Hochschule Wismar
University of Technology, Business and Design
Fakultät für Ingenieurwissenschaften

Forschungsseminar
Videosynopsis und Klassifikation

von: Patrice Matz

Betreuer: Prof. Dr. Litschke

Datum: Januar 2021

Abstrakt

In dieser Arbeit wird ein Verfahren zur Videosynopsis und -klassifizierung erläutert. Das Verfahren basiert auf der Extraktion von Änderungen zwischen Einzelbildern und der Aggregation dieser in Ebenen.

Es wird ein objektorientierter und komponentenweise parallelisierter Ansatz verfolgt und in der Programmiersprache Python 3 unter Zuhilfenahme von NumPy, OpenCV, Tensorflow und weiteren Frameworks implementiert.

0 Inhalt

1 Ziel

Es wird ein Algorithmus angestrebt mit welchem alle relevanten Bewegungen aus einem Video extrahiert werden. Stehen einzelne Kontouren zeitlich oder räumlich im Zusammenhang sind diese zu Ebenen zu aggregieren.

Diese Ebenen sind anschließend entsprechend des Inhaltes zu klassifizieren, sodass eine Suche nach bestimmten Ereignissen in einem Video folgen könnte. Die Suche selbst wird nicht betrachtet, stattdessen soll eine Liste von Objekten aus dem COCO Datensatz in Ebenen identifiziert und die Ebenen anschließend mit diesen annotiert werden.

2 Konzept

In diesem Kapitel werden die Architektur, Datenstrukturen, Klassifizierungsansätze und das zugrundeliegende Konzept erläutert.

Grundsätzlich ist es möglich zwei Bilder einer statischen Kamera miteinander zu vergleichen und somit Änderungen zu finden. Dies ist ein häufig eingesetztes Verfahren der Bild- und Videoverarbeitung. Werden aufeinander folgende Bilder eines Videos miteinander verglichen und die Unterschiede nacheinander abgespielt wirkt dies auf einen Zuschauer als würden alle Änderungen in einem Video abgespielt werden. Logisch besteht aber keine Verbindung zwischen den einzelnen Konturen, somit ist keine Filterung oder weitere Verarbeitung möglich.

Weitere Verarbeitungsschritte könnte die Feststellung von zeitlichen oder räumlichen Mustern, die Klassifizierung und Annotierung von Bewegungen oder ähnliches sein. Um dies zu ermöglichen muss eine logische Verbindung zwischen den einzelnen Konturen hergestellt werden. In Abb. 1 ist der Datenfluss zwischen den Komponenten vereinfacht dargestellt. Die logische Verbindung zwischen Konturen wird über die Aggregation in Ebenen oder Layer realisiert. Eine extrahierte Kontur wird mit den letzten n Konturen aller Layer verglichen. Die Layer dürfen hierbei nicht älter als x Frames sein. Gibt es eine Überschneidung zwischen der aktuellen Kontur und einem existierenden Layer wird die Kontur dem Layer hinzugefügt. Gibt es weitere Überschneidungen werden alle gefundenen Layer kombiniert. Dies kann als transitive Assoziation interpretiert werden.

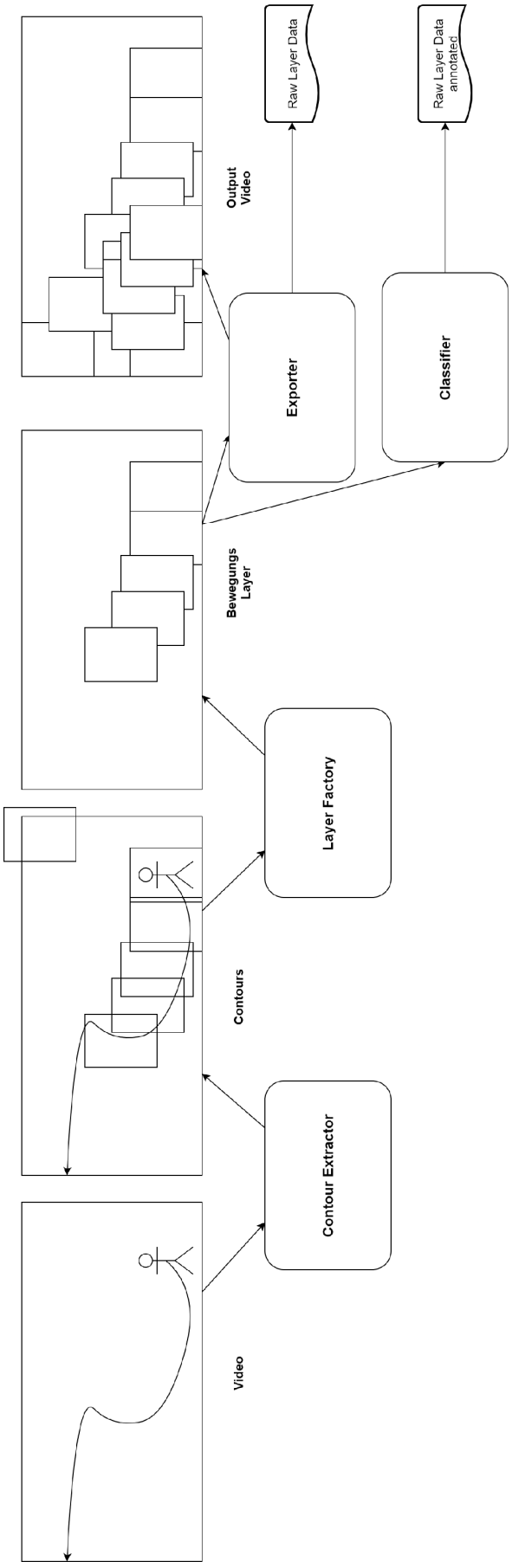


Abb. 1: Vereinfachter Datenfluss

2.1 Architektur

Dieses Unterkapitel dient der Herleitung der genutzten Architektur. Das Grundprinzip wird an einem vereinfachten Modell mit drei Komponenten erläutert. Anschließend wird eine erweiterte Architektur mit sechs parallelisierten Komponenten vorgestellt.

2.1.1 Drei Komponenten Modell

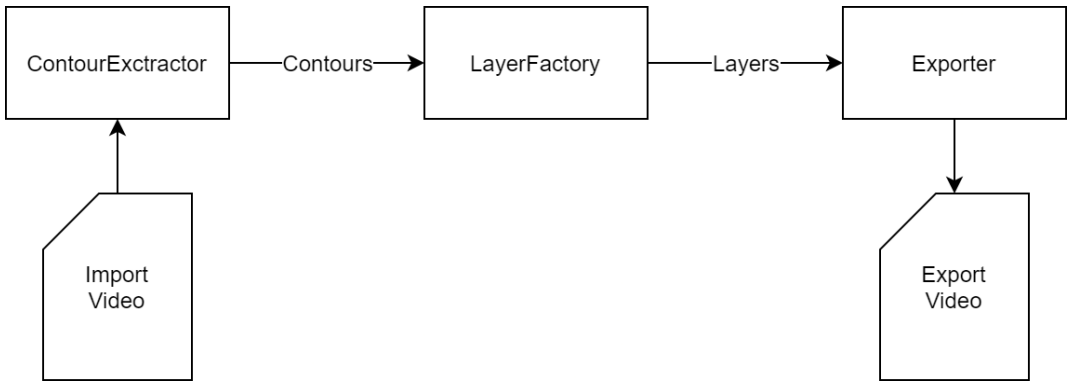


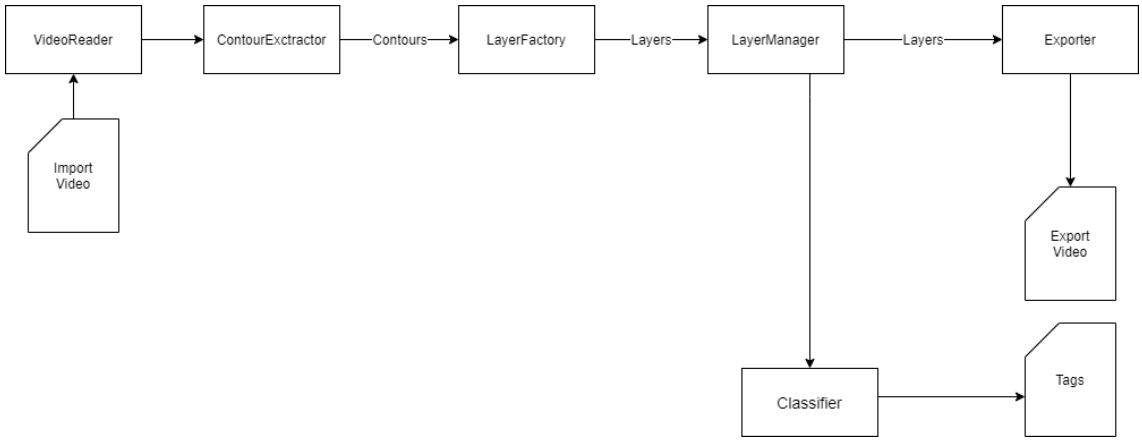
Abb. 2: Vereinfachte Architektur mit drei Komponenten

Wie in Abb. 2 dargestellt besteht die grundlegende Architektur aus drei Komponenten. Jede Komponente entspricht einer Klasse. Jede Klasse hat genau eine Aufgabe. Das „ContourExtractor“-Objekt extrahiert Konturen, diese dienen als Input einer Funktion des „LayerFactory“-Objektes, die Layer Factory wiederum produziert eine Menge von Layer Objekten, welche vom Exporter in ein Video gerendert werden. Auf die verwendeten Algorithmen wird in Kapitel 2.4 eingegangen.

Diese Architektur ist vollständig sequenziell. Somit ist die Verarbeitung eines einzigen Videos sehr zeitaufwendig, da zur Verfügung stehende Ressourcen nicht vollständig ausgelastet werden.

2.1.2 Sechs Komponenten Modell

Um die Ressourcenauslastung zu erhöhen und die Verwaltung der Layer zu verbessern werden zwei neue Komponenten eingefügt und bestehende Komponenten parallelisiert.



2. **Abb. 3: Architektur mit sechs Komponenten**

Die neuen Komponenten sind das „VideoReader“-Objekt und das „LayerManagement“-Objekt. Das LM-Objekt enthält eine Menge von Classifier-Objekten, welche jeweils ein Classifier-Interface Implementieren.

Das VR-Objekt spawnt einen neuen Thread, welcher das Video dekodiert und die dekodierten Frames in einem Buffer speichert. Auf den hierzu verwendeten Algorithmus wird in Kapitel 2.4.1 eingegangen.

Das LM-Objekt wurde eingefügt, um Funktionen zu aggregieren, die alle Layer betreffen, wie z.B. das Klassifizieren oder Rausfiltern von Layern mit geringer Qualität.

2.3 Datenstrukturen

Der verwendete Algorithmus benötigt zwei komplexe Datenstrukturen, diese werden in diesem Unterkapitel erläutert.

2.3.1 Konturen

Aus jedem Einzelbild können eine Menge von Konturen extrahiert werden, diese müssen mit einer Zuordnung des Bildes aus welchem sie entnommen wurden abgelegt werden. Hierzu wird ein Dictionary verwendet, welches die „Frame Number“ als Key und eine Liste von Konturen als Value speichert. Die Konturen werden nicht mit ihrem

Inhalt gespeichert, da dies große Mengen an Hauptspeicher benötigen würde. Stattdessen werden die X- und Y-Koordinaten, die Breite und Höhe gespeichert.

Daraus resultiert eine Datenstruktur welche wie folgt aussehen könnte.

```
Konturen = {

0: [(10, 10, 20, 50), (100, 100, 100, 50), (150, 150, 80, 35), …],

1: [(12, 9, 20, 50), (105, 104, 100, 50), (150, 160, 80, 35), …],

…

}
```

Listing 1: Datenstruktur extrahierter Konturen

2.3.2 Ebenen

Extrahierte Konturen sollen zu Ebenen zusammengefasst werden. Jede Ebene enthält eine Bewegung. Auch wenn mehrere Bewegungen zum selben Zeitpunkt stattfinden wird für jede dieser Bewegungen eine separate Ebene erstellt. Dies erleichtert das Bereinigen der Ebenen von Störungen, da Störungen somit meist in einer eigenen Ebene liegen und somit die Qualität der andere Ebenen nicht beeinflussen. Als hochwertig wird eine Ebene bezeichnet, wenn sie möglichst wenig Störungen und Unterbrechung beinhaltet und der Inhalt relevant ist. Bewegende Blätter z. B. sind irrelevant, eine Person oder ein Hund die nachts über ein Grundstück laufen sind hingegen relevant.

Ebenen sind komplexere Datenstrukturen mit spezifischen Funktionen, daher wird ein Layer Objekt erstellt, dieses kann in Listing 2 gefunden werden.

```
Class Layer:

      startFrame = None

      lastFrame = None

      length = None

      data = []

      bounds = []

      __init__()

      __len__()

      add()
```

Listing 2: Datenstruktur Layer

Eine Ebene / Layer ist eine Sammlung von zusammenhängenden Konturen, die Datenstruktur spiegelt dies wider. Das „Bounds“ Array ähnelt dem Konturen Dictionary, nur werden die Schlüssel durch den Index ersetzt. Zusätzlich werden nur überlappende Konturen gespeichert.

2.4 Algorithmen

Die zentralen Algorithmen dieses Forschungsseminares werden in diesem Unterkapitel erläutert. Im Kapitel 3 Implementierung wird auf die Details der Implementierung eingegangen.

2.4.1 Konturenextraktion

Die Konturenextraktion basiert auf dem Vergleich zweier Bilder und dem Speichern der Unterschiede. Dieses Verfahren ist genauer als nötig und Störungsanfällig bei Bewegungsrauschen und Helligkeitsänderungen. Daher wird eine Reihe weiterer Schritte eingefügt.

Algorithmus zur Extraktion von Konturen:

1. Berechnen des Durchschnitts der vorherigen X Bilder
2. Bilder herunterskalieren
3. Farbraum nach Graustufen konvertieren
4. Gaußschen Weichzeichner anwenden
5. Unterschied der Graustufen berechnen
6. Binarisierung
7. Dilatation
8. OpenCV::findContours()
9. Konturen nach Größe filtern
10. Speichern der Konturen in einer Datenstruktur

2.4.2 Ebenenaggregation

Die Ebenenaggregation ist einfach verglichen mit der Konturenextraktion. Es wird über alle Layer iteriert, ist das Layer älter als die Toleranz zulässt wird es übersprungen. Anschließend werden die Konturen der letzten fünf Frames des jeweiligen Layers mit der aktuellen Kontur verglichen, gibt es eine Überlappung wird die Kontur in das Layer eingefügt.

Kann eine Kontur zu mehr als einem Layer hinzugefügt werden, wird sie dem ersten Layer hinzugefügt und mit allen nachfolgenden Layern kombiniert. Somit ist eine wird eine transitive Assoziation eingefügt.

2.5 Klassifizierung

Die Klassifizierung von Bildern und Videos hat in den letzten Jahren immer mehr an Bedeutung gewonnen. Nicht zuletzt aufgrund von Fortschritten im Bereich selbstfahrender Fahrzeuge, Gesichtserkennung, Bildverarbeitung oder Robotik. Hierbei scheinen Machine Learning Ansätze am vielversprechendsten. Aus diesem Grund werden sie auch in diesem Projekt verwendet. Tensorflow ist ein beliebtes Framework, welches sowohl in der Forschung als auch in der Wirtschaft eingesetzt wird, daher wird es auch in diesem Projekt eingesetzt.

Auch wenn neuronale Netze in dieser Arbeit bevorzugt werden bedeutet dies nicht das andere Klassifizierungsansätze ungeeignet sind, besonders in Kombination mit anderen Ansätzen können die Annotierungen deutlich aussagekräftiger werden. Ein NN basierter Klassifikator mag ein Layer mit dem Tag „Truck“ annotieren, ein weiterer simplerer Klassifikator könnte die primäre Farbe des Objektes feststellen. Das Tagging von Ebenen kann mit beliebig vielen Klassifikatoren erfolgen, somit wäre anschließend eine deutliche genauere Filterung möglich.

2.5.1 Neuronale Netze

Neuronale Netze können, neben anderen Verfahren, zur Klassifizierung von z. B. Bildern eingesetzt werden. Neuronale Netze bestehen aus einer Eingabeschicht, einer versteckten Schicht und einer Ausgabeschicht, wobei die verstecke Schicht Null, bis N Schichten enthalten kann. Jede dieser Schichten enthält Nodes, welche mit Nodes folgender Schichten verbunden sind. Ein einfaches Neuronales Netz ist in Abb. 2 dargestellt.

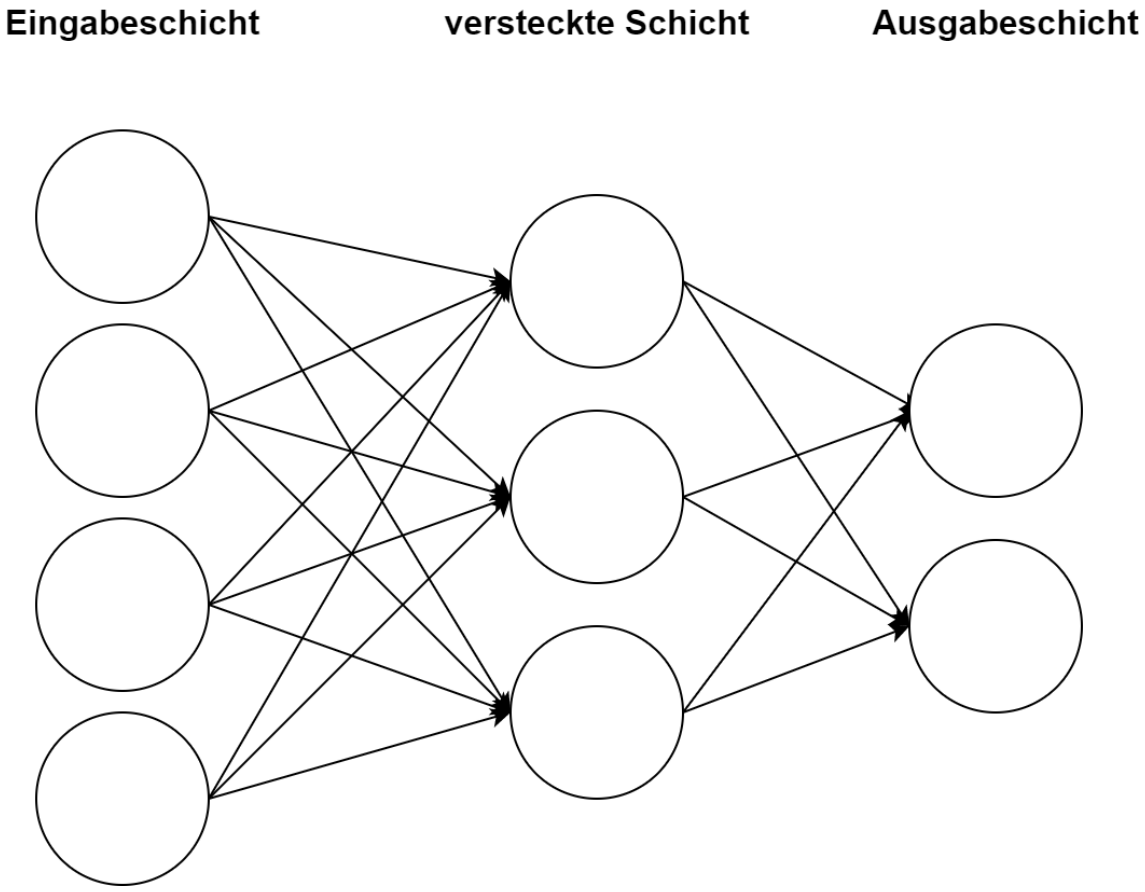


Abb. 4: Neuronales Netz mit einer versteckten Schicht

Das Verhalten eines Neuronalen Netzes und der Umgang mit Eingaben hängt von einer Vielzahl von Faktoren ab, unteranderem der Anzahl der versteckten Schichten, der Anzahl der Nodes pro Schicht, der Anzahl der Trainingszyklen, der Art, Anzahl und Qualität der Trainingsdaten und weiteren. Neuronal Netze sind ein komplexer Themenbereich, der in dieser Arbeit nicht tiefgründig untersucht werden kann.

Im Rahmen dieser Arbeit werden neuronale Netze als eine Implementierung des „Classifier-Interfaces“ betrachtet. Es wird ein vortrainiertes neuronales Netz verwendet, welches mit Bildern des COCO-Datensatzes (Common Objects in Context) trainiert wurde. Das bedeutet, dass Bildausschnitte als Eingabe eines Neuronalen Netzes dienen und die Ausgabe als Index für eine Klasse des COCO-Datensatzes. Da das neuronale Netz somit als Black-Box betrachtet wird ist ein tiefes Verständnis von neuronalen Netzen nicht für das Verständnis dieser Arbeit nötig.

3 Implementierung

In diesem Kapitel wird die Implementierung der bisher vorgestellten Konzepte erläutert. Der Aufbau dieses Kapitels ähnelt hierbei der Processing Pipeline. Es wird in Kapitel 3.2 mit der synthetischen Generierung eines Validierungsvideos begonnen. Anschließend wird erläutert wird die Konturenextraktion implementiert wurde, hierbei wird ebenfalls auf Erkenntnisse zu Farbräumen und Referenzbildern eingegangen. Anschließend folgt ein Unterkapitel zur Aggregation von Konturen zu Layern mithilfe einer Layer Factory und der Verwaltung und Klassifikation dieser mithilfe des Layer Managers und Classifiern. Abschließend wird auf das Objekt zum Exportieren der Videos und der erzeugten Daten eingegangen.

3.1 Vorgehen

Die Qualität der Videoverarbeitung und besonders die Videozusammenfassung kann schwer quantifizierbar sein, da die Ideallösung nicht immer bekannt ist. Daher wurde mit der Implementierung eines Tools zur synthetischen Generierung eines Videos mit definierten Ereignissen begonnen. Mithilfe der so erzeugten Videos konnten erzeugt Ergebnisse und somit auch das verwendete Verfahren validiert werden, da bereits bekannt ist wie viele Ereignisse mit welcher Dauer zu extrahieren sind.

Videos liegen aufgrund ihres hohen Speicherbedarfs meist in komprimierter Form vor. Sollen ein Video verarbeitet oder analysiert werden bedeutet dies daher oft, dass die Einzelbilder erst dekomprimiert bzw. decodiert werden müssen. Der Zeitaufwand für das Decodieren eins Videos, Frame für Frame ist nicht vernachlässigbar. Daher wird ein „VideoReader“-Objekt geschaffen, welches die Dekodierung der Videos in einen eigenen Thread auslagert und die dekodierten Frames in einem Buffer für die weitere Verarbeitung ablegt.

Somit sind die Voraussetzungen für eine effiziente und zielgerichtete Arbeit and der eigentlichen Video-Synopsis Komponente geschaffen.

3.2 Videosynthetisierung

Die Videosynthetisierung erfolgt mit dem Programm „gen.py“. Das Programm verfügt über kein CLI, da es in der Programmiersprache Python geschrieben ist können aber Einstellungen geändert werden, ohne dass eine erneute Kompilierung erfolgen muss.

In dem Programm können die Auflösung der Videos, die FPS, die Länges des Videos und die Anzahl der Ereignisse spezifiziert werden. Es ist nur ein Ereignis gleichzeitig sichtbar. Die Zeit zwischen Ereignissen ergibt sich als Länge des Videos / Anzahl der Ereignisse.

Ein Ereignis besteht aus dem Erscheinen eines Rechtecks mit zufälliger Dimensionierung, an einer zufälligen Position und der Bewegung dieses in eine Zufällige aber konstante Richtung, zusätzlich wird dem Rechteck eine zufällige Farbezugewiesen. So können die Rechtecke und die Farbtreue nach der Video-Synopsis besser validiert werden.

Die Implementierung kann in Listing 3 eingesehen werden.

```
def genVideo():
    writer = imageio.get_writer(outputPath, fps=fps)
    writer.append_data(np.zeros(shape=[1080, 1920, 3], dtype=np.uint8))
    writer.append_data(np.zeros(shape=[1080, 1920, 3], dtype=np.uint8))

    for i in range(numberOfEvents):
        objectWidth = (5 + random.randint(0, 5)) * xmax / 100
        objectHeight = (10 + random.randint(-5, 5)) * ymax / 100

        objectX = random.randint(0, xmax)
        objectY = random.randint(0, ymax)

        objectSpeedX = random.randint( 1 ,5 )
        objectSpeedY = random.randint( 1, 5 )
        color = getRandomColorString()

        for j in range(int(fps*length*60 / numberOfEvents)):
            objectX -= objectSpeedX
            objectY -= objectSpeedY

            objectShape = [
                (objectX, objectY),
                (objectX + objectWidth, objectY + objectHeight)
            ]
            img = Image.new("RGB", (xmax, ymax))
            img1 = ImageDraw.Draw(img)
            img1.rectangle(objectShape, fill = color)
            writer.append_data(np.array(img))

    writer.close()
```

Listing 3: Programm zur Generierung eines synthetischen Videos

3.3 VideoReader Objekt

Wie bereits in Kapitel 3.1 erwähnt dient der Videoreader dem Handling des Dekodierens eines Videos in einem separaten Thread. Der Videoreader ist hierbei wenig mehr als ein Wrapper um das OpenCV VideoCapture Objekt und einen Buffer in Form einer Queue. Die Konfiguration erfolgt mit einem „Configuration“-Objekt, welches alle einstellbaren Parameter der Applikation enthält.

Es existieren zwei Funktionen für das Befüllen des Buffers, die „readFrames“-Funktion und die „readFramesByList“-Funktion. Die erstgenannte Funktion liest alle Frames eines Videos in den Buffer, diese Funktion ist für die Konturenextraktion besonders relevant. Die zuletzt genannte Funktion hingegen ist für den Export des Videos relevant, da hier eine List von Frames aus welchen Bewegungen extrahiert wurden übergeben werden kann. Somit müssen nur relevante Frames dekodiert werden.

3.4 Konturenextraktion

3.4.1 Farbräume

3.4.2 Referenzbilder

3.5 Ebenenaggregation

3.5.1 Ebenenerzeugung

3.5.2 Ebenenmanagement

3.6 Classifier

3.6.1 Interface

3.6.2 OpenCV

3.6.3 Tensorflow

3.7 Export

4 Auswertung

4.1 Benchmarks

4.2 Weitere Beobachtungen

5 Zusammenfassung

6 Literaturverzeichnis

7 Abbildungsverzeichnis

8 Listingverzeichnis