerrung?

Zusammenfassung der Beobachtungen

- Einstufung der Fehler und ihre Wichtigkeit
- Fehlertoleranz
- Ursachenanalyse
- Verbesserungsvorschläge
- Interssenskonflikte(Performance und Zeitaufwand)
- Beurteilung der Ressourcen (Zeit, Vorwissen, erworbene Kenntnisse)
- Effektivität
- Effizienz
- Zufriedenstellung/Akzeptanz
- Aufgabenangemessenheit
- Erlernbarkeit
- Erwartungskonformität

5.1. Planung

Die vorgegebenen Phasen aus der Aufgabenstellung dieses Projektes stellten sich als sehr hilfreich für ein sinnvolles schrittweises Vorgehen und dessen Umsetzung heraus, besonders im Hinblick auf fehlendes Vorwissen aller Gruppenmitglieder in der Thematik Maschine Learning. Während die erste Phase problemlos ablief, gestaltete sich der praktische Einstieg in Phase zwei dagegen relativ langwierig bis wir einen ersten Prototypen unseres Detektors erstellen konnten. Als Folge dessen war es uns nicht vollständig möglich das Ziel eines Reiterpaardetektors bereits in Phase 2 fertigzustellen, da dieser an einigen Stellen noch Fehler aufwies, welche jedoch direkt zu Beginn der dritten Phase behoben wurden. Ebenfalls sollte in der zweiten Phase das Labelling weiterer Daten umgesetzt werden, wobei die entsprechend vereinfachte Benutzung dieses Ziels erst mithilfe der GUI in Phase drei hinzugefügt wurde. Diese zeitlichen Verschiebungen konnten jedoch in mit effizienter Aufgabenteilung gut bewältigt werden und stellten kein Hindernis für die letzte Phase dar. Durch die recht allgemeine Formulierung der Ziele in Phase drei, die einen Fokus auf robustes und flüssiges Tracking, haben wir die daraus abgeleiteten Gruppenziele in den Vordergrund gestellt. Obwohl einige Konzepte, die aus Effizienz- oder Performancegründen verworfen wurden, erprobt wurden, konnten die Projektziele eingehalten werden.

5.2. Umsetzung und Zielerreichung

Im Folgende betrachten wir die Umsetzung der einzelnen Ziele und die Einstufigung von bestehenden Fehler sowie deren Wichtigkeit

Detektor

Der erste Schritt für die autonome Kameraführung war die Erstellung eines Detektors für Reiter und Pferd sowie später auch für Reiterpaare. Diesen konnten wir erfolgreich mithilfe von Maschine Learning umsetzen, indem wir die erstellten Trainingsdaten der ersten Phase als Grundlage genutzt haben und den Detektor später um weitere Datensätze erweitert haben. Mit Wahl von Mask RCNN haben wir schon am Ende der zweiten Phase ausreichende Genauigkeit erreicht, dass wir problemlos zusätzliche Daten mit weiteren Videomaterial labeln konnten, obwohl sich die Umgebung und die Reiterpaare unterschieden. Die Schnelligkeit des Labelings konnten wir durch eine graphische Benutzeroberfläche gewährleisten, mit welcher ein Nutzer die möglichen Detektionen beurteilen kann. Trotz der erfolgten Erweiterung des Detektors ist es in jeder neuen Umgebung auch weiterhin nötig diesen weiter zu trainieren, da durchaus fehlerhafte Erkennungen auftreten können. Das Detektieren von Reiterpaaren konnten wir simpel mithilfe der Überdeckung von Reiter und Pferd lösen. Als

5. Ergebnisauswertung

Folge dessen konnten wir einzelne Reiter und Pferde ausschließen, was in neuen Umgebungen robusteres Detektieren zur Folge hat.

Tracker

Um eine flüssige und robuste Verfolgung des Reiterpaars zu ermöglichen haben wir viele grundlegende Problematiken betrachtet und schrittweise gelöst. Neben der Behandlung der Bildränder haben wir auch die Problematik von Verdeckung und dem fehlerhaften oder fehlenden detektieren von mehreren und einzelnen Reitern behandelt. Mit den errechneten Bounding Boxen konnten wir die jeweiligen RoIs der einzelnen Frames bestimmen und mit der Anwendung eines Gaußfilters auch eine angenehme Videoqualität ermöglichen.

Abschnitt unvollständig

5.3. Ausblick

Durch die zeitlichen Vorgaben konnten nicht alle Ideen und Ansätze ausprobiert werden, jedoch sehen wir in einigen Aspekten großes Potenzial zur Verbesserung und Erweiterung.

Der erstellte Detektor für Reiterpaare sollte sowohl für eine Vielzahl von Umgebungen mit anderen Reitern und Pferden trainiert werden, als auch für verschiedene Reitdisziplinen wie beispielsweise Springreiten oder Westernreiten. Um eine noch robustere Verfolgung eines Reiterpaars zu ermöglichen sind weitere Sonderfälle, wie das Kreuzen und längeres Verdecken von Reitern zu beachten. Ebenfalls sind Ausnahmesituationen wie Stürze von Interesse, da in diesem Fall Reiter und Pferd nicht mehr als Paar erkannt würden. Ein ausgiebiger Praxistest würde uns an dieser Stelle Aufschluss über nötige Erweiterungen und Problematiken geben.

Im Hinblick auf den Einsatz für Live Tracking müsste der bestehende Smart Camera Operator an einigen Stellen modifiziert werden, da im Rahmen des Projektes nur mit Aufzeichnungen gearbeitet wurde. Besonders müsste die Performance dafür deutlich erhöht werden, damit die Erkennung für ca. 25 fps erreicht werden kann. Davon ausgehend könnten dann auch Kamerabewegungen ermöglicht werden, die dem fokussierten Reiterpaar mittels Translationen und Rotationen folgt.

6. Schlussfolgerung und Ausblick

Ausblick:

- Einsatzmöglichkeiten für Turniere und Trainingsstunden

Abschnitt fehlt

A. Videos

Alle Videos befinden sich in einem Sciebo-Ordner , in dem die Ergebnis- und Siegerehrungsvideos getrennt zu finden sind.

A.1. Ergebnisse

Auflistung und Links

A.2. Siegerehrung

Auflistung und Links