# Modellvisualisierung

May 22, 2022

[1]:
```
import os
import sys
import random
import math
import re
import time
import numpy as np
import tensorflow as tf
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
# Der Pfad der Projektdatei Mask_CNN
ROOT_DIR = os.path.abspath("C:/Users/majd4/Desktop/Bachelorarbeit/
 ↪Bachelor-Arbeit-Daten/MaskRCNNProjekt/MaskRCNN_2/Mask_RCNN")
# In Sytsem Bibliothek hinzufügen
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
# Die Datei für die logs angeben
MODEL_DIR = os.path.join(ROOT_DIR, "logs")
```

```
Using TensorFlow backend.
```

[2]:
```
config = Tipvortexcavitation.TipvortexcavitationConfig()
Tipvortexcavitation_DIR = os.path.join(ROOT_DIR, "datasets/Tipvortexcavitation")
```

Backbone ist dafür da, um interessante Merkmale aus dem Eingabebild herauszunehmen oder zu extrahieren oder zu Filtern. Backbone strides stellt die verschiebung der Filter dar Die Strides stellen die Verschiebungen dar

[3]:
```
# config wird die Funktionsweise der TipvortexcavitationConfig funktion␣
 ↪übernommen und
# einpaar werte überschrieben
```

```python
class InferenceConfig(config.__class__):
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

config = InferenceConfig()
config.display()
```

```
Configurations:
BACKBONE                       resnet101
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     1
BBOX_STD_DEV                   [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE         None
DETECTION_MAX_INSTANCES        100
DETECTION_MIN_CONFIDENCE       0.7
DETECTION_NMS_THRESHOLD        0.3
FPN_CLASSIF_FC_LAYERS_SIZE     1024
GPU_COUNT                      1
GRADIENT_CLIP_NORM             5.0
IMAGES_PER_GPU                 1
IMAGE_CHANNEL_COUNT            3
IMAGE_MAX_DIM                  1024
IMAGE_META_SIZE                14
IMAGE_MIN_DIM                  800
IMAGE_MIN_SCALE                0
IMAGE_RESIZE_MODE              square
IMAGE_SHAPE                    [1024 1024    3]
LEARNING_MOMENTUM              0.9
LEARNING_RATE                  0.0001
LOSS_WEIGHTS                   {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0,
'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0, 'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE                 14
MASK_SHAPE                     [28, 28]
MAX_GT_INSTANCES               100
MEAN_PIXEL                     [123.7 116.8 103.9]
MINI_MASK_SHAPE                (56, 56)
NAME                           Tipvortexcavitation
NUM_CLASSES                    2
POOL_SIZE                      7
POST_NMS_ROIS_INFERENCE        1000
POST_NMS_ROIS_TRAINING         2000
PRE_NMS_LIMIT                  6000
ROI_POSITIVE_RATIO             0.33
RPN_ANCHOR_RATIOS              [0.5, 1, 2]
RPN_ANCHOR_SCALES              (32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE              1
```

```
RPN_BBOX_STD_DEV                [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD               0.7
RPN_TRAIN_ANCHORS_PER_IMAGE     256
STEPS_PER_EPOCH                 10
TOP_DOWN_PYRAMID_SIZE           256
TRAIN_BN                        False
TRAIN_ROIS_PER_IMAGE            200
USE_MINI_MASK                   True
USE_RPN_ROIS                    True
VALIDATION_STEPS                20
WEIGHT_DECAY                    0.0001
```

[4]:
```
DEVICE = "/cpu:0"
TEST_MODE = "inference"
```

[5]:
```
def get_ax(rows=1, cols=1, size=16):
    _, ax = plt.subplots(rows, cols, figsize=(size*cols, size*rows))
    return ax
```

[6]:
```
dataset = Tipvortexcavitation.TipvortexcavitationDataset()
dataset.load_Tipvortexcavitation(Tipvortexcavitation_DIR, "train")
dataset.prepare()

print("Images: {}\nClasses: {}".format(len(dataset.image_ids), dataset.
 ↪class_names))
```

```
Images: 84
Classes: ['BG', 'Tipvortexcavitation']
```

[7]:
```
with tf.device(DEVICE):
    model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR,
                              config=config)
```

```
WARNING:tensorflow:From
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\keras\backend\tensorflow_backend.py:517: The name tf.placeholder is
deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\keras\backend\tensorflow_backend.py:74: The name tf.get_default_graph
is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\keras\backend\tensorflow_backend.py:4138: The name tf.random_uniform is
```

```
deprecated. Please use tf.random.uniform instead.

WARNING:tensorflow:From
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\keras\backend\tensorflow_backend.py:1919: The name
tf.nn.fused_batch_norm is deprecated. Please use
tf.compat.v1.nn.fused_batch_norm instead.

WARNING:tensorflow:From
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\keras\backend\tensorflow_backend.py:3976: The name tf.nn.max_pool is
deprecated. Please use tf.nn.max_pool2d instead.

WARNING:tensorflow:From
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\keras\backend\tensorflow_backend.py:2018: The name
tf.image.resize_nearest_neighbor is deprecated. Please use
tf.compat.v1.image.resize_nearest_neighbor instead.

WARNING:tensorflow:From C:\Users\majd4\Desktop\Bachelorarbeit\Bachelor-Arbeit-
Daten\MaskRCNNProjekt\MaskRCNN_2\Mask_RCNN\mrcnn\model.py:341: The name tf.log
is deprecated. Please use tf.math.log instead.

WARNING:tensorflow:From C:\Users\majd4\Desktop\Bachelorarbeit\Bachelor-Arbeit-
Daten\MaskRCNNProjekt\MaskRCNN_2\Mask_RCNN\mrcnn\model.py:399: where (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From C:\Users\majd4\Desktop\Bachelorarbeit\Bachelor-Arbeit-
Daten\MaskRCNNProjekt\MaskRCNN_2\Mask_RCNN\mrcnn\model.py:423: calling
crop_and_resize_v1 (from tensorflow.python.ops.image_ops_impl) with box_ind is
deprecated and will be removed in a future version.
Instructions for updating:
box_ind is deprecated, use box_indices instead
WARNING:tensorflow:From C:\Users\majd4\Desktop\Bachelorarbeit\Bachelor-Arbeit-
Daten\MaskRCNNProjekt\MaskRCNN_2\Mask_RCNN\mrcnn\model.py:720: The name
tf.sets.set_intersection is deprecated. Please use tf.sets.intersection instead.

WARNING:tensorflow:From C:\Users\majd4\Desktop\Bachelorarbeit\Bachelor-Arbeit-
Daten\MaskRCNNProjekt\MaskRCNN_2\Mask_RCNN\mrcnn\model.py:722: The name
tf.sparse_tensor_to_dense is deprecated. Please use tf.sparse.to_dense instead.

WARNING:tensorflow:From C:\Users\majd4\Desktop\Bachelorarbeit\Bachelor-Arbeit-
Daten\MaskRCNNProjekt\MaskRCNN_2\Mask_RCNN\mrcnn\model.py:772: to_float (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
```

```
Use `tf.cast` instead.
```

[8]:
```
#model.keras_model.summary()
```

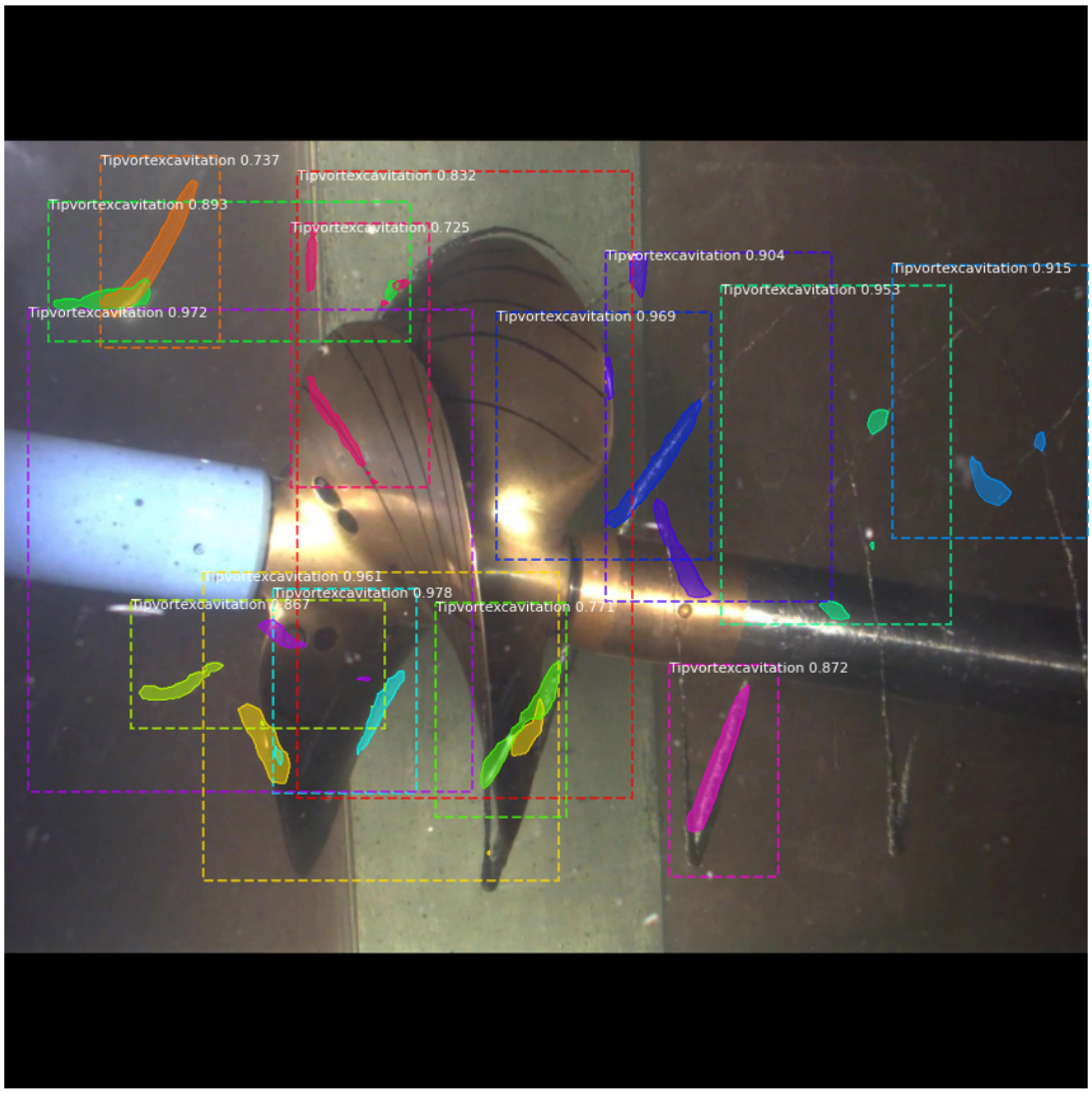
[9]:
```
# weights_path = model.find_last()
weights_path = "C:/Users/majd4/Desktop/Bachelorarbeit/Bachelor-Arbeit-Daten/
 ↪MaskRCNNProjekt/MaskRCNN_2/Mask_RCNN/mask_rcnn_tipvortexcavitation_0100.h5"
# Gewichte Laden
print("Loading weights ", weights_path)
model.load_weights(weights_path, by_name=True)
```

```
Loading weights  C:/Users/majd4/Desktop/Bachelorarbeit/Bachelor-Arbeit-
Daten/MaskRCNNProjekt/MaskRCNN_2/Mask_RCNN/mask_rcnn_tipvortexcavitation_0100.h5
WARNING:tensorflow:From
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\keras\backend\tensorflow_backend.py:174: The name
tf.get_default_session is deprecated. Please use
tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\keras\backend\tensorflow_backend.py:181: The name tf.ConfigProto is
deprecated. Please use tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\keras\backend\tensorflow_backend.py:186: The name tf.Session is
deprecated. Please use tf.compat.v1.Session instead.

WARNING:tensorflow:From
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\keras\backend\tensorflow_backend.py:190: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\keras\backend\tensorflow_backend.py:199: The name
tf.is_variable_initialized is deprecated. Please use
tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From
C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNNN\lib\site-
packages\keras\backend\tensorflow_backend.py:206: The name
tf.variables_initializer is deprecated. Please use
tf.compat.v1.variables_initializer instead.
```

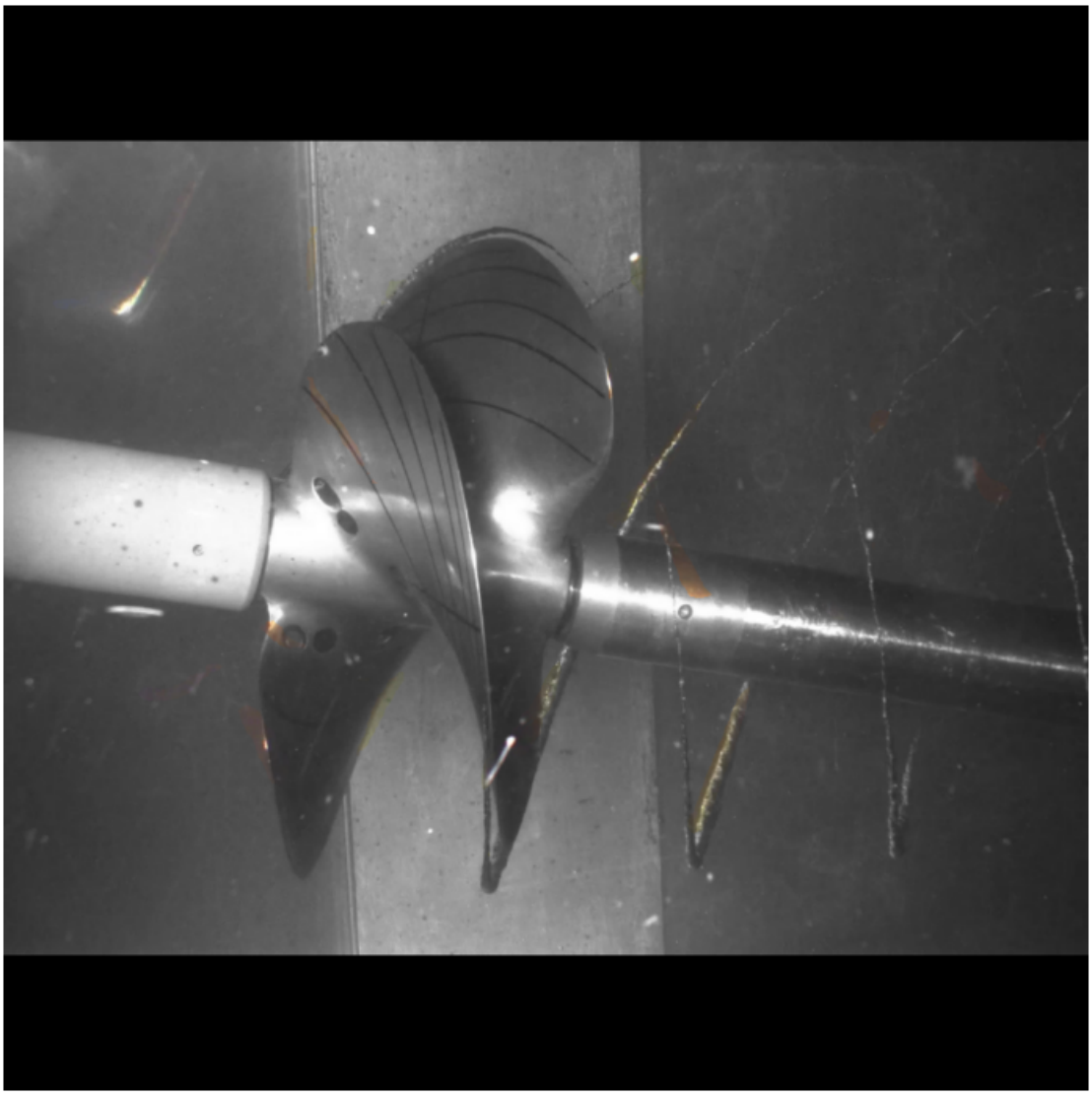
[10]:
```
image_id = random.choice(dataset.image_ids)
image, image_meta, gt_class_id, gt_bbox, gt_mask =\
    modellib.load_image_gt(dataset, config, image_id, use_mini_mask=False)
info = dataset.image_info[image_id]
print("image ID: {}.{} ({}) {}".format(info["source"], info["id"], image_id,
                                       dataset.image_reference(image_id)))
# Objekterkennung ausführen
results = model.detect([image], verbose=1)
# Ergebnisse anzeigen
ax = get_ax(1)
r = results[0]
visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'],
                            dataset.class_names, r['scores'], ax=ax,
                            title="Predictions")
log("gt_class_id", gt_class_id)
log("gt_bbox", gt_bbox)
log("gt_mask", gt_mask)
```

```
image ID: Tipvortexcavitation.Bb Gesamt0001 13-09-27 08-55-31-1 29.jpg (79)
C:\Users\majd4\Desktop\Bachelorarbeit\Bachelor-Arbeit-
Daten\MaskRCNNProjekt\MaskRCNN_2\Mask_RCNN\datasets/Tipvortexcavitation\train\Bb
Gesamt0001 13-09-27 08-55-31-1 29.jpg
Processing 1 images
image                    shape: (1024, 1024, 3)       min:    0.00000  max:
255.00000  uint8
molded_images            shape: (1, 1024, 1024, 3)    min: -123.70000  max:
151.10000  float64
image_metas              shape: (1, 14)               min:    0.00000  max:
1024.00000  int32
anchors                  shape: (1, 261888, 4)        min:   -0.35390  max:
1.29134  float32
gt_class_id              shape: (5,)                  min:    1.00000  max:
1.00000  int32
gt_bbox                  shape: (5, 4)                min:  201.00000  max:
1020.00000  int32
gt_mask                  shape: (1024, 1024, 5)       min:    0.00000  max:
1.00000  bool
```

Predictions




[11]:
```
splash = Tipvortexcavitation.color_splash(image, r['masks'])
display_images([splash], cols=1)
```

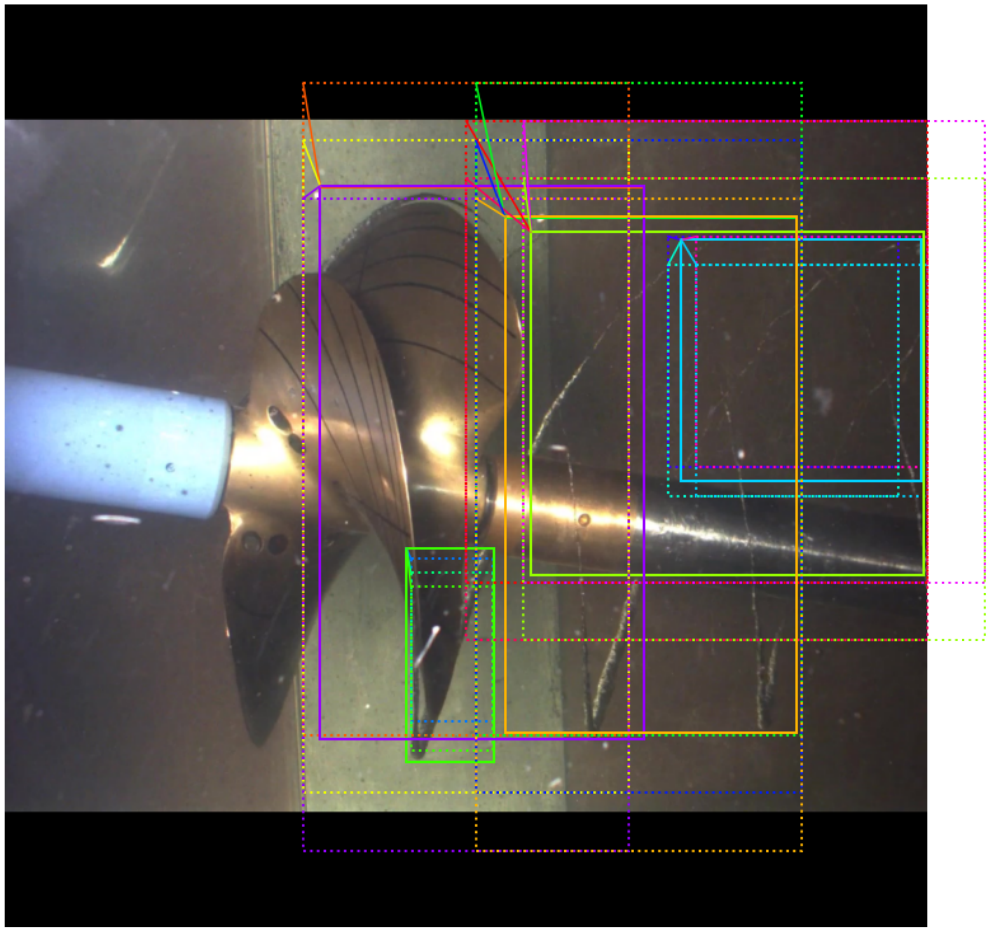


[12]:
```
# Region Prposal Network Schlägt Regionen vor, wo das gewollte oder erkannte␣
 ↪Objekt zu sehen ist
# Dieses Mathching, das bedeutet,  das Muster wird gefunden und wird eine␣
 ↪Quadrate darüber gezeichnet
# Positive anchor 1, -1 das Mactching wird nicht gefunden
# and 0 for neutral anchors. die Bounding boxs werden nach dem Muster␣
 ↪durchgesucht
# werden klassifiziert nach postiv anker 1, negitiver anker -1, neutral anker 0
target_rpn_match, target_rpn_bbox = modellib.build_rpn_targets(
    image.shape, model.anchors, gt_class_id, gt_bbox, model.config)
log("target_rpn_match", target_rpn_match)
log("target_rpn_bbox", target_rpn_bbox)
positive_anchor_ix = np.where(target_rpn_match[:] == 1)[0]
```

```
negative_anchor_ix = np.where(target_rpn_match[:] == -1)[0]
neutral_anchor_ix = np.where(target_rpn_match[:] == 0)[0]
positive_anchors = model.anchors[positive_anchor_ix]
negative_anchors = model.anchors[negative_anchor_ix]
neutral_anchors = model.anchors[neutral_anchor_ix]
log("positive_anchors", positive_anchors)
log("negative_anchors", negative_anchors)
log("neutral anchors", neutral_anchors)

# Wenden Sie Verfeinerungsdeltas auf positive Anker an
refined_anchors = utils.apply_box_deltas(
    positive_anchors,
    target_rpn_bbox[:positive_anchors.shape[0]] * model.config.RPN_BBOX_STD_DEV)
log("refined_anchors", refined_anchors, )
```

```
target_rpn_match         shape: (261888,)             min:   -1.00000  max:
1.00000  int32
target_rpn_bbox          shape: (256, 4)              min:   -1.47763  max:
1.36833  float64
positive_anchors         shape: (17, 4)               min:   85.96133  max:
1088.00000  float64
negative_anchors         shape: (239, 4)              min:  -32.00000  max:
1082.50967  float64
neutral anchors          shape: (261632, 4)           min: -362.03867  max:
1322.03867  float64
refined_anchors          shape: (17, 4)               min:  200.99997  max:
1020.00000  float32
```

[13]:
```
# Positive Anker vor der Verfeinerung anzeigen (gepunktet) und
# nach Verfeinerung
visualize.draw_boxes(image, boxes=positive_anchors,␣
↪refined_boxes=refined_anchors, ax=get_ax())
```

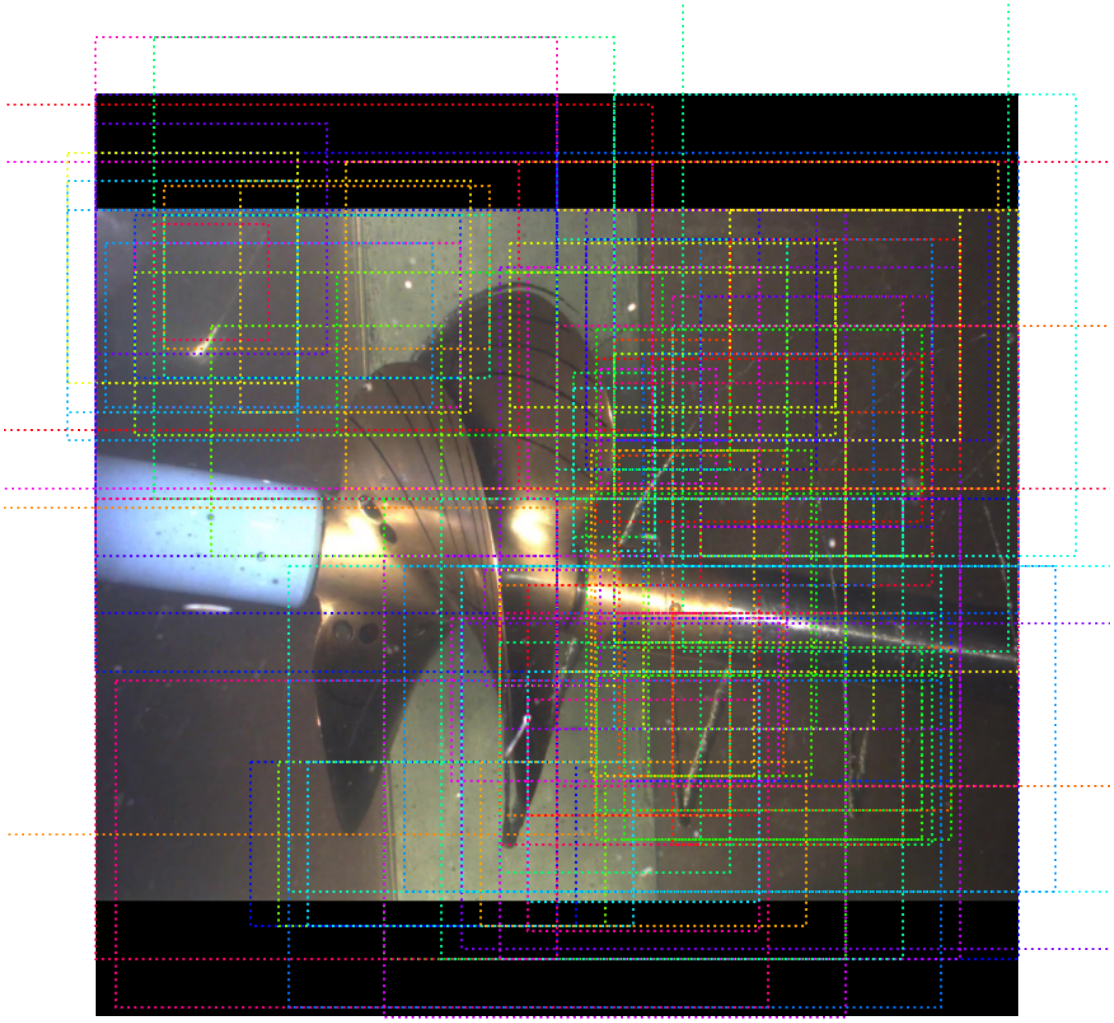


[14]:
```
# das RPN-Unterdiagramm ausführen
pillar = model.keras_model.get_layer("ROI").output  # node to start searching␣
 ↪from

# non maximum suppression
nms_node = model.ancestor(pillar, "ROI/rpn_non_max_suppression:0")
if nms_node is None:
    nms_node = model.ancestor(pillar, "ROI/rpn_non_max_suppression/
 ↪NonMaxSuppressionV2:0")
if nms_node is None: #TF 1.9-1.10
    nms_node = model.ancestor(pillar, "ROI/rpn_non_max_suppression/
 ↪NonMaxSuppressionV3:0")
```

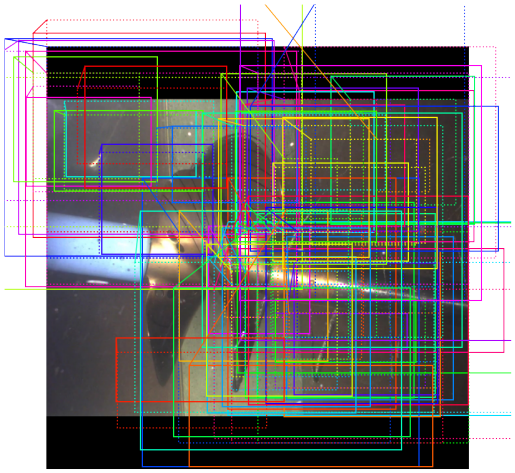
```
rpn = model.run_graph([image], [
    ("rpn_class", model.keras_model.get_layer("rpn_class").output),
    ("pre_nms_anchors", model.ancestor(pillar, "ROI/pre_nms_anchors:0")),
    ("refined_anchors", model.ancestor(pillar, "ROI/refined_anchors:0")),
    ("refined_anchors_clipped", model.ancestor(pillar, "ROI/
 ↪refined_anchors_clipped:0")),
    ("post_nms_anchor_ix", nms_node),
    ("proposals", model.keras_model.get_layer("ROI").output),
])
```

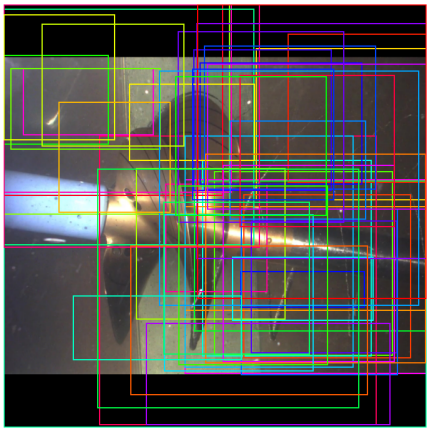
```
rpn_class                shape: (1, 261888, 2)        min:    0.00000  max:
1.00000  float32
pre_nms_anchors          shape: (1, 6000, 4)          min:   -0.29134  max:
1.29134  float32
refined_anchors          shape: (1, 6000, 4)          min: -16062.56543  max:
16063.24121  float32
refined_anchors_clipped  shape: (1, 6000, 4)          min:    0.00000  max:
1.00000  float32
post_nms_anchor_ix       shape: (1000,)               min:    0.00000  max:
3146.00000  int32
proposals                shape: (1, 1000, 4)          min:    0.00000  max:
1.00000  float32
```

[15]:
```
# Top-Anker nach Punktzahl anzeigen (vor der Verfeinerung)
limit = 100
sorted_anchor_ids = np.argsort(rpn['rpn_class'][:,:,1].flatten())[::-1]
visualize.draw_boxes(image, boxes=model.anchors[sorted_anchor_ids[:limit]],␣
 ↪ax=get_ax())
```

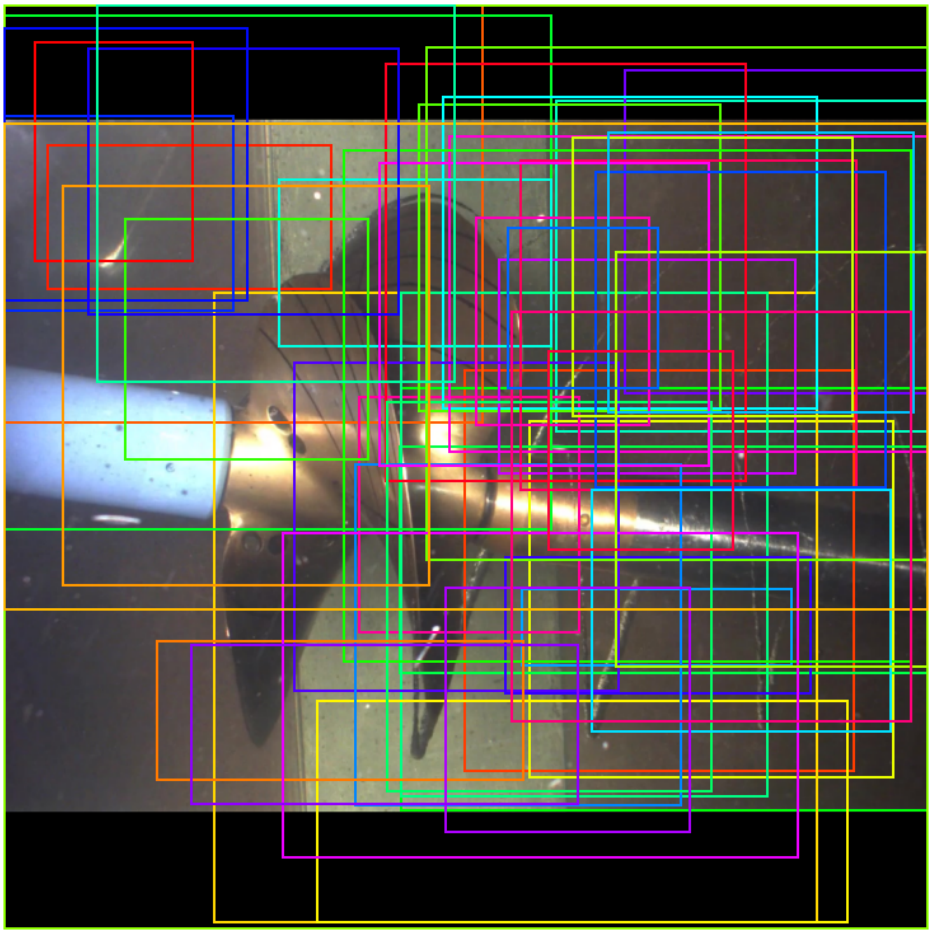


[16]:
```
#Zeige Top-Anker mit Verfeinerung. Dann mit Clipping auf Bildgrenzen
limit = 50
ax = get_ax(1, 2)
pre_nms_anchors = utils.denorm_boxes(rpn["pre_nms_anchors"][0], image.shape[:2])
refined_anchors = utils.denorm_boxes(rpn["refined_anchors"][0], image.shape[:2])
refined_anchors_clipped = utils.denorm_boxes(rpn["refined_anchors_clipped"][0],␣
↪image.shape[:2])
visualize.draw_boxes(image, boxes=pre_nms_anchors[:limit],
                     refined_boxes=refined_anchors[:limit], ax=ax[0])
visualize.draw_boxes(image, refined_boxes=refined_anchors_clipped[:limit],␣
↪ax=ax[1])
```





[17]:
```
# Verfeinerte Anker nach nicht maximaler Unterdrückung anzeigen
limit = 50
ixs = rpn["post_nms_anchor_ix"][:limit]
visualize.draw_boxes(image, refined_boxes=refined_anchors_clipped[ixs],␣
 ↪ax=get_ax())
```



[18]:
```
#
mrcnn = model.run_graph([image], [
    ("proposals", model.keras_model.get_layer("ROI").output),
    ("probs", model.keras_model.get_layer("mrcnn_class").output),
    ("deltas", model.keras_model.get_layer("mrcnn_bbox").output),
    ("masks", model.keras_model.get_layer("mrcnn_mask").output),
    ("detections", model.keras_model.get_layer("mrcnn_detection").output),
])
```

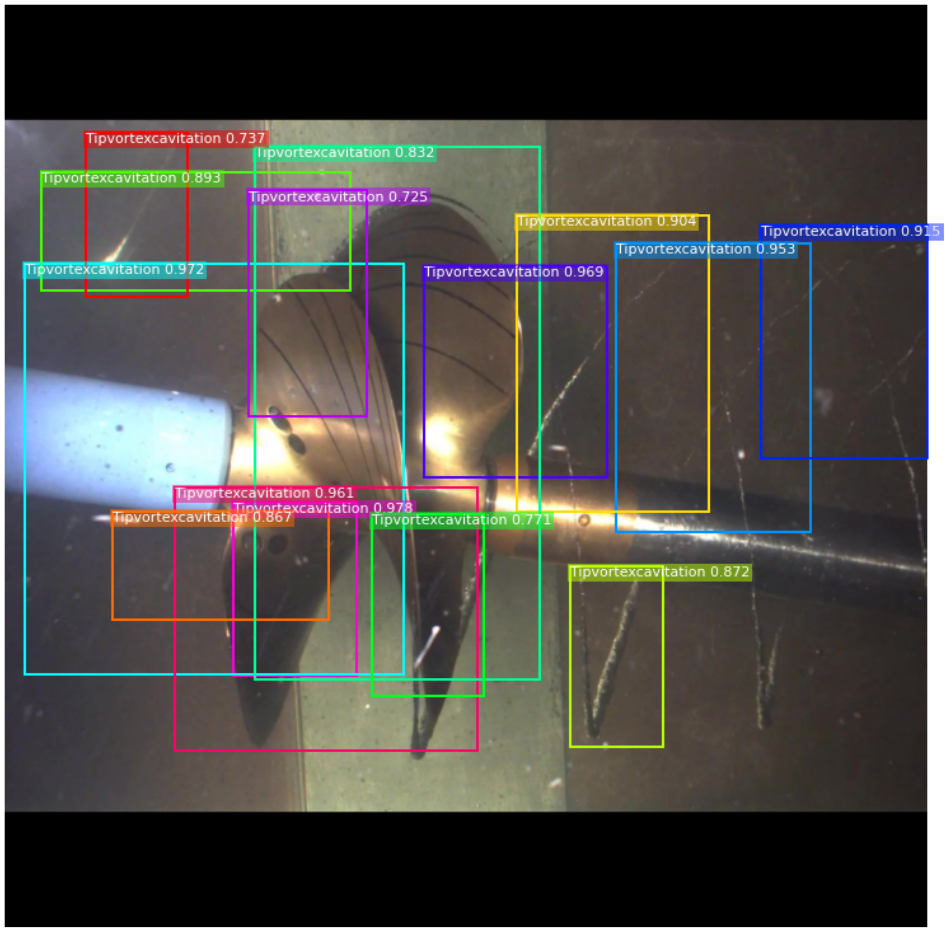
```
proposals                shape: (1, 1000, 4)           min:    0.00000  max:
1.00000  float32
probs                    shape: (1, 1000, 2)           min:    0.00000  max:
1.00000  float32
```

```
deltas                   shape: (1, 1000, 2, 4)       min:   -7.25453  max:
5.85932  float32
masks                    shape: (1, 100, 28, 28, 2)   min:    0.00000  max:
0.89839  float32
detections               shape: (1, 100, 6)           min:    0.00000  max:
1.00000  float32
```

[19]:
```
#
#
det_class_ids = mrcnn['detections'][0, :, 4].astype(np.int32)
det_count = np.where(det_class_ids == 0)[0][0]
det_class_ids = det_class_ids[:det_count]
detections = mrcnn['detections'][0, :det_count]

print("{} detections: {}".format(
    det_count, np.array(dataset.class_names)[det_class_ids]))

captions = ["{} {:.3f}".format(dataset.class_names[int(c)], s) if c > 0 else ""
            for c, s in zip(detections[:, 4], detections[:, 5])]
visualize.draw_boxes(
    image,
    refined_boxes=utils.denorm_boxes(detections[:, :4], image.shape[:2]),
    visibilities=[2] * len(detections),
    captions=captions, title="Detections",
    ax=get_ax())
```

```
14 detections: ['Tipvortexcavitation' 'Tipvortexcavitation'
'Tipvortexcavitation'
 'Tipvortexcavitation' 'Tipvortexcavitation' 'Tipvortexcavitation'
 'Tipvortexcavitation' 'Tipvortexcavitation' 'Tipvortexcavitation'
 'Tipvortexcavitation' 'Tipvortexcavitation' 'Tipvortexcavitation'
 'Tipvortexcavitation' 'Tipvortexcavitation']
```

Detections



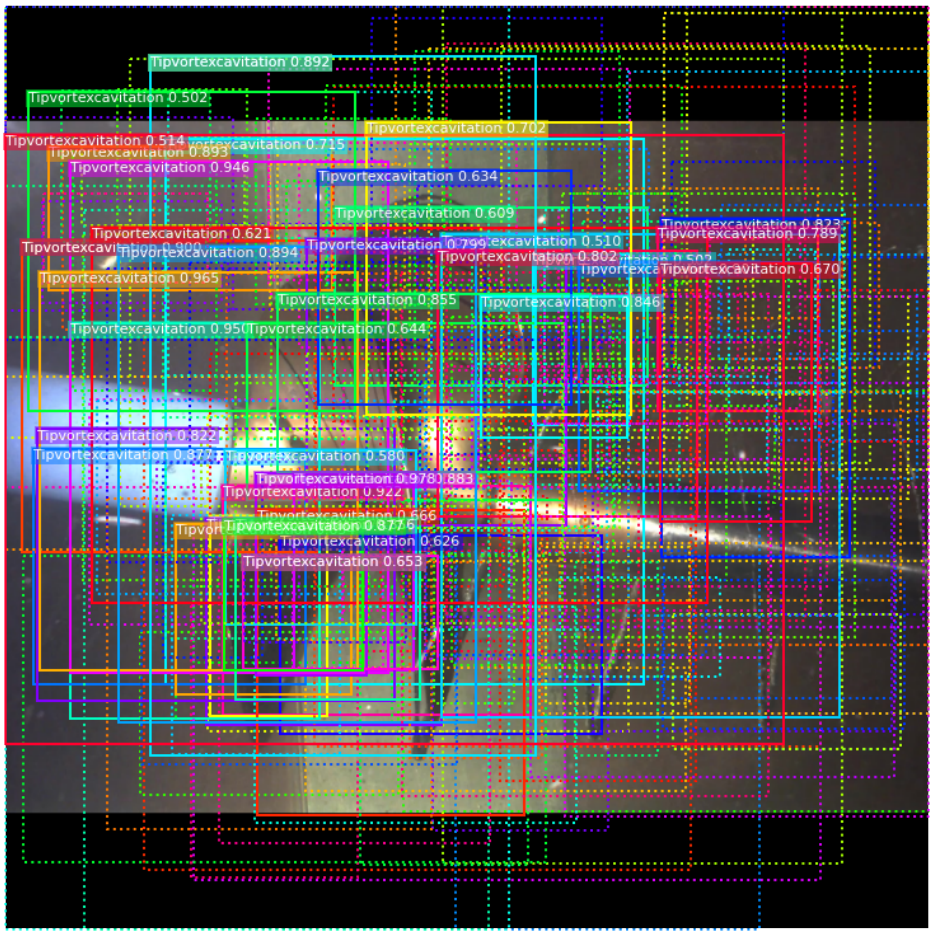

[20]:
```
# Vorschläge sind normalisierte Koordinaten. Skaliere die Vorschaläge in␣
 ↪Bildkoordinaten
h, w = config.IMAGE_SHAPE[:2]
proposals = np.around(mrcnn["proposals"][0] * np.array([h, w, h, w])).astype(np.
 ↪int32)

# Class ID, score, and mask per proposal
# Klasse ID, Ergebnis, und Maske pro Vorschlag
roi_class_ids = np.argmax(mrcnn["probs"][0], axis=1)
roi_scores = mrcnn["probs"][0, np.arange(roi_class_ids.shape[0]), roi_class_ids]
roi_class_names = np.array(dataset.class_names)[roi_class_ids]
roi_positive_ixs = np.where(roi_class_ids > 0)[0]
```

```
# Wie viele ROIs vs leere Zeilen?
print("{} Valid proposals out of {}".format(np.sum(np.any(proposals, axis=1)),␣
 ↪proposals.shape[0]))
print("{} Positive ROIs".format(len(roi_positive_ixs)))

# Class counts
# Klassen anzahl
print(list(zip(*np.unique(roi_class_names, return_counts=True))))
```

```
1000 Valid proposals out of 1000
231 Positive ROIs
[('BG', 769), ('Tipvortexcavitation', 231)]
```

[21]:
```
# Zeige eine zufällige Auswahl von Vorschlägen an.
# Als Hintergrund eingestufte Vorschläge sind gepunktet
# der Rest zeigt seine Klasse und sein Selbstvertrauen
# Bounding box und Wahrscheinlichkeit. Die Wahrscheinlichkeit ist dann␣
 ↪Tipvortexcavitation
# Die Wahrscheinlichkeit ist niedrig, dann handelt es sich um den Hintergrund.
# niedrig bedeutet, dass die Klasse nicht gefunden wird
limit = 200
ixs = np.random.randint(0, proposals.shape[0], limit)
captions = ["{} {:.3f}".format(dataset.class_names[c], s) if c > 0 else ""
            for c, s in zip(roi_class_ids[ixs], roi_scores[ixs])]
visualize.draw_boxes(image, boxes=proposals[ixs],
                     visibilities=np.where(roi_class_ids[ixs] > 0, 2, 1),
                     captions=captions, title="ROIs Before Refinement",
                     ax=get_ax())
```

ROIs Before Refinement

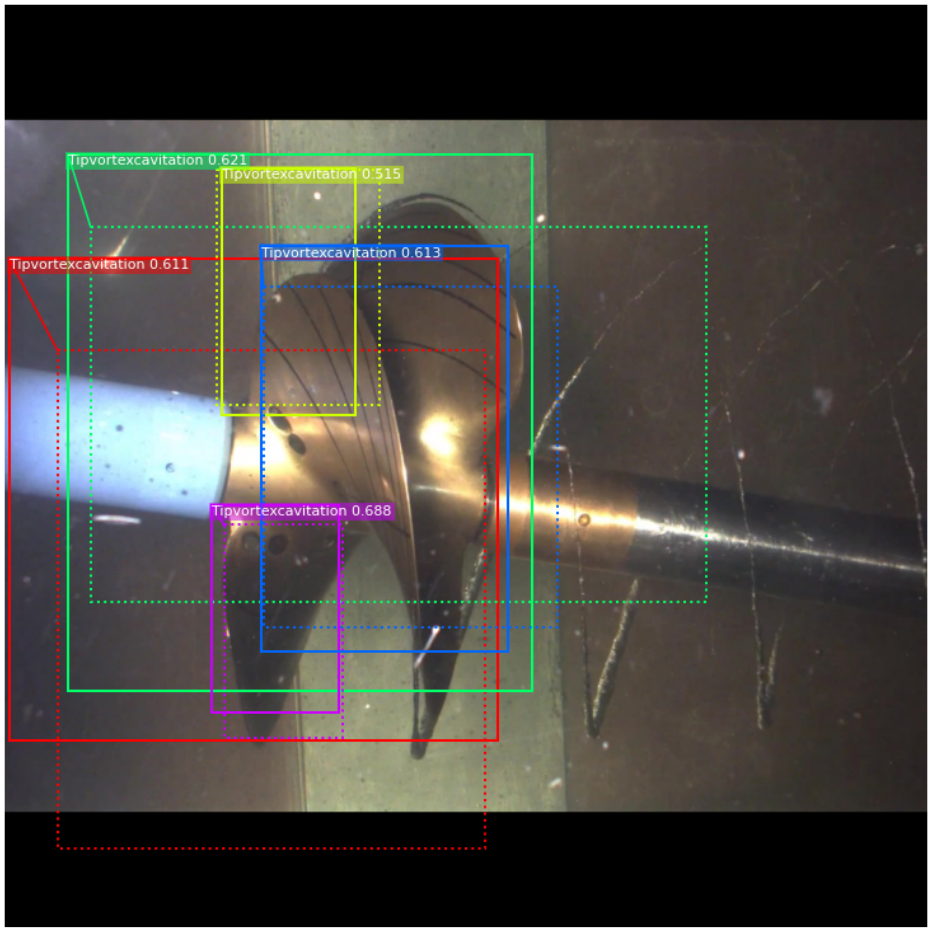



[22]:
```
# Klassenspezifische Verschiebungen des Begrenzungsrahmens.
roi_bbox_specific = mrcnn["deltas"][0, np.arange(proposals.shape[0]),␣
↪roi_class_ids]
log("roi_bbox_specific", roi_bbox_specific)


# Wenden Sie Begrenzungsrahmentransformationen an
# Shape: [N, (y1, x1, y2, x2)]
refined_proposals = utils.apply_box_deltas(
    proposals, roi_bbox_specific * config.BBOX_STD_DEV).astype(np.int32)
log("refined_proposals", refined_proposals)
```

```
# Zeige positive Vorschläge
# ids = np.arange(roi_boxes.shape[0])  # alle anzeigen
limit = 5
ids = np.random.randint(0, len(roi_positive_ixs), limit)  # Display random␣
↪sample
captions = ["{} {:.3f}".format(dataset.class_names[c], s) if c > 0 else ""
            for c, s in zip(roi_class_ids[roi_positive_ixs][ids],␣
↪roi_scores[roi_positive_ixs][ids])]
visualize.draw_boxes(image, boxes=proposals[roi_positive_ixs][ids],
                     refined_boxes=refined_proposals[roi_positive_ixs][ids],
                     visibilities=np.where(roi_class_ids[roi_positive_ixs][ids]␣
↪> 0, 1, 0),
                     captions=captions, title="ROIs After Refinement",
                     ax=get_ax())
```

```
roi_bbox_specific        shape: (1000, 4)             min:   -4.76556  max:
4.18748  float32
refined_proposals        shape: (1000, 4)             min: -372.00000  max:
1124.00000  int32
```

ROIs After Refinement




[23]:
```
# Als Hintergrund klassifizierte Kästchen entfernen
keep = np.where(roi_class_ids > 0)[0]
print("Keep {} detections:\n{}".format(keep.shape[0], keep))
```

```
Keep 231 detections:
[  1  18  29  31  35  36  44  57  63  64  65  73  74  82  87  88  91  93
  95  96  97 101 102 104 105 107 109 114 118 119 120 125 128 131 132 134
 136 137 143 147 148 165 166 168 178 181 184 191 193 196 203 206 210 212
 213 219 221 224 225 231 234 236 237 241 243 244 245 246 258 259 260 263
 267 271 273 276 277 280 281 284 285 291 297 311 313 315 321 323 324 332
 333 336 338 339 341 342 357 358 359 362 363 364 367 368 370 385 386 395
 397 400 418 423 426 428 433 436 437 443 445 449 450 451 453 454 458 476
 482 491 492 497 499 507 508 515 516 519 528 533 537 538 545 558 561 565
```

```
 567 574 575 581 582 587 589 594 605 606 608 610 613 621 628 635 640 643
 650 655 659 664 666 667 668 671 699 710 713 724 727 729 735 745 746 752
 755 760 778 781 784 790 793 796 804 809 810 812 813 827 828 832 833 836
 837 838 843 844 845 846 848 849 853 857 860 864 869 875 878 879 890 895
 902 909 918 924 933 953 959 961 965 968 974 979 986 989 997]
```

[24]:
```
# Entferne Erkennungen mit geringer Zuverlässigkeit(Konfidenz )
keep = np.intersect1d(keep, np.where(roi_scores >= config.
 ↪DETECTION_MIN_CONFIDENCE)[0])
print("Remove boxes below {} confidence. Keep {}:\n{}".format(
    config.DETECTION_MIN_CONFIDENCE, keep.shape[0], keep))
```

```
Remove boxes below 0.7 confidence. Keep 138:
[  1  18  35  44  57  63  64  73  74  87  96  97 102 107 109 119 128 134
 136 143 147 148 165 168 178 191 193 196 206 210 212 213 221 224 225 231
 234 236 241 243 244 245 246 258 259 260 263 267 271 273 276 277 280 297
 323 324 332 333 336 338 339 341 342 358 359 363 364 367 368 370 385 400
 418 433 436 450 451 453 454 482 515 516 519 537 538 565 574 575 581 582
 605 606 610 613 621 635 650 664 666 667 671 699 710 713 724 729 735 752
 755 760 778 784 790 804 810 813 832 833 836 843 844 845 848 853 864 869
 875 879 895 902 924 953 959 965 968 979 986 989]
```

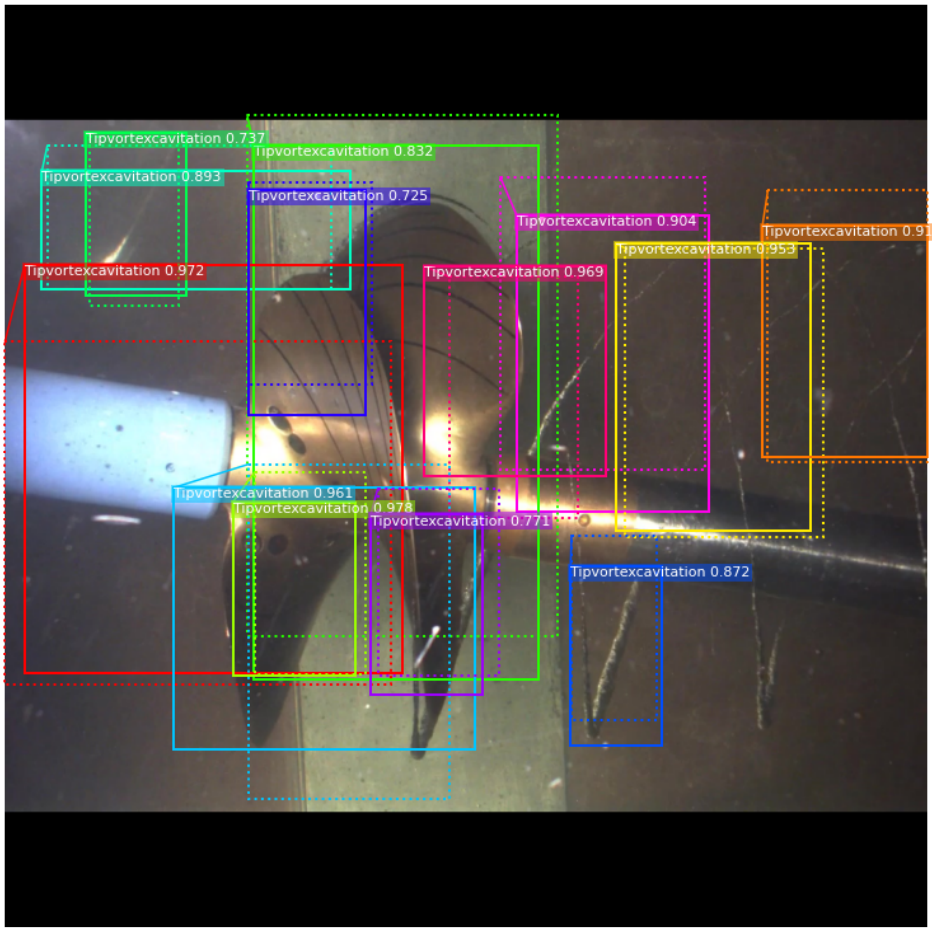
[25]:
```
# die Non-Max-Unterdrückung pro Klasse anwenden
pre_nms_boxes = refined_proposals[keep]
pre_nms_scores = roi_scores[keep]
pre_nms_class_ids = roi_class_ids[keep]

nms_keep = []
for class_id in np.unique(pre_nms_class_ids):
    # Pick detections of this class
    # Erkennungen dieser Klasse auswählen
    ixs = np.where(pre_nms_class_ids == class_id)[0]
    # NMS anwenden
    class_keep = utils.non_max_suppression(pre_nms_boxes[ixs],
                                           pre_nms_scores[ixs],
                                           config.DETECTION_NMS_THRESHOLD)
    # Map indicies
    # Kartenindizes
    class_keep = keep[ixs[class_keep]]
    nms_keep = np.union1d(nms_keep, class_keep)
    print("{:22}: {} -> {}".format(dataset.class_names[class_id][:20],
                                   keep[ixs], class_keep))

keep = np.intersect1d(keep, nms_keep).astype(np.int32)
print("\nKept after per-class NMS: {}\n{}".format(keep.shape[0], keep))
```

```
Tipvortexcavitation    : [  1  18  35  44  57  63  64  73  74  87  96  97 102 107
109 119 128 134
```

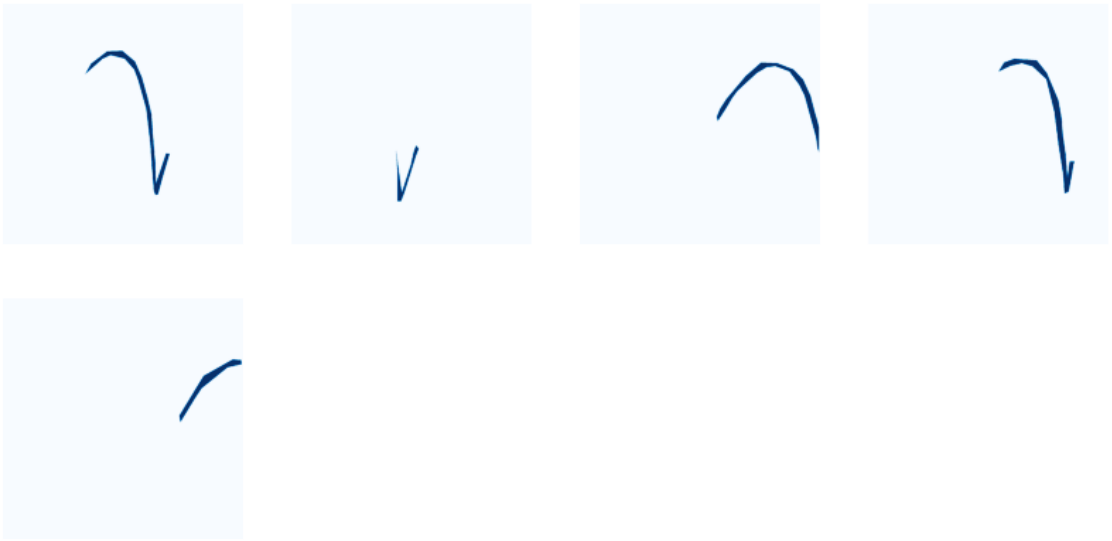
```
 136 143 147 148 165 168 178 191 193 196 206 210 212 213 221 224 225 231
 234 236 241 243 244 245 246 258 259 260 263 267 271 273 276 277 280 297
 323 324 332 333 336 338 339 341 342 358 359 363 364 367 368 370 385 400
 418 433 436 450 451 453 454 482 515 516 519 537 538 565 574 575 581 582
 605 606 610 613 621 635 650 664 666 667 671 699 710 713 724 729 735 752
 755 760 778 784 790 804 810 813 832 833 836 843 844 845 848 853 864 869
 875 879 895 902 924 953 959 965 968 979 986 989] -> [574 147 213 245 134 234
107   1 451 143 605 515 755]

Kept after per-class NMS: 13
[  1 107 134 143 147 213 234 245 451 515 574 605 755]
```

[26]:
```python
# Endgültige Erkennungen anzeigen
ixs = np.arange(len(keep))  # Display all
# ixs = np.random.randint(0, len(keep), 10)  # zufällige Probe anzeigen
captions = ["{} {:.3f}".format(dataset.class_names[c], s) if c > 0 else ""
            for c, s in zip(roi_class_ids[keep][ixs], roi_scores[keep][ixs])]
visualize.draw_boxes(
    image, boxes=proposals[keep][ixs],
    refined_boxes=refined_proposals[keep][ixs],
    visibilities=np.where(roi_class_ids[keep][ixs] > 0, 1, 0),
    captions=captions, title="Detections after NMS",
    ax=get_ax())
```

Detections after NMS




[27]:
```
display_images(np.transpose(gt_mask, [2, 0, 1]), cmap="Blues")
```



[28]:
```
# Die Vorhersagen Über die Maske erhalten
mrcnn = model.run_graph([image], [
    ("detections", model.keras_model.get_layer("mrcnn_detection").output),
    ("masks", model.keras_model.get_layer("mrcnn_mask").output),
])# Masken
det_boxes = utils.denorm_boxes(mrcnn["detections"][0, :, :4], image.shape[:2])
det_mask_specific = np.array([mrcnn["masks"][0, i, :, :, c]
                              for i, c in enumerate(det_class_ids)])
det_masks = np.array([utils.unmold_mask(m, det_boxes[i], image.shape)
                      for i, m in enumerate(det_mask_specific)])
log("det_mask_specific", det_mask_specific)
log("det_masks", det_masks)

# Erkennungen erhalten class IDs. Schneide die Zeros von dem Padding
det_class_ids = mrcnn['detections'][0, :, 4].astype(np.int32)
det_count = np.where(det_class_ids == 0)[0][0]
det_class_ids = det_class_ids[:det_count]

print("{} detections: {}".format(
    det_count, np.array(dataset.class_names)[det_class_ids]))
```

```
detections               shape: (1, 100, 6)           min:    0.00000  max:
1.00000  float32
masks                    shape: (1, 100, 28, 28, 2)   min:    0.00000  max:
0.89839  float32
det_mask_specific        shape: (14, 28, 28)          min:    0.00000  max:
0.89839  float32
det_masks                shape: (14, 1024, 1024)      min:    0.00000  max:
1.00000  bool
```

```
14 detections: ['Tipvortexcavitation' 'Tipvortexcavitation'
'Tipvortexcavitation'
 'Tipvortexcavitation' 'Tipvortexcavitation' 'Tipvortexcavitation'
 'Tipvortexcavitation' 'Tipvortexcavitation' 'Tipvortexcavitation'
 'Tipvortexcavitation' 'Tipvortexcavitation' 'Tipvortexcavitation'
 'Tipvortexcavitation' 'Tipvortexcavitation']
```

[29]:
```
# Masken
det_boxes = utils.denorm_boxes(mrcnn["detections"][0, :, :4], image.shape[:2])
det_mask_specific = np.array([mrcnn["masks"][0, i, :, :, c]
                              for i, c in enumerate(det_class_ids)])
det_masks = np.array([utils.unmold_mask(m, det_boxes[i], image.shape)
                      for i, m in enumerate(det_mask_specific)])
log("det_mask_specific", det_mask_specific)
log("det_masks", det_masks)
```

```
det_mask_specific        shape: (14, 28, 28)          min:    0.00000  max:
0.89839  float32
det_masks                shape: (14, 1024, 1024)      min:    0.00000  max:
1.00000  bool
```

[30]:
```
display_images(det_mask_specific[:4] * 255, cmap="Blues", interpolation="none")
```

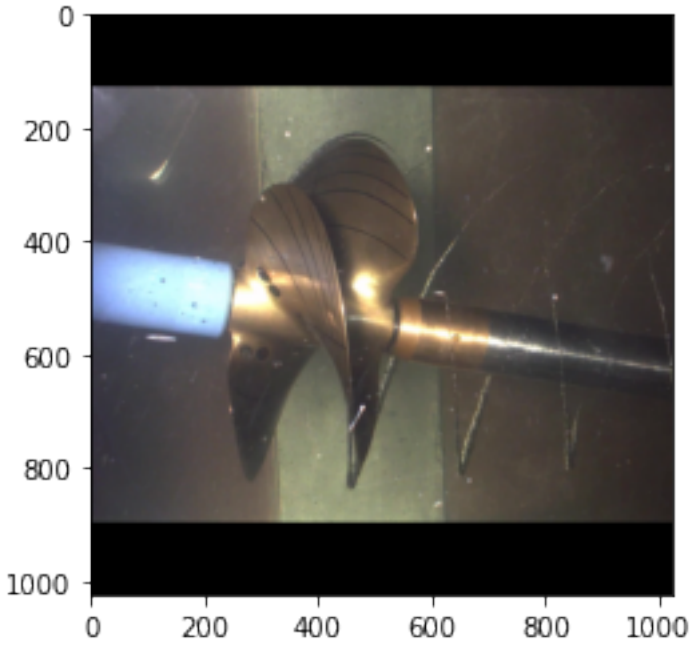
[31]:
```
display_images(det_masks[:4] * 255, cmap="Blues", interpolation="none")
```

[32]:
```
# die aktivierierungsfunktionen für die layers erhalten
activations = model.run_graph([image], [
```

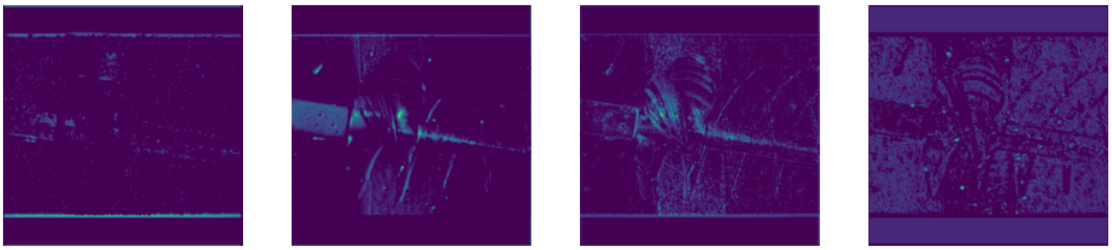
```
    ("input_image",         tf.identity(model.keras_model.
↪get_layer("input_image").output)),
    ("res2c_out",           model.keras_model.get_layer("res2c_out").output),
    ("res3c_out",           model.keras_model.get_layer("res3c_out").output),
    ("res4w_out",           model.keras_model.get_layer("res4w_out").output),  #␣
↪for resnet101
    ("rpn_bbox",            model.keras_model.get_layer("rpn_bbox").output),
    ("roi",                 model.keras_model.get_layer("ROI").output),
])
```

```
input_image              shape: (1, 1024, 1024, 3)    min: -123.70000  max:
151.10001  float32
res2c_out                shape: (1, 256, 256, 256)    min:    0.00000  max:
28.32854  float32
res3c_out                shape: (1, 128, 128, 512)    min:    0.00000  max:
35.13713  float32
res4w_out                shape: (1, 64, 64, 1024)     min:    0.00000  max:
52.77522  float32
rpn_bbox                 shape: (1, 261888, 4)        min:  -80.50720  max:
488.81476  float32
roi                      shape: (1, 1000, 4)          min:    0.00000  max:
1.00000  float32
```

[33]:
```
# Eingangsbild (normalisiert)
_ = plt.imshow(modellib.unmold_image(activations["input_image"][0],config))
```



[34]:
```
# Backbone Eigenschaftskarte
display_images(np.transpose(activations["res2c_out"][0,:,:,:4], [2, 0, 1]),␣
↪cols=4)
```



[ ]:
```

```