Team 4 - Task 2

Milos Aleksic, Dominik Bepple, Robin Fink,
Ataullah Shinwari, Sebastian Sätzler, Bastian Tilk

Aufteilung

- Task 2a - Data Preprocessing (Milos Aleksic & Robin Fink)
- Task 2b - Classification of frames (Bastian Tilk & Dominik Bepple)
- Task 2c - Object localization (Sebastian Sätzler & Ataullah Shinwari)

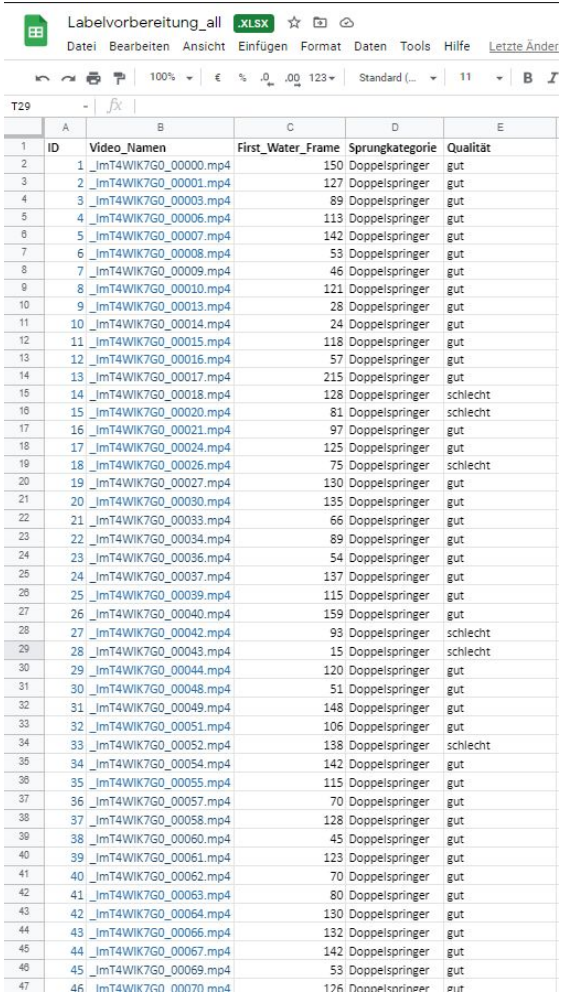
Task 2a

Steps

- Label Vorbereitung
- Videos in Frames zerlegen und labeln
- Filter mit Beispielen
- Datasets für Gruppe 2b erstellen
- Datasets für Gruppe 2c erstellen

Label Vorbereitung

- First_Water_Frame identifiziert
- Sprungkategorie bestimmt
- Qualität bestimmt
    - Bsp. für Schlecht
        - Falsche Sicht
        - Nahaufnahme
        - Video hängt
        - Springer berührt das Wasser nicht



Labelvorbereitung_all .XLSX

Datei Bearbeiten Ansicht Einfügen Format Daten Tools Hilfe Letzte Änder

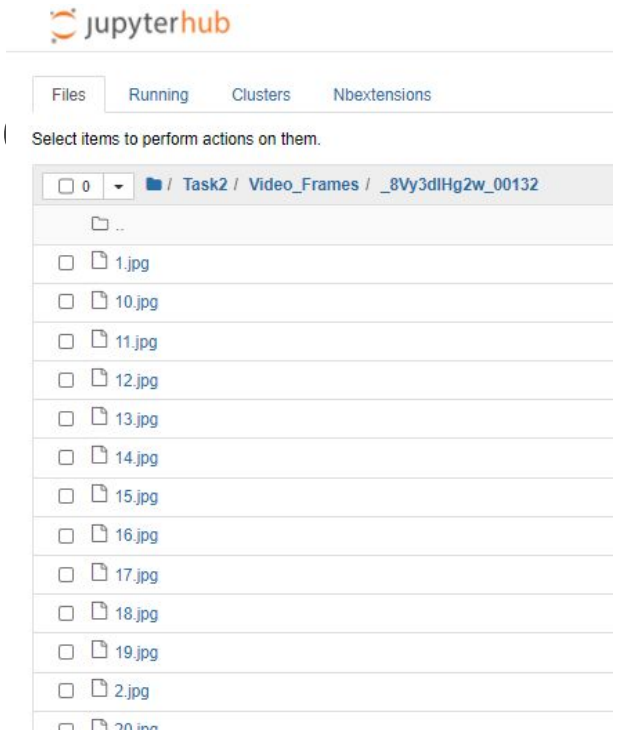
100% € % .0 .00 123 Standard (... 11

T29

	A	B	C	D	E
1	ID	Video_Namen	First_Water_Frame	Sprungkategorie	Qualität
2	1	_lmT4WIK7G0_00000.mp4	150	Doppelspringer	gut
3	2	_lmT4WIK7G0_00001.mp4	127	Doppelspringer	gut
4	3	_lmT4WIK7G0_00003.mp4	89	Doppelspringer	gut
5	4	_lmT4WIK7G0_00006.mp4	113	Doppelspringer	gut
6	5	_lmT4WIK7G0_00007.mp4	142	Doppelspringer	gut
7	6	_lmT4WIK7G0_00008.mp4	53	Doppelspringer	gut
8	7	_lmT4WIK7G0_00009.mp4	46	Doppelspringer	gut
9	8	_lmT4WIK7G0_00010.mp4	121	Doppelspringer	gut
10	9	_lmT4WIK7G0_00013.mp4	28	Doppelspringer	gut
11	10	_lmT4WIK7G0_00014.mp4	24	Doppelspringer	gut
12	11	_lmT4WIK7G0_00015.mp4	118	Doppelspringer	gut
13	12	_lmT4WIK7G0_00016.mp4	57	Doppelspringer	gut
14	13	_lmT4WIK7G0_00017.mp4	215	Doppelspringer	gut
15	14	_lmT4WIK7G0_00018.mp4	128	Doppelspringer	schlecht
16	15	_lmT4WIK7G0_00020.mp4	81	Doppelspringer	schlecht
17	16	_lmT4WIK7G0_00021.mp4	97	Doppelspringer	gut
18	17	_lmT4WIK7G0_00024.mp4	125	Doppelspringer	gut
19	18	_lmT4WIK7G0_00026.mp4	75	Doppelspringer	schlecht
20	19	_lmT4WIK7G0_00027.mp4	130	Doppelspringer	gut
21	20	_lmT4WIK7G0_00030.mp4	135	Doppelspringer	gut
22	21	_lmT4WIK7G0_00033.mp4	66	Doppelspringer	gut
23	22	_lmT4WIK7G0_00034.mp4	89	Doppelspringer	gut
24	23	_lmT4WIK7G0_00036.mp4	54	Doppelspringer	gut
25	24	_lmT4WIK7G0_00037.mp4	137	Doppelspringer	gut
26	25	_lmT4WIK7G0_00039.mp4	115	Doppelspringer	gut
27	26	_lmT4WIK7G0_00040.mp4	159	Doppelspringer	gut
28	27	_lmT4WIK7G0_00042.mp4	93	Doppelspringer	schlecht
29	28	_lmT4WIK7G0_00043.mp4	15	Doppelspringer	schlecht
30	29	_lmT4WIK7G0_00044.mp4	120	Doppelspringer	gut
31	30	_lmT4WIK7G0_00048.mp4	51	Doppelspringer	gut
32	31	_lmT4WIK7G0_00049.mp4	148	Doppelspringer	gut
33	32	_lmT4WIK7G0_00051.mp4	106	Doppelspringer	gut
34	33	_lmT4WIK7G0_00052.mp4	138	Doppelspringer	schlecht
35	34	_lmT4WIK7G0_00054.mp4	142	Doppelspringer	gut
36	35	_lmT4WIK7G0_00055.mp4	115	Doppelspringer	gut
37	36	_lmT4WIK7G0_00057.mp4	70	Doppelspringer	gut
38	37	_lmT4WIK7G0_00058.mp4	128	Doppelspringer	gut
39	38	_lmT4WIK7G0_00060.mp4	45	Doppelspringer	gut
40	39	_lmT4WIK7G0_00061.mp4	123	Doppelspringer	gut
41	40	_lmT4WIK7G0_00062.mp4	70	Doppelspringer	gut
42	41	_lmT4WIK7G0_00063.mp4	80	Doppelspringer	gut
43	42	_lmT4WIK7G0_00064.mp4	130	Doppelspringer	gut
44	43	_lmT4WIK7G0_00066.mp4	132	Doppelspringer	gut
45	44	_lmT4WIK7G0_00067.mp4	142	Doppelspringer	gut
46	45	_lmT4WIK7G0_00069.mp4	53	Doppelspringer	gut
47	46	_lmT4WIK7G0_00070.mp4	126	Doppelspringer	gut

Videos in Frames zerle

- Opencv Bibliothek verwendet
- Pro Video +100 - 230 Bilder
- 392 Videos ≈ 60.000 Bilder in Verzeichnisstruktur
- Herausforderung:
  - Speicherformat
  - Anzeige
  - Wie labeln?



Filterung des Datensatzes

- Excel in DataFrame umgewandelt
- Ermöglicht filtern des Datensatzes nach bleiben
- Basis zur Erstellung von Datensätzen

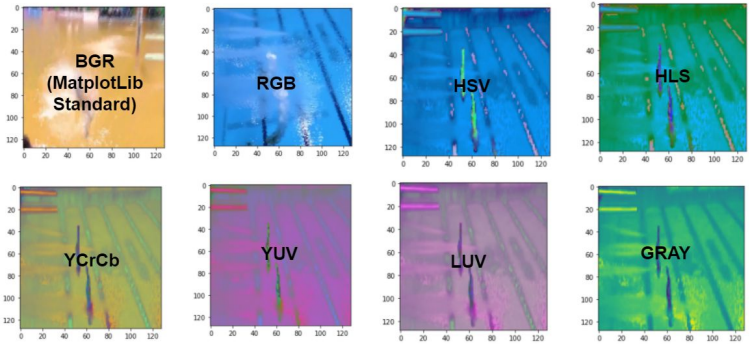
	ID	Video_Namen	First_Water_Frame	Sprungkategorie	Qualität	\
0	1	_lmT4WlK7G0_00000.mp4	150	Doppelspringer	gut	
1	2	_lmT4WlK7G0_00001.mp4	127	Doppelspringer	gut	
2	3	_lmT4WlK7G0_00003.mp4	89	Doppelspringer	gut	
3	4	_lmT4WlK7G0_00006.mp4	113	Doppelspringer	gut	
4	5	_lmT4WlK7G0_00007.mp4	142	Doppelspringer	gut	
5	6	_lmT4WlK7G0_00008.mp4	53	Doppelspringer	gut	
6	7	_lmT4WlK7G0_00009.mp4	46	Doppelspringer	gut	
7	8	_lmT4WlK7G0_00010.mp4	121	Doppelspringer	gut	
8	9	_lmT4WlK7G0_00013.mp4	28	Doppelspringer	gut	
9	10	_lmT4WlK7G0_00014.mp4	24	Doppelspringer	gut	
10	11	_lmT4WlK7G0_00015.mp4	118	Doppelspringer	gut	
11	12	_lmT4WlK7G0_00016.mp4	57	Doppelspringer	gut	
12	13	_lmT4WlK7G0_00017.mp4	215	Doppelspringer	gut	
13	14	_lmT4WlK7G0_00018.mp4	128	Doppelspringer	schlecht	
14	15	_lmT4WlK7G0_00020.mp4	81	Doppelspringer	schlecht	
15	16	_lmT4WlK7G0_00021.mp4	97	Doppelspringer	gut	
16	17	_lmT4WlK7G0_00024.mp4	125	Doppelspringer	gut	
17	18	_lmT4WlK7G0_00026.mp4	75	Doppelspringer	schlecht	

```
df = df[df.Sprungkategorie == "Einzelspringer"]
df = df[df.Qualität == "gut"]
```

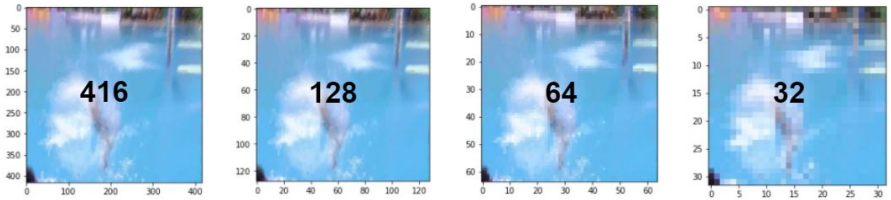
	ID	Video_Namen	First_Water_Frame	Sprungkategorie	Qualität	\
83	84	_8Vy3dlHg2w_00133.mp4	27	Einzelspringer	gut	
84	85	_8Vy3dlHg2w_00136.mp4	73	Einzelspringer	gut	
85	86	_8Vy3dlHg2w_00139.mp4	37	Einzelspringer	gut	
86	87	_8Vy3dlHg2w_00142.mp4	23	Einzelspringer	gut	
87	88	_8Vy3dlHg2w_00145.mp4	84	Einzelspringer	gut	
88	89	_8Vy3dlHg2w_00146.mp4	89	Einzelspringer	gut	
89	90	_8Vy3dlHg2w_00148.mp4	45	Einzelspringer	gut	
90	91	_8Vy3dlHg2w_00149.mp4	100	Einzelspringer	gut	
91	92	_8Vy3dlHg2w_00150.mp4	29	Einzelspringer	gut	
92	93	_8Vy3dlHg2w_00152.mp4	67	Einzelspringer	gut	
93	94	_8Vy3dlHg2w_00154.mp4	95	Einzelspringer	gut	
94	95	_8Vy3dlHg2w_00157.mp4	46	Einzelspringer	gut	
95	96	_8Vy3dlHg2w_00159.mp4	82	Einzelspringer	gut	
96	97	_8Vy3dlHg2w_00162.mp4	56	Einzelspringer	gut	
97	98	_8Vy3dlHg2w_00165.mp4	54	Einzelspringer	gut	
98	99	_8Vy3dlHg2w_00167.mp4	60	Einzelspringer	gut	
99	100	_8Vy3dlHg2w_00168.mp4	32	Einzelspringer	gut	
100	101	_8Vy3dlHg2w_00171.mp4	41	Einzelspringer	gut	

Beispiele

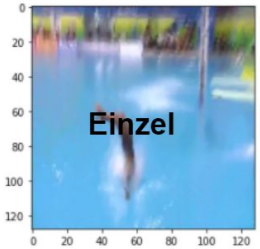
Filter

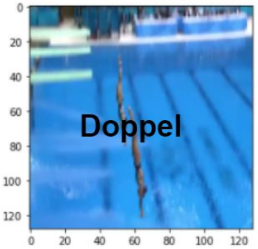


Pixel Varianten



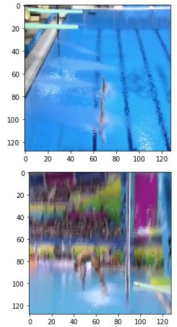
Einzelspringer und Doppelspringer



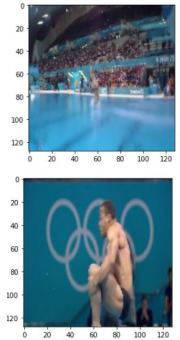


Gute und Schlechte

Gut

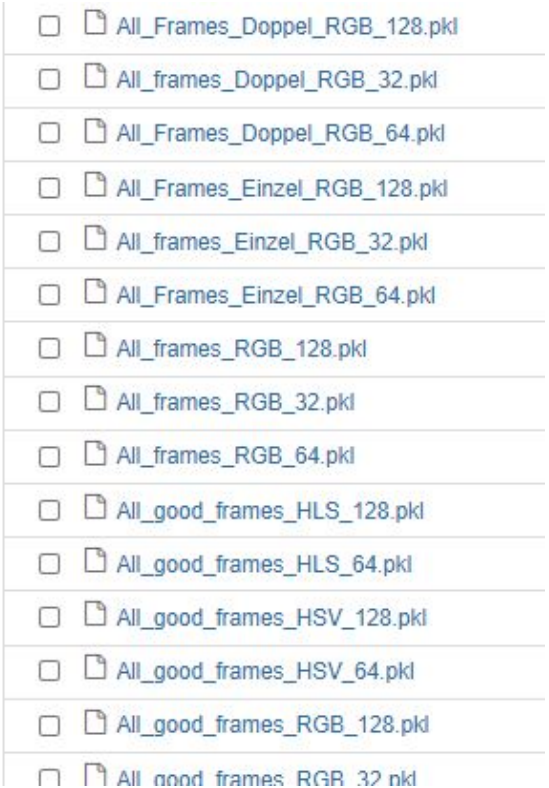


Schlecht

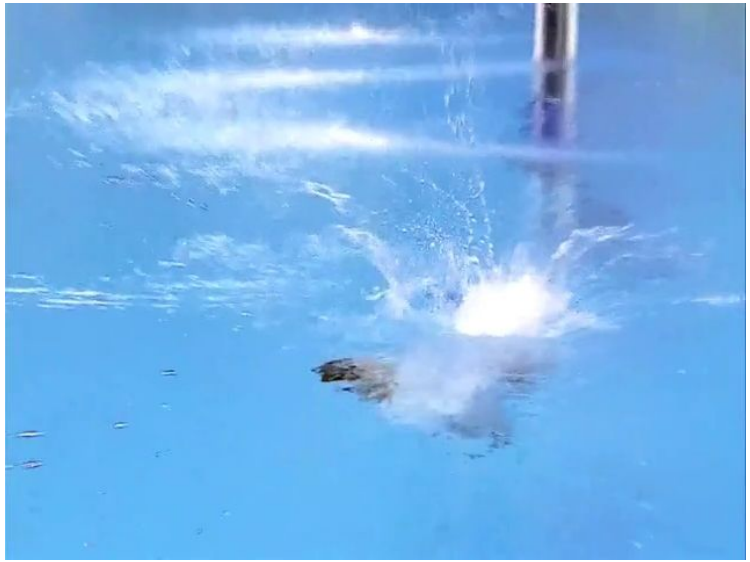


.pkl Datasets erstellt

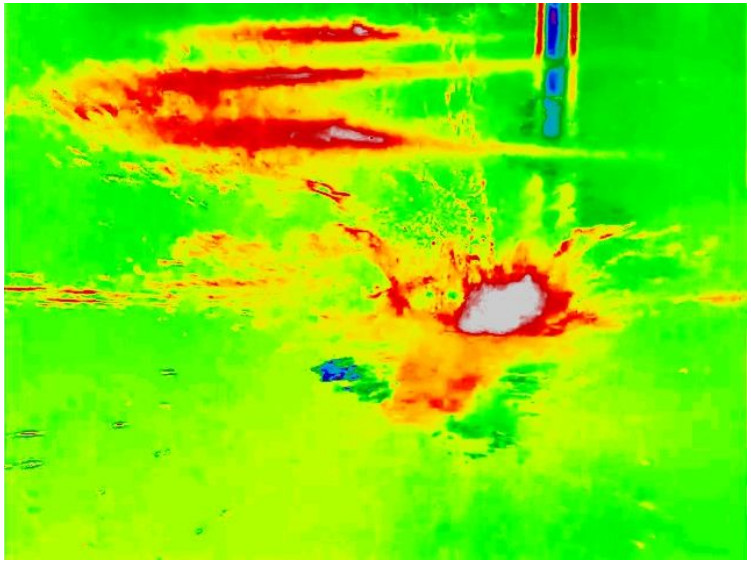
- Labeling:
  - Alle Bilder hinter dem First Water Frame werden in Liste Frames gespeichert. Label wird in separater Liste gespeichert
  - Listen werden Dictionary zugewiesen, Label des zugehörigen Frames über Index auslesbar
- Parameter bsp. Auflösung, Farbton und Inhalt(df) flexibel anpassbar
- Speicherung in Pickle = Python Struktur zur (de)-serialisierung von Objekten
- Hinterlegt sind die Pixelwerte im Pickle



Datensets für Gruppe 2c



Normal



Filter “nipy_spectral”

Task 2b

Verwendete Layers

Conv2D: ein 2D Convolution-Layer, der für jeden einzelnen Pixel eine Faltung mit n Filtern durchführt

MaxPooling2D: aus jedem 2 × 2 Quadrat aus Neuronen des Convolutional Layers wird nur die Aktivität des aktivsten (daher "Max") Neurons für die weiteren Berechnungsschritte beibehalten

Flatten: formt die dreidimensionale Ausgabe der Convolution in einen eindimensionalen Vektor um (ist wichtig für den Dense Layer)

Dense: Berechnet aus den Outputs des vorherigen Layer die angegebene Anzahl an Neuronen und erleichtert so die Klassifizierung

Modell - few layers

```
"""few_layers"""
model = models.Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(res, res, channel)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes))
model.summary()
```

Res = Die Bildauflösung

Channel = Anzahl der Farbwerte; RGB daher 3 Farbwerte

num_classes = Klassifikation in 2 Klassen; Air und Water

“few layers” hat nur ein Convolution Layer

Modell - standard

```
"""standard"""
model = models.Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(res, res, channel)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes))
model.summary()
```

1 zusätzlicher MaxPooling-Layer

Modelle - minimal

```
"""minimal"""
model = models.Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(res, res, channel)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(num_classes))
model.summary()

"""minimal_1"""
model = models.Sequential()
model.add(Conv2D(128, (3, 3), activation='relu', input_shape=(res, res, channel)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(num_classes))
model.summary()

"""minimal_2"""
model = models.Sequential()
model.add(Conv2D(32, (7, 7), activation='relu', input_shape=(res, res, channel)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(num_classes))
model.summary()
```

- Minimale Anzahl an Layern
- 2 verschiedene Anpassungen des Convolution-Layers
- Erhöhte Anzahl an Filtern
- Größere Faltungsmatrix

Auswahl eines Modells

Datensatz:

Alle guten und schlechten Bilder

Anzahl Bilder: 46864

Format: 64x64

Farbraum: RGB

Split: 80:20

Epochen: 10

Modellname	Validation Loss (Epoche 10)	Validation Accuracy (Epoche 10)	Test Accuracy (20%)	Training Time
Standard	0,0189	99.41%	99.41%	226,3 sec
Few Layers	0,0242	99,23%	99,20%	110,12 sec
Minimal	0,0286	99,09%	99,18%	91,2sec
Minimal 1 (mehr Filter)	0,0099	99,68%	98,99%	340,7 sec
Minimal 2 (größere Matrix)	0,0421	98,44%	98,61%	165,56 sec

Pixelgröße

Model: Few Layers
Split: 80:20
Epochen: 10
Alle guten und schlechten Bilder
Farbraum: RGB

Pixelgröße	**Validation Loss (Epoche 10)**	**Validation Accuracy (Epoche 10)**	**Test Accuracy (Few Layers Modell)**	**Training Time**
32x32	0,0414	98,69%	98,73%	31,04s
64x64	0,0242	99,23%	99,20%	110,12s
128x128 (alt)	0,0361	98,81%	98,84%	540,5s

Verschiedene Samples

Model: Few Layers
Split: 80:20
Epochen: 10
Farbraum: RGB
Format: 64x64

Art des Datensatzes	Anzahl Trainingsdaten	Anzahl Testdaten	Test Accuracy (Few Layers Modell)	Training Time
Alle Frames	37498	9375	99,20%	110,12s
Alle Frames (gute Frames)	30820	7705	98,62%	102s
Einzelspringer (gute Frames)	23772	5943	99,29%	72,59s
Synchronspringer (gute Frames)	7048	1771	98,98%	22,96s

Vergleich der Vorhersagen (Modelle mit guten Frames)

Label: Airframe (6 Bilder vor First_Waterframe)

Alle Frames-Modell : Airframe
Einzelspringer-Modell: Waterframe

Label: Airframe (2 Bilder vor First_Waterframe)

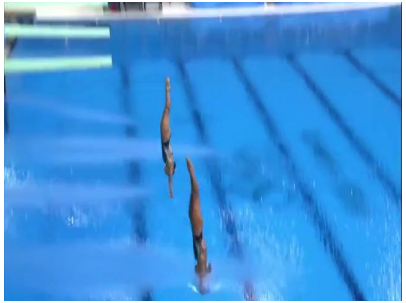
Alle Frames-Modell : Airframe
Einzelspringer-Modell: Waterframe

Farbskalierungen

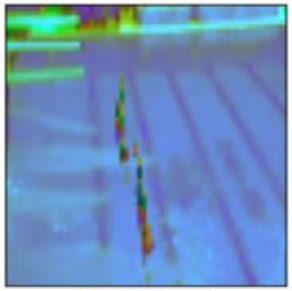
Art des Datensatzes	RGB	HLS	HSV	YCrCb
Alle Frames (gute Frames)	Val.Loss: 0,0469 Val.Acc.: 98,69% Test Acc.: 98,62%	Val.Loss: 0,000007 Val. Acc.: 100% Test Acc.: 100%	Val.Loss: 0,0268 Val.Acc.: 99,12% Test Acc.: 99,17%	Val.Loss: 0,0245 Val.Acc.: 99.09% Test Acc.: 99,19%
Einzelspringer (gute Frames)	Val.Loss: 0,0149 Val.Acc.: 99,41% Test Acc.: 99,29%	Val.Loss: 0,0168 Val.Acc.: 99,45% Test Acc.: 99,11%	Val.Loss: 0,015 Val. Acc.: 99,41% Test Acc.: 99,34%	Val.Loss: 0,0175 Val.Acc.: 99,33% Test Acc.: 98,91%

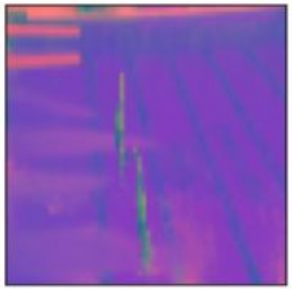
Analyse HLS-Filter

RGB 640x480

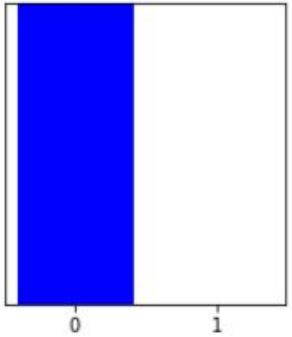


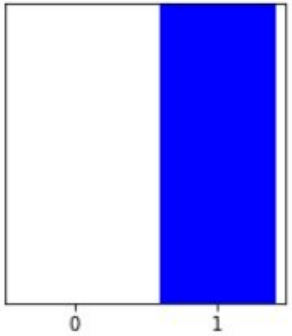
HLS 64x64





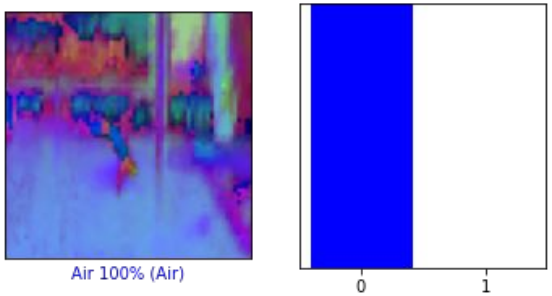
Klassifizierung



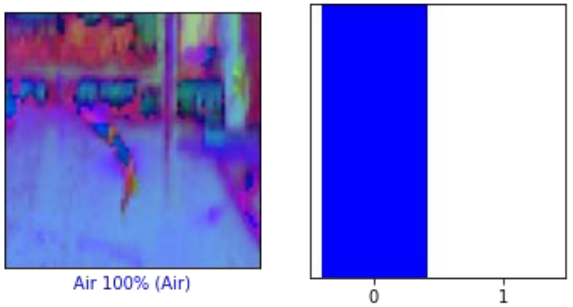


- Lilastich bei allen Waterframes
- Möglicherweise Fehler bei der Prozessierung

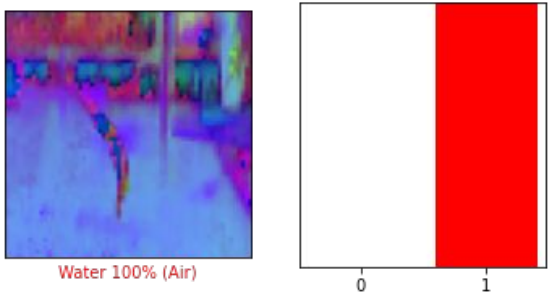
Einzelspringer (gute Frames) HSV Filter - Prediction



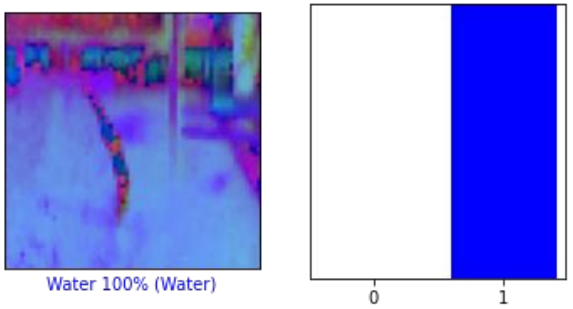
Frame 24



Frame 25

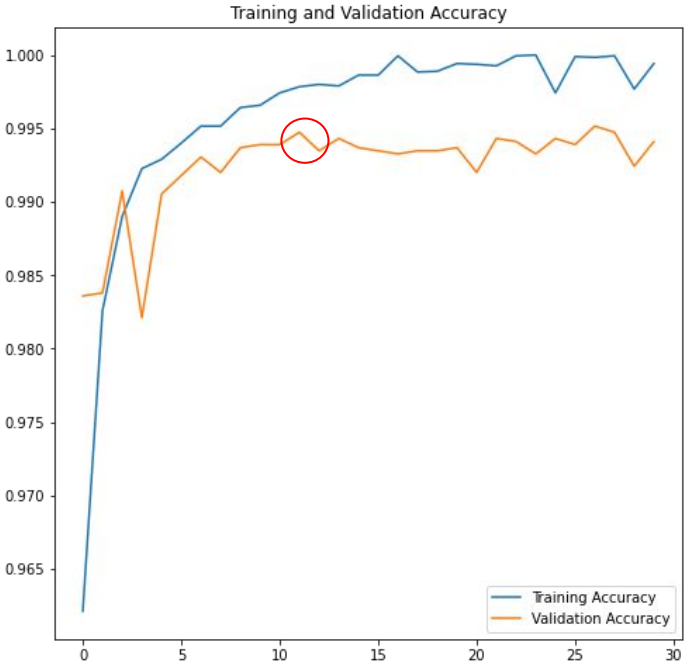


Frame 26



Frame 27 (erstes Waterframe)

Übersicht der Accuracies pro Epoche



Höchste Validation Accuracy bei Epoche 12: 99,47%
→ Modell auf 12 Epochen trainieren und erneut auf Testdaten anwenden

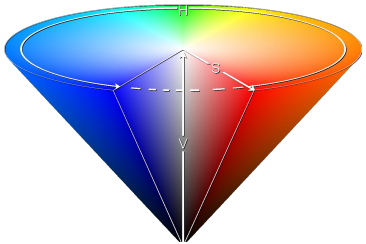
```
Epoch 12/12
595/595 [==============================] - 7s 12ms/step - loss: 0.0048 - accuracy: 0.9981 - val_loss: 0.0138 - val_accuracy: 0.
9945
INFO:tensorflow:Assets written to: /home/dl4/Task2/Modelle/BastianTest/Good_Frames_Einzel_HSV_64_model/assets
Modelbuilding read fertig
(5943, 64, 64, 3)
Good_Frames_Einzel_HSV_64_model
186/186 [==============================] - 1s 4ms/step - loss: 0.0199 - accuracy: 0.9924
Restored model, accuracy: 99.24%
```

Test-Accuracy mit 10 Epochen: 99,34%
Test-Accuracy mit 12 Epochen: 99,24%

Zusammenfassung

Top-Modell: Good_Frames_Einzel_HSV_64

Farbschema: HSV



Bildgröße: 64x64 Pixel

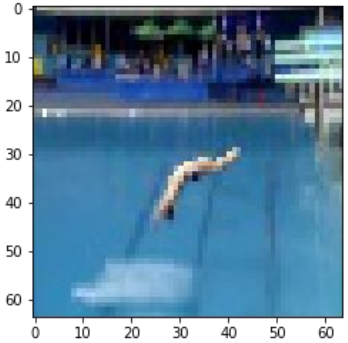
Epochen: 10

Layer-Modell: Few Layers

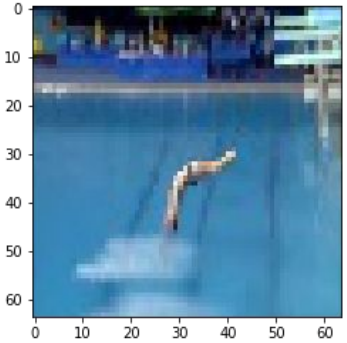
```
Epoch 10/10
595/595 [==============================] - 8s 13ms/step - loss: 0.0102 - accuracy: 0.9965 - val_loss: 0.0150 - val_accuracy: 0.
9941
INFO:tensorflow:Assets written to: /home/dl4/Task2/Modelle/BastianTest/Good_Frames_Einzel_HSV_64_model/assets
Modelbuilding read fertig
(5943, 64, 64, 3)
Good_Frames_Einzel_HSV_64_model
186/186 [==============================] - 1s 4ms/step - loss: 0.0184 - accuracy: 0.9934
Restored model, accuracy: 99.34%
```

Praxistest: Wird der erste Waterframe erkannt?

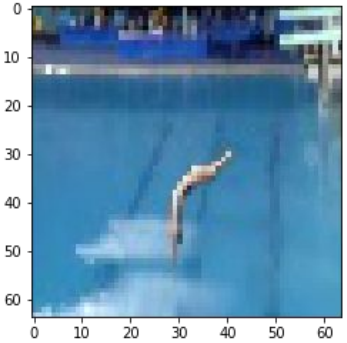
Ergebnis



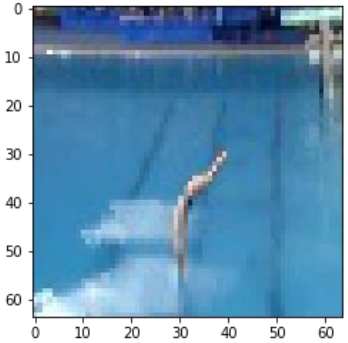
Air-Frame 84
(Air)



Air-Frame 85
(Water)



Air-Frame 86
(Water)



Water-Frame 87
(Water)

→ Modell erkennt den ersten Waterframe 2 Bilder zu früh

→ Alle Bilder davor und danach richtig erkannt

Task 2c

2c Analyse “Waterframes”/Wasserspritzeranalyse

2.0 Überblick über den Prozess

2.1 Objektlokalisierung via Object Detection

- Architektur
- Erstellung Trainingsdaten
- Parameter
- Evaluierung
  - Qualitative Analyse
  - IoU
  - Unser Test

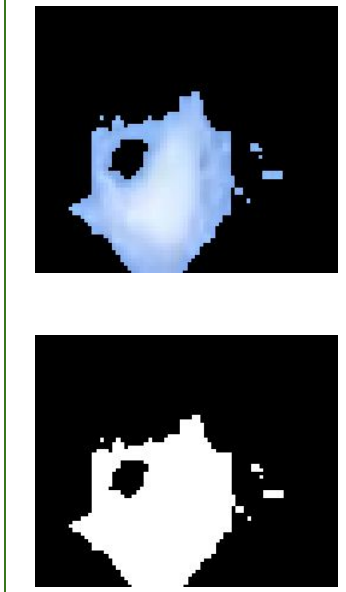
2.2 Volumenberechnung mit Masken

Überblick Prozess

Input

Object Detection

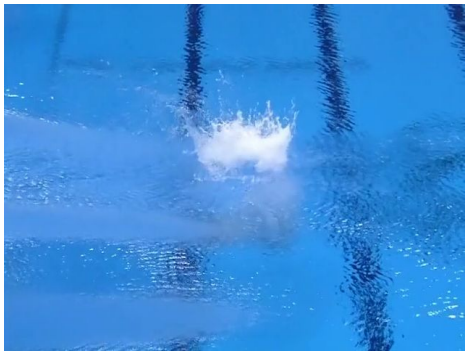
Mask Operation



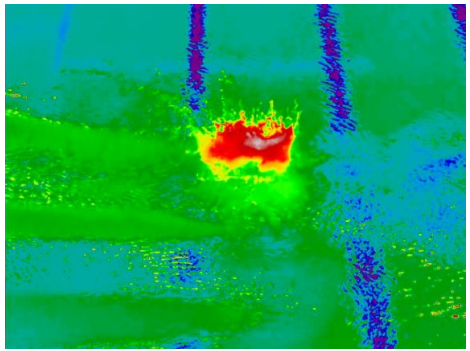
Results:	BB width: 77 BB height: 68		Splash Volume: 1200

Datensätze

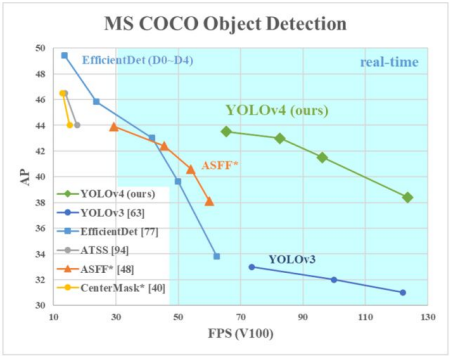
Normal



Filter “nipy_spectral”

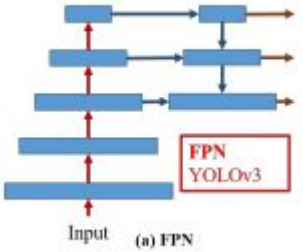


Object Detection Architektur



YOLOv4 (Bochkosvskiy et al.):

- Data augmentation als integraler Bestandteil des Modells
- Backbone: CSPDarknet-53 (Wang et al.)
  - Intuitiv als Feature Extractor zu verstehen
  - “Bereitet” die Inputs für die eigentliche Feature Detection vor
- Head: YOLOv3 (Redmond et al.)
  - Für die Feature Detection verantwortlich
- Feature Pyramid Network / Multi scale feature maps
  - Nicht-dominante Features gehen durch ganze Faltungsoperationen verloren:
  - ⇒ Detections werden in unterschiedlichen Phasen der hidden layers durchgeführt



Erstellung der Trainingsdaten

- 371 Frames von unterschiedlichsten Videos (auf denen Spritzer zu sehen sind)
- Spritzer manuell mit LabelImg umrahmt → Bounding Boxes (Label: “splash”)
- Data augmentation:
  - Horizontal Flip
  - Zoom
  - Blur
- ⇒ Insgesamt 1113 Trainingsbilder




Parameter

- Training über Darknet
  - Framework von Alexey Bochkovskiy (yolov3)
- Batchsize = 32
  - GPU Beschleunigung erlaubt höhere Batchsizes
- Steps = 2000
  - Konvention: [classes*2000]
- Filter = 16
  - Konvention: [(classes+5)*3]

Trainigszeit: 3:20

```
[convolutional]
size=1
stride=1
pad=1
filters=24
activation=linear
```

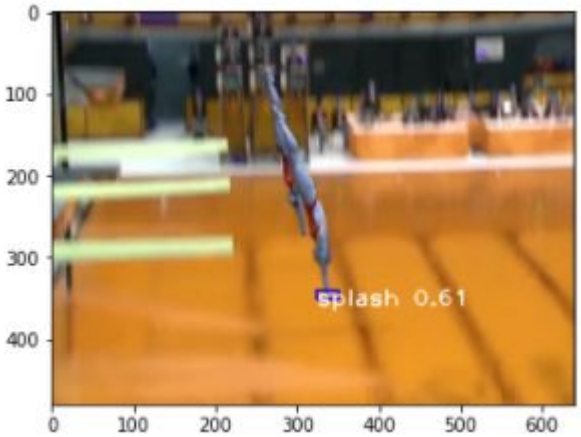
```
[yolo]
mask = 6,7,8
anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401
classes=3
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
scale_x_y = 1.05
iou_thresh=0.213
cls_normalizer=1.0
iou_normalizer=0.07
iou_loss=ciou
nms_kind=greedynms
beta_nms=0.6
max_delta=5
```

Evaluierung der Object Detection

- Qualitative Analyse der Object Detection
- Normale Farbewerte vs. Filter
- Intersection over Union (IoU)

Qualitative Analyse

Probleme der Object Detection



Fehlklassifizierungen mit niedriger Confidence (0.61)

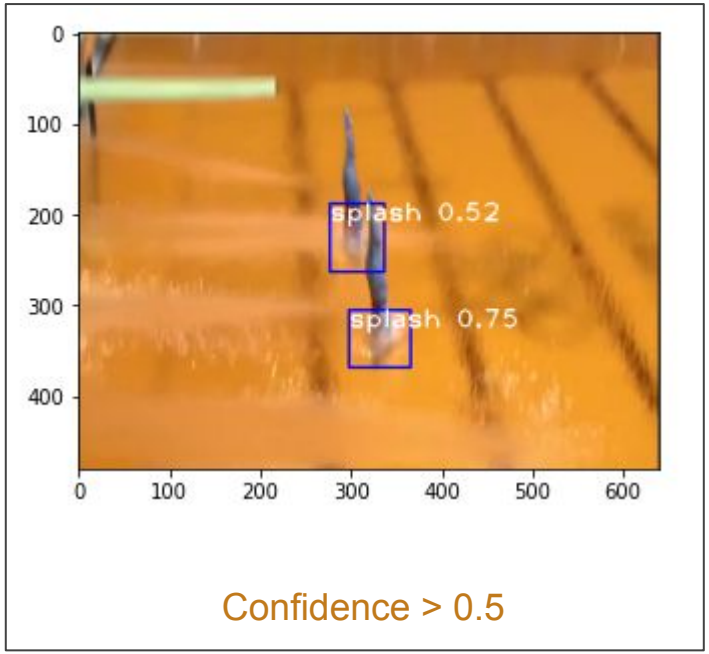
Lösung

Schwellwert der Confidence höher setzen:
z.B: auf 0.7

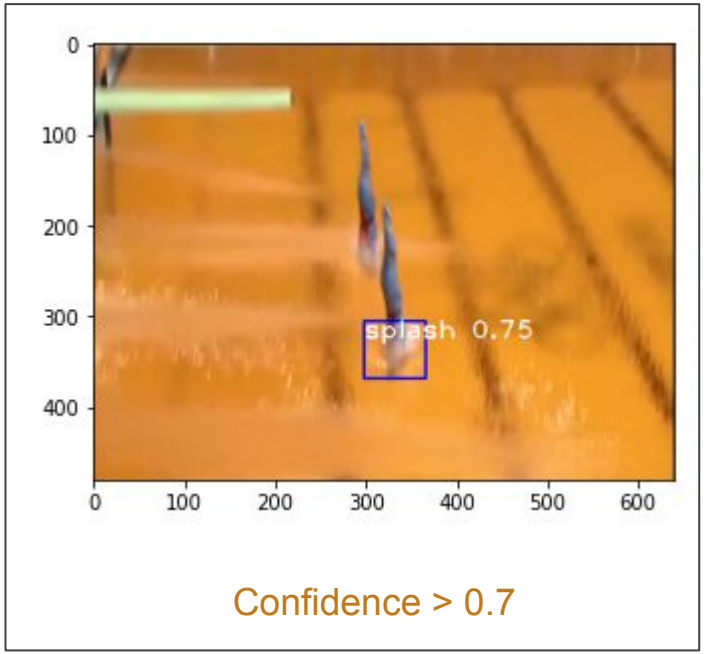
Qualitative Analyse

Konsequenzen eines höheren Schwellwertes

Schwellwert:

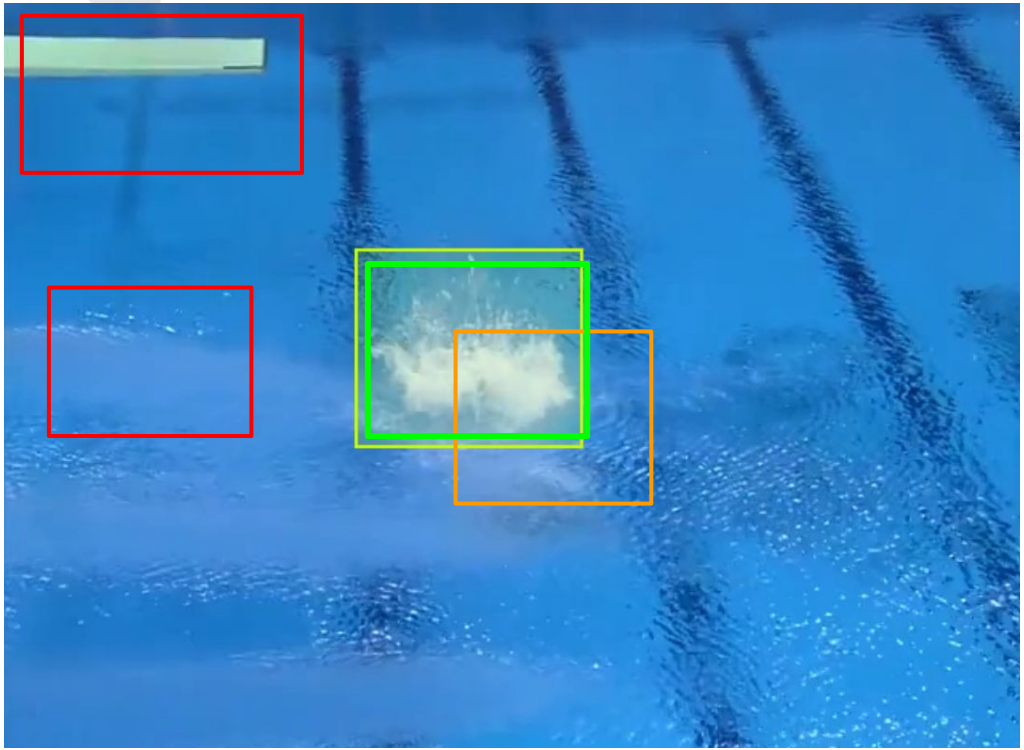


Confidence > 0.5



Confidence > 0.7

Intersection over Union



Intersection over Union (IoU)

- Prominente Methode um Object Detection Performance zu messen

Unser Test

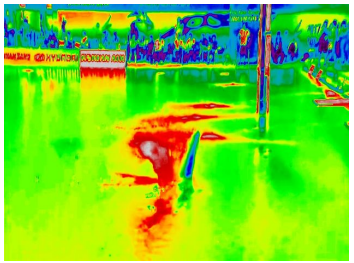
- 100 Test Bilder mit Wasserspritzer
- Test, ob Wasserspritzer erkannt
- Auch Bilder genommen, die schwer zu erkennen sind

Normal

```
Watersplash erkannt:    48
Watersplash nicht erkannt:      52
```

Filter

```
Watersplash erkannt:    18
Watersplash nicht erkannt:      82
```



Fazit

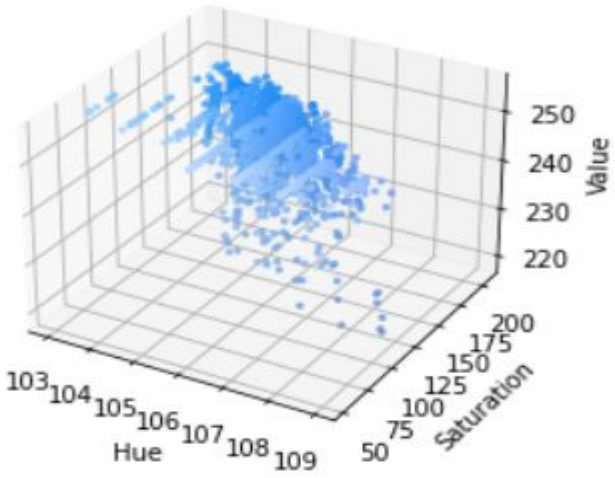
- Klassifizierung nicht optimal
  - Bedingt durch wenige Trainingsdaten
    - OD profitiert von sehr vielen Daten (Deshalb MS COCO Dataset zum Beispiel)
- Object Detection tat sich mit unbearbeiteten Bildern leichter

Changing Colorspace

- OpenCV liest standardmäßig Bilder im BGR-Format → mit `cvtColor` von BGR2RGB

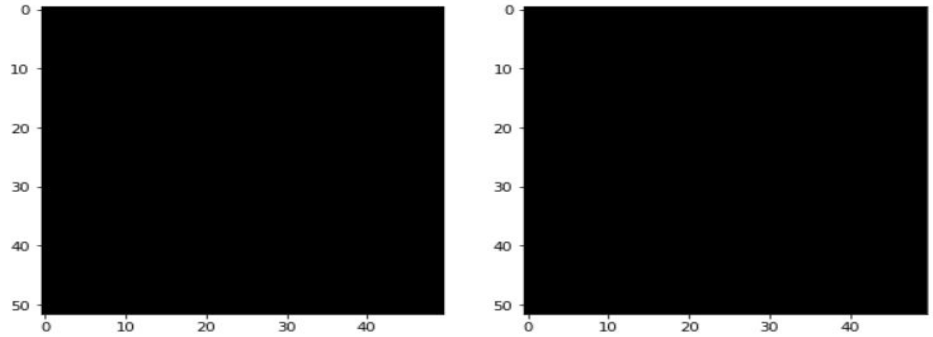
Erstellung von Maske

- Visualisierung von Frame im HSV-Farbraum
- HSV steht für Farbton, Sättigung und Wert (oder Helligkeit)
- Im HSV-Raum ist das Splash (Weiße) viel stärker lok
- Auswählen eines Bereichs (obere und untere Grenze)
- von Diagramm ablesen, oder mit einem Tool geht es
- Mit `cv2.inRange()` das Frame mit einem Schwellenwer

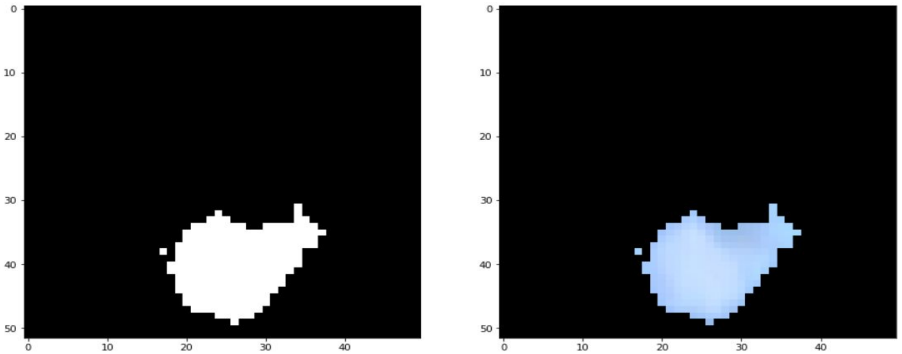


- Ausgabe davon eine Binäre Maske (wobei Werte von 1en Werte innerhalb des Bereichs und Nullen Werte außerhalb des Bereichs anzeigen)
- Um zu sehen, was das genau bewirkt hat, sehen wir uns sowohl die Maske als auch das Originalbild mit der Maske oben an

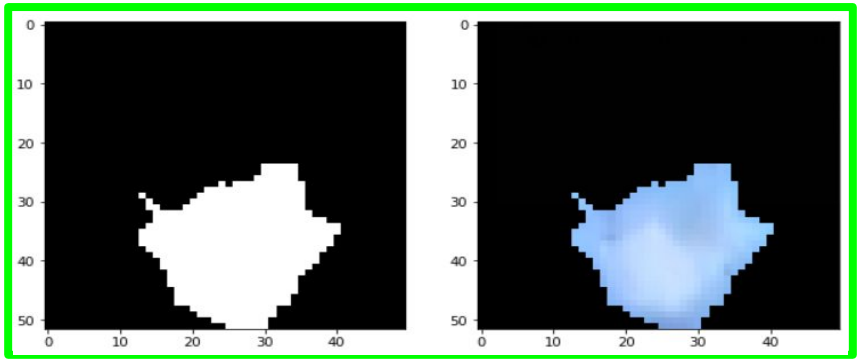
Grenzwerte:(172,20,255)



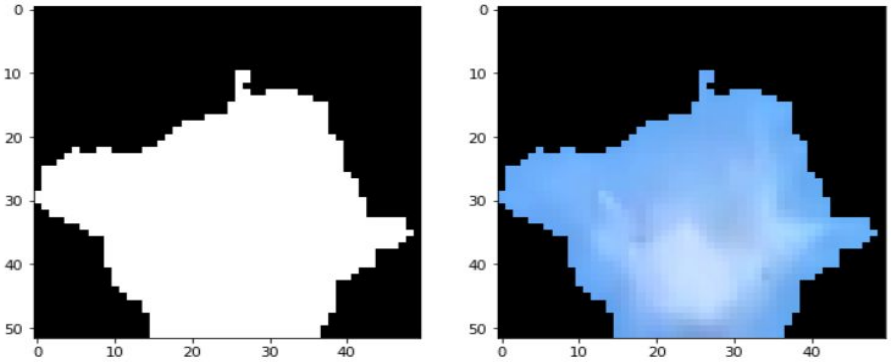
Grenzwerte: (172,90,255)



Grenzwerte:(172,115,255)

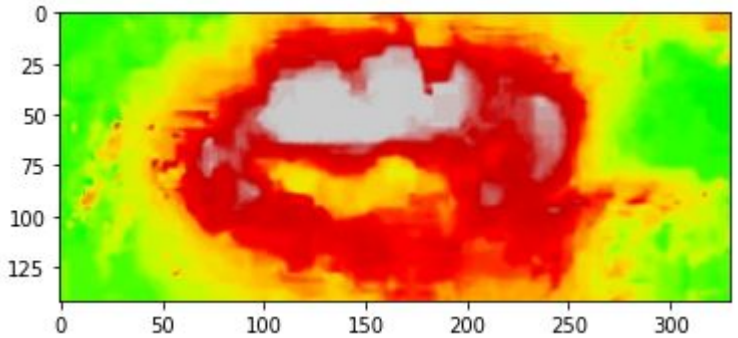


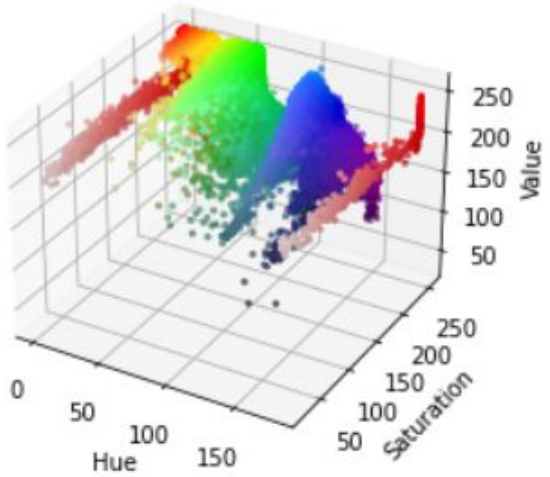
Grenzwerte: (172,150,255)



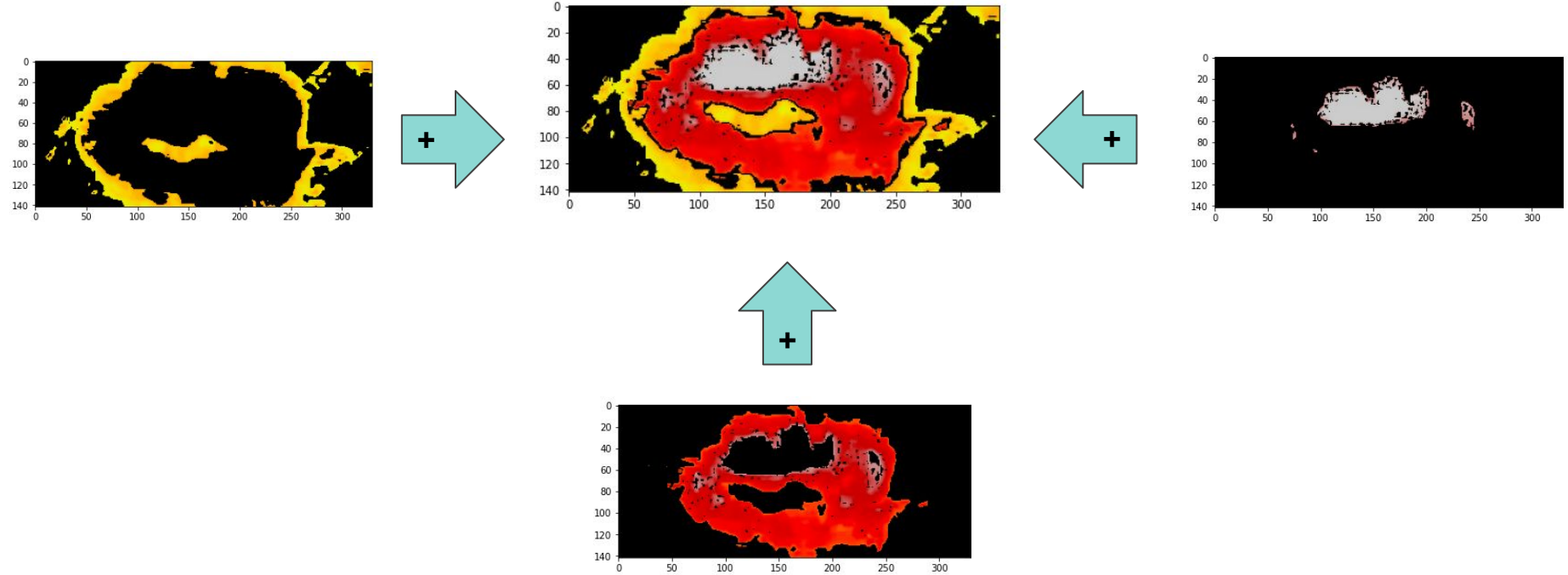
Erstellung von Maske(n) mit bearbeiteten Bildern

- Gleiche Schritte wie mit unbearbeiteten Bildern
- Jedoch **statt eine Maske haben wir drei Masken**





Kombination von drei Masken



Ausgabe der Pixelanzahl

Unbearbeitetes Bild

```
print("Pixelanzahl: "+str(np.count_nonzero(mask)))
```

```
Pixelanzahl: 219
```

Bearbeitetes Bild

```
#Ausgabe von Pixelanzahl
print("Pixelanzahl: "+str(np.count_nonzero(final_final_mask)))
```

```
Pixelanzahl: 26935
```

Vielen Dank für Ihre Aufmerksamkeit!