Datenvisualisierung

May 21, 2022

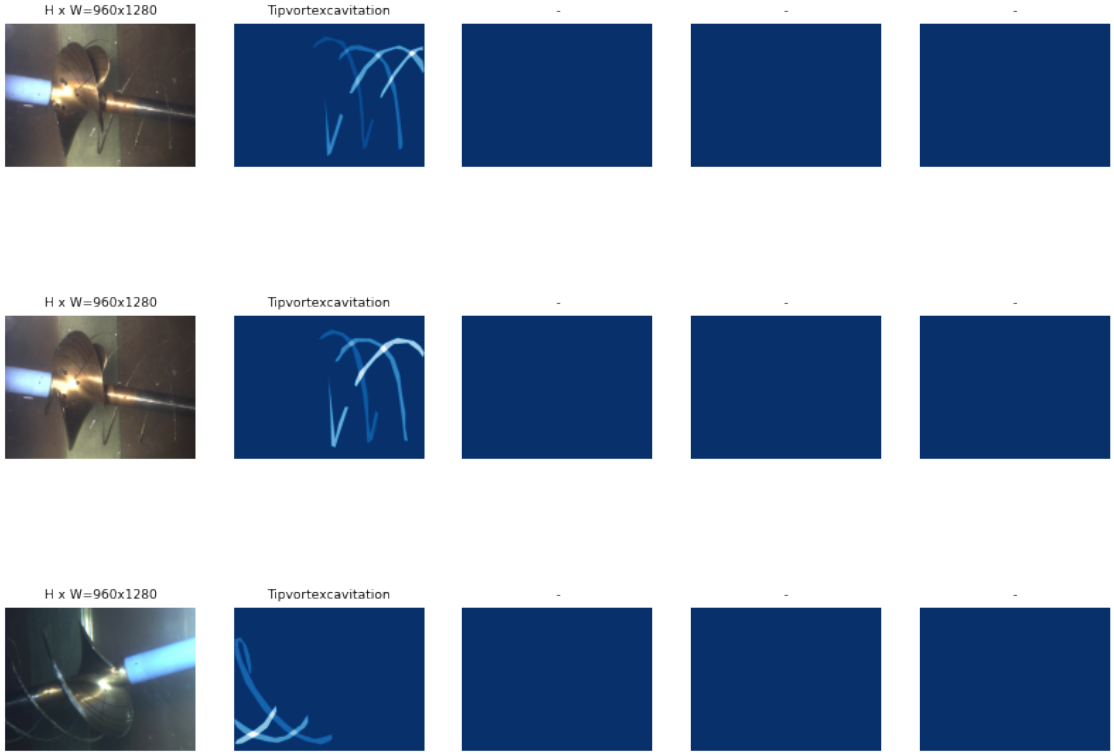
[8]:
```
import os
import sys
import itertools
import math
import logging
import json
import re
import random
from collections import OrderedDict
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.lines as lines
from matplotlib.patches import Polygon

# Der Pfad der Mask R_CNN Projektdatei

ROOT_DIR = os.path.abspath("C:/Users/majd4/Desktop/Bachelorarbeit/
 ↪Bachelor-Arbeit-Daten/MaskRCNNProjekt/MaskRCNN_2/Mask_RCNN")

# Mask R_CNN importieren
sys.path.append(ROOT_DIR)
from mrcnn import utils
from mrcnn import visualize
from mrcnn.visualize import display_images
import mrcnn.model as modellib
from mrcnn.model import log

from samples.Tipvortexcavitation import Tipvortexcavitation

%matplotlib inline
```

[9]:
```
config = Tipvortexcavitation.TipvortexcavitationConfig()
Tipvortexcavitationdir = os.path.join(ROOT_DIR, "datasets/Tipvortexcavitation")
```

[10]:
```
# Datensatz laden.
dataset = Tipvortexcavitation.TipvortexcavitationDataset()
dataset.load_Tipvortexcavitation(Tipvortexcavitationdir, "train")


dataset.prepare()

print("Image Count: {}".format(len(dataset.image_ids)))
print("Class Count: {}".format(dataset.num_classes))
for i, info in enumerate(dataset.class_info):
    print("{:3}. {:50}".format(i, info['name']))
```

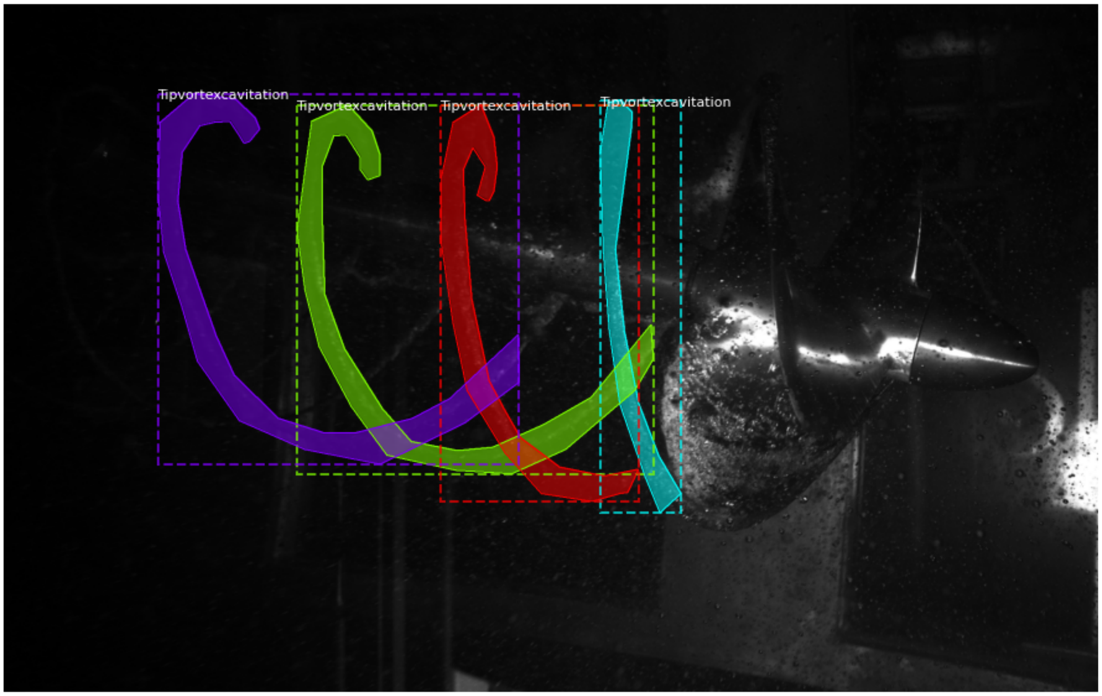
```
Image Count: 84
Class Count: 2
  0. BG
  1. Tipvortexcavitation
```

[11]:
```
# 4 Bilder zufällig auswählen, laden und visualisieren
# Masken erzeugen
image_ids = np.random.choice(dataset.image_ids, 5)
for image_id in image_ids:
    image = dataset.load_image(image_id)
    mask, class_ids = dataset.load_mask(image_id)
    visualize.display_top_masks(image, mask, class_ids, dataset.class_names)
```



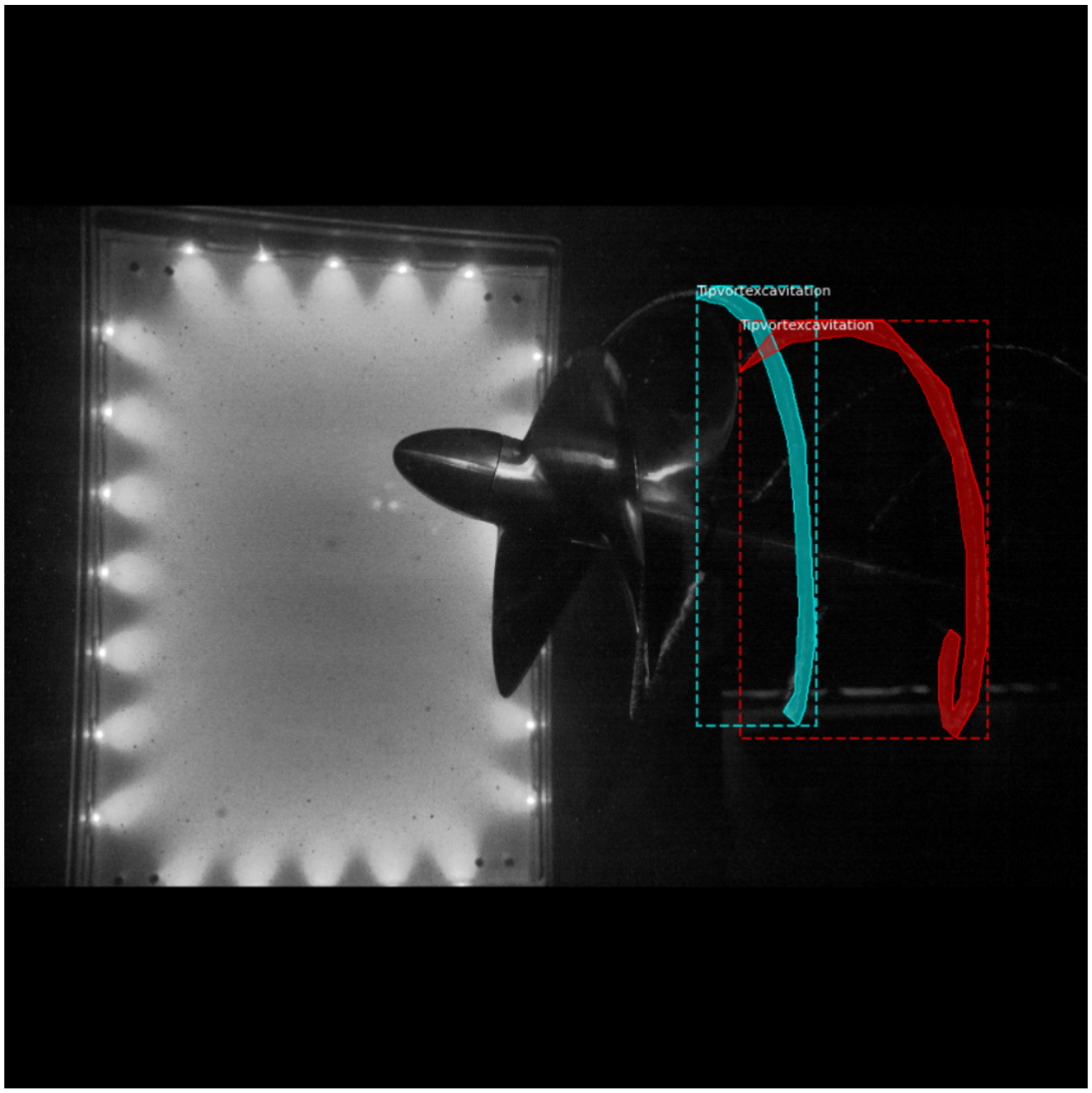
[12]:
```
# zufälliges Bild und zufällige Maske auswählen
image_id = random.choice(dataset.image_ids)
image = dataset.load_image(image_id)
mask, class_ids = dataset.load_mask(image_id)

#  Bounding box berechnen
bbox = utils.extract_bboxes(mask)

# Bild anzeigen, die Position und das Shape von Bounding box ausgeben.
# Bouding box begrenzt das Objekt von dem Gesamtbild in einem Rahmen

print("image_id ", image_id, dataset.image_reference(image_id))
log("image", image)
log("mask", mask)
log("class_ids", class_ids)
log("bbox", bbox)

visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```

```
image_id  43 C:\Users\majd4\Desktop\Bachelorarbeit\Bachelor-Arbeit-Daten\MaskRCN
NProjekt\MaskRCNN_2\Mask_RCNN\datasets/Tipvortexcavitation\train\Neue Videos_000
0000002_2021_11_17_15_41_30_414_2021_11_17_14_41_30_411_226502590347_229108.png
image                    shape: (1216, 1936, 3)       min:    0.00000  max:
255.00000  uint8
mask                     shape: (1216, 1936, 4)       min:    0.00000  max:
1.00000  bool
class_ids                shape: (4,)                  min:    1.00000  max:
1.00000  int32
bbox                     shape: (4, 4)                min:  157.00000  max:
1199.00000  int32
```



[13]:
```
# zufälliges Bild und zufällige Maske hochladen
image_id = np.random.choice(dataset.image_ids, 1)[0]
image = dataset.load_image(image_id)
mask, class_ids = dataset.load_mask(image_id)
original_shape = image.shape
# Größe ändern.
image, window, scale, padding, _ = utils.resize_image(
    image,
    min_dim=  config.IMAGE_MIN_DIM,
    max_dim=config.IMAGE_MAX_DIM,
    mode=config.IMAGE_RESIZE_MODE)
mask = utils.resize_mask(mask, scale, padding)

bbox = utils.extract_bboxes(mask)

print("image_id: ", image_id, dataset.image_reference(image_id))
print("Original shape: ", original_shape)
log("image", image)
log("mask", mask)
log("class_ids", class_ids)
log("bbox", bbox)

visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```

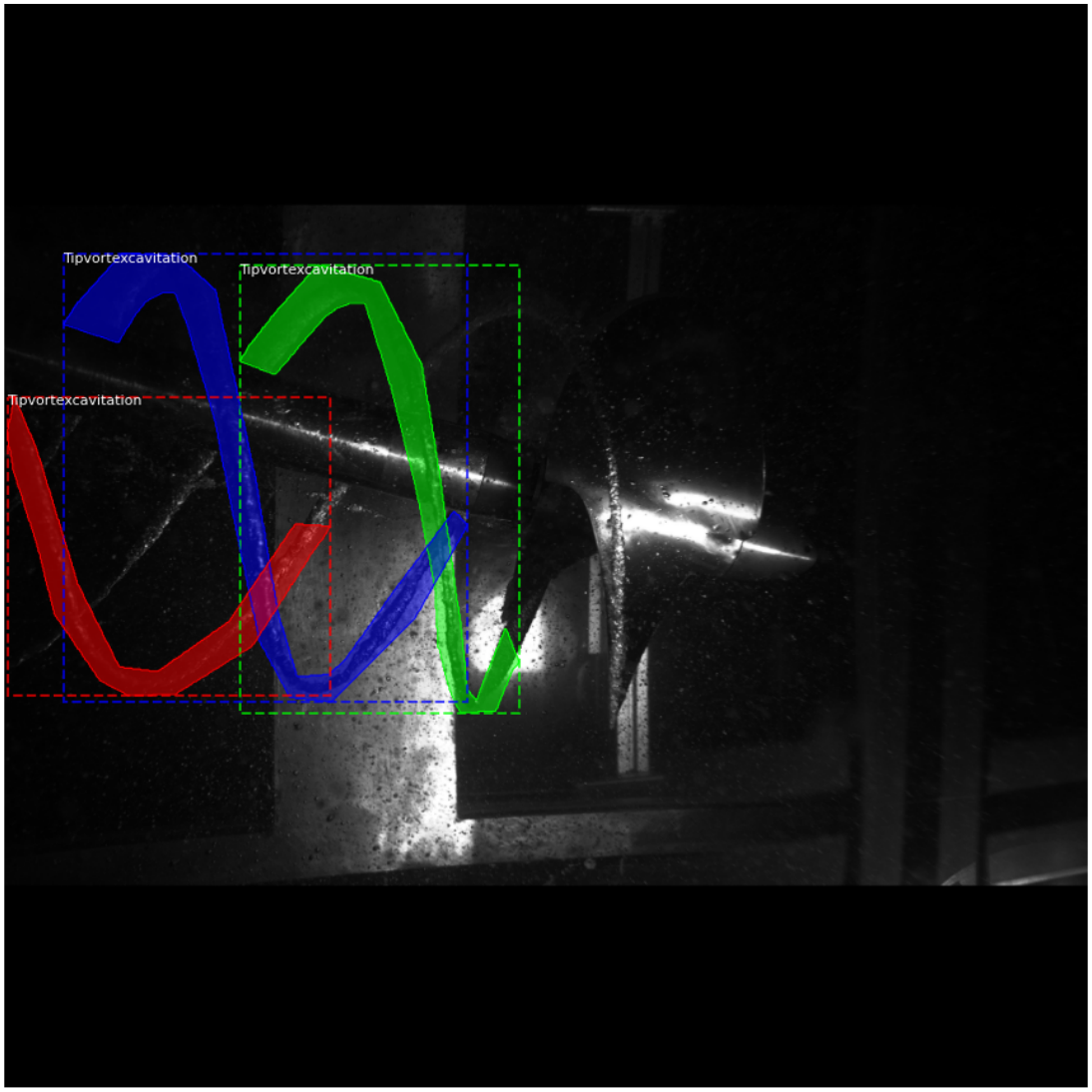
```
image_id:  62 C:\Users\majd4\Desktop\Bachelorarbeit\Bachelor-Arbeit-Daten\MaskRC
NNProjekt\MaskRCNN_2\Mask_RCNN\datasets/Tipvortexcavitation\train\Pictures_00000
00001_2021_11_17_13_18_19_162_2021_11_17_12_18_19_163_140590445308_121084.png
Original shape:  (1216, 1936, 3)
image                    shape: (1024, 1024, 3)       min:    0.00000  max:
255.00000  uint8
mask                     shape: (1024, 1024, 2)       min:    0.00000  max:
1.00000  bool
class_ids                shape: (2,)                  min:    1.00000  max:
1.00000  int32
bbox                     shape: (2, 4)                min:  265.00000  max:
930.00000  int32
```



[14]:
```
image_id = np.random.choice(dataset.image_ids, 1)[0]
image, image_meta, class_ids, bbox, mask = modellib.load_image_gt(
    dataset, config, image_id, use_mini_mask=False)

log("image", image)
log("image_meta", image_meta)
log("class_ids", class_ids)
log("bbox", bbox)
log("mask", mask)

display_images([image]+[mask[:,:,i] for i in range(min(mask.shape[-1], 7))])
```

```
image                    shape: (1024, 1024, 3)       min:    0.00000  max:
255.00000  uint8
image_meta               shape: (14,)                 min:    0.00000  max:
1936.00000  float64
class_ids                shape: (3,)                  min:    1.00000  max:
1.00000  int32
bbox                     shape: (3, 4)                min:    3.00000  max:
669.00000  int32
mask                     shape: (1024, 1024, 3)       min:    0.00000  max:
1.00000  bool
```

[15]:
```
visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```



[16]:
```
image, image_meta, class_ids, bbox, mask = modellib.load_image_gt(
    dataset, config, image_id, augment=True, use_mini_mask=True)
log("mask", mask)
display_images([image]+[mask[:,:,i] for i in range(min(mask.shape[-1], 7))])
```

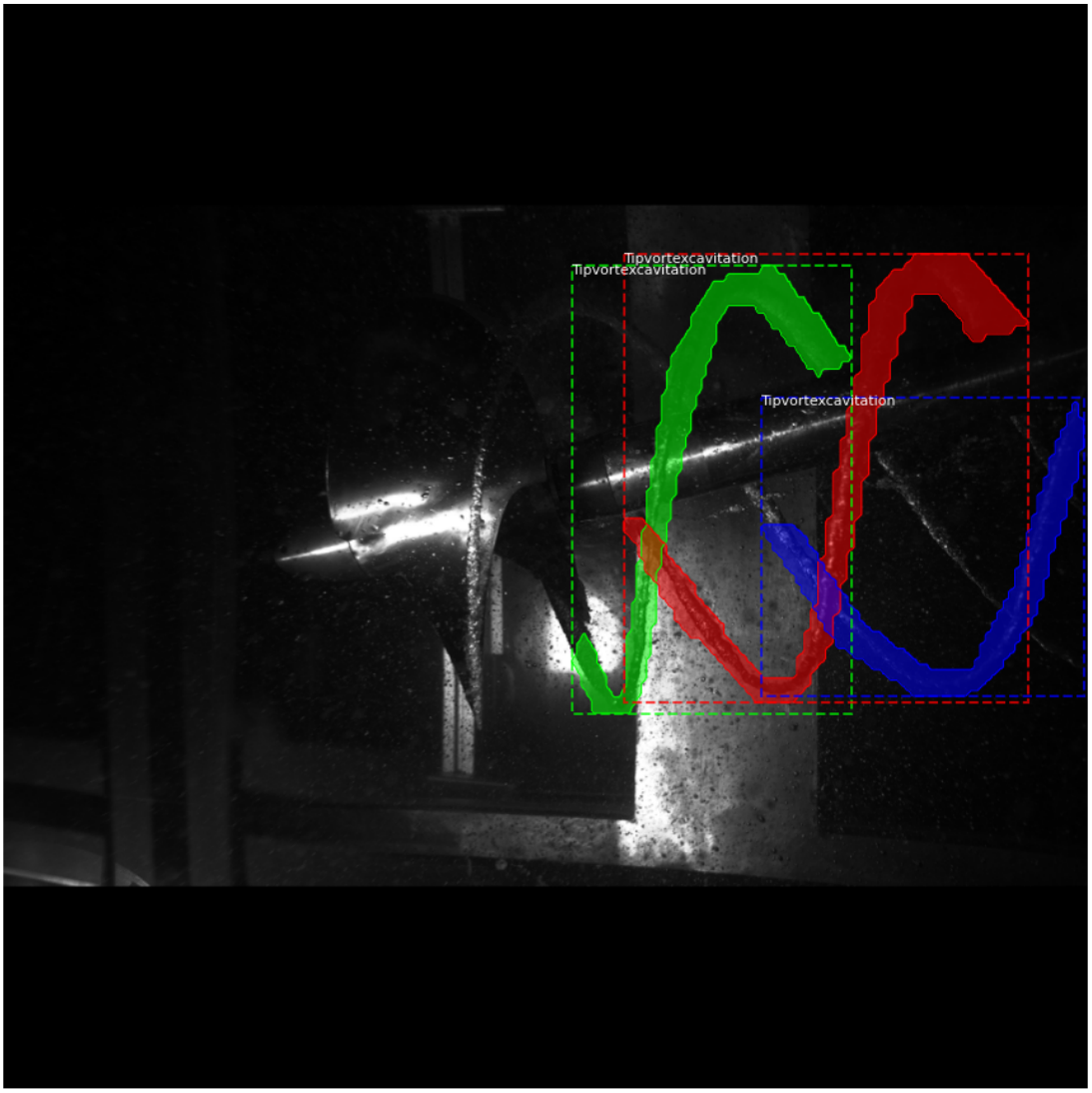
```
WARNING:root:'augment' is deprecated. Use 'augmentation' instead.
```

```
mask                     shape: (56, 56, 3)           min:    0.00000  max:
1.00000  bool
```

```
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\skimage\transform\_warps.py:830: FutureWarning: Input image dtype is
bool. Interpolation is not defined with bool data type. Please set order to 0 or
explicitely cast input image to another data type. Starting from version 0.19 a
```

```
ValueError will be raised instead of this warning.
  order = _validate_interpolation_order(image.dtype, order)
```

[17]:
```
mask = utils.expand_mask(bbox, mask, image.shape)
visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```



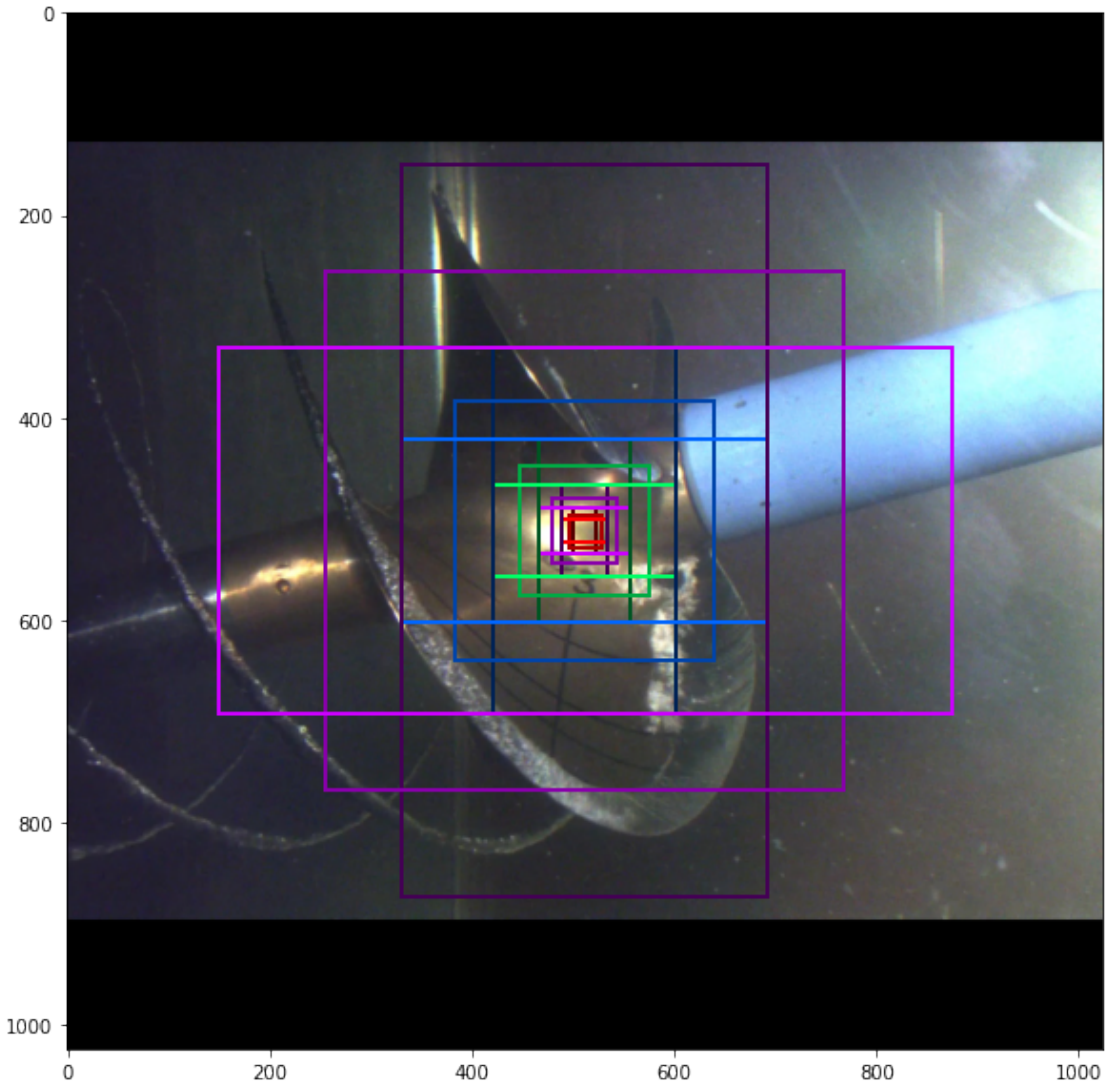
[18]:
```
# Rahmen erstellen
backbone_shapes = modellib.compute_backbone_shapes(config, config.IMAGE_SHAPE)
anchors = utils.generate_pyramid_anchors(config.RPN_ANCHOR_SCALES,
                                         config.RPN_ANCHOR_RATIOS,
                                         backbone_shapes,
                                         config.BACKBONE_STRIDES,
                                         config.RPN_ANCHOR_STRIDE)

# Informationen über die Rahmen Ausgaben
num_levels = len(backbone_shapes)
anchors_per_cell = len(config.RPN_ANCHOR_RATIOS)
print("Count: ", anchors.shape[0])
print("Scales: ", config.RPN_ANCHOR_SCALES)
print("ratios: ", config.RPN_ANCHOR_RATIOS)
print("Anchors per Cell: ", anchors_per_cell)
print("Levels: ", num_levels)
anchors_per_level = []
for l in range(num_levels):
    num_cells = backbone_shapes[l][0] * backbone_shapes[l][1]
    anchors_per_level.append(anchors_per_cell * num_cells // config.
 ↪RPN_ANCHOR_STRIDE**2)
    print("Anchors in Level {}: {}".format(l, anchors_per_level[l]))
```

```
Count:  261888
Scales:  (32, 64, 128, 256, 512)
ratios:  [0.5, 1, 2]
Anchors per Cell:  3
Levels:  5
Anchors in Level 0: 196608
Anchors in Level 1: 49152
Anchors in Level 2: 12288
Anchors in Level 3: 3072
Anchors in Level 4: 768
```

[19]:
```
# ein zufälliges Bild zeichnen und laden
image_id = np.random.choice(dataset.image_ids, 1)[0]
image, image_meta, _, _, _ = modellib.load_image_gt(dataset, config, image_id)
fig, ax = plt.subplots(1, figsize=(10, 10))
ax.imshow(image)
levels = len(backbone_shapes)

for level in range(levels):
    colors = visualize.random_colors(levels)
    #  den Index der Rahmen in der Mitte des Bildes Berechnen
```

```
    level_start = sum(anchors_per_level[:level]) # Summe der Rahmen der␣
↪vorherigen Ebenen
    level_anchors = anchors[level_start:level_start+anchors_per_level[level]]
    print("Level {}. Anchors: {:6}  Feature map Shape: {}".format(level,␣
↪level_anchors.shape[0],
                                                                  ␣
↪backbone_shapes[level]))
    center_cell = backbone_shapes[level] // 2
    center_cell_index = (center_cell[0] * backbone_shapes[level][1] +␣
↪center_cell[1])
    level_center = center_cell_index * anchors_per_cell
    center_anchor = anchors_per_cell * (
        (center_cell[0] * backbone_shapes[level][1] / config.
↪RPN_ANCHOR_STRIDE**2) \
        + center_cell[1] / config.RPN_ANCHOR_STRIDE)
    level_center = int(center_anchor)



    for i, rect in enumerate(level_anchors[level_center:
↪level_center+anchors_per_cell]):
        y1, x1, y2, x2 = rect
        p = patches.Rectangle((x1, y1), x2-x1, y2-y1, linewidth=2,␣
↪facecolor='none',
                              edgecolor=(i+1)*np.array(colors[level]) /␣
↪anchors_per_cell)
        ax.add_patch(p)
```

```
Level 0. Anchors: 196608  Feature map Shape: [256 256]
Level 1. Anchors:  49152  Feature map Shape: [128 128]
Level 2. Anchors:  12288  Feature map Shape: [64 64]
Level 3. Anchors:   3072  Feature map Shape: [32 32]
Level 4. Anchors:    768  Feature map Shape: [16 16]
```



[20]:
```
random_rois = 2000
g = modellib.data_generator(
    dataset, config, shuffle=True, random_rois=random_rois,
    batch_size=4,
    detection_targets=True)
```

[21]:
```
if random_rois:
    [normalized_images, image_meta, rpn_match, rpn_bbox, gt_class_ids,␣
 ↪gt_boxes, gt_masks, rpn_rois, rois], \
    [mrcnn_class_ids, mrcnn_bbox, mrcnn_mask] = next(g)

    log("rois", rois)
    log("mrcnn_class_ids", mrcnn_class_ids)
```

```
    log("mrcnn_bbox", mrcnn_bbox)
    log("mrcnn_mask", mrcnn_mask)
else:
    [normalized_images, image_meta, rpn_match, rpn_bbox, gt_boxes, gt_masks], _␣
 ↪= next(g)

log("gt_class_ids", gt_class_ids)
log("gt_boxes", gt_boxes)
log("gt_masks", gt_masks)
log("rpn_match", rpn_match, )
log("rpn_bbox", rpn_bbox)
image_id = modellib.parse_image_meta(image_meta)["image_id"][0]
print("image_id: ", image_id, dataset.image_reference(image_id))


mrcnn_class_ids = mrcnn_class_ids[:,:,0]
```

```
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\skimage\transform\_warps.py:830: FutureWarning: Input image dtype is
bool. Interpolation is not defined with bool data type. Please set order to 0 or
explicitely cast input image to another data type. Starting from version 0.19 a
ValueError will be raised instead of this warning.
  order = _validate_interpolation_order(image.dtype, order)
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\skimage\transform\_warps.py:830: FutureWarning: Input image dtype is
bool. Interpolation is not defined with bool data type. Please set order to 0 or
explicitely cast input image to another data type. Starting from version 0.19 a
ValueError will be raised instead of this warning.
  order = _validate_interpolation_order(image.dtype, order)
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\skimage\transform\_warps.py:830: FutureWarning: Input image dtype is
bool. Interpolation is not defined with bool data type. Please set order to 0 or
explicitely cast input image to another data type. Starting from version 0.19 a
ValueError will be raised instead of this warning.
  order = _validate_interpolation_order(image.dtype, order)
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\skimage\transform\_warps.py:830: FutureWarning: Input image dtype is
bool. Interpolation is not defined with bool data type. Please set order to 0 or
explicitely cast input image to another data type. Starting from version 0.19 a
ValueError will be raised instead of this warning.
  order = _validate_interpolation_order(image.dtype, order)
```

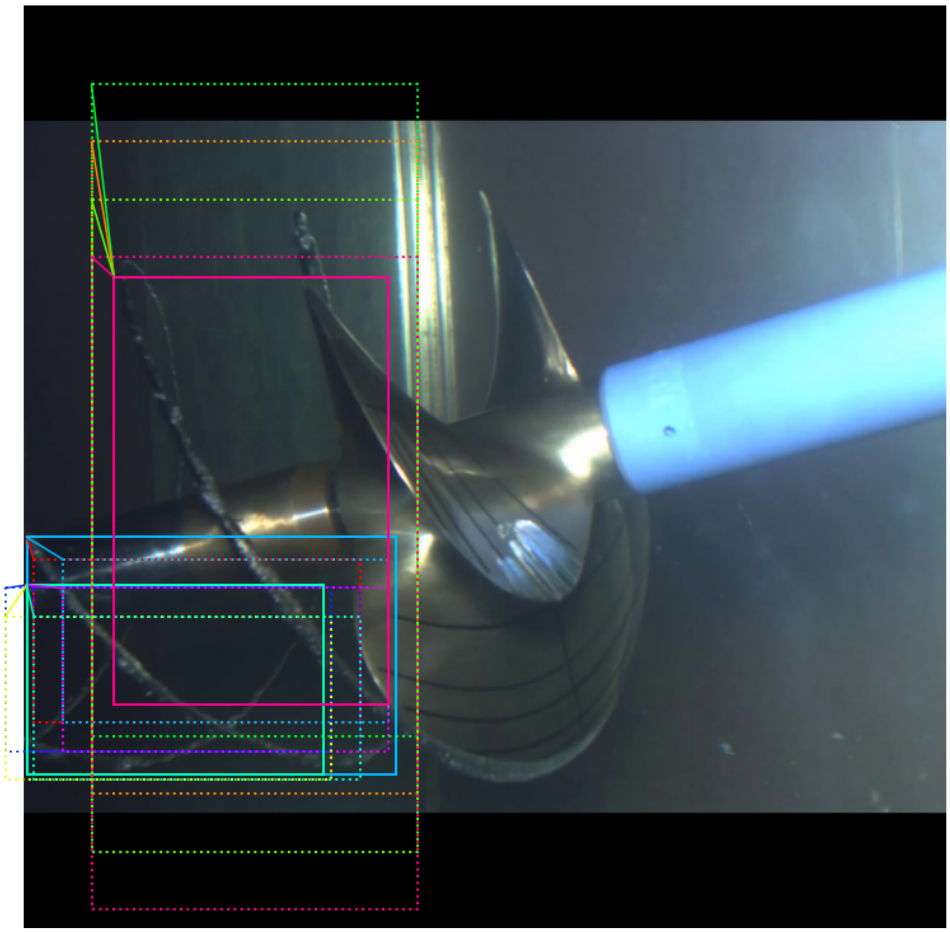
```
rois                     shape: (4, 200, 4)           min:    0.00000  max:
1023.00000  int32
mrcnn_class_ids          shape: (4, 200, 1)           min:    0.00000  max:
1.00000  int32
mrcnn_bbox               shape: (4, 200, 2, 4)        min:   -4.13265  max:
3.31828  float32
```

```
mrcnn_mask               shape: (4, 200, 28, 28, 2)   min:    0.00000  max:
1.00000  float32
gt_class_ids             shape: (4, 100)              min:    0.00000  max:
1.00000  int32
gt_boxes                 shape: (4, 100, 4)           min:    0.00000  max:
1018.00000  int32
gt_masks                 shape: (4, 56, 56, 100)      min:    0.00000  max:
1.00000  bool
rpn_match                shape: (4, 261888, 1)        min:   -1.00000  max:
1.00000  int32
rpn_bbox                 shape: (4, 256, 4)           min:   -3.61840  max:
1.93359  float64
image_id:  13 C:\Users\majd4\Desktop\Bachelorarbeit\Bachelor-Arbeit-Daten\MaskRC
NNProjekt\MaskRCNN_2\Mask_RCNN\datasets/Tipvortexcavitation\train\Stb Gesamt0001
13-09-26 14-39-46-2 03.jpg
```

[22]:
```
b = 0

# originales Bild wiederherstellen
sample_image = modellib.unmold_image(normalized_images[b], config)

# Rahmenverschiedbungen berechnen
indices = np.where(rpn_match[b] == 1)[0]
refined_anchors = utils.apply_box_deltas(anchors[indices], rpn_bbox[b, :
 ↪len(indices)] * config.RPN_BBOX_STD_DEV)
log("anchors", anchors)
log("refined_anchors", refined_anchors)

# liste für positive Rahmen bekommen
positive_anchor_ids = np.where(rpn_match[b] == 1)[0]
print("Positive anchors: {}".format(len(positive_anchor_ids)))
negative_anchor_ids = np.where(rpn_match[b] == -1)[0]
print("Negative anchors: {}".format(len(negative_anchor_ids)))
neutral_anchor_ids = np.where(rpn_match[b] == 0)[0]
print("Neutral anchors: {}".format(len(neutral_anchor_ids)))


for c, n in zip(dataset.class_names, np.bincount(mrcnn_class_ids[b].flatten())):
    if n:
        print("{:23}: {}".format(c[:20], n))

# Positive Rahmen anzeigen. Positiv, bedeutet, dass der Rahmen ein Objekt␣
 ↪beinhaltet
fig, ax = plt.subplots(1, figsize=(16, 16))
visualize.draw_boxes(sample_image, boxes=anchors[positive_anchor_ids],
                     refined_boxes=refined_anchors, ax=ax)
```

```
anchors                  shape: (261888, 4)           min: -362.03867  max:
```

```
1322.03867  float64
refined_anchors          shape: (11, 4)               min:    3.00000  max:
853.99994  float32
Positive anchors: 11
Negative anchors: 245
Neutral anchors: 261632
BG                     : 134
Tipvortexcavitation    : 66
```



[23]:
```
# ein Rahmen ist negativ, wenn der Rahmen kein Tipvortexkavitation beinhaltet
visualize.draw_boxes(sample_image, boxes=anchors[negative_anchor_ids])
```