



THEMA :

DETEKTION UND KLASSEFIKATION VON KAVITATIONERSCHEINUNGEN AM PROPELLER MIT METHODEN KÜNSTLICHER INTELLIGENZ

NAME:	ALYAN
VORNAME:	MAJD
MATRIKEL-NR.:	217203553
STUDIENGANG:	INFORMATIONSTECHNIK / TECHNISCHE INFORMATIK
DOZENTIN:	DR.-ING. ERIC EBERT PROF.DR.-ING. NILS DAMASCHKE
INSTITUT:	INSITITUT FÜR ALLGEMEINE ELEKTROTECHNIK— OPTO-ELEKTRONIK UND PHOTONISCHE SYSTEME
FAKULTÄT:	FAKULITÄT FÜR INFORMATIK UND ELEKTROTECHNIK
ABGABEDATUM:	04.04.2022

Bachelorarbeit

für Herrn **Majd Alyan** (Matrikel-Nr. 217203553)

Thema: **Detektion und Klassifikation von Kavitationserscheinungen am Propeller mit Methoden künstlicher Intelligenz**

Aufgabenstellung:

In der Propellerforschung, speziell der Kavitationsuntersuchung ist es bisher ein Problem während einer Messung zu spezifizieren, welche Kavitationsart gerade vorliegt. Als Ausgangspunkt der Arbeit sind zu Verfügung gestellten Beispielabbildungen der Kavitationserscheinungen aus Universitätsprojekten vorhanden. Weiterhin gibt es eine Übersicht der ITTC, wie die Kavitationsarten zu bewerten sind. Darüber hinaus existiert eine Software die trainierte Mask R-CNNs auf Bilder anwenden kann. Mit dieser Arbeit soll versucht werden, ein vorhandenes KI-Modell zu trainieren was die verschiedenen Kavitationsarten unterscheiden und im Bild als Umriss/Maske markieren kann. Das an den Beispieldaten trainierte Modell soll im Anschluss auf reale Daten angewendet werden und danach bewertet werden, wie gut die Erkennungsleistung ist.

Folgende Punkte gilt es zu bearbeiten:

- Literaturrecherche & Einarbeitung zum Thema für den Stand der Technik:
 - Objekterkennung & -segmentierung mittels künstlicher Intelligenz
 - Kavitationsuntersuchung an Propellern
 - Andere automatische Ansätze zur Problemlösung
- Evaluation der möglichen Ansätze zu Segmentierung der einzelnen Kavitationsarten
- Umsetzen des Trainingsprozesses anhand der Beispielabbildungen und von ITTC Informationen/Rechercheergebnissen
- Anwendung des trainierten Modells auf dem Modell unbekannten Beispieldatensätzen
- Optimierung des vorhandenen C# Codes zur Anwendung des Mask R-CNNs was zur Problemlösung im Moment favorisiert wird.
- Beschaffung und Aufsetzen der notwendigen Hard- und Softwareplattformen zur Erledigung der Aufgabenstellung

Bearbeitungszeit: 20 Wochen

Betreuer: Dr.-Ing. E. Ebert

Tag der Ausgabe: 15.11.2021

Prof. Dr.-Ing. N. Damaschke

Tag der Abgabe: 04.04.2022

Inhaltsverzeichnis

Inhaltsverzeichnis

1 Danksagung	5
2 Einleitung	6
3 Stand der Technik	8
3.1 Objekterkennung und -segmentierung mittels künstlicher Intelligenz	8
3.1.1 Künstliche Neuronale Netzwerke(KNN)	8
3.1.2 Faltende neuronale Netze(FNN)	10
3.1.3 Faster R-CNN	15
3.1.4 Mask R-CNN	17
3.2 Kavitationsuntersuchung an Propellern	18
4 Umsetzung	19
4.1 Datensätze	20
4.1.1 Ballondatensatz	20
4.1.2 Tipvortexcavitationdatensatz	20
4.1.3 Sheetcavitationdatensatz	22
4.2 Modell	23
4.3 Trainieren und Validieren	24
4.3.1 Das Trainieren von dem Ballondatensatz	24
4.3.2 Das Trainieren von dem Tipvortexcavitationdatensatz	25
5 Testen und Ergebnisse	29
6 Zusammenfassung	32
7 Literaturverzeichnis	33
8 Tabellenverzeichnis	36
9 Abbildungsverzeichnis	37
10 Anhang	38
10.1 VGG Image Annotator(Annotation tool)	38
10.2 Umgebungseinstellung von Anaconda	42
10.3 Mask R-CNN Konfiguration	43
10.4 Tip vortex cavitation code	45
10.5 Visualisierungscode der Daten und des Modells:	55
10.5.1 Daten in jupyterlab	55
10.5.2 Modell in jupyterlab(resnet50)	73
10.5.3 resnet101	115
11 Eidesstattliche Versicherung	139

Abkürzungsverzeichnis

KNN: Künstliche Neuronale Netze

CNN: Convolutional Neural Network (auf Englisch)

FNN: Faltende Neuronale Netze(auf Deutsch)

Fast R-CNN: Fast Region-based Convolutional Neural Network

Faster R-CNN: Faster Region-based Convolutional Neural Network

FCN: Fully Convolutional Network

Mask R-CNN: Mask Region based convolutionl neural Neural Network

VIA: VGG Image Annotator

ILSVRC: ImageNet Large Scale Visual Recognition Challenge

1 Danksagung

An erste Stelle möchte ich meinem Betreuer, Doktoringenieur Eric Ebert, danken, der mich unterstützt hat und mich richtungsweisend begleitet hat. Ich danke meiner Familie für den motivierenden Beistand während meines gesamten Studiums. Ich danke der Leser und Leserinnen, die meine Arbeit lesen.

2 Einleitung

Kavitation ist ein Problem in der Propellerforschung und beschreibt die Bildung und das Zusammenfallen der Blasen, dabei entstehen Hohlräume und werden Stücke von dem Propeller gerissen. Dadurch wird der Propeller zerstört [5, 28]. Daher ist Kavitation in der Hydraulik unerwünscht, da sie nicht nur das Material beschädigt. Sondern auch den Wirkungsgrad reduziert [4]. Kavitation geschieht in der Pumpe und am Propeller. Beim Implodieren von Blasen entstehen Geräusche. Diese Geräusche hören sich wie Knall an [16]. Aufgrund der Geräusche kann das U-Boot geortet werden [6]. Es gibt mehrere Arten von Kavitationen, die sind schwer erkennbar und die folgenden Arten sind: Einzelblasenkavitation (Bubble Kavitation), Schichtkavitation (Sheet Kavitation), Wolkenkavitation (Cloud Kavitation), Wirbelkavitation (Vortex Kavitation). Außerdem gibt es mehrere Orte, die unterschieden werden: Flügel spitzen(Tip Vortex Kavitation), Nabe (Hub Vortex Kavitation)[5]. In Figure (1) wird gezeigt, welche Arten von Kavitationen es gibt. Die Bilder, die in der Arbeit sind, wurden in der Schiffbau-Versuchsanstalt von Potsdam aufgenommen. Die verschiedenen Arten und Formen von Kavitationen sind nicht so leicht(schwer)zu erkennen und klassifizieren. In der künstlichen Intelligenz gibt es verschiedenen Arten oder Modelle von Algorithmen und Ansätze, die mit Bildern arbeiten können. Die Lösung, die in dieser Arbeit verfolgt wird, ist Mask R-CNN für Instanzsegmentierung. Es wird auch auf andere Modelle im Stand der Technik eingegangen, sowie in der Tabelle 1 vorgestellt wird. Alle diese Modelle sind Grundlagen, um den Algorithmus von dem Mask R-CNN zu verstehen. Mask R-CNN ist Mask Region based Convolutional Neural Networks und ist eine Erweiterung von Faster R-CNN. Faster R-CNN macht ein Rechteck um das Objekt im Bild. Mit diesen Rechtecken wird die Lage und die Position von dem Objekt bestimmt. Das heißt, es kann sein, dass mehrere verschiedene Objekte im Bild sind und die werden alle mit einem Rechteck begrenzt, aber mit unterschiedlichen Namen. Mask R-CNN macht noch einen Schritt nach vorne und gibt unterschiedliche Farben für die Objekte, die die verschiedene Klassen besitzen[12, 14].

Framework oder System	Anwendungsbereich	Referenzen
KNN	Regressionsproblem und Klassifikationsproblem	[11]
FNN	Klassifikationsproblem	[11], [31], [8]
R-CNN	Objekterkennung	[13]
Fast R-CNN	Objekterkennung	[12]
Faster R-CNN	Objekterkennung	[26]
FCN	Semantische Segmentierung	[22]
Mask R-CNN	Instanzsegmentierung	[14]

Tabelle 1: Bekannte Algorithmen aus der Künstlichen Intelligenz für Bildverarbeitung

Objekterkennung, Semantische Segmentierung und Instanzsegmentierung sind Aufgaben, die zur Bildverarbeitung gehören. Es gibt in der künstlichen Intelligenz Algorithmen, die auf die faltende neuronale Netzwerke basieren. Die Faltenden neuronale Netzwerke werden für Klassifikation von Bildern in Mnist, Mnist Fashion, Cifar-10, Cifar-100, Imagenet [17] und Coco [21] datensätze verwendet. Faltende neuronale Netzwerke haben kleine Fehlerrate im vergleich zur künstlichen neuronalen Netwerken

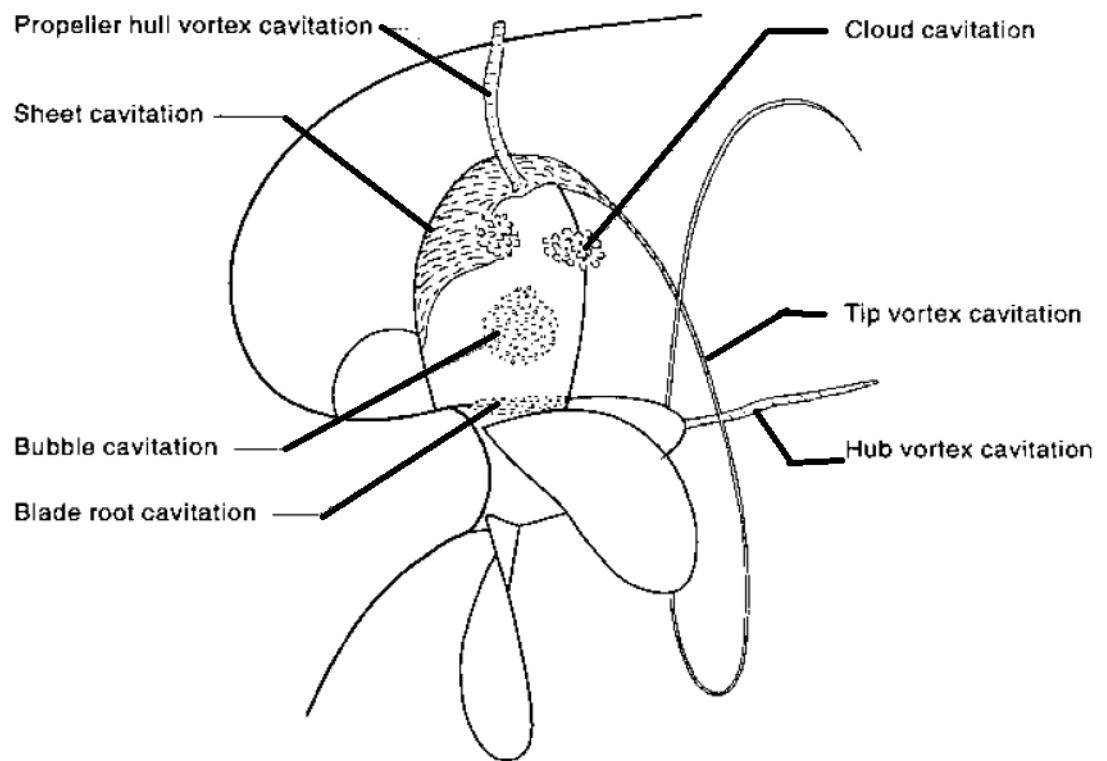


Abbildung 1: Kavitationsarten

1. Propeller hull vortex cavitation
2. Sheet cavitation
3. Bubble cavitation
4. Blade root cavitation
5. Cloud cavitation
6. Tip vortex cavitation
7. Hub vortex cavitation

[1]

3 Stand der Technik

3.1 Objekterkennung und -segmentierung mittels künstlicher Intelligenz

3.1.1 Künstliche Neuronale Netzwerke(KNN)

KNN können einfach als Graph verstanden werden. Neuronen sind wie Knoten des Graphes und die Verbindung als Kanten im Sinne von Graphentheorie. Aber haben auch Ähnlichkeit mit der biologischen Neuronen im Gehirn. Ein Netz besteht aus einer Menge von Knoten. Jeder Knoten hat seine Eingaben, seine Gewichte, seine Ausgaben und eine Aktivierungsfunktion oder einen Schwellenwert [18]. In dieser Arbeit [23] wird der Schwellenwert als Bias beschrieben. Ein neuronales Netz besteht aus Schichten. Jede Schicht besteht aus Neuronen. Die erste Schicht sind die Eingaben, da werden die Daten eingegeben und wird auch darin spezifiziert, welches Shape die Daten haben. Die Anzahl der Neuronen in der Ausgabeschicht ist gleich die angegebene Klassen, die das Netz vorhersagen muss. Die Ausgabe ist eine Zahl. Man kann die Ausgabe beschränken durch eine Aktivierungsfunktion in der Ausgabeschicht. Die Abbildung(2) zeigt, wie ein neuronales Netz für ein Klassifikationsproblem aufgebaut ist. Hier das Netz lernt zu unterscheiden zwischen orangem und blauem Punkt. Das Netz hat zwei Eingaben(x_1, x_2) und 3 verborgene Schichten. Die erste verborgene Schicht hat 8 Neuronen, die zweite verborgene Schicht hat auch 8 Neuronen und die dritte Schicht hat 2 Neurons. Die Aktivierungsfunktion ist Rectifier-Funktion($\text{ReLU}(x) = \max(0, x)$). Neuronale Netze werden für zwei Arten von Problemen genutzt: Die erste Art ist Regression und die zweite ist Klassifikation. Es gibt Binary Klassifikation und Multiclass Klassifikation. Binary Klassifikation ist für zwei Objekte zum Beispiel: so wie im Figure (1), ob ein Punkt Blau oder Orange ist. Multiclass Klassifikation werden die Bilder in Klassen eingeteilt. Zum Beispiel: ob ein Bild Pizza oder Steak oder Sushi ist [7].

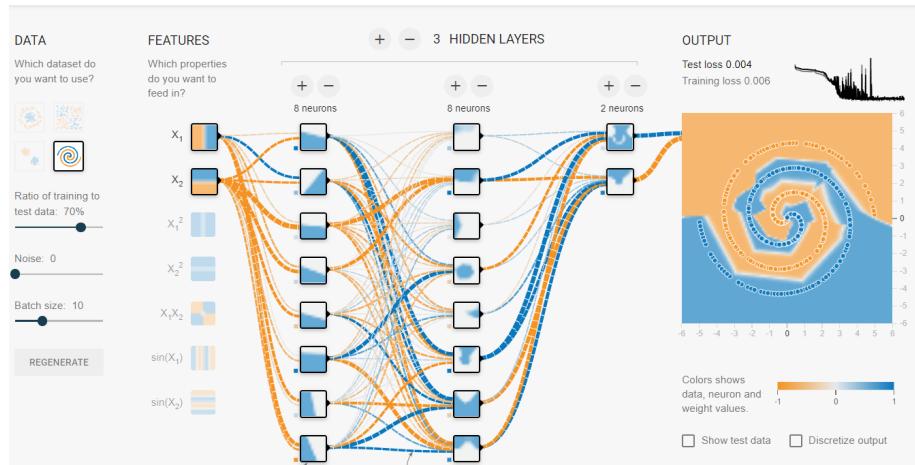


Abbildung 2: Ein KNN in A Neural Network playground für ein Klassifikationsproblem

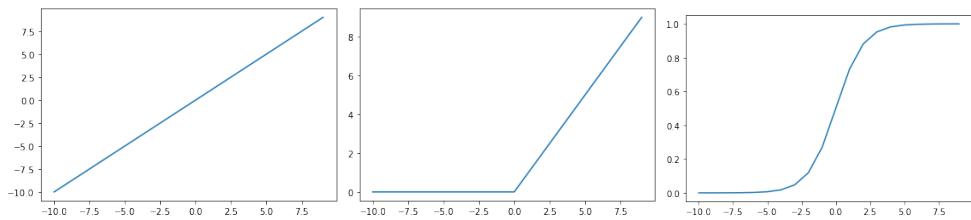


Abbildung 3: links: Lineare Funktion, Mitte: Rectifier Funktion, rechts: sigmoid Funktion

Die meist verbreite Aktivierungsfunktionen in Machine Learning und Deep Learning. Diese Aktivierungsfunktionen werden in künstliche neuronale Netzwerke und faltende neuronale Netzwerke verwendet [11]

3.1.2 Faltende neuronale Netze(FNN)

Faltende neuronale Netzwerke(FNN) sind nach dem visuellen Cortex nachgebildet, Netzwerkarchitekturen aus dem Deep Learning und werden für Objekterkennung benutzt. Deep Learning ist Teilbereich von dem Machine Learning. Faltende neuronale Netzwerke wurden in Systeme für Bildersuche(Google Lens), selbstfahrende Autos(Tesla) und automatische Klassifikation von Videos(YouTube) und Sprachverarbeitung(Siri, Alexa) verwendet. Besondere weise bei FNN sind alle Schichten des Netzwerks nicht miteinander verbunden, das bedeutet vergleichsweise zu KNN weniger Gewichte. Bei KNN sind die Neuronen alle miteinander verbunden. Das Problem ist die vollständigen Verbindungen zwischen den Neuronen. Zum Beispiel enthält ein KNN für einen Datensatz mit 100 Bilder mit 10000 Pixel bei der ersten Schicht 1000 Neuronen und bei der zweiten Schicht 10000. Das bedeutet, zwischen erster Schicht und zweiter Schicht gibt es 10 Millionen Verbindungen. FNN verwenden zur Lösung dieses Problem teilweise verbundene Schichten. FNN besteht aus mehreren Schichten von Convolutional Layer, Pooling Layer, Flatten Layer und Fully Connected Layer [25]. Es gibt mehrere Architekturen von CNNs so wie das LeNet-5[19], AlexNet[17], GoogLeNet[29] und ResNet[15]. Die erste Schicht bei einem Faltenenden neuronalen Netz erkennt Linien und Kanten. Die weitere Schichten bauen auf die einfache Merkmale auf und machen komplexe Strukturen. FNN können in Pythoncode mithilfe der Bibliotheken TensorFlow und PyTorch geschrieben werden. Für das Schreiben von FNN Modellen in Browser gibt es Google colab mit Grafikkarte support und TPU(Tensor Processing Unit) für schnellere Ausführung. Anaconda ist auch eine Umgebung für das Erstellen, Trainieren, Evaluieren oder Auswerten und Verbessern durch Experimentieren, Speichern und Testen auf neue Daten.

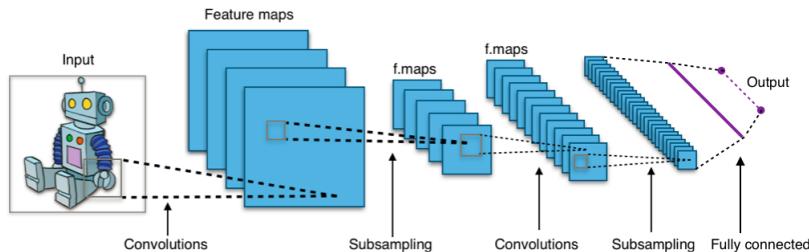


Abbildung 4: Faltende Neuronale Netze

Faltung Schicht: Die Faltung ist definiert als:

$$\int_{-\infty}^{\infty} s(\tau)g(t-\tau)d\tau = s(t) * g(t)$$

[27]

im Tensorflowcode: `tf.keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid' or 'same', activation=relu)`. Die strides sind die Verschiebungen τ

Bild:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 1 & 0 \\ 0 & 5 & 1 & 1 & 3 & 2 & 0 \\ 0 & 2 & 3 & 3 & 1 & 1 & 0 \\ 0 & 1 & 2 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Filter:

$$\begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Ergebnis : Faltung des Bildes mit dem Filter mit einer Verschiebung von 2. jedes Mal wird die Faltung durchgeführt wird eine Matrix(3,3) herausgenommen und jedes Element mit dem anderen multipliziert und dann alle zusammenaddiert. die Summe wird in das Ergebnis eingetragen. Beispiel: Bei dem $0*1 + 0*0 + 0*-1 + 0*0 + 1*0 + 2*1 + 0*0 + 5*0 + 1*1 = 3$ und dann wird der Filter zweimal geschoben. Das Bild ist ein 5 x 5 Matrix durch Padding oder auffüllung von Nullen wird auf 7 x 7 erweitert[9].

$$\begin{pmatrix} 3 & 7 & 0 \\ 4 & 0 & 3 \\ -1 & 5 & 1 \end{pmatrix}$$

Pooling Schicht: Im Featuremap stecken hier die wichtigste Eigenschaften, die in den Bildern durch die Faltung gefunden wurden. Durch das Pooling wird die Größe des Featuremaps reduziert, dadurch sinkt der Rechenaufwand und der Speicherbedarf und wird es am Ende für die Fully-connected Layer einfacher Muster zu finden. Pooling schützt gegen Überanpassung [10].

Max-Pooling: Im Max-Pooling verwendet der Layer eine Maximumsfunktion. Diese Maximumsfunktion berechnet den größten Wert aus der Matrix. In einigen Fällen geht durch das Max-Pooling Information verloren. Bei dem Beispiel unten wird ein MaxPool die Größe (2,2) und mit Verschiebung von 2. Es wird ausgewählt zwischen 11 und 9. Aber was genommen wird, ist 11. Beim Max-Pooling werden Werte maximiert. Es können zwei hohe Werte gegeben, was beide relevant für das Netz sind und es wird durch das Max-Pooling ein Wert genommen.[32].

$$\begin{pmatrix} 1 & 5 & 4 & 5 \\ 7 & 10 & 9 & 5 \\ 3 & 4 & 6 & 7 \\ 11 & 9 & 8 & 9 \end{pmatrix} \xrightarrow{\quad} \begin{pmatrix} \boxed{11} & \boxed{9} \end{pmatrix}$$

$$\max(1,5,7,10) = 10, \max(4,5,9,5)=9, \max(3,4,11,9)=11, \max(6,7,8,9) = 9$$

$$\begin{pmatrix} 10 & 9 \\ 11 & 9 \end{pmatrix}$$

Average-Pooling: Die Mittelwertfunktion berechnet die Summe der Matrixwerte durch die Anzahl.

$$\frac{2+2+4+8}{4} = \frac{16}{4} = 4, \quad \frac{16+8+2+10}{4} = \frac{36}{4} = 9, \quad \frac{1+2+4+1}{4} = 2, \quad \frac{20+20+20+4}{4} = \frac{64}{4} = 16$$

$$\left(\begin{array}{cccc} 2 & 2 & 16 & 8 \\ 4 & 8 & 2 & 10 \\ 1 & 2 & 20 & 20 \\ 4 & 1 & 20 & 4 \end{array} \right) \quad \left(\begin{array}{c|c} \hline & \end{array} \right) \quad \left(\begin{array}{cc} 4 & 9 \\ 2 & 16 \end{array} \right)$$

[23]

Flatten Schicht: Die Merkmalkarte(das Feature map) wird in einen dimensionalen Vektor geschrieben, damit die Merkmalkarte in das neuronalen Netz oder die Fully-connected Layers(schichten) verarbeitet oder um Muster in den Daten zu finden. Die Ausgabe von dem Flattenlayer ist ein Vektor.

Vollständig verbundenes neuronales Netzwerk: Das Netzwerk besteht aus einem Neuron oder mehreren Schichten von Neuronen. Sie dienen zur Klassifizierung von Merkmalen. Diese Merkmale sind sehr wichtig, um jedem Objekt eine Klasse zuordnen zu können. Die vorherige Schicht ist der Flatten Schicht. Das heißt, dass das vollständig verbundene neuronale Netzwerk die Eingaben in Form von einem Vektor bekommt. [32].

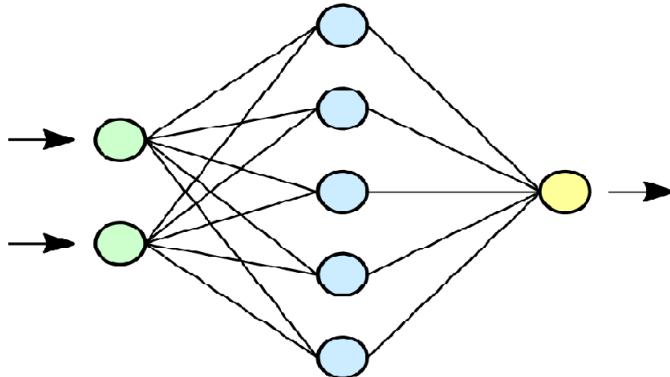


Abbildung 5: Vollständig verbundenes neuronales Netzwerk

Architekturen von CNNs

LetNet-5: Die LetNet-5- Architktur wird in diesem Paper [19] erwähnt. Die LetNet-5-Architektur wird in diesem Paper erwähnt und wurde im Jahr 1998 erstellt. In Figure 5 wird die LetNet-Architektur nachgebildet und auf den Cifar 10 datensatz trainiert. Die Genauigkeit(accuracy) liegt bei 90 % und die Fehlerquote bei 20 %. In Figure 5 besteht die LetNet-5-Architektur aus drei Convlution Layern, 2 Average-Poolaing-Layern, Flattenlayer und zwei Dense-layern.

Schicht	Art	Maps	Größe	Kernel-Größe	Schrittweite	Aktivierung
Out	Vollst. verbunden	-	10	-	-	Softmax
F6	Vollst. verbunden	-	84	-	-	tanh
C5	Faltung	120	1x1	5x5	1	tanh
S4	Avg-Pooling	16	5x5	2x2	2	tanh
C3	Faltung	16	10x10	5x5	1	tanh
S2	Avg-Pooling	6	14x14	2x2	2	tanh
C1	Faltung	6	28x28	5x5	1	tanh
In	Eingabe	1	28x28	-	-	-

Tabelle 2: LeNet-5-Architektur

Layer (type)	Output Shape	Param #
<hr/>		
C1 (Conv2D)	(None, 28, 28, 6)	456
S2 (AveragePooling2D)	(None, 14, 14, 6)	0
C3 (Conv2D)	(None, 10, 10, 16)	2416
S4 (AveragePooling2D)	(None, 5, 5, 16)	0
C5 (Conv2D)	(None, 1, 1, 120)	48120
flatten_42 (Flatten)	(None, 120)	0
F6 (Dense)	(None, 84)	10164
Out (Dense)	(None, 10)	850
<hr/>		
Total params: 62,006		
Trainable params: 62,006		
Non-trainable params: 0		

Abbildung 6: LeNet-5 Modell

AlexNet: AlexNet ist ein Modell in deep learning, was so ähnlich aussieht wie LetNet, aber AlexNet ist tiefer, hat mehr Schichten als LetNet. Die Faltungsschichten sind bei AlexNet übereinander gestapelt, nicht so wie LeNet-5. bei LeNet-5 zwischen zwei Faltungsschichten gibt es ein Avg-pooling. Bei AlexNet ist das Avg-Pooling mit Max-Pooling getauscht. In ILSVRC 2012 gewinnt AlexNet den Wettbewerb mit einer Fehlerquote von 17%. Um die Überanpassung(overfitting) zu reduzieren, benutzen die Autoren von AlexNet zwei Regularisierungstechniken. Bei erster Technik haben die Autoren eine Dropoutquote von 50% während des Trainings auf die Ausgänge von den zwei Schichten F9 und F10 angewendet. Bei zweiter Technik haben die Autoren data Augmentation auf die Bilder angewendet. Data Augmentation definiert als Änderung in Bildern zu machen. Änderung bedeutet Verschieben der Trainingsbilder, die Bilder horizontal oder vertikal drehen und Veränderung der Lage des Lichtes.

Schicht	Art	Maps	Größe	Kernel-Größe	Schrittweite	Padding	Aktivierung
Out	Vollst. verbunden	-	1000	-	-	-	Softmax
F10	Vollst. verbunden	-	4096	-	-	-	ReLU
F9	Vollst. verbunden	-	4096	-	-	-	ReLU
S8	Max pooling	256	6 x 6	3 x 3	2	VALID	-
C7	Faltung	256	13x13	3x3	1	SAME	ReLU
C6	Faltung	384	13x13	3x3	1	SAME	ReLU
C5	Faltung	384	13x13	3x3	1	SAME	RELU
S4	Max-Pooling	256	13x13	3x3	2	VALID	-
C3	Faltung	256	27x27	5x5	1	SAME	RELU
S2	Max-Pooling	96	27x27	3x3	2	VALID	-
C1	Faltung	96	55x55	11x11	4	SAME	RELU
In	Eingabe	3	224x224	-	-	-	-

[11]

Tabelle 3: AlexNet-Architektur

3.1.3 Faster R-CNN

Faster R-CNN ist Faster Region-based Convolutional Neural Networks und wird in diesem Paper [26] vorgestellt. Der Grund für die Entwicklung von Faster R-CNN ist, dass das Fast R-CNN mit Selective Search langsam war. Stattdessen arbeitet Faster R-CNN mit Region Proposal Netzwerk(RPN). RPN schlägt Region of Interest (ROI). Diese ROIs haben zwei Parameter: die Koordinaten der Box und eine Wahrscheinlichkeit oder Score, was angibt, mit welcher Wahrscheinlichkeit diese Box ein Objekt enthält. Faster R-CNN hat zwei Ausgaben. Die erste Ausgabe ist ein Label für das Objekt. Die zweite Ausgabe ist eine Box. Diese Box begrenzt ein Objekt. Jedes Objekt im Bild erhält eine Box und eine bestimmte Klasse. Faster R-CNN ordnet jedem Objekt eine Klasse und eine Box zu. Faster R-CNN enthält zwei Stufen. Die erste Stufe wird Region Propsoal Network(RPN) genannt. Das Region Proposal Network schlägt boundingboxes oder Boxen für Objekt vor. Der zweite Schritt im Faster R-CNN ist Featureextraction(Merkmalextraktion). Die Merkmalsextraktion benutzt Roipoolmethode. Der zweite Schritt nimmt Eigenschaften mithilfe der Roipoolmethode von jeder Box oder Kiste heraus oder jeder vorgeschlagenen Box heraus. In Figure 10 enthält Faster R-CNN mehrere Faltungsschichten und Poolingschichten. Diese Faltungsschichten und Poolingschichten nehmen Eigenschaften heraus und speichern diese Eigenschaften in eine Merkmalkarte(Map). Diese Merkmalkarte ist eine Menge von Tensoren oder Matrixen, was sie durch Faltung und Pooling erzeugt wurden. Das Faster R-CNN hat zwei Ausgaben, die Klassenbezeichnung des Objekts und die Koordinaten der ROI, in welcher sich das Objekt befindet [24], [14].

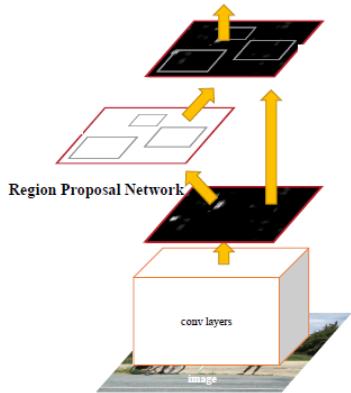


Abbildung 7: Faster R-CNN

In Abbildung 7 ist ein Bild mit Faltungsschichten, Region Proposal Network und classifier. Diese Faltungsschichten extrahieren Merkmalenkarten(feature maps) heraus. Ein Region Proposal Network hat zwei Schritte. Ein Schritt ist Erzeuge Kisten oder Boxen. Zweiter Schritt berechnet Intersection over Union (IOU). Klassifikator nimmt die Regionen von dem Bild oder die Boxen und gibt jedes Objekt ein bestimmtes Label oder bestimmten Name. Faster R-CNN enthält faltende Schichten, Region Propsal Network und RoI pooling. Die faltende Schichten extrahieren Merkmale aus dem Bild. Diese Merkmale helfen dem Region Proposal Network(RPN), die Objekte im Bild aufwendig zu machen. RPN enthält Non Maximum Suppression. Das Non Maximum Suppression nimmt die Box mit der höchsten Wahrscheinlichkeit heraus [26].

3.1.4 Mask R-CNN

Mask R-CNN ist Mask Region Based Convolutional Neural Networks. Dieses System wird entwickelt für Instanzsegmentierung und wird in diesem Paper [14] vorgestellt. Mask R-CNN wird verwendet in mehreren Arbeiten zum Beispiel: so wie in dieser Arbeit [24] wird das Mask R-CNN auf medizinische Bilder von den Augen angewendet, in anderer Arbeit wird versucht mit dem Mask R-CNN ein System aufzubauen, um Menschen mit Demenz zu helfen [3]. In anderer Anwendung wird das Mask R-CNN zur Erkennung von Werbung am Straßenrand verwendet[30]. Mask R-CNN wird auf verschiedene Probleme in unterschiedlichen Bereichen benutzt. Mask R-CNN wird auf den Coco Datensatz trainiert. Coco ist weiterverbereiteter Datensatz . Coco datensatz enthält 91 Arten von Objekten, 2,5 million Objekte und 328000 Bilder. Mask R-CNN erreicht eine Durchschnittliche Genauigkeit von 35,7 auf den Coco datensatz und wird mit dem Modell ResNet-101-FPN trainiert. Mask R-CNN erkennt Objekte im Bild, erweitert das Faster R-CNN mit einer Maske für jedes Objekt im Bild. Mask R-CNN hat vier Ausgaben. Die erste Ausgabe ist die Maske, die zweite ist bounding box, die dritte ist eine Wahrscheinlichkeit und die vierte ist eine Kategorie. In diesem Paper [14] wird vorgestellt und wird mit verschiedenen Faltungs-Backbone-Architekturen für Merkmalsextraktion implementiert. es wird im Paper zwischen Mask R-CNN[14] und FCIS (Fully Convolutional Instance Segmentation) [20] verglichen. FCIS zeigt Fehler, wenn mehrere Objekte sich teilweise decken. Das bedeutet, dass FCIS nicht unterscheiden kann, ob der Teil zu dem Objekt gehört. Aber Mask R-CNN schneidet die Objekte ohne Überlappung heraus. Mask R-CNN basiert auf Faster R-CNN, Die Entwicklung von Mask R-CNN basiert auf Faster R-CNN und FCN. Mask R-CNN kombiniert zwei Algorithmen:

1. Faster R-CNN: bounding box erzeugung, für jedes Objekt im Bild ist
2. FCN(Fully convolutional Network):Semantic Segmentation(Semantische Segmentierung)

Ergibt aus den beiden Kombinierten Algorithmen Instance Segmentation(Instanz Segementierung)[14].

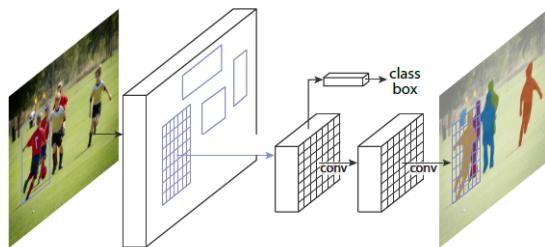


Abbildung 8: Mask R-CNN grundlegende Struktur
[14]

3.2 Kavitationsuntersuchung an Propellern

Bilder werden in Versuchsanstalt Potsdam aufgenommen. Der Versuchsanstalt ist dafür da, damit Schiffe auf dem Wasser getestet werden und Propeller mit hohen Geschwindigkeiten durchlaufen lässt, damit es gewusst wird, ab welchen Druck und welche Temperatur der Propeller im Wasser Kavitiert oder einen Hohlraum bildet

In diesem Versuchsanstalt wird es mit dem Propeller die Folgende Experimente durchgeführt(Widerstand und Propulsion, Propeller und Kavitation, Dynamik und Akustik, Computational Fluid Dynamics(CDF)) . Die Experimente werden mit dem Propeller gemacht wie das Bild von der Versuchstanstall zeigt.

Orte für Kavitationsorte zum Beispiel Schiffbau-Versuchsanstalt Potsdam GmbH,



Abbildung 9: Kavitationstunnel K15A von Kempf und Remmers

Das Bild ist aus dieser Website [<https://www.sva-potsdam.de/kavitationstunnel/>]

Hamburgische Schiffbau-Versuchsanstalt GmbH, Versuchsanstalt für Binnenschiffbau e.V. , UT 2 Versuchsanstalt für Wasserbau und Schiffbau (VWS) (Berlin)

4 Umsetzung

In der Umsetzung wird das Programm Tipvortexexcavitaion in Python geschrieben. Im Anhang 11.4 Tip Vortex Kavitation code steht das Programm zur Verfügung. Die Daten sind getrennt zwischen Trainingsdaten(traindatei) und Validierungsdaten(valdatei). Die traindatei enthält 22 Bilder und 3 Annotationsdateien. Annotationsdateien sind Tipvortexexcavitation_coco, Tipvortexexcavitation_csv und Tipvortexexcavitation_json. Es wird mit dem JSON experimentiert. Nach dem Hochladen der Daten und der Durchführung werden segmentierte Bilder ausgegeben.

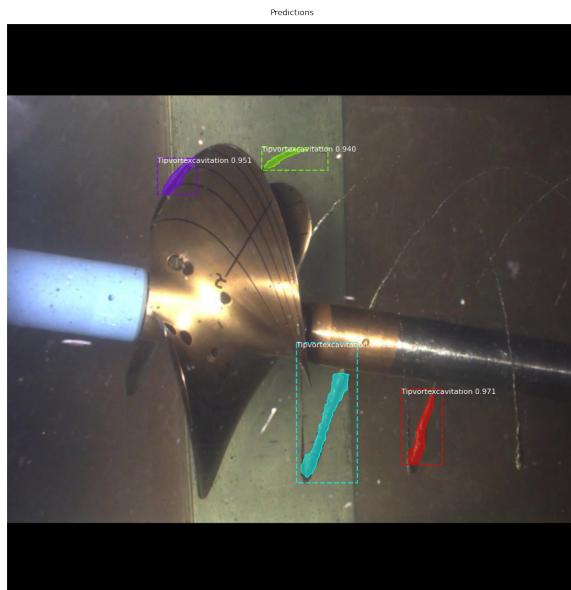


Abbildung 10: Die Ausgabe von dem Mask R-CNN

Das Mask R-CNN wird auf den Validierungsdatensatz angewendet. Die Konfigurationen für das Mask R-CNN sind faltende neuronale Netzwerke(Netze) für Merkmalsextraktion: resnet101, die Lerningsrate $0,001=10^{-3}$ und Anzahl der Epochen: 10. Im Anhang 11.3 sind die Konfigurationen von dem Mask R-CNN zu sehen.

Der Code ist im Anhang mit Kommentare und basiert auf das Repository [2]. Der Code nutzt die Datei von dem mrcnn. mrcnndatei hat sechs Teilprogramme(Codes), die in Python geschrieben wurden . mrcnn enthält `__init__`, `config`, `modell`, `parallel_model`, `utils` und `visualize`. In mrcnn ist der Pythoncode vom Mask R-CNN ohne Trainieren. Config macht die Einstellungen für das Mask R-CNN. Diese Einstellungen sind im Anhang. Das Programm Tipvortexexcavitation nimmt die Funktionen, die in der mrcnndatei implementiert sind. Im Programm werden die Konfigurationen überschrieben. Zum Beispiel, ob ein GPU oder ein CPU benutzt wird, Anzahl der Bilder, die in der Grafikkarte aufgenommen werden können. Es hängt von dem Speicher der Grafikkarte ab. Im Paper wird das Mask R-CNN auf den Cocodatensatz und Ballondatensatz trainiert. In der Arbeit wird das Mask R-CNN Programm mit den annotier-

ten Bildern von Tip Vortex Kavitation trainiert. Das Programm hat andere Konfiguration. Im Paper wird das Programm mit einer Grafikkarte von 12 GB Speicher. Jedes Bild verbraucht 6 GB. Das heißt im Originalpaper Bilder Pro GPU gleich zwei. Bei dem Tipvortextcavitationprogramm wird auf 1 gesetzt, weil der Prozessor AMD Ryzen 73700U mit Radeon Vega Mobile Gfx 2.30 GHz ist. In Jupyter Lab wird die Visualisierung von den Daten und dem Modell gemacht.

4.1 Datensätze

Computervisionssysteme benötigen eine große Menge von Hand annotierte Daten, damit die Systeme die richtige Form und Figur lernen können. Dieser Schritt ist aufwendig und nimmt so viel Zeit in Anspruch

4.1.1 Ballondatensatz

Ballon ist Einführungsbeispiel, wie die Datensätze in VGG Image Annotator in JSON Format erzeugt werden. VGG Image Annotator ist ein Bildanmerkungstool, mit dem Bereiche in einem Bild definiert und Textbeschreibungen dieser Bereiche erstellt werden können(ist ein Bildanmerkungstool, das verwendet werden kann, um Regionen in einem Bild zu definieren und Textbeschreibungen dieser Regionen zu erstellen). In VGG Image Annotator gibt es mehrere Möglichkeiten, um Regionen zu annotieren.

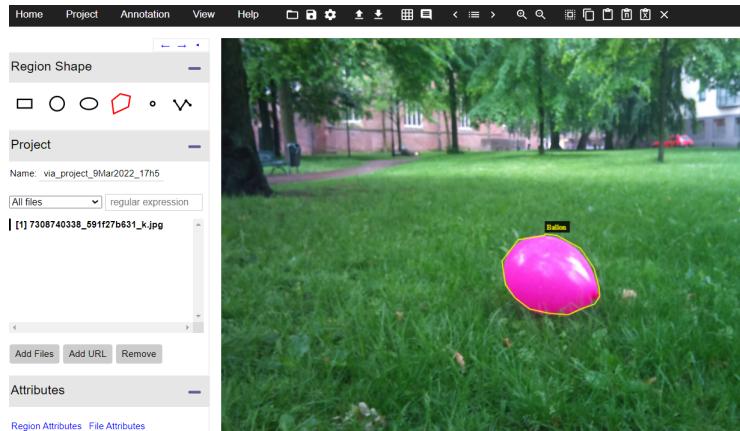


Abbildung 11: Benutzeroberfläche von VGG Image Annotator (VIA)
Im Anhang wird erklärt, wie VGG Image Annotator benutzt wird und welche Funktionen das Programm bereitstellt.

4.1.2 Tipvortextcavitationdatensatz

Bilder werden in VGG Image Annotator annotiert. Das bedeutet, dass mit Anmerkung versehen wird. Wie in Figure 15 gemacht wird. Auf der Basis von der Kavitationsforschung und Bildern wird die Tipvortextkavitation gesucht, die eine besondere Form hat. Diese Form ist Wirbel, die aus dem Propeller hinausgehen. Die Form des Spitzewirbels wird in den Bildern den von Versuchen entnommen und annotiert, um den Algorithmus von Mask R-CNN trainieren

zu können. Polygon wird in VGG Image Annotator benutzt, um Tipvortexcavitation zu schneiden. In den zwei Bildern werden Bilder in der Postdam Schiffbau-Versuchsanstalt. so viel Bilder werden annotiert. 33 Bilder werden

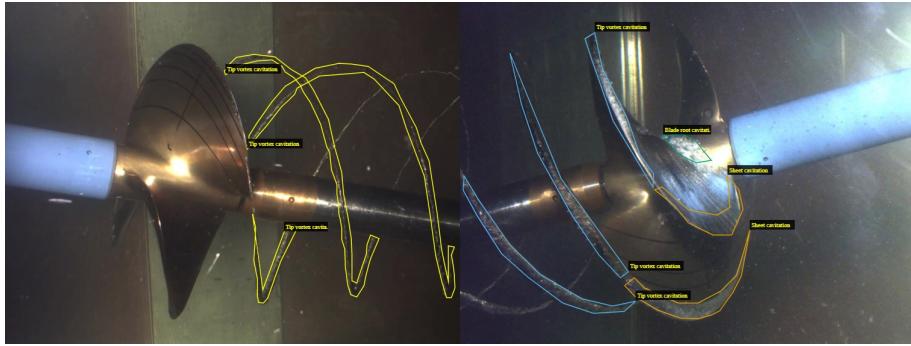


Abbildung 12: Markiertes TipvortexcavitationBild

Hier wird das Bild in VGG Image Annotator markiert. Man kann Tip vortex cavitation erkennen, indem die erste Abbildung 1 anguckt wird, dann erkennt man, dass die Tip vortex cavitation einen Wirbel um den Propeller bildet und Spitzen hat. Im zweiten Bild werden mehrere Kavitationen annotiert.

ausgewählt und geteilt zwischen 22 Bilder für Trainingsdatensatz und 11 Bilder für Validierungsdatensatz.

4.1.3 Sheetcavitationdatensatz

Sheetkavitationen haben Schaumiges Aussehen an den Rändern von dem Propeller. Scheetkavitation hat die Eigenschaft, dass sie dünn, glatt und transparent sind.

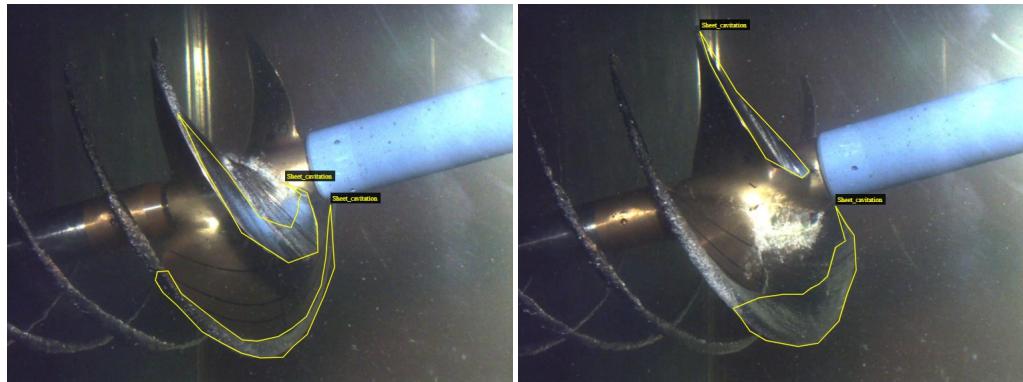


Abbildung 13: Annotiertes Sheetcavitationbild

4.2 Modell

Das Modell von dem Mask R-CNN erzeugt bounding boxes(Begrenzungsrahmen, bedeutet Rahmen um das Objekt) und segmentierten Masken und basiert auf dem Feature Pyramid Network(FPN) und ResNet101. In Mask R-CNN wird zwei Netzwerkarchitekturen für Merkmalsextraktion verwendet. Diese Netzwerkarchitekturen sind Resnet50 und Resnet101. Mask R-CNN wird mit dem Resnet50 und Resnet101 trainiert. Im Anhang Modell in JupyterLab (resnet50) gibt es ein Modell zwischen Seite 3 und 22. Modell enthält mehrere Blöcke oder Schritten, um das Eingabebild Instanz zu segmentieren. Die Merkmale werden mit dem Resnet50-Modell extrahiert.(FPN). (RPN).(ROI). Es gibt zwei Modelle, womit der Algorithmus von Mask R-CNN trainiert werden kann, die Resnet50 und Resnet101. Resnet50 ist Netzarchitektur mit 50 Layer hintereinandergeschaltet. Das Resnet 101-Modell wird in Mask R-CNN für die Coco-datasets und ausgeführt, mit 5 Frames per Sekunde und erreicht Mask Average Precision(AP) 35.7 [#He2017]. Das Ergebnis wurde in diesem Paper präsentiert. Tabelle (Vergleich zwischen Resnet50 und Resnet101 von dem Aufbau her, Struktur). Wer ist das beste für das Trainieren Resnet50 oder Resnet 101. Vergleich in Bilder, wer kann von den zwei Modellen mehr segmentieren. vergleichen segmentiertes Bild mit resnet-50 erzeugt und zwischen segmentiertes Bild mit resnet101 erzeugt mit verschiedenen Lernraten.

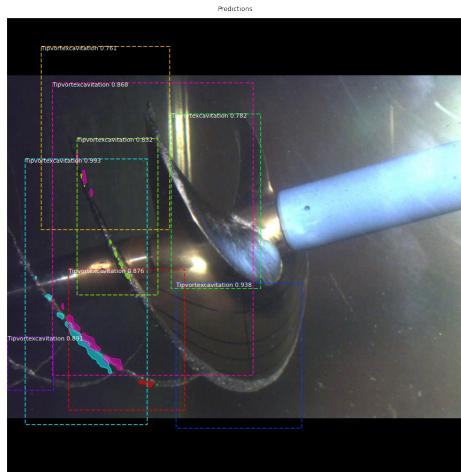


Abbildung 14: Instanzsegmentiertes Bild mit Resnet50

4.3 Trainieren und Validieren

Das Trainieren wird mit der Umgebung von Anaconda durchgeführt. Die Schritte stehen im Anhang 10.2 . Das Trainieren von den Datensätzen mit dem Modell(Die Merkmalextraktion). Die Implementierung wird mit dem Mask R-CNN wird mit Programmiersprache Python geschrieben. Tensorflow und Keras sind wichtig für das Programm, um bessere Ergebnisse bei dem Trainieren wird auf GPU-Support von Nvidia verwiesen. Der Algorithmus wird auf zwei Klassen trainiert, die eine Klasse von Figuren ist Tipvortexkavitation und die Andere Klasse ist der Hintergrund. Der Instanzsegmentierer hat eine Loss-Funktion, die in der 10 Epochen minimiert wird. Die Lossfunktion = $rpn_class_loss + rpn_bbox_loss + mrcnn_class_loss + mrcnn_bbox_loss + mrcnn_mask_loss$

4.3.1 Das Trainieren von dem Ballondatensatz

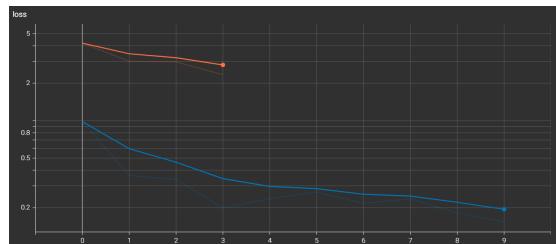


Abbildung 15: Lossfunktion von dem Ballondatensatz

Epochen	balloon20220220T0146	balloon20220304T2205
1	4.196	0.9822
2	3.447	0.5954
3	3.202	0.4635
4	2.804	0.3423
5	-	0.2964
6	-	0.2842
7	-	0.2566
8	-	0.2479
9	-	0.2212
10	-	0.1942

Tabelle 4: Vergleich zwischen zwei Modellen

das erste Modell wird für vier Epochen trainiert und das zweite Modell wird für zehn Epochen trainiert. bei dem Zweiten Modell wird das erste Modell verwendet, um die Erkennungsleistung zu verbessern. Das Modell für Merkmalextraktion ist Resnet101



Abbildung 16: Instanzsegmentierte Bilder von dem Ballondatensatz
 Hier wird verglichen zwischen zwei Bildern. Das Modell balloon20220220T0146 für 4 Epochen und wird auf das Bild angewendet. Das Ballon wird nicht erkannt. Aufgrund der hohen Lossfunktion. Bei dem zweiten Bild wird das Modell balloon20220304T2205 für 10 Epochen trainiert und auf das Bild angewendet.

4.3.2 Das Trainieren von dem Tipvortexcavitationdatensatz

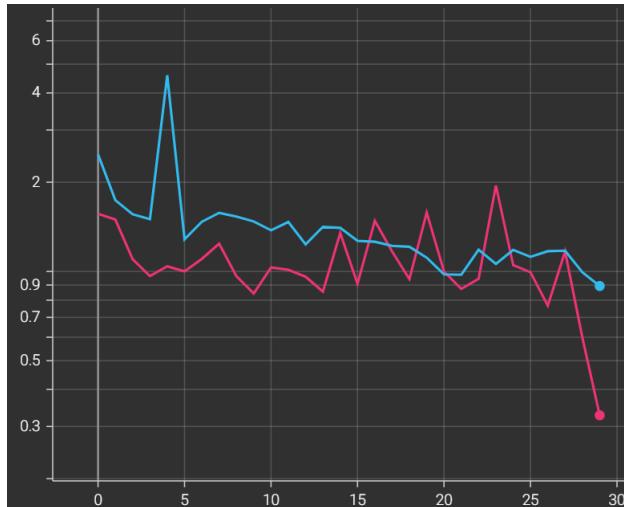


Abbildung 17: Vergleich zwei Modellen tipvortexcavitation20220424T1954 und tipvortexcavitation20220425T1108
 Die Lossfunktion für die zwei Modelle. Das rote Modell ist besser als das blaue Modell. Die Lossfunktion von dem roten Modell liegt bei 0.3273. Durch Transferlearning verbessert sich das Modell mit 0.5 als das blaue Modell.

Epochen	tipvortexexcavitation20220424T1954(Blau)	tipvortexexcavitation20220425T1108(Rot)
1	2.481	1.565
2	1.74	1.496
3	1.559	1.1
4	1.5	0.9656
5	4.589	1.001
6	1.284	1.001
7	1.471	1.103
8	1.576	1.241
9	1.531	0.9632
10	1.474	0.8425
11	1.377	1.032
12	1.468	1.013
13	1.234	0.9598
14	1.415	0.8555
15	1.403	1.348
16	1.268	0.9091
17	1.26	1.482
18	1.22	1.168
19	1.212	0.9427
20	1.113	1.58
21	0.9779	0.9993
22	0.9748	0.8733
23	1.184	0.9442
24	1.06	1.951
25	1.182	1.051
26	1.121	0.9938
27	1.17	0.7673
28	1.175	1.168
29	0.9938	0.598
30	0.8933	0.3273

Tabelle 5: Vergleich zwischen zwei Modellen

Die Tabelle zeigt das Modell in Rot besser als blau. in Rot wird das Transferlearning benutzt, um bessere Erkennungsleistung zu bekommen .Das Blaue Modell wird mit ihm den Algorithmus für das Rote Modell trainiert. In einem neuronalen Netz wird Transfer learning benutzt, damit die Trainingsdaten reduziert werden.

Die beide Modelle haben dieselbe Konfiguration:

```

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 0.01
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 10
WEIGHT_DECAY 0.0001

```

Abbildung 18: Die Konfiguration von dem Modell tipvortexcavation20220424T1954 und tipvortexcavitation20220425T1108

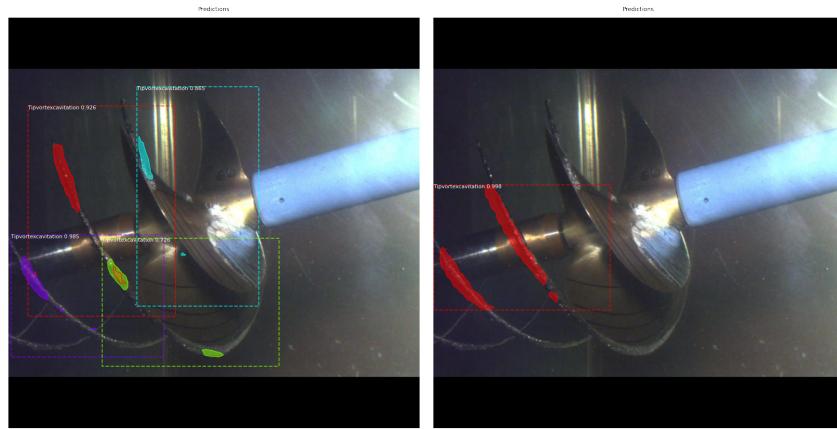


Abbildung 19: Instanzsegmentierte Bilder von Tipvortexcavitationdatensatz
Das Mask R-CNN wird auf die Bilder von dem Tipvortexcavitationdatensatz angewendet. Tipvortexcavitationdatensatz enthält zwei Dateien train und val. In der traindatei gibt es 22 Bilder mit einer Jsondatei. Die Jsondatei enthält die Position, die Ausrichtung und wo die Kavitation zusehen ist. Auf das Bild links wird das Modell tipvortexcavitation20220424T1954 angewendet. Auf das Bild rechts wird das Modell tipvortexcavitation20220425T1108 angewendet. das Modell rechts verwendet die Gewichte von dem Modell links.

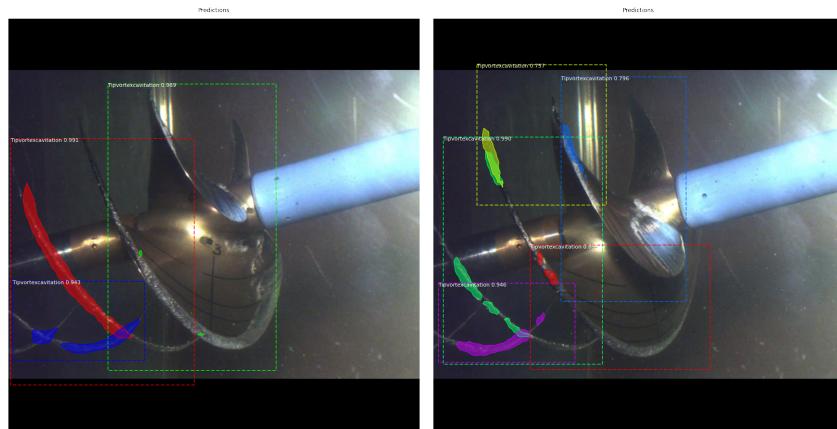


Abbildung 20: Instanzsegmentierte Bilder von Tipvortexcavitationdatensatz
Instanzsegmentierte Bilder und Bounding box erzeugen. Das trainierte Modell wird auf den Validierungsdatensatz angewendet. Der Valideirungsdatensatz(val_datensatz) enthält 14 Bilder. resnet101 das ist für 20 Epochen

5 Testen und Ergebnisse

In diesem Kapitel werden die Modelle auf neue Bilder getestet, die Ergebnisse gezeigt und was aus der Umsetzung und dem Stand der Technik schlussgefolgert werden. Hier werden die zwei Modelle mit dem Splash Color Effekt getestet werden, die oben genannt werden.

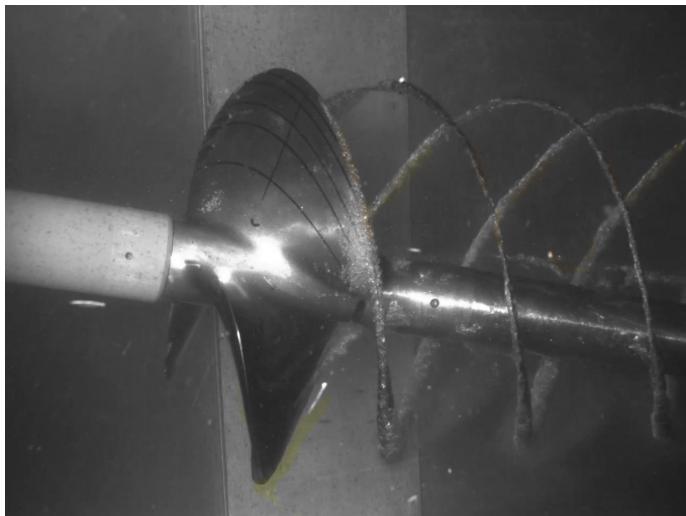


Abbildung 21: color splash Effekt angewendet auf ein Bild
Modell (1): tipvortextcavitation20220424T1954(Blau)

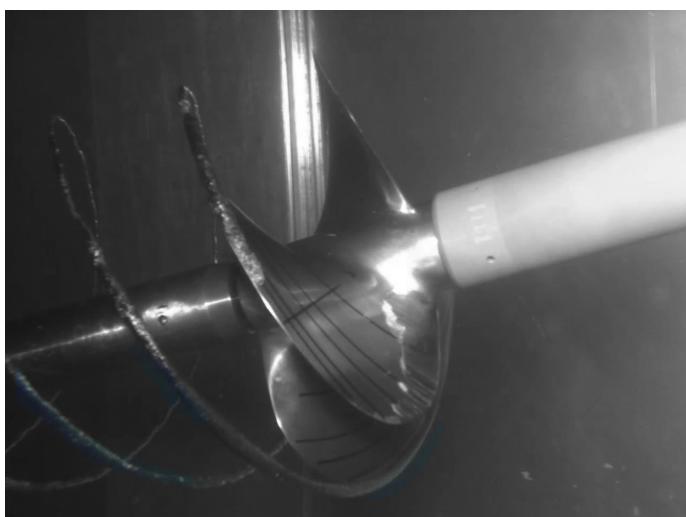
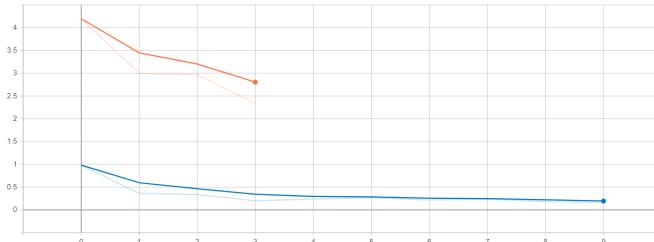


Abbildung 22: color splash Effekt angewendet auf ein Bild
Modell(2):tipvortextcavitation20220425T1108(Rot)

Bei der Annotation der Bilder muss aufgepasst werden, welche Bilder mit dem Tool VIA bearbeitet werden. Diese Bilder werden in eine separat Datei gespeichert, damit gewusst wird, mit welchen Bildern der Algorithmus trainiert wird. Bei einem CPU dauert das Trainieren viel lange als GPU, weil das GPU so viel ALU besitzt. GPU führt mehr viele Berechnungen als CPU, kann mit Bildern effizienter umgehen. Dadurch kann das GPU die Faltung und Matrixmultiplikation viel schneller als das CPU berechnen. Deswegen ist es sinnvoller, mit dem GPU zu arbeiten, wenn faltendes neuronales Netz sowie (R-CNN, Fast R-CNN, Faster R-CNN und Mask R-CNN) auf Bilder trainiert wird [10]. Es wird aus der Umsetzung herausgefunden, dass beim Trainieren von neuronalen Netzwerken auf folgende Punkte geachtet wird:

- für welche dauer trainiert wird.
- mit welchem Modell für Das Trainieren verwendet wird
- Für welche Lernrate das Modell trainiert wird, Wenn die Lernrate zu gering klein Beispiel $0.00001 = 10^{-5}$ eingestellt wird, dann werden die Gewichte zu langsam angepasst.
- Die Epochen habe ich die meiste Zeit(ungefähr 90 % der Experimente) zahn ausgewählt wegen der Beschränkten Leistung des Prozessors, das kann man in das Figure 18 sehen.



Name	Smoothed	Value	Step	Time	Relative
balloon20220220T0146	2.804	2.337	3	Sun Feb 20, 03:35:49	1h 23m 8s
balloon20220304T2205	0.1942	0.1543	9	Sat Mar 5, 01:50:31	3h 22m 37s

Abbildung 23: Der Lossfunktionsverlauf von Balloondatensatz



Abbildung 24: Der Lossfunktionsverlauf von Tipvortexexcavationdatensatz



Abbildung 25: Der Lossfunktionsverlauf von Sheetcavitationdatensatz

6 Zusammenfassung

In dieser Arbeit wird angefangen, mit dem Problem der Kavitation zu erklären und was sie von Folgen auf den Propeller hat, wenn sie nicht schnell behandelt wird. Es gibt sieben Kavitationsarten, die den Propeller verhindern können, ihre Arbeit zu machen. Es gibt verschiedene Lösungen in der künstlichen Intelligenz, die im Stand der Technik auf diese Lösungen eingegangen werden. Diese Lösung, die damit in der Arbeit weitergemacht wird, ist Mask R-CNN für Instanz Segmentierung. Mask R-CNN ist ein System, was als Eingabe die annotierte Bilder nimmt und daraus extrahiert das Mask R-CNN durch Resnet50 und Resnet101 die Merkmale und nutzt diese Merkmale, um die Region of Interests vorschlagen zu können. Bei der Umsetzung wird VIA Tool zum Annotieren verwendet und werden so Bilder annotiert für Multiclass Klassifikation und für Binary Klassifikation. Dann wird das Mask R-CNN verwendet, um das Binary Klassifikation Problem zu lösen. Es wird zuerst probiert, mit dem Ballon Datensatz, was schon von Facebook AI Research(FAIR) Gruppe implementiert wird. in deisem Paper [14] und befindet sich auf dieses Repository [2]. Das Modell wird wieder zweimal trainiert, wird für vier Epochen und das zweite Modell für 10 Epochen ausgewählt und wird zwischen den beiden verglichen. Es wird herausgefunden, dass das Modell mit zehn Epochen besser als das Modell mit vier Epochen ist. Es wird das Mask R-CNN mit dem Tip vortex Kavitation Datensatz trainiert und auch zwei Modelle genommen. Dann das erste Modell ist ohne Transfer Learning und hat nicht gute Ergebnisse geliefert. Aber das zweite Modell, das gezeigt hat, dass Transfer Learning nützlich ist, weil die Datenmenge gleich bleibt, aber das zweite Modell die Gewichte von einem trainierten Modell hat. Deswegen wird eine Verbesserung in der Lossfunktion beobachtet, obwohl die Epochen und Bilder gleich geblieben sind. Was auch eine Reduzierung in der Bildermenge bringt, wenn zwischen die zwei Netzwerkarchitekturen verglichen werden Resnet50 und Resnet101, dann wird es herausgefunden, dass Resnet101 gute Erkennungsleistung hat besser als Resnet50 mit kleinem Datensatz von annotierten Bildern. Wenn es soviel Daten und kleine Grafikkarte gibt, dann wird Resnet50 benutzt und natürlich für Echtzeitanwendung, weil Resnet50 schneller ist. Aber trotzdem lohnt es sich, dass das Resnet 101 probiert wird. Es gibt andere Varianten neben Resnet. Diese Varianten sind so wie zum Beipsiel wie LeNet-5, AlexNet, GoogLeNet und VGGNet. Im Testen wird der color splash effekt auf die Bilder von Propeller und Video angewendet. in der Egebnisse werden die Lossfunktionsverläufe vorgestellt von dem Ballon datensatz, Tipvortexexcavationdatensatz und Sheetcavitationdatensatz. Das Mask R-CNN braucht viel zeit. Zum Beispiel bei dem Experiment Tipvortexexcavation für 40 Epochen, was in Figure 25 vorgestellt wird. das zwölf Stunden braucht, das bedeutet, dass sehr gute Rechenleistung gebraucht wird, um schneller zu sein.

7 Literaturverzeichnis

Literatur

- [1] Testing and extrapolation methodspropulsion; cavitation description of cavitation appearances. *ITTC - Recommended Procedures and Guidelines*, 2002.
- [2] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow, 2017.
- [3] Aydt, Miriam. Entwicklung einer Machine-Learning-basierten Anwendung mit interaktiven Bildern für die Biografie- und Erinnerungspflege für Menschen mit Demenz. 2020.
- [4] Bernstein, Herbert. Bauelemente der Hydraulik. In *Elektropneumatische und elektrohydraulische Bauelemente in der Mechatronik*, pages 49–152. Springer, 2022.
- [5] Robert Bronsart. *Glossar Schiffstechnik*. Seehafen Verlag, Hamburg, 2016.
- [6] JS Carlton. 9 - cavitation. In JS Carlton, editor, *Marine Propellers and Propulsion (Second Edition)*, pages 206–246. Butterworth-Heinemann, Oxford, second edition edition, 2007.
- [7] Daniel Bourke. 03. Convolutional Neural Networks and Computer Vision with TensorFlow.
- [8] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. March 2016.
- [9] Eggers, Sebastian. Automatische Erkennung von Textelementen in Bildschirmaufnahmen für die Eye Tracking Datenanalyse.
- [10] Geldermann, Torben. *Implementierung eines faltenden neuronalen Netzwerks auf einer NVIDIA-CUDA-Plattform für Detektionsaufgaben*. PhD thesis, Hochschule für angewandte Wissenschaften Hamburg, 2019.
- [11] Géron, Aurélien. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and techniques to Build Intelligent systems*. O'Reilly Media, 2019.
- [12] Ross Girshick. Fast r-cnn. April 2015.
- [13] Girshick, Ross and Donahue, Jeff and Darrell, Trevor and Malik, Jitendra. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2015.
- [14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. March 2017.
- [15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. December 2015.

- [16] Kleiser, Georg. Einführung in die Strömungslehre.
- [17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.
- [18] Rudolf Kruse, Christian Borgelt, Christian Braune, Frank Klawonn, Christian Moewes, and Matthias Steinbrecher. *Computational Intelligence*. Springer Fachmedien Wiesbaden, 2015.
- [19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [20] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. November 2016.
- [21] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context. May 2014.
- [22] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. November 2014.
- [23] Maasjosthusmann, Robin and Lehrbass, Frank. *Explainable Artificial Intelligence: Analyse und Visualisierung des Lernprozesses eines Convolutional Neural Network zur Erkennung deutscher Straßenverkehrsschilder*. Number 26. ifes Schriftenreihe, 2021.
- [24] Moser, Artur. Automatische Erkennung von Biomarkern in OCT-Aufnahmen und Berechnung eines Riskscores für altersabhängige Makuladegeneration.
- [25] Sebastian Raschka. *Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow : Konzepte, Tools und Techniken fÄr intelligente Systeme*. O'Reilly Dpunkt.verlag, Heidelberg, 2018.
- [26] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. June 2015.
- [27] Prof. Dr.-Ing. Sascha Spors. Beschreibung von lti-systemen im zeitbereich. *Signal- und Systemtheorie*, Sommersemester 2019.
- [28] Spurk, Joseph H and Aksel, Nuri. *Strömungslehre*, volume 4. Springer, 1989.
- [29] Szegedy, Christian and Liu, Wei and Jia, Yangqing and Sermanet, Pierre and Reed, Scott and Anguelov, Dragomir and Erhan, Dumitru and Vanhoucke, Vincent and Rabinovich, Andrew. Going Deeper With Convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

- [30] Voithofer, Mario. Verwendung von Deep Learning für die Erkennung von Werbung am Straßenrand in Dashcam-Videos. Master's thesis, Interactive Media; FH Oberösterreich – Fakultät für Informatik, Kommunikation und Medien, 4232 Hagenberg, Austria, 2019.
- [31] Zijie J. Wang, Robert Turko, Omar Shaikh, Haekyu Park, Nilaksh Das, Fred Hohman, Minsuk Kahng, and Duen Horng Chau. Cnn explainer: Learning convolutional neural networks with interactive visualization. April 2020.
- [32] Ziegler, Michael. Object Recognition with Convolutional Neural Networks.

8 Tabellenverzeichnis

Tabellenverzeichnis

1	Bekannte Algorithmen aus der Künstlichen Intelligenz für Bildverarbeitung	6
2	LeNet-5-Architektur	13
3	AlexNet-Architektur	14
4	Vergleich zwischen zwei Modellen	24
5	Vergleich zwischen zwei Modellen	26
6	Tasten nur im Bildfokus verfügbar	40
7	Tasten immer vorhanden	41

9 Abbildungsverzeichnis

Abbildungsverzeichnis

1	Kavitationsarten	7
2	Ein KNN in A Neural Network playground für ein Klassifikationsproblem	8
3	links: Lineare Funktion, Mitte: Rectifier Funktion, rechts: sigmoid Funktion	9
4	Faltende Neuronale Netze	10
5	Vollständig verbundenes neuronales Netzwerk	12
6	LeNet-5 Modell	13
7	Faster R-CNN	16
8	Mask R-CNN grundlegende Struktur	17
9	Kavitationstunnel K15A von Kempf und Remmers	18
10	Die Ausgabe von dem Mask R-CNN	19
11	Benutzeroberfläche von VGG Image Annotator (VIA)	20
12	Markiertes TipvortexcavitationBild	21
13	Annotiertes Sheetcavitationbild	22
14	Instanzsegmentiertes Bild mit Resnet50	23
15	Lossfunktion von dem Ballondatensatz	24
16	Instanzsegmentierte Bilder von dem Ballondatensatz	25
17	Vergleich zwei Modellen tipvortexcavitation20220424T1954 und tipvortexcavitation20220425T1108	25
18	Die Konfiguration von dem Modell tipvortexcavitation20220424T1954 und tipvortexcavitation20220425T1108	27
19	Instanzsegmentierte Bilder von Tipvortexcavitationdatensatz	28
20	Instanzsegmentierte Bilder von Tipvortexcavitationdatensatz	28
21	color splash Effekt angewendet auf ein Bild	29
22	color splash Effekt angewendet auf ein Bild	29
23	Der Lossfunktionsverlauf von Balloondatensatz	30
24	Der Lossfunktionsverlauf von Tipvortexcavitationdatensatz	31
25	Der Lossfunktionsverlauf von Sheetcavitationdatensatz	31
26	VGG Image Annotator Oberfläche	39

10 Anhang

10.1 VGG Image Annotator(Annotation tool)

Bilder werden mit Vgg Image Annotator(VIA) annotiert. Das bedeutet, in diesem Schritt werden Segmente von den Bildern klassifiziert als (Tip vortex cavitation, Hub vortex cavitation, Cloud cavitation, Blade root cavitation, Bubble cavitation, Sheet cavitation, Propeller hull vortex cavitation).

Hier ist Einleitung, wie man mit VGG zurecht kommt und wie man Bilder in VGG annotieren kann.

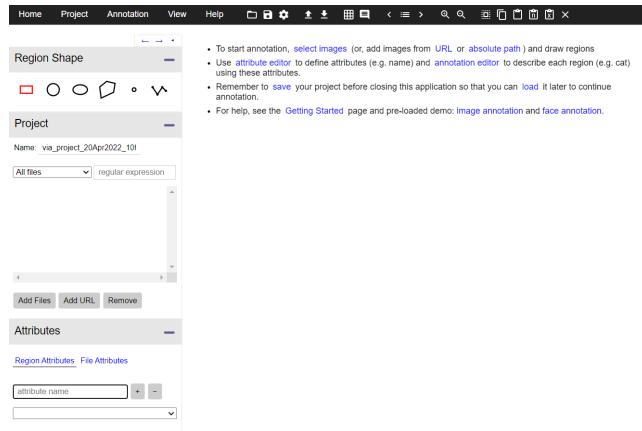


Abbildung 26: VGG Image Annotator Oberfläche um Bilder in VGG Image Annotator bearbeiten zu können:

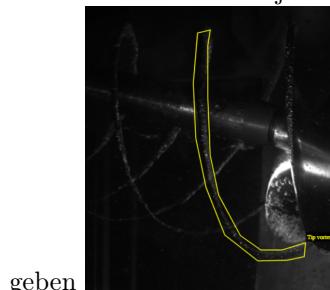
1. man muss diese Taste klicken. dann kann man eine Menge von Bildern auswählen.
2. den Name für ein Merkmal oder die Klasse von dem Objekt(Kavitationsarten, Personen, Tiere, Pflanzen, Krebsarten.....) eingeben, dann Klickt man auf .

3. Der Name taucht auf . dann als Type dropdown wird ausgewählt.

4. Hier gibt man die Objekte, die annotiert werden und die später erkannt werden oder segmentiert werden.

5. Hier wird Polygon region shape ausgewählt

6. Am Ende wird das Objekt auf dem Bild Markiert. Die Sorte wird eingegeben



Tasten	Aktion
	Ausgewählte Region bewegen
	Alle Regionen auswählen
	Ausgewählte Regionen kopieren
	Ausgewählte Regionen einfügen
	Ausgewählte Regionen löschen
	Vergrößern/verkleinern (Mauszeiger befindet sich über dem Bild)
	Regionsbezeichnung ein-/ausschalten
	Regionsgrenze ein-/ausschalten
	das Zeichnen von Polyshape beenden
	letzte Polyshape-Ecke löschen

Tabelle 6: Tasten nur im Bildfokus verfügbar

Tasten	Aktion
	Zum nächsten/vorherigen Bild wechseln
	Vergrößern/verkleinern/zurücksetzen
	Regionenbezeichnung aktualisieren
	Bereichsfarbe aktualisieren
	Anmerkungseditor einschalten und ausschalten
	Zum ersten Bild springen
	Zum letzten Bild springen
	Mehrere Bilder springen
	Mehrere Bilder springen
	Laufende Aktion abbrechen

Tabelle 7: Tasten immer vorhanden

10.2 Umgebungseinstellung von Anaconda

Das ist hier eine Anleitung, wie die Umgebung von Mask_RCNN unter Windows mit Anaconda läuft. man muss die richtige versionen von den Bibliotheken installieren, weil man oft fehler bekommt.

- mkdir MaskRCNN
- cd MaskRCNN
- conda create --name Matterprot_MaskRCNN python=3.6.13 tensorflow==1.15.0 Keras==2.2.4 h5py==2.8.0 pip
- conda activate Matterprot_MaskRCNN
- conda install -c conda-forge jupyterlab
- git clone https://github.com/matterport/Mask_RCNN.git
- cd Mask_RCNN
- pip install -r requirements.txt
- git clone https://github.com/phiferriere/cocoapi.git
- pip install pycocotools

Damit die Bibliotheken richtig von Coco(cocoapi, pycocotools) funktionieren, muss man sicherstellen, dass Visual Studio 2022 oder 2019 oder 2015 installiert ist.

der Fehler ist: error: command `cl.exe` failed: No such file or directory

Link: <https://stackoverflow.com/questions/41724445/python-pip-on-windows-command-cl-exe-failed>

hier wird erklärt, wieso man cl.exe installieren muss

- pip install git+<https://github.com/phiferriere/cocoapi.git#subdirectory=PythonAPI>
- wget https://github.com/matterport/Mask_RCNN/releases/download/v2.1/mask_rcnn_balloon.h5 Mask_RCNN mask_rcnn_balloon.h5. Das Funktioniert nicht
- python setup.py install
- jupyter lab

10.3 Mask R-CNN Kofiguration

```

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      0.001
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	50
WEIGHT_DECAY	0.0001

10.4 Tip vortex cavitation code

Der Code basiert auf diese Arbeit [2]. Dieses Repository [2] wird in der Arbeit verwendet.

Tip vortex cavitation code

April 22, 2022

""” Mask R-CNN wird mit Tipvortexcavitationdatensatz trainiert. Farbspritzereffekt(color splash effect) wird im Algorithmus Implementiert

1 Ein neues Modell wird mit den vortrainierten Coco-Gewichten trainiert

python3 Tipvortexcavitation.py train –dataset=/path/to/Tipvortexcavitation/dataset –weights=coco für Linux der Pfad und die Gewichte von Coco dataset.

python Tipvortexcavitation.py train –dataset=C:/Users/majd4/Desktop/Bachelorarbeit/Bachelor-Arbeit-Daten/MaskRCNNProjekt/MaskRCNN_2/Mask_RCNN/datasets/Tipvortexcavitation –weights=coco

2 Fortsetzen des Trainierens von einem Modell, was schon trainiert wurde

python3 Tipvortexcavitation.py train –dataset=/path/to/Tipvortexcavitation/dataset –weights=last für Linux

python Tipvortexcavitation.py train –dataset=C:/Users/majd4/Desktop/Bachelorarbeit/Bachelor-Arbeit-Daten/MaskRCNNProjekt/MaskRCNN_2/Mask_RCNN/datasets/Tipvortexcavitation –weights=last

3 Ein neues Modell wird mit den ImageNet-Gewichten trainiert

python3 Tipvortexcavitation.py train –dataset=/path/to/Tipvortexcavitation/dataset –weights=imagenet für Linux

python Tipvortexcavitation.py train –dataset=C:/Users/majd4/Desktop/Bachelorarbeit/Bachelor-Arbeit-Daten/MaskRCNNProjekt/MaskRCNN_2/Mask_RCNN/datasets/Tipvortexcavitation –weights=imagenet

4 Farbspritzer(color splash) auf ein Bild anwenden

python3 Tipvortexcavitation.py splash –weights=/path/to/weights/file.h5 –image= für Linux

```

python Tipvortexcavitation.py splash -weights=C:/Users/majd4/Desktop/Bachelorarbeit/Bachelor-
Arbeit- Daten/MaskRCNNProjekt/MaskRCNN_2/Mask_RCNN/mask_rcnn_tipvortexcavitation_0010.h5
-image=C:/Users/majd4/Desktop/Bachelorarbeit/Bachelor-Arbeit-Daten/MaskRCNNProjekt/MaskRCNN_2/M
# Farbspritzer (color splash) auf ein Video mit Gewichten von der Logsdatei anwenden python3
Tipvortexcavitation.py splash -weights=last -video= für Linux

python Tipvortexcavitation.py splash -weights=last -video= für Windows

python Tipvortexcavitation.py splash -weights=C:/Users/majd4/Desktop/Bachelorarbeit/Bachelor-
Arbeit- Daten/MaskRCNNProjekt/MaskRCNN_2/Mask_RCNN/mask_rcnn_tipvortexcavitation_0010.h5
-video=C:/Users/majd4/Desktop/Bachelorarbeit/Bachelor-Arbeit-Daten/MaskRCNNProjekt/MaskRCNN_2/M
"""

```

```

[47]: import os
import sys
import json
import datetime
import numpy as np
import skimage.draw

# der Pfad der Datei von dem Mask R-CNN Projekt
ROOT_DIR = os.path.abspath("C:/Users/majd4/Desktop/Bachelorarbeit/
                           →Bachelor-Arbeit-Daten/MaskRCNNProjekt/MaskRCNN_2/Mask_RCNN")

# Maske RCNN importieren
sys.path.append(ROOT_DIR)
from mrcnn.config import Config
from mrcnn import model as modellib, utils
from mrcnn import visualize

# Pfad zur Datei mit den trainierten Gewichten
# was vorher trainiert wurde, natürlich wird es nochmals benutzt, um ein neues Modell zu trainieren
# es wird das neue Modell besser in der erkennung von Tipvortexcavitation
WEIGHTS_PATH = os.path.join(ROOT_DIR, "mask_rcnn_tipvortexcavitation_0010.h5")

# hier ist die Datei, wo die logs gespeichert wurden
# Epochen von dem Training
DEFAULT_LOGS_DIR = os.path.join(ROOT_DIR, "logs")

#####
# Konfigurationen
#####

# Es ist hilfreich diese Website zu lesen, wenn man Änderung auf den Code vornehmen möchte
# https://github.com/matterport/Mask_RCNN/wiki

```

```
[48]: class TipvortexcavitationConfig(Config):
    """Konfiguration für das trainieren mit dem Tipvortexcavitationdatensatz
    Abgeleitet von dem basic Config Klasse und werden einige Werte
    →überschrieben
    """
    # ANZAHL DER zu verwendenden GPUs. Wenn nur eine CPU verwendet wird, muss
    →dies auf 1 gesetzt werden.
    # es ist abhängig von dem Prozessor und die Grafikkarte, die man hat
    # hier Auf meinem Laptop nutze ich AMD Ryzen 7 3700U with Radeon Vega
    →Mobile Gfx 2.30 GHz
    GPU_COUNT = 1

    #Die Konfiguration wird einen erkennbaren Namen gegeben
    NAME = "Tipvortexcavitation"

    # verwende eine GPU mit 12 GB Speicher, die zwei Bilder aufnehmen kann.
    # Passe nach unten an, wenn eine kleinere GPU verwendet wird.
    IMAGES_PER_GPU = 1

    # Anzahl der Klassen (einschließlich Hintergrund)
    NUM_CLASSES = 1 + 1 # Background + Tipvortexcavitation

    # Anzahl der Farbkanäle pro Bild.
    # RGB = 3 (Bilder nur mit Farbe),
    # grayscale = 1 (Bilder nur Schwarz und Weiß),
    # RGB-D = 4 (Bilder werden von einer Speziellen Kamera aufgenommen.
    # jedes Pixel im Bildem ist mit einem Intensitätswert verbunden)
    IMAGE_CHANNEL_COUNT = 3

    # Backbone Netzwerk die Architektur
    # Unterstützte Werte sind: resnet50, resnet101.
    BACKBONE = "resnet101"

    # Anzahl der Trainingsschritte pro Epoche
    STEPS_PER_EPOCH = 10

    # Anzahl der Validierungsschritte, die am Ende jeder Trainingsepoche
    →ausgeführt werden.

    VALIDATION_STEPS = 50
```

```
[49]: class TipvortexcavitationDataset(utils.Dataset):

    def load_Tipvortexcavitation(self, dataset_dir, subset):
        """Lade eine Teilmenge des Tip Vortex Cavitation-Datensatzes.
        """

```

```

# Klassen hinzufügen.
self.add_class("Tipvortextexcavation", 1, "Tipvortextexcavation")

# Trainings- oder Validierungsdatensatz?
assert subset in ["train", "val"]
dataset_dir = os.path.join(dataset_dir, subset)

annotations = json.load(open(os.path.join(dataset_dir, "Tipvortextexcavation_json.json")))
annotations = list(annotations.values()) # 

# Annotationen.unannotierte Bilder überspringen.
annotations = [a for a in annotations if a['regions']]

# Bilder hinzufügen
for a in annotations:
    # Erhalte die x, y Koordinaten der Punkte der Polygone
    # Die Polygone werden in den shape_attributes gespeichert
    if type(a['regions']) is dict: #annotated
        polygons = [r['shape_attributes'] for r in a['regions'].values()]
    else: # unanotated
        polygons = [r['shape_attributes'] for r in a['regions']]

    # load_mask() benötigt die Bildgröße, um Polygone in Masken zu konvertieren.
    image_path = os.path.join(dataset_dir, a['filename'])
    image = skimage.io.imread(image_path)
    height, width = image.shape[:2]

    self.add_image(
        "Tipvortextexcavation",
        image_id=a['filename'],
        path=image_path,
        width=width, height=height,
        polygons=polygons)

def load_mask(self, image_id):
    """Erzeuge Instanz für ein Bild.
    zurückgegeben:

    """
    image_info = self.image_info[image_id]
    if image_info["source"] != "Tipvortextexcavation":

```

```

    return super(self.__class__, self).load_mask(image_id)

    # Erzeuge Polygone zum einem bitmap mask of shape
    # [Höhe, Breite, Instanz_Anzahl]
    info = self.image_info[image_id]
    mask = np.zeros([info["height"], info["width"], len(info["polygons"])],
                   dtype=np.uint8)
    for i, p in enumerate(info["polygons"]):

        rr, cc = skimage.draw.polygon(p['all_points_y'], p['all_points_x'])
        mask[rr, cc, i] = 1

    return mask.astype(np.bool), np.ones([mask.shape[-1]], dtype=np.int32)

def image_reference(self, image_id):
    """Gibt den Pfad des Bildes zurück."""
    info = self.image_info[image_id]
    if info["source"] == "Tipvortexcavitation":
        return info["path"]
    else:
        super(self.__class__, self).image_reference(image_id)

```

```

[50]: def train(model):
    """ das Modell trainieren."""
    # Trainingsdatensatz(Trainingsdatenmenge).
    dataset_train = TipvortexcavitationDataset()
    dataset_train.load_Tipvortexcavitation(args.dataset, "train")
    dataset_train.prepare()

    # Validierungsdatensatz(Validierungsdatenmenge)
    dataset_val = TipvortexcavitationDataset()
    dataset_val.load_Tipvortexcavitation(args.dataset, "val")
    dataset_val.prepare()

    # keine Notwendigkeit, alle Schichten zu trainieren.
    print("Training network heads")
    model.train(dataset_train, dataset_val,
                learning_rate=config.LEARNING_RATE,
                epochs=10,
                layers='all')

```

```
[51]: def color_splash(image, mask):
    """Farbspritzer-Effekt anwenden.
    Bild: RGB Bild [Höhe, Breite, 3]
    mask:Instanzsegmentierung mask [Höhe, Breite, Anzahl der Instanzen]

    es wird schwarz weißes Bild mit Farbe zurückgegeben
    """
    # Erstelle ein schwarzweiße Kopie von dem Farbigen Bild.
    gray = skimage.color.gray2rgb(skimage.color.rgb2gray(image)) * 255
    # Kopiere die gefärbte Maske von dem Originalbild
    if mask.shape[-1] > 0:
        mask = (np.sum(mask, -1, keepdims=True) >= 1)
        splash = np.where(mask, image, gray).astype(np.uint8)
    else:
        splash = gray.astype(np.uint8)
    return splash
```

```
[52]: def detect_and_color_splash(model, image_path=None, video_path=None):
    assert image_path or video_path

    # Bild oder Video?
    if image_path:

        # das Modell für erkennung ausführen und erzeuge ↵
        ↵Farbspritzereffekt(color splash effect)
        print("Running on {}".format(args.image))
        # Bild lesen
        image = skimage.io.imread(args.image)
        # Objekte erkennen
        r = model.detect([image], verbose=1)[0]
        # Farbspritzer
        splash = color_splash(image, r['masks'])
        # Ausgabe speichern
        file_name = "splash_{:%Y%m%dT%H%M%S}.png".format(datetime.datetime.now())
        skimage.io.imsave(file_name, splash)
    elif video_path:
        import cv2

        vcapture = cv2.VideoCapture(video_path)
        width = int(vcapture.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(vcapture.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = vcapture.get(cv2.CAP_PROP_FPS)
```

```

file_name = "splash_{:%Y%m%dT%H%M%S}.avi".format(datetime.datetime.
now())
vwriter = cv2.VideoWriter(file_name,
                          cv2.VideoWriter_fourcc(*'MJPG'),
                          fps, (width, height))

count = 0
success = True
while success:
    print("frame: ", count)
    # Das nächste Bild lesen
    success, image = vcapture.read()
    if success:
        # OpenCV gibt die Bilder als BGR zurück und wandelt zu RGB um
        image = image[..., ::-1]
        # Objekte erkennen
        r = model.detect([image], verbose=0)[0]
        # Farbspritzer(Farbe spritzen)
        splash = color_splash(image, r['masks'])
        # RGB -> BGR um ein Bild in einem Video zu speichern
        splash = splash[..., ::-1]
        # dem Videoschreiber ein Bild hinzufügen
        vwriter.write(splash)
        count += 1
    vwriter.release()
print("Speichere zu ", file_name)

```

```

[ ]: #####
# Trainieren
#####

if __name__ == '__main__':
    import argparse

    #
    parser = argparse.ArgumentParser(
        description='Trainiere Mask R-CNN, um Tipvortexcavitation zu erkennen')
    parser.add_argument("command",
                       metavar="<command>",
                       help="'train' or 'splash'")
    parser.add_argument('--dataset', required=False,
                       metavar="/path/to/balloon/dataset/",
                       help='Directory of the Tipvortexcavitation dataset')
    parser.add_argument('--weights', required=True,
                       metavar="/path/to/weights.h5",
                       help="Path to weights .h5 file or 'coco'")

```

```

parser.add_argument('--logs', required=False,
                    default=DEFAULT_LOGS_DIR,
                    metavar="/path/to/logs/",
                    help='Logs and checkpoints directory (default=logs/)')
parser.add_argument('--image', required=False,
                    metavar="path or URL to image",
                    help='Image to apply the color splash effect on')
parser.add_argument('--video', required=False,
                    metavar="path or URL to video",
                    help='Video to apply the color splash effect on')
args = parser.parse_args()

# Argumente validieren
if args.command == "train":
    assert args.dataset, "Argument --dataset is required for training"
elif args.command == "splash":
    assert args.image or args.video,\n        "Provide --image or --video to apply color splash"

print("Weights: ", args.weights)
print("Dataset: ", args.dataset)
print("Logs: ", args.logs)

# Konfigurationen
if args.command == "train":
    config = TipvortexexcavationConfig()
else:
    class InferenceConfig(TipvortexexcavationConfig):
        # setz die Batchgröße(batch size) auf 1.da wir führen Inferenz auf ↵
        ↵ ein Bild nach dem anderen aus
        # Batchgröße(Batch size). Batch size = GPU_COUNT * IMAGES_PER_GPU
        GPU_COUNT = 1
        IMAGES_PER_GPU = 1
    config = InferenceConfig()
config.display()

# ein Modell erzeugen
if args.command == "train":
    model = modellib.MaskRCNN(mode="training", config=config,
                               model_dir=args.logs)
else:
    model = modellib.MaskRCNN(mode="inference", config=config,
                               model_dir=args.logs)

# Gewichtete Datei zum Laden auswählen
if args.weights.lower() == "coco":
    weights_path = WEIGHTSS_PATH

```

```

# Gewichte Datei herunterladen
if not os.path.exists(weights_path):
    utils.download_trained_weights(weights_path)
elif args.weights.lower() == "last":
    # letzte trainierte Gewichte finden oder das letzte trainierte Modell
    ↵mit den Gewichten finden
    weights_path = model.find_last()
elif args.weights.lower() == "imagenet":
    # Das neue Modell wird mit den Gewichten von Imagenetmodell
    ↵Initialisieren
    weights_path = model.get_imagenet_weights()
else:
    weights_path = args.weights

# Gewichte laden
print("Gewichte laden ", weights_path)
if args.weights.lower() == "coco":

    model.load_weights(weights_path, by_name=True, exclude=[
        "mrcnn_class_logits", "mrcnn_bbox_fc",
        "mrcnn_bbox", "mrcnn_mask"])
else:
    model.load_weights(weights_path, by_name=True)

# Trainieren or evaluieren
if args.command == "train":
    train(model)
elif args.command == "splash":
    detect_and_color_splash(model, image_path=args.image,
                           video_path=args.video)
else:
    print("{} ist nicht erkannt. "
          "benutze 'trainieren' oder 'splash'".format(args.command))

```

10.5 Visualisierungscode der Daten und des Modells:

10.5.1 Daten in jupyterlab

Datenvisualisierung

April 30, 2022

```
[57]: 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
```

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

```
[59]: # 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: 22
 Class Count: 2
 0. BG
 1. Tipvortexcavitation

```
[60]: # 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)
```





```
[61]: # 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)

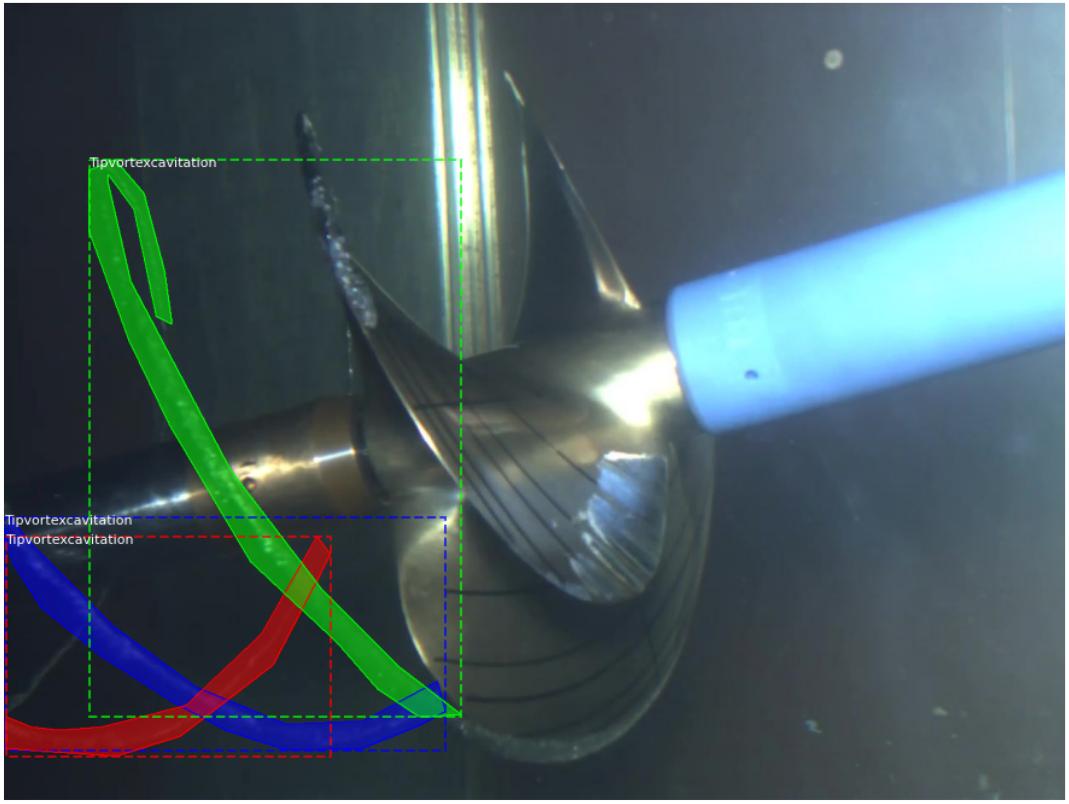
# Bounding box berechnen
bbox = utils.extract_bboxes(mask)

# 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 14 C:\Users\majd4\Desktop\Bachelorarbeit\Bachelor-Arbeit-Daten\MaskRCN
NProjekt\MaskRCNN_2\Mask_RCNN\datasets\Tipvortextcavitation\train\Stb_Gesamt0001
13-09-26 14-39-46-2 04.jpg
image           shape: (960, 1280, 3)      min:  19.00000  max:
255.00000  uint8
mask           shape: (960, 1280, 3)      min:  0.00000  max:
1.00000  bool
class_ids       shape: (3,)                  min:  1.00000  max:
1.00000  int32
bbox           shape: (3, 4)                 min:  2.00000  max:
907.00000  int32
```



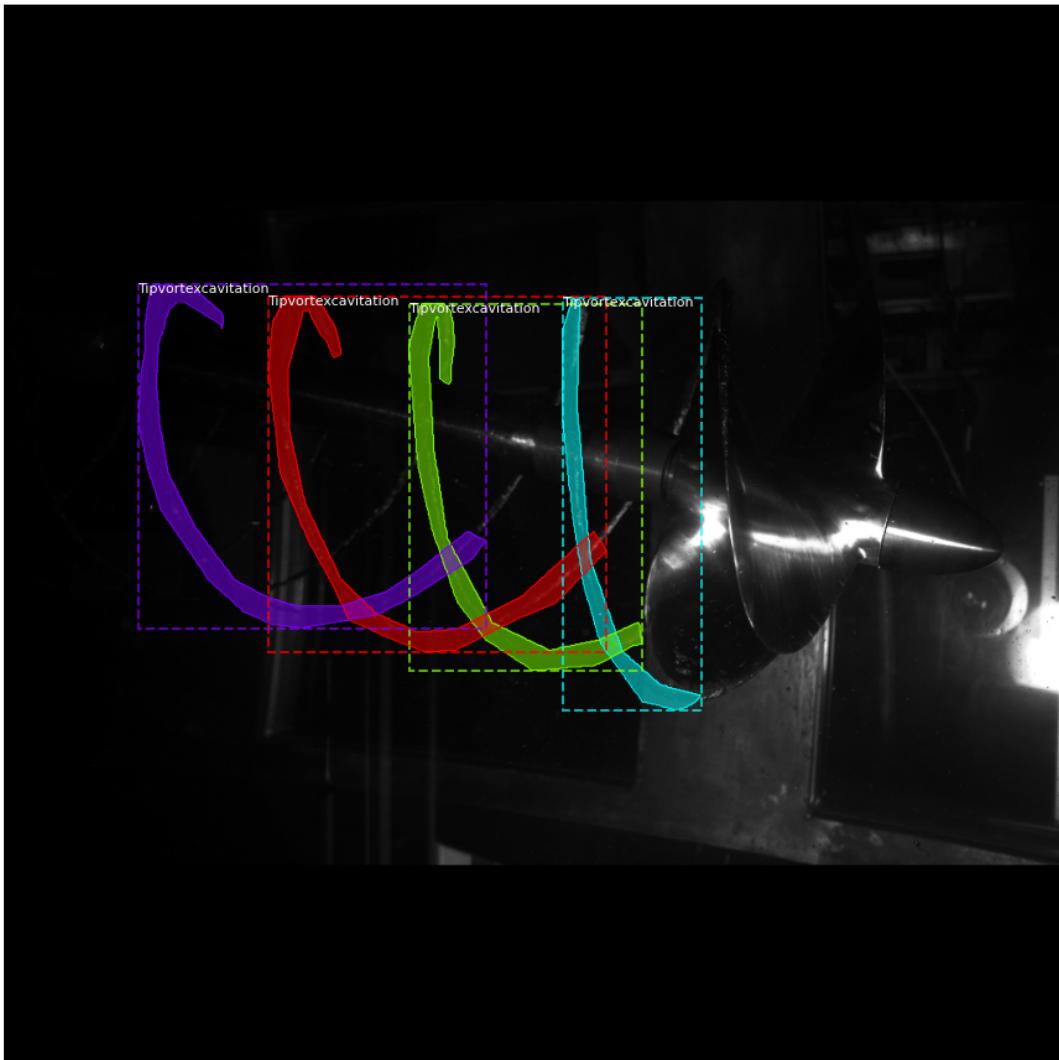
```
[62]: # zufälliges Bild und zufällige Maske hochladen
image_id = np.random.choice(dataset.image_ids, 1)[0]
image = dataset.load_image(image_id)
mask, class_ids = dataset.load_mask(image_id)
original_shape = image.shape
# Größe ändern.
image, window, scale, padding, _ = utils.resize_image(
    image,
    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: 2 C:\Users\majd4\Desktop\Bachelorarbeit\Bachelor-Arbeit-Daten\MaskRCNN  
NProjekt\MaskRCNN_2\Mask_RCNN\datasets\Tipvortexcavitation\train\Neue Videos_000  
0000001_2021_11_17_15_29_16_979_2021_11_17_14_29_16_968_219168312474_225394.png  
Original shape: (1216, 1936, 3)  
image shape: (1024, 1024, 3) min: 0.00000 max:  
255.00000 uint8  
mask shape: (1024, 1024, 4) min: 0.00000 max:  
1.00000 bool  
class_ids shape: (4,) min: 1.00000 max:  
1.00000 int32  
bbox shape: (4, 4) min: 130.00000 max:  
682.00000 int32
```



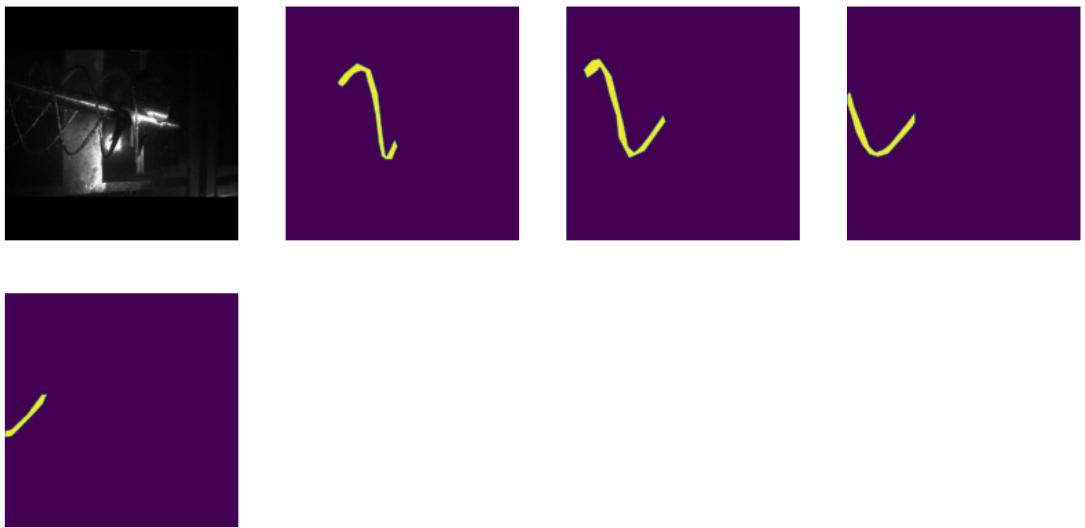
```
[63]: image_id = np.random.choice(dataset.image_ids, 1)[0]
image, image_meta, class_ids, bbox, mask = modellib.load_image_gt(
    dataset, config, image_id, use_mini_mask=False)

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

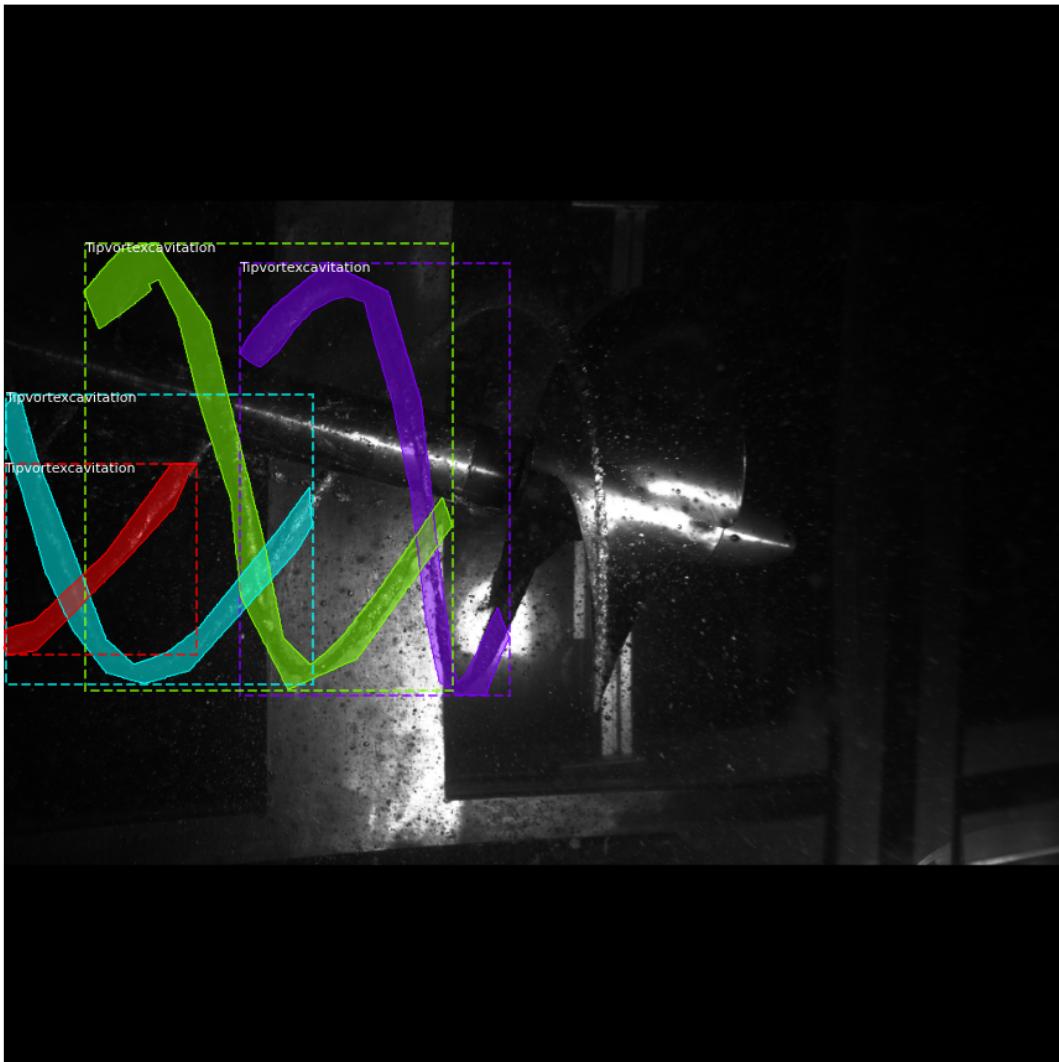
display_images([image]+[mask[:, :, i] for i in range(min(mask.shape[-1], 7))])
```

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: (4,)          min: 1.00000 max:
1.00000 int32
bbox                 shape: (4, 4)        min: 0.00000 max:
668.00000 int32
mask                 shape: (1024, 1024, 4) min: 0.00000 max:
1.00000 bool
```



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



```
[65]: 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.

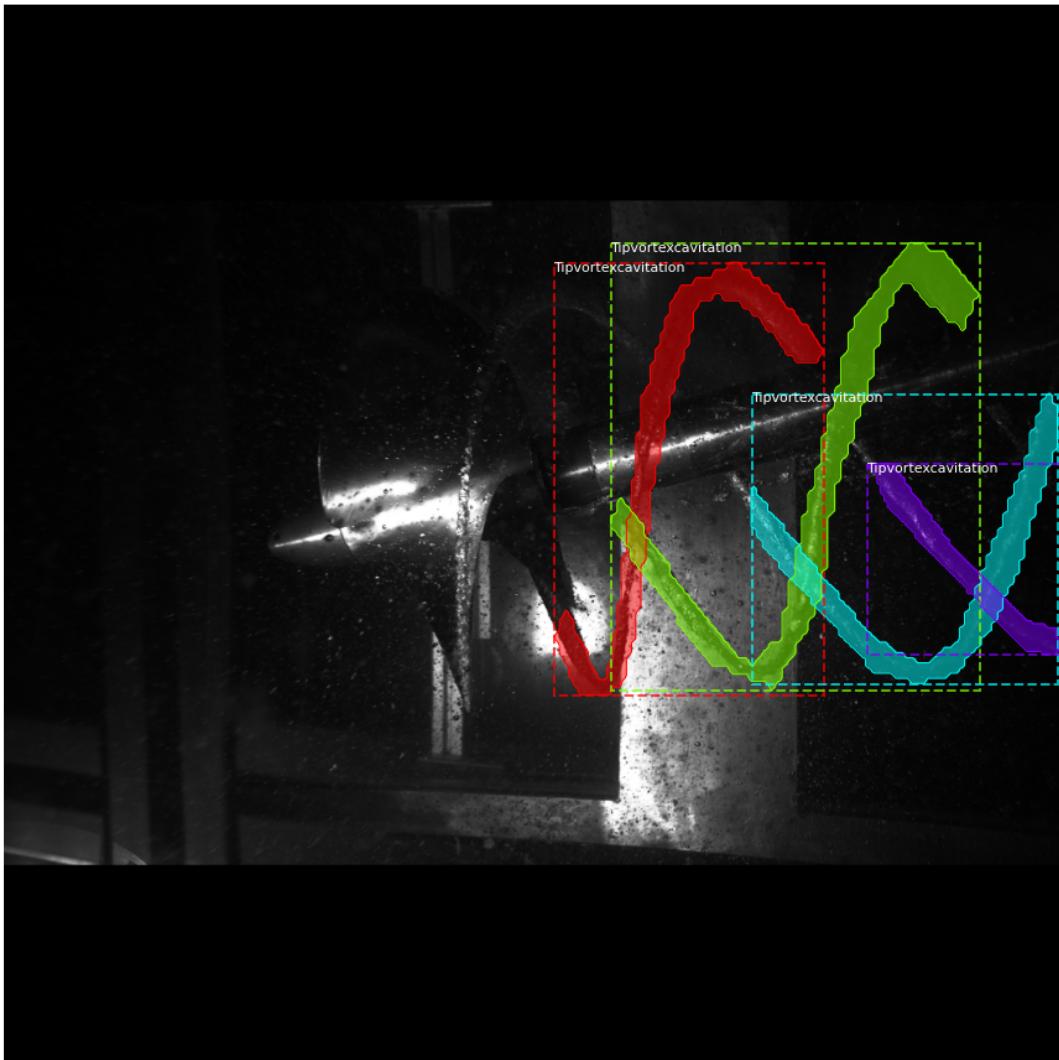
mask shape: (56, 56, 4) min: 0.00000 max:
1.00000 bool

C:\Users\majd4\anaconda3\envs\Matterprot_MaskRCNN\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)
```



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



```
[67]: # 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

```

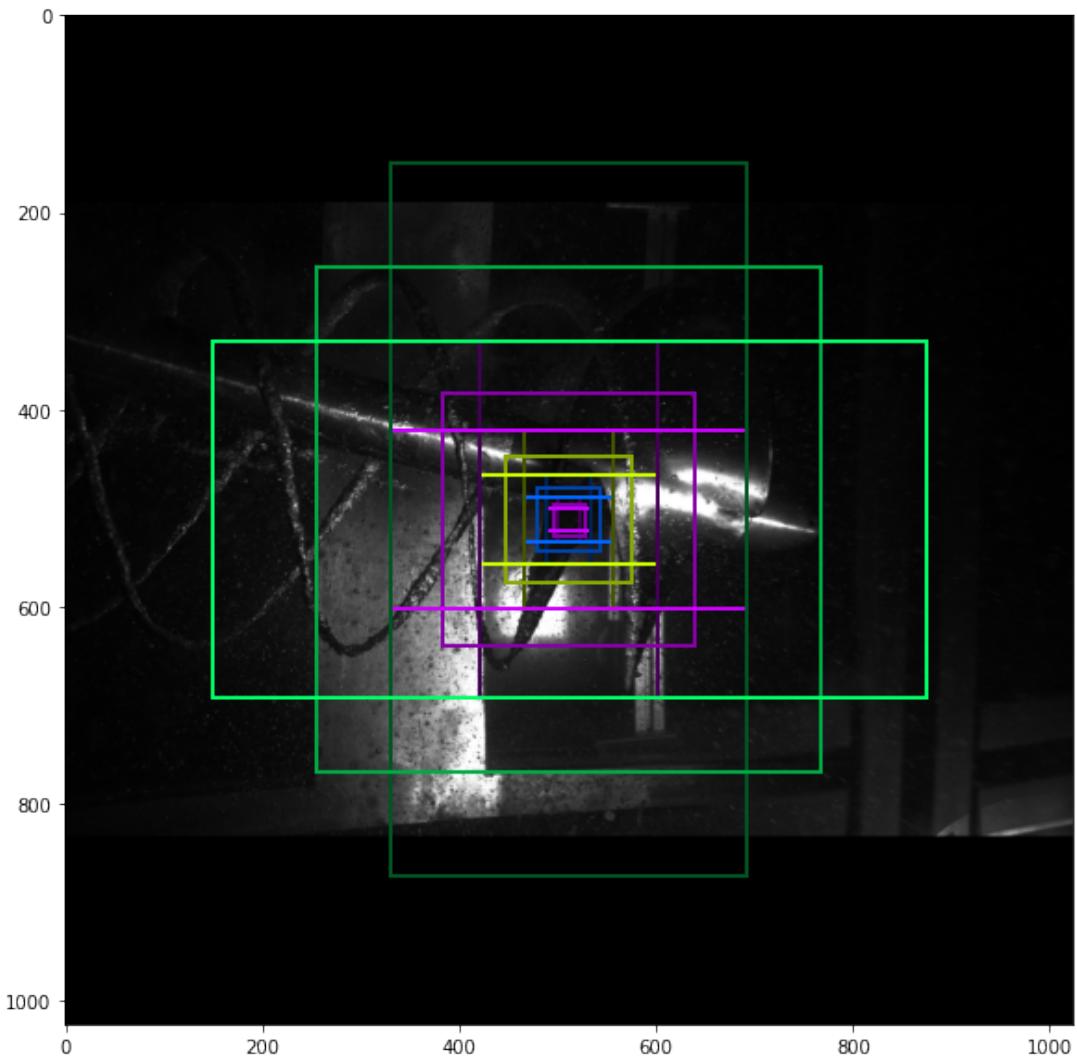
[68]: # ein zufälliges Bild zeichnen und laden
image_id = np.random.choice(dataset.image_ids, 1)[0]
image, image_meta, _, _, _ = modellib.load_image_gt(dataset, config, image_id)
fig, ax = plt.subplots(1, figsize=(10, 10))
ax.imshow(image)
levels = len(backbone_shapes)

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]



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

```
[70]: 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_MaskRCNN\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_MaskRCNN\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_MaskRCNN\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_MaskRCNN\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: -3.30287 max:
3.25301 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:
860.00000 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: -2.50199 max:
3.14883 float64
image_id: 15 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 05.jpg

```

```
[71]: 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)

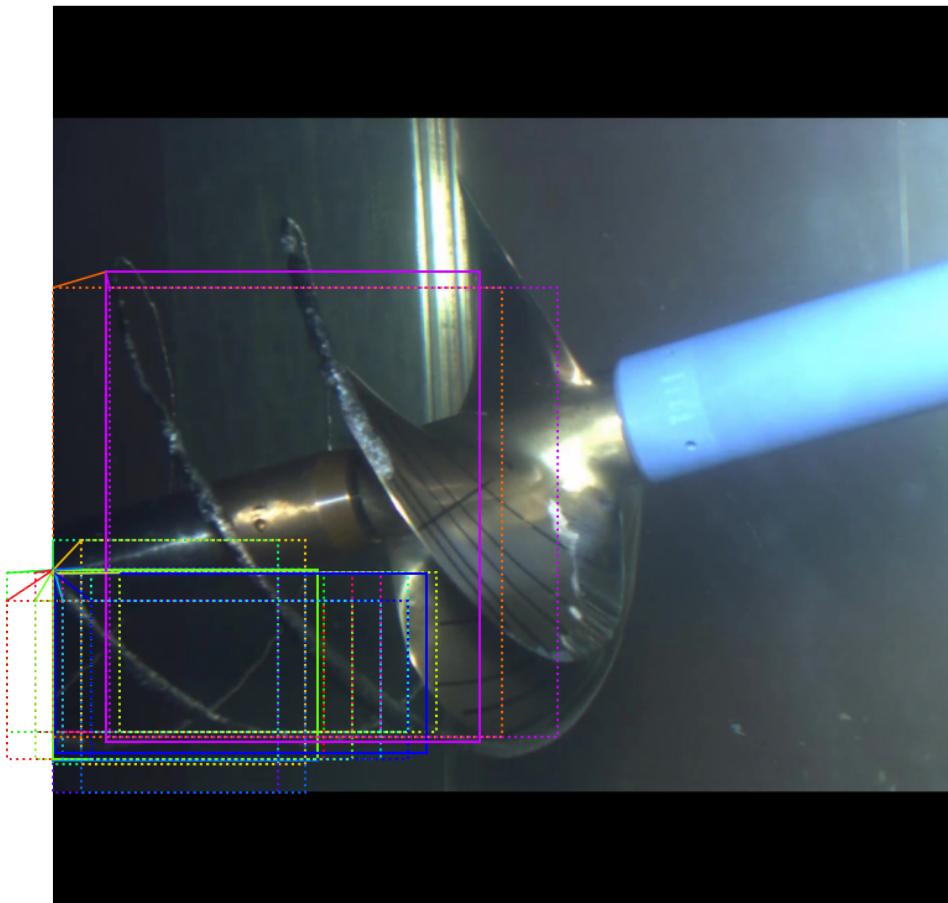
# liste für positive Rahmen bekommen
positive_anchor_ids = np.where(rpn_match[b] == 1)[0]
print("Positive anchors: {}".format(len(positive_anchor_ids)))
negative_anchor_ids = np.where(rpn_match[b] == -1)[0]
print("Negative anchors: {}".format(len(negative_anchor_ids)))
neutral_anchor_ids = np.where(rpn_match[b] == 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("{}: {}".format(c[:20], n))

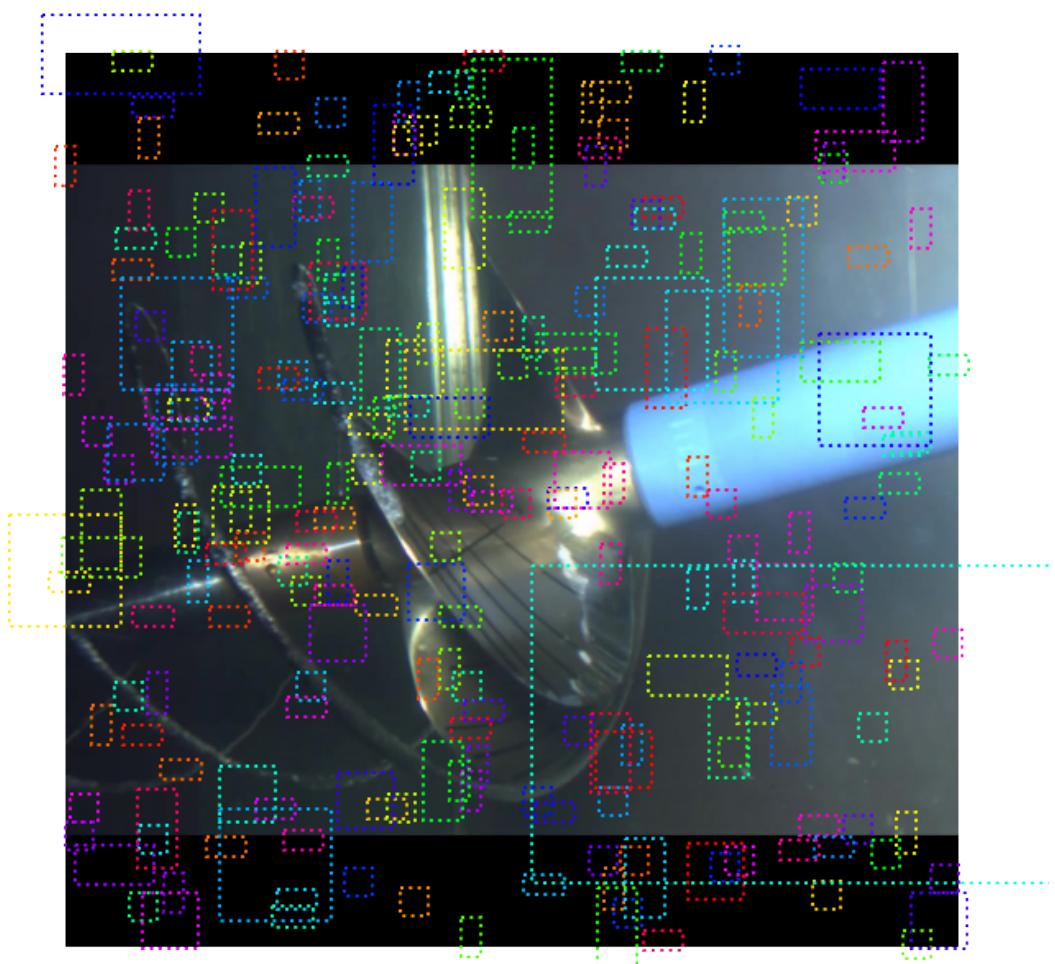
# Positive Rahmen anzeigen. Positiv, bedeutet, dass der Rahmen ein Objekt
# beinhaltet
fig, ax = plt.subplots(1, figsize=(16, 16))
visualize.draw_boxes(sample_image, boxes=anchors[positive_anchor_ids],
                     refined_boxes=refined_anchors, ax=ax)
```

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

```
1322.03867 float64
refined_anchors           shape: (15, 4)          min: 0.00000 max:
859.00000 float32
Positive anchors: 15
Negative anchors: 241
Neutral anchors: 261632
BG                      : 135
Tipvortexexcavation     : 65
```



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



10.5.2 Modell in jupyterlab(resnet50)

Resnet50

April 23, 2022

```
[79]: 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")
```

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

```
[81]: # config wird die Funktionsweise der TipvortexcavitationConfig funktion
    ↪übernommen und
# einpaar werte überschrieben
class InferenceConfig(config.__class__):
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

config = InferenceConfig()
```

```
config.display()
```

Configurations:

BACKBONE	resnet50
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	0.02
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	Tipvortexexcavitation
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   10
WEIGHT_DECAY       0.0001
```

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

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

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

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

```
Images: 14
Classes: ['BG', 'Tipvortexcavitation']
```

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

```
[86]: model.keras_model.summary()
```

```
-----
Layer (type)                 Output Shape            Param #     Connected to
=====
=====
input_image (InputLayer)      (None, None, None, 3 0
-----
zero_padding2d_4 (ZeroPadding2D (None, None, None, 3 0
input_image[0] [0]
-----
conv1 (Conv2D)                (None, None, None, 6 9472
zero_padding2d_4[0] [0]
-----
```

```
bn_conv1 (BatchNorm)           (None, None, None, 6 256      conv1[0] [0]
-----
activation_121 (Activation)   (None, None, None, 6 0       bn_conv1[0] [0]
-----
max_pooling2d_4 (MaxPooling2D) (None, None, None, 6 0
activation_121[0] [0]
-----
res2a_branch2a (Conv2D)       (None, None, None, 6 4160
max_pooling2d_4[0] [0]
-----
bn2a_branch2a (BatchNorm)     (None, None, None, 6 256
res2a_branch2a[0] [0]
-----
activation_122 (Activation)   (None, None, None, 6 0
bn2a_branch2a[0] [0]
-----
res2a_branch2b (Conv2D)       (None, None, None, 6 36928
activation_122[0] [0]
-----
bn2a_branch2b (BatchNorm)     (None, None, None, 6 256
res2a_branch2b[0] [0]
-----
activation_123 (Activation)   (None, None, None, 6 0
bn2a_branch2b[0] [0]
-----
res2a_branch2c (Conv2D)       (None, None, None, 2 16640
activation_123[0] [0]
-----
res2a_branch1 (Conv2D)        (None, None, None, 2 16640
max_pooling2d_4[0] [0]
-----
bn2a_branch2c (BatchNorm)     (None, None, None, 2 1024
res2a_branch2c[0] [0]
-----
bn2a_branch1 (BatchNorm)      (None, None, None, 2 1024
res2a_branch1[0] [0]
```

```
-----  
add_49 (Add)           (None, None, None, 2 0  
bn2a_branch2c[0] [0]  
bn2a_branch1[0] [0]  
  
-----  
res2a_out (Activation)      (None, None, None, 2 0      add_49[0] [0]  
  
-----  
res2b_branch2a (Conv2D)      (None, None, None, 6 16448      res2a_out[0] [0]  
  
-----  
bn2b_branch2a (BatchNorm)    (None, None, None, 6 256  
res2b_branch2a[0] [0]  
  
-----  
activation_124 (Activation)    (None, None, None, 6 0  
bn2b_branch2a[0] [0]  
  
-----  
res2b_branch2b (Conv2D)      (None, None, None, 6 36928  
activation_124[0] [0]  
  
-----  
bn2b_branch2b (BatchNorm)    (None, None, None, 6 256  
res2b_branch2b[0] [0]  
  
-----  
activation_125 (Activation)    (None, None, None, 6 0  
bn2b_branch2b[0] [0]  
  
-----  
res2b_branch2c (Conv2D)      (None, None, None, 2 16640  
activation_125[0] [0]  
  
-----  
bn2b_branch2c (BatchNorm)    (None, None, None, 2 1024  
res2b_branch2c[0] [0]  
  
-----  
add_50 (Add)           (None, None, None, 2 0  
bn2b_branch2c[0] [0]                      res2a_out[0] [0]  
  
-----  
res2b_out (Activation)      (None, None, None, 2 0      add_50[0] [0]  
  
-----
```

```
-----  
res2c_branch2a (Conv2D)           (None, None, None, 6 16448      res2b_out[0] [0]  
-----  
bn2c_branch2a (BatchNorm)        (None, None, None, 6 256  
res2c_branch2a[0] [0]  
-----  
activation_126 (Activation)      (None, None, None, 6 0  
bn2c_branch2a[0] [0]  
-----  
res2c_branch2b (Conv2D)           (None, None, None, 6 36928  
activation_126[0] [0]  
-----  
bn2c_branch2b (BatchNorm)        (None, None, None, 6 256  
res2c_branch2b[0] [0]  
-----  
activation_127 (Activation)      (None, None, None, 6 0  
bn2c_branch2b[0] [0]  
-----  
res2c_branch2c (Conv2D)           (None, None, None, 2 16640  
activation_127[0] [0]  
-----  
bn2c_branch2c (BatchNorm)        (None, None, None, 2 1024  
res2c_branch2c[0] [0]  
-----  
add_51 (Add)                   (None, None, None, 2 0  
bn2c_branch2c[0] [0]                         res2b_out[0] [0]  
-----  
res2c_out (Activation)          (None, None, None, 2 0      add_51[0] [0]  
-----  
res3a_branch2a (Conv2D)           (None, None, None, 1 32896      res2c_out[0] [0]  
-----  
bn3a_branch2a (BatchNorm)        (None, None, None, 1 512  
res3a_branch2a[0] [0]  
-----  
activation_128 (Activation)      (None, None, None, 1 0
```

bn3a_branch2a[0] [0]

res3a_branch2b (Conv2D) (None, None, None, 1 147584
activation_128[0] [0]

bn3a_branch2b (BatchNorm) (None, None, None, 1 512
res3a_branch2b[0] [0]

activation_129 (Activation) (None, None, None, 1 0
bn3a_branch2b[0] [0]

res3a_branch2c (Conv2D) (None, None, None, 5 66048
activation_129[0] [0]

res3a_branch1 (Conv2D) (None, None, None, 5 131584 res2c_out[0] [0]

bn3a_branch2c (BatchNorm) (None, None, None, 5 2048
res3a_branch2c[0] [0]

bn3a_branch1 (BatchNorm) (None, None, None, 5 2048
res3a_branch1[0] [0]

add_52 (Add) (None, None, None, 5 0
bn3a_branch2c[0] [0]
bn3a_branch1[0] [0]

res3a_out (Activation) (None, None, None, 5 0 add_52[0] [0]

res3b_branch2a (Conv2D) (None, None, None, 1 65664 res3a_out[0] [0]

bn3b_branch2a (BatchNorm) (None, None, None, 1 512
res3b_branch2a[0] [0]

activation_130 (Activation) (None, None, None, 1 0
bn3b_branch2a[0] [0]

```
-----  
res3b_branch2b (Conv2D)           (None, None, None, 1 147584  
activation_130[0][0]  
-----  
bn3b_branch2b (BatchNorm)        (None, None, None, 1 512  
res3b_branch2b[0][0]  
-----  
activation_131 (Activation)      (None, None, None, 1 0  
bn3b_branch2b[0][0]  
-----  
res3b_branch2c (Conv2D)           (None, None, None, 5 66048  
activation_131[0][0]  
-----  
bn3b_branch2c (BatchNorm)        (None, None, None, 5 2048  
res3b_branch2c[0][0]  
-----  
add_53 (Add)                   (None, None, None, 5 0  
bn3b_branch2c[0][0]             res3a_out[0][0]  
-----  
res3b_out (Activation)          (None, None, None, 5 0             add_53[0][0]  
-----  
res3c_branch2a (Conv2D)           (None, None, None, 1 65664             res3b_out[0][0]  
-----  
bn3c_branch2a (BatchNorm)        (None, None, None, 1 512  
res3c_branch2a[0][0]  
-----  
activation_132 (Activation)      (None, None, None, 1 0  
bn3c_branch2a[0][0]  
-----  
res3c_branch2b (Conv2D)           (None, None, None, 1 147584  
activation_132[0][0]  
-----  
bn3c_branch2b (BatchNorm)        (None, None, None, 1 512  
res3c_branch2b[0][0]  
-----
```

```
activation_133 (Activation)      (None, None, None, 1 0
bn3c_branch2b[0] [0]

-----
res3c_branch2c (Conv2D)          (None, None, None, 5 66048
activation_133[0] [0]

-----
bn3c_branch2c (BatchNorm)        (None, None, None, 5 2048
res3c_branch2c[0] [0]

-----
add_54 (Add)                   (None, None, None, 5 0
bn3c_branch2c[0] [0]
                                         res3b_out[0] [0]

-----
res3c_out (Activation)          (None, None, None, 5 0
add_54[0] [0]

-----
res3d_branch2a (Conv2D)          (None, None, None, 1 65664
                                         res3c_out[0] [0]

-----
bn3d_branch2a (BatchNorm)        (None, None, None, 1 512
res3d_branch2a[0] [0]

-----
activation_134 (Activation)      (None, None, None, 1 0
bn3d_branch2a[0] [0]

-----
res3d_branch2b (Conv2D)          (None, None, None, 1 147584
activation_134[0] [0]

-----
bn3d_branch2b (BatchNorm)        (None, None, None, 1 512
res3d_branch2b[0] [0]

-----
activation_135 (Activation)      (None, None, None, 1 0
bn3d_branch2b[0] [0]

-----
res3d_branch2c (Conv2D)          (None, None, None, 5 66048
activation_135[0] [0]

-----
bn3d_branch2c (BatchNorm)        (None, None, None, 5 2048
```

```
res3d_branch2c[0] [0]
-----
add_55 (Add)           (None, None, None, 5 0
bn3d_branch2c[0] [0]             res3c_out[0] [0]
-----
res3d_out (Activation)      (None, None, None, 5 0      add_55[0] [0]
-----
res4a_branch2a (Conv2D)      (None, None, None, 2 131328      res3d_out[0] [0]
-----
bn4a_branch2a (BatchNorm)    (None, None, None, 2 1024
res4a_branch2a[0] [0]
-----
activation_136 (Activation)  (None, None, None, 2 0
bn4a_branch2a[0] [0]
-----
res4a_branch2b (Conv2D)      (None, None, None, 2 590080
activation_136[0] [0]
-----
bn4a_branch2b (BatchNorm)    (None, None, None, 2 1024
res4a_branch2b[0] [0]
-----
activation_137 (Activation)  (None, None, None, 2 0
bn4a_branch2b[0] [0]
-----
res4a_branch2c (Conv2D)      (None, None, None, 1 263168
activation_137[0] [0]
-----
res4a_branch1 (Conv2D)       (None, None, None, 1 525312      res3d_out[0] [0]
-----
bn4a_branch2c (BatchNorm)    (None, None, None, 1 4096
res4a_branch2c[0] [0]
-----
bn4a_branch1 (BatchNorm)     (None, None, None, 1 4096
res4a_branch1[0] [0]
```

```
-----  
add_56 (Add)           (None, None, None, 1 0  
bn4a_branch2c[0] [0]  
bn4a_branch1[0] [0]  
-----  
res4a_out (Activation)      (None, None, None, 1 0      add_56[0] [0]  
-----  
res4b_branch2a (Conv2D)      (None, None, None, 2 262400      res4a_out[0] [0]  
-----  
bn4b_branch2a (BatchNorm)    (None, None, None, 2 1024  
res4b_branch2a[0] [0]  
-----  
activation_138 (Activation)   (None, None, None, 2 0  
bn4b_branch2a[0] [0]  
-----  
res4b_branch2b (Conv2D)      (None, None, None, 2 590080  
activation_138[0] [0]  
-----  
bn4b_branch2b (BatchNorm)    (None, None, None, 2 1024  
res4b_branch2b[0] [0]  
-----  
activation_139 (Activation)   (None, None, None, 2 0  
bn4b_branch2b[0] [0]  
-----  
res4b_branch2c (Conv2D)      (None, None, None, 1 263168  
activation_139[0] [0]  
-----  
bn4b_branch2c (BatchNorm)    (None, None, None, 1 4096  
res4b_branch2c[0] [0]  
-----  
add_57 (Add)           (None, None, None, 1 0  
bn4b_branch2c[0] [0]                                res4a_out[0] [0]  
-----  
res4b_out (Activation)      (None, None, None, 1 0      add_57[0] [0]  
-----
```

```
res4c_branch2a (Conv2D)           (None, None, None, 2 262400      res4b_out[0] [0]
-----
bn4c_branch2a (BatchNorm)        (None, None, None, 2 1024
res4c_branch2a[0] [0]
-----
activation_140 (Activation)     (None, None, None, 2 0
bn4c_branch2a[0] [0]
-----
res4c_branch2b (Conv2D)           (None, None, None, 2 590080
activation_140[0] [0]
-----
bn4c_branch2b (BatchNorm)        (None, None, None, 2 1024
res4c_branch2b[0] [0]
-----
activation_141 (Activation)     (None, None, None, 2 0
bn4c_branch2b[0] [0]
-----
res4c_branch2c (Conv2D)           (None, None, None, 1 263168
activation_141[0] [0]
-----
bn4c_branch2c (BatchNorm)        (None, None, None, 1 4096
res4c_branch2c[0] [0]
-----
add_58 (Add)                   (None, None, None, 1 0
bn4c_branch2c[0] [0]
                                         res4b_out[0] [0]
-----
res4c_out (Activation)          (None, None, None, 1 0
add_58[0] [0]
-----
res4d_branch2a (Conv2D)           (None, None, None, 2 262400      res4c_out[0] [0]
-----
bn4d_branch2a (BatchNorm)        (None, None, None, 2 1024
res4d_branch2a[0] [0]
-----
activation_142 (Activation)     (None, None, None, 2 0
bn4d_branch2a[0] [0]
```

```
-----  
res4d_branch2b (Conv2D)           (None, None, None, 2 590080  
activation_142[0] [0]  
  
-----  
bn4d_branch2b (BatchNorm)        (None, None, None, 2 1024  
res4d_branch2b[0] [0]  
  
-----  
activation_143 (Activation)      (None, None, None, 2 0  
bn4d_branch2b[0] [0]  
  
-----  
res4d_branch2c (Conv2D)           (None, None, None, 1 263168  
activation_143[0] [0]  
  
-----  
bn4d_branch2c (BatchNorm)        (None, None, None, 1 4096  
res4d_branch2c[0] [0]  
  
-----  
add_59 (Add)                   (None, None, None, 1 0  
bn4d_branch2c[0] [0]                         res4c_out[0] [0]  
  
-----  
res4d_out (Activation)          (None, None, None, 1 0)           add_59[0] [0]  
  
-----  
res4e_branch2a (Conv2D)           (None, None, None, 2 262400)       res4d_out[0] [0]  
  
-----  
bn4e_branch2a (BatchNorm)        (None, None, None, 2 1024  
res4e_branch2a[0] [0]  
  
-----  
activation_144 (Activation)      (None, None, None, 2 0  
bn4e_branch2a[0] [0]  
  
-----  
res4e_branch2b (Conv2D)           (None, None, None, 2 590080  
activation_144[0] [0]  
  
-----  
bn4e_branch2b (BatchNorm)        (None, None, None, 2 1024  
res4e_branch2b[0] [0]
```

```
-----  
activation_145 (Activation)      (None, None, None, 2 0  
bn4e_branch2b[0] [0]  
  
-----  
res4e_branch2c (Conv2D)          (None, None, None, 1 263168  
activation_145[0] [0]  
  
-----  
bn4e_branch2c (BatchNorm)        (None, None, None, 1 4096  
res4e_branch2c[0] [0]  
  
-----  
add_60 (Add)                   (None, None, None, 1 0  
bn4e_branch2c[0] [0]                         res4d_out[0] [0]  
  
-----  
res4e_out (Activation)          (None, None, None, 1 0           add_60[0] [0]  
  
-----  
res4f_branch2a (Conv2D)          (None, None, None, 2 262400           res4e_out[0] [0]  
  
-----  
bn4f_branch2a (BatchNorm)        (None, None, None, 2 1024  
res4f_branch2a[0] [0]  
  
-----  
activation_146 (Activation)      (None, None, None, 2 0  
bn4f_branch2a[0] [0]  
  
-----  
res4f_branch2b (Conv2D)          (None, None, None, 2 590080  
activation_146[0] [0]  
  
-----  
bn4f_branch2b (BatchNorm)        (None, None, None, 2 1024  
res4f_branch2b[0] [0]  
  
-----  
activation_147 (Activation)      (None, None, None, 2 0  
bn4f_branch2b[0] [0]  
  
-----  
res4f_branch2c (Conv2D)          (None, None, None, 1 263168  
activation_147[0] [0]  
-----
```

bn4f_branch2c (BatchNorm) (None, None, None, 1 4096
res4f_branch2c[0] [0]

add_61 (Add) (None, None, None, 1 0
bn4f_branch2c[0] [0] res4e_out[0] [0]

res4f_out (Activation) (None, None, None, 1 0 add_61[0] [0]

res5a_branch2a (Conv2D) (None, None, None, 5 524800 res4f_out[0] [0]

bn5a_branch2a (BatchNorm) (None, None, None, 5 2048
res5a_branch2a[0] [0]

activation_148 (Activation) (None, None, None, 5 0
bn5a_branch2a[0] [0]

res5a_branch2b (Conv2D) (None, None, None, 5 2359808
activation_148[0] [0]

bn5a_branch2b (BatchNorm) (None, None, None, 5 2048
res5a_branch2b[0] [0]

activation_149 (Activation) (None, None, None, 5 0
bn5a_branch2b[0] [0]

res5a_branch2c (Conv2D) (None, None, None, 2 1050624
activation_149[0] [0]

res5a_branch1 (Conv2D) (None, None, None, 2 2099200 res4f_out[0] [0]

bn5a_branch2c (BatchNorm) (None, None, None, 2 8192
res5a_branch2c[0] [0]

bn5a_branch1 (BatchNorm) (None, None, None, 2 8192
res5a_branch1[0] [0]

```
-----  
add_62 (Add)           (None, None, None, 2 0  
bn5a_branch2c[0] [0]  
bn5a_branch1[0] [0]  
  
-----  
res5a_out (Activation)   (None, None, None, 2 0      add_62[0] [0]  
  
-----  
res5b_branch2a (Conv2D)    (None, None, None, 5 1049088     res5a_out[0] [0]  
  
-----  
bn5b_branch2a (BatchNorm)   (None, None, None, 5 2048  
res5b_branch2a[0] [0]  
  
-----  
activation_150 (Activation)   (None, None, None, 5 0  
bn5b_branch2a[0] [0]  
  
-----  
res5b_branch2b (Conv2D)    (None, None, None, 5 2359808  
activation_150[0] [0]  
  
-----  
bn5b_branch2b (BatchNorm)   (None, None, None, 5 2048  
res5b_branch2b[0] [0]  
  
-----  
activation_151 (Activation)   (None, None, None, 5 0  
bn5b_branch2b[0] [0]  
  
-----  
res5b_branch2c (Conv2D)    (None, None, None, 2 1050624  
activation_151[0] [0]  
  
-----  
bn5b_branch2c (BatchNorm)   (None, None, None, 2 8192  
res5b_branch2c[0] [0]  
  
-----  
add_63 (Add)           (None, None, None, 2 0  
bn5b_branch2c[0] [0]                      res5a_out[0] [0]  
  
-----  
res5b_out (Activation)   (None, None, None, 2 0      add_63[0] [0]  
-----
```

```
-----  
res5c_branch2a (Conv2D)           (None, None, None, 5 1049088      res5b_out[0] [0]  
-----  
bn5c_branch2a (BatchNorm)        (None, None, None, 5 2048  
res5c_branch2a[0] [0]  
-----  
activation_152 (Activation)      (None, None, None, 5 0  
bn5c_branch2a[0] [0]  
-----  
res5c_branch2b (Conv2D)           (None, None, None, 5 2359808  
activation_152[0] [0]  
-----  
bn5c_branch2b (BatchNorm)        (None, None, None, 5 2048  
res5c_branch2b[0] [0]  
-----  
activation_153 (Activation)      (None, None, None, 5 0  
bn5c_branch2b[0] [0]  
-----  
res5c_branch2c (Conv2D)           (None, None, None, 2 1050624  
activation_153[0] [0]  
-----  
bn5c_branch2c (BatchNorm)        (None, None, None, 2 8192  
res5c_branch2c[0] [0]  
-----  
add_64 (Add)                   (None, None, None, 2 0  
bn5c_branch2c[0] [0]                         res5b_out[0] [0]  
-----  
res5c_out (Activation)          (None, None, None, 2 0             add_64[0] [0]  
-----  
fpn_c5p5 (Conv2D)              (None, None, None, 2 524544      res5c_out[0] [0]  
-----  
fpn_p5upsampled (UpSampling2D)  (None, None, None, 2 0             fpn_c5p5[0] [0]  
-----  
fpn_c4p4 (Conv2D)              (None, None, None, 2 262400      res4f_out[0] [0]  
-----
```

```

-----  

fpn_p4add (Add)           (None, None, None, 2 0  

fpn_p5upsampled[0] [0]          fpn_c4p4[0] [0]  

-----  

fpn_p4upsampled (UpSampling2D)  (None, None, None, 2 0      fpn_p4add[0] [0]  

-----  

fpn_c3p3 (Conv2D)           (None, None, None, 2 131328      res3d_out[0] [0]  

-----  

fpn_p3add (Add)             (None, None, None, 2 0  

fpn_p4upsampled[0] [0]          fpn_c3p3[0] [0]  

-----  

fpn_p3upsampled (UpSampling2D) (None, None, None, 2 0      fpn_p3add[0] [0]  

-----  

fpn_c2p2 (Conv2D)           (None, None, None, 2 65792      res2c_out[0] [0]  

-----  

fpn_p2add (Add)             (None, None, None, 2 0  

fpn_p3upsampled[0] [0]          fpn_c2p2[0] [0]  

-----  

fpn_p5 (Conv2D)              (None, None, None, 2 590080      fpn_c5p5[0] [0]  

-----  

fpn_p2 (Conv2D)              (None, None, None, 2 590080      fpn_p2add[0] [0]  

-----  

fpn_p3 (Conv2D)              (None, None, None, 2 590080      fpn_p3add[0] [0]  

-----  

fpn_p4 (Conv2D)              (None, None, None, 2 590080      fpn_p4add[0] [0]  

-----  

fpn_p6 (MaxPooling2D)        (None, None, None, 2 0      fpn_p5[0] [0]  

-----  

rpn_model (Model)            [(None, None, 2), (N 1189394      fpn_p2[0] [0]  

                           fpn_p3[0] [0]  

                           fpn_p4[0] [0]  

                           fpn_p5[0] [0]  

                           fpn_p6[0] [0]

```

```

-----  

rpn_class (Concatenate)      (None, None, 2)      0      rpn_model[1][1]  

                                         rpn_model[2][1]  

                                         rpn_model[3][1]  

                                         rpn_model[4][1]  

                                         rpn_model[5][1]  

-----  

rpn_bbox (Concatenate)      (None, None, 4)      0      rpn_model[1][2]  

                                         rpn_model[2][2]  

                                         rpn_model[3][2]  

                                         rpn_model[4][2]  

                                         rpn_model[5][2]  

-----  

input_anchors (InputLayer)   (None, None, 4)      0  

-----  

ROI (ProposalLayer)         (None, 1000, 4)      0      rpn_class[0][0]  

                                         rpn_bbox[0][0]  

input_anchors[0][0]  

-----  

input_image_meta (InputLayer) (None, 14)          0  

-----  

roi_align_classifier (PyramidRO (None, 1000, 7, 7, 2 0      ROI[0][0]  

input_image_meta[0][0]  

                                         fpn_p2[0][0]  

                                         fpn_p3[0][0]  

                                         fpn_p4[0][0]  

                                         fpn_p5[0][0]  

-----  

mrcnn_class_conv1 (TimeDistribu (None, 1000, 1, 1, 1 12846080  

roi_align_classifier[0][0]  

-----  

mrcnn_class_bn1 (TimeDistribute (None, 1000, 1, 1, 1 4096  

mrcnn_class_conv1[0][0]  

-----  

activation_154 (Activation)    (None, 1000, 1, 1, 1 0  

mrcnn_class_bn1[0][0]  

-----  

mrcnn_class_conv2 (TimeDistribu (None, 1000, 1, 1, 1 1049600

```

```
activation_154[0] [0]
-----
mrcnn_class_bn2 (TimeDistribute (None, 1000, 1, 1, 1 4096
mrcnn_class_conv2[0] [0]
-----
activation_155 (Activation)      (None, 1000, 1, 1, 1 0
mrcnn_class_bn2[0] [0]
-----
pool_squeeze (Lambda)           (None, 1000, 1024)   0
activation_155[0] [0]
-----
mrcnn_class_logits (TimeDistrib (None, 1000, 2)       2050
pool_squeeze[0] [0]
-----
mrcnn_bbox_fc (TimeDistributed) (None, 1000, 8)       8200
pool_squeeze[0] [0]
-----
mrcnn_class (TimeDistributed)   (None, 1000, 2)       0
mrcnn_class_logits[0] [0]
-----
mrcnn_bbox (Reshape)           (None, 1000, 2, 4)   0
mrcnn_bbox_fc[0] [0]
-----
mrcnn_detection (DetectionLayer (None, 100, 6)       0           ROI[0] [0]
mrcnn_class[0] [0]
mrcnn_bbox[0] [0]
input_image_meta[0] [0]
-----
lambda_12 (Lambda)             (None, 100, 4)       0
mrcnn_detection[0] [0]
-----
roi_align_mask (PyramidROIAlign (None, 100, 14, 14, 0           lambda_12[0] [0]
input_image_meta[0] [0]          fpn_p2[0] [0]
                                         fpn_p3[0] [0]
                                         fpn_p4[0] [0]
                                         fpn_p5[0] [0]
```

```
-----  
mrcnn_mask_conv1 (TimeDistribut (None, 100, 14, 14, 590080  
roi_align_mask[0] [0]  
-----  
mrcnn_mask_bn1 (TimeDistributed (None, 100, 14, 14, 1024  
mrcnn_mask_conv1[0] [0]  
-----  
activation_157 (Activation)      (None, 100, 14, 14, 0  
mrcnn_mask_bn1[0] [0]  
-----  
mrcnn_mask_conv2 (TimeDistribut (None, 100, 14, 14, 590080  
activation_157[0] [0]  
-----  
mrcnn_mask_bn2 (TimeDistributed (None, 100, 14, 14, 1024  
mrcnn_mask_conv2[0] [0]  
-----  
activation_158 (Activation)      (None, 100, 14, 14, 0  
mrcnn_mask_bn2[0] [0]  
-----  
mrcnn_mask_conv3 (TimeDistribut (None, 100, 14, 14, 590080  
activation_158[0] [0]  
-----  
mrcnn_mask_bn3 (TimeDistributed (None, 100, 14, 14, 1024  
mrcnn_mask_conv3[0] [0]  
-----  
activation_159 (Activation)      (None, 100, 14, 14, 0  
mrcnn_mask_bn3[0] [0]  
-----  
mrcnn_mask_conv4 (TimeDistribut (None, 100, 14, 14, 590080  
activation_159[0] [0]  
-----  
mrcnn_mask_bn4 (TimeDistributed (None, 100, 14, 14, 1024  
mrcnn_mask_conv4[0] [0]  
-----  
activation_160 (Activation)      (None, 100, 14, 14, 0  
mrcnn_mask_bn4[0] [0]
```

```

-----
mrcnn_mask_deconv (TimeDistribu (None, 100, 28, 28, 262400
activation_160[0][0]

-----
mrcnn_mask (TimeDistributed)      (None, 100, 28, 28, 514
mrcnn_mask_deconv[0][0]
=====

=====
Total params: 44,662,942
Trainable params: 44,603,678
Non-trainable params: 59,264
-----
```

```
[87]: #weights_path = model.find_last()
weights_path = "C:/Users/majd4/Desktop/Bachelorarbeit/Bachelor-Arbeit-Daten/
    ↪MaskRCNNProjekt/MaskRCNN_2/Mask_RCNN/Tipvortex2.h5"
# Gewichte Laden
print("Loading weights ", weights_path)
model.load_weights(weights_path, by_name=True)
```

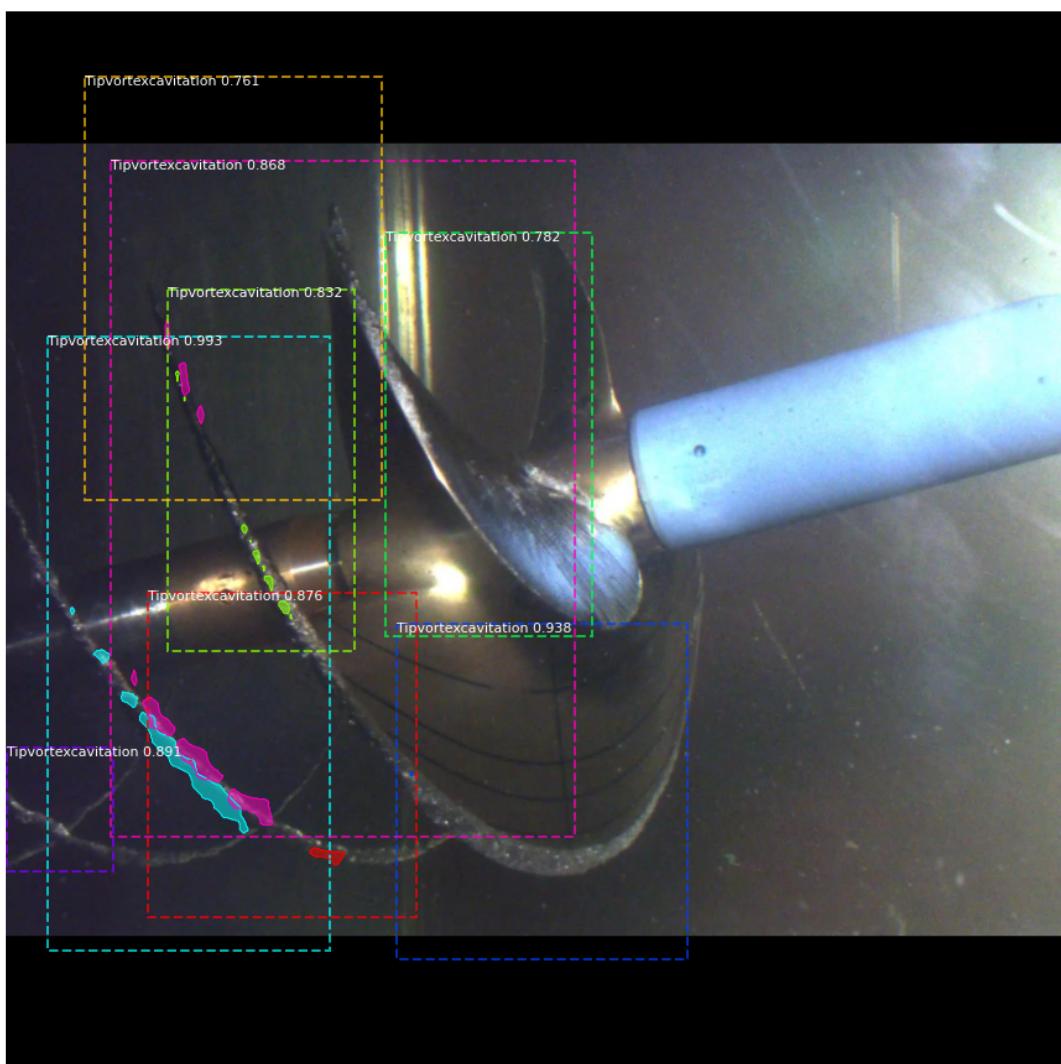
Loading weights C:/Users/majd4/Desktop/Bachelorarbeit/Bachelor-Arbeit-Daten/MaskRCNNProjekt/MaskRCNN_2/Mask_RCNN/Tipvortex2.h5

```
[88]: 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: {}.\n{} ({} {})".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)
```

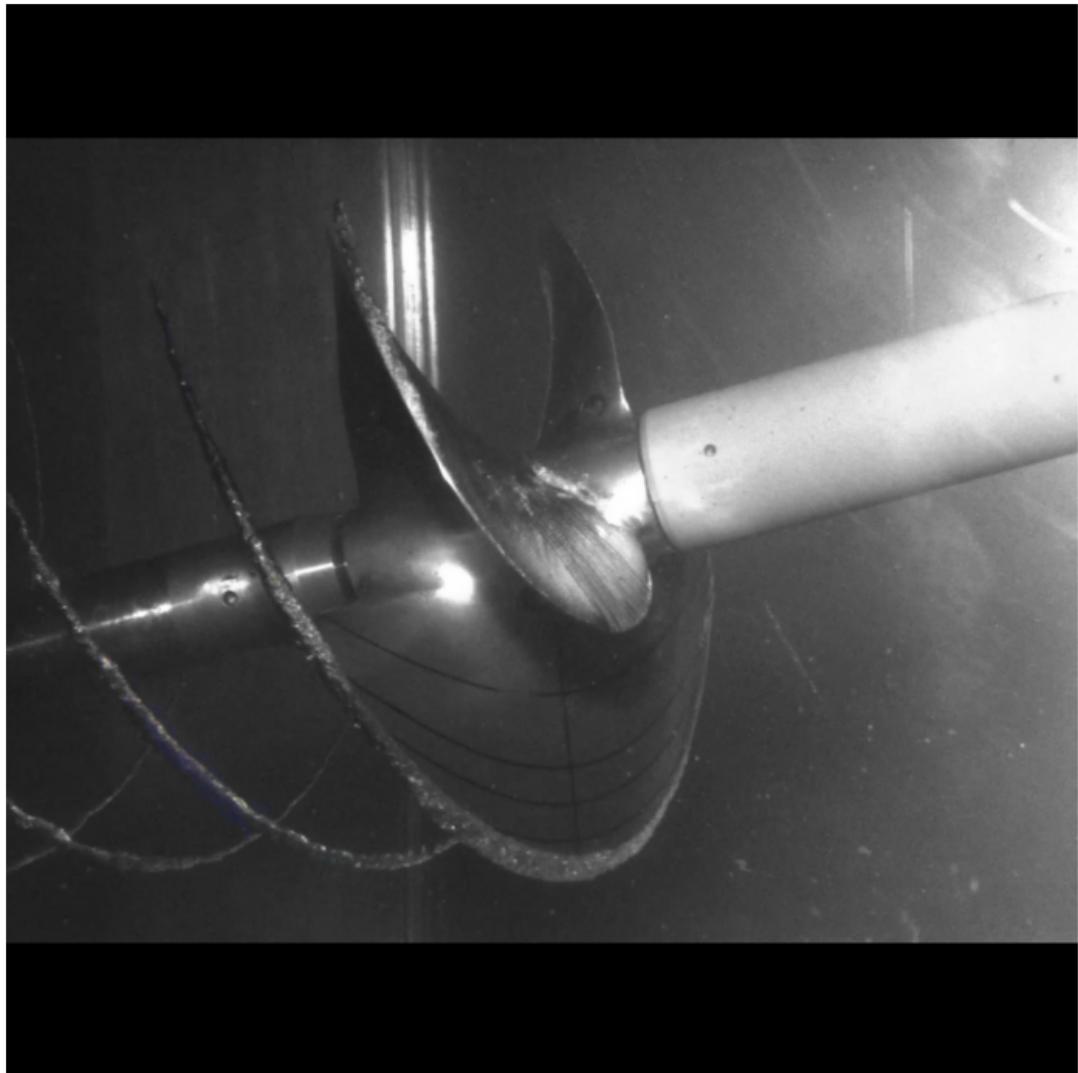
image ID: Tipvortextcavitation.Stb Gesamt0001 13-09-27 10-41-00-2 10.jpg (11)
C:\Users\majd4\Desktop\Bachelorarbeit\Bachelor-Arbeit-Daten\MaskRCNNProjekt\MaskRCNN_2\Mask_RCNN\datasets\Tipvortextcavitation\val\Stb
Gesamt0001 13-09-27 10-41-00-2 10.jpg
Processing 1 images

```
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: (3,)                 min:  1.00000 max:  
1.00000 int32  
gt_bbox                          shape: (3, 4)                min:  3.00000 max:  
838.00000 int32  
gt_mask                          shape: (1024, 1024, 3)      min:  0.00000 max:  
1.00000 bool
```

Predictions



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



```
[90]: # Region Prposal Network Schlägt Regionen vor, wo das gewollte oder erkannte Objekt zu sehen ist
# Dieses Matching, das bedeutet, das Muster wird gefunden und wird eine Quadrate darüber gezeichnet
# Positive anchor 1, -1 das Matching wird nicht gefunden
# and 0 for neutral anchors. die Bounding boxes werden nach dem Muster durchgesucht
```

```

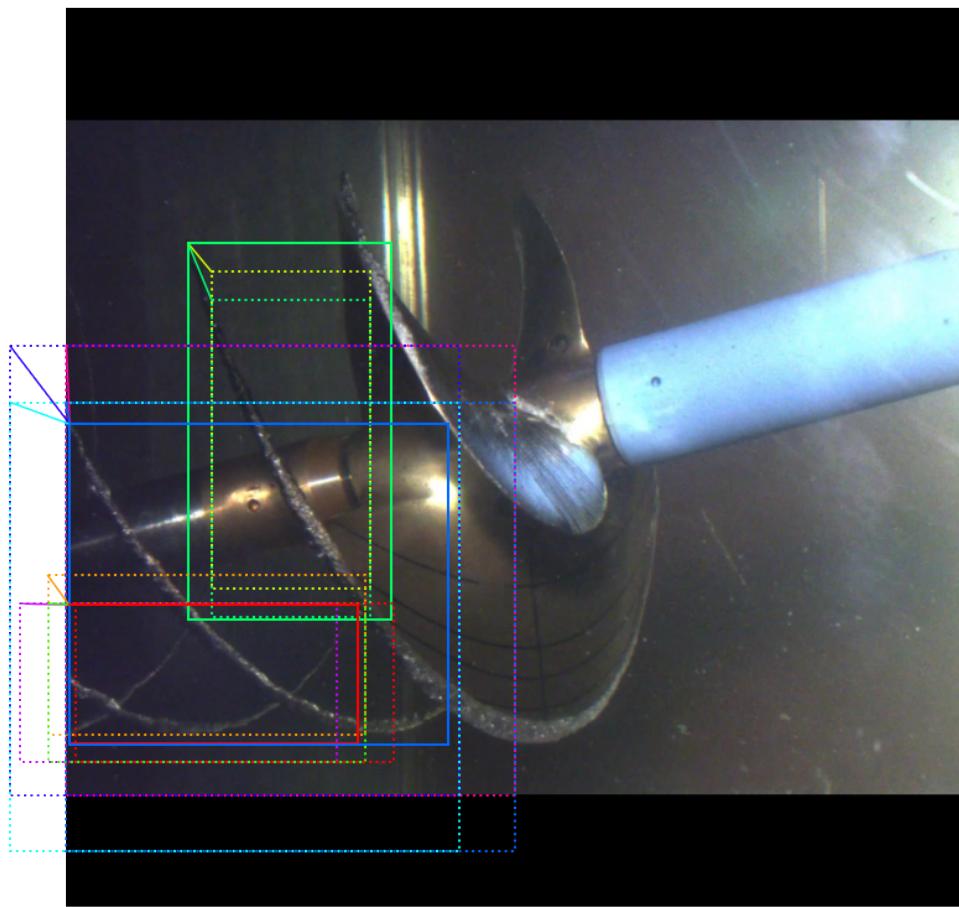
# werden klassifiziert nach positiv anker 1, negitiver anker -1, neutral anker 0
target_rpn_match, target_rpn_bbox = modellib.build_rpn_targets(
    image.shape, model.anchors, gt_class_id, gt_bbox, model.config)
log("target_rpn_match", target_rpn_match)
log("target_rpn_bbox", target_rpn_bbox)
positive_anchor_ix = 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: -1.69214 max:
1.24296 float64		
positive_anchors	shape: (10, 4)	min: -64.00000 max:
960.00000 float64		
negative_anchors	shape: (246, 4)	min: -181.01934 max:
1152.00000 float64		
neutral anchors	shape: (261632, 4)	min: -362.03867 max:
1322.03867 float64		
refined_anchors	shape: (10, 4)	min: 3.00000 max:
838.00000 float32		

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



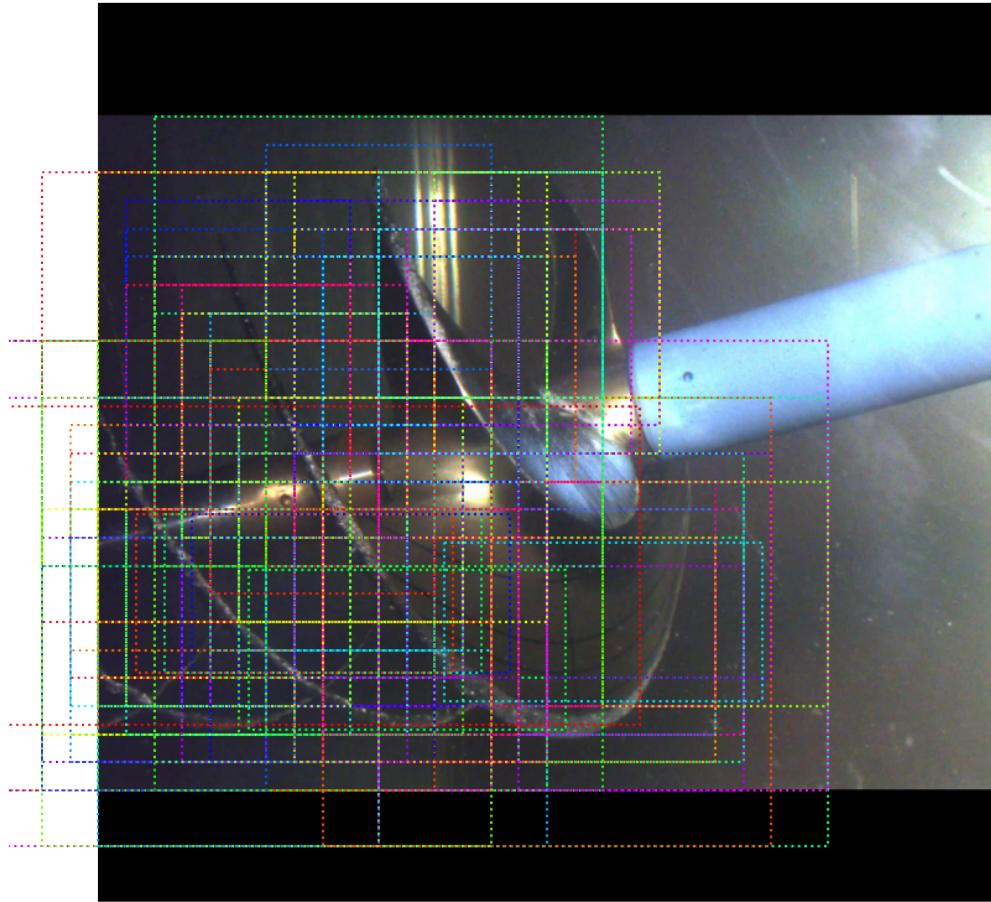
```
[92]: # das RPN-Unterdiagramm ausführen
pillar = model.keras_model.get_layer("ROI").output # node to start searching
from

# 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")
```

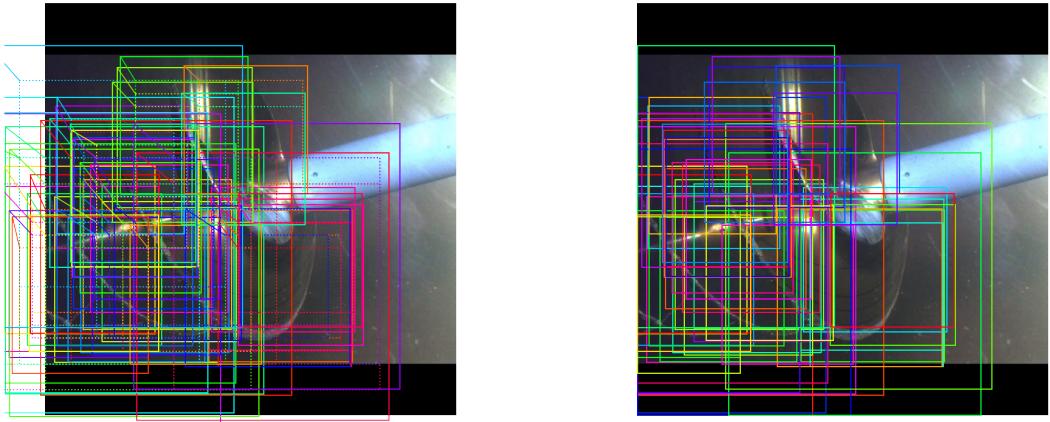
```
rpn = model.run_graph([image], [
    ("rpn_class", model.keras_model.get_layer("rpn_class").output),
    ("pre_nms_anchors", model.ancestor(pillar, "ROI/pre_nms_anchors:0")),
    ("refined_anchors", model.ancestor(pillar, "ROI/refined_anchors:0")),
    ("refined_anchors_clipped", model.ancestor(pillar, "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: -0.35390	max:
	1.29134 float32		
refined_anchors	shape: (1, 6000, 4)	min: -806.05688	max:
	806.27625 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:
	5806.00000 int32		
proposals	shape: (1, 1000, 4)	min: 0.00000	max:
	1.00000 float32		

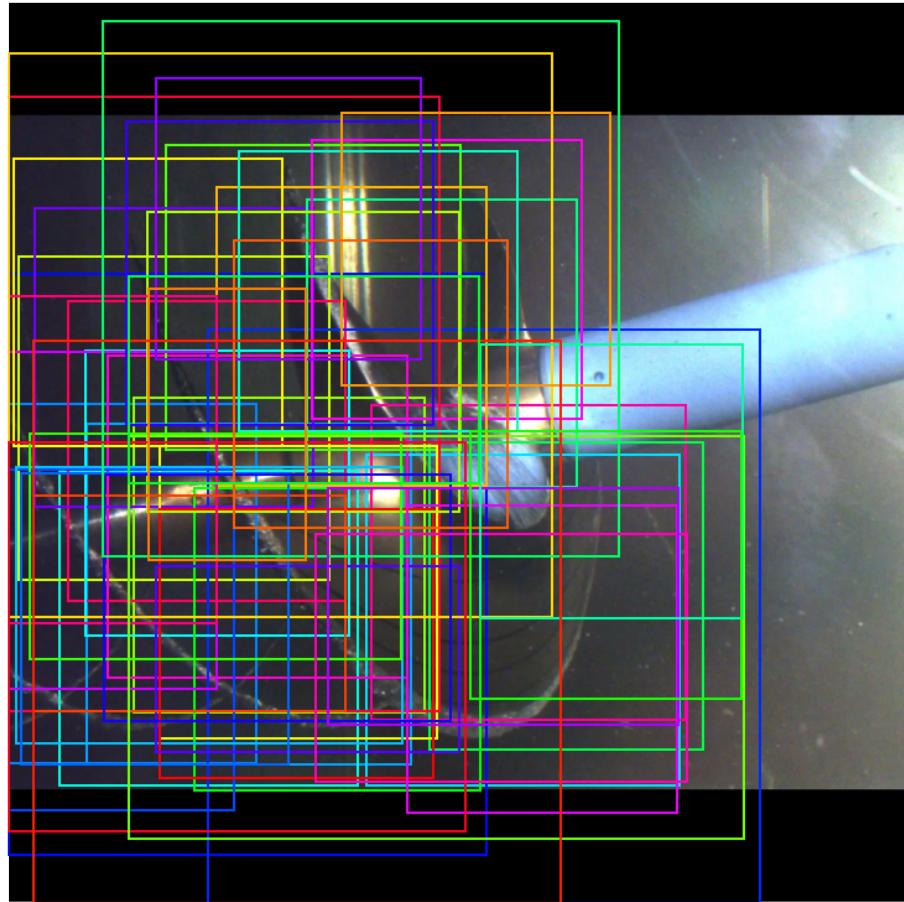
```
[93]: # Top-Anker nach Punktzahl anzeigen (vor der Verfeinerung)
limit = 100
sorted_anchor_ids = np.argsort(rpn['rpn_class'][:, :, 1].flatten())[::-1]
visualize.draw_boxes(image, boxes=model.anchors[sorted_anchor_ids[:limit]], ↪
    ↪ax=get_ax())
```



```
[94]: #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])
```



```
[95]: # 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())
```



```
[96]: #
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),
])

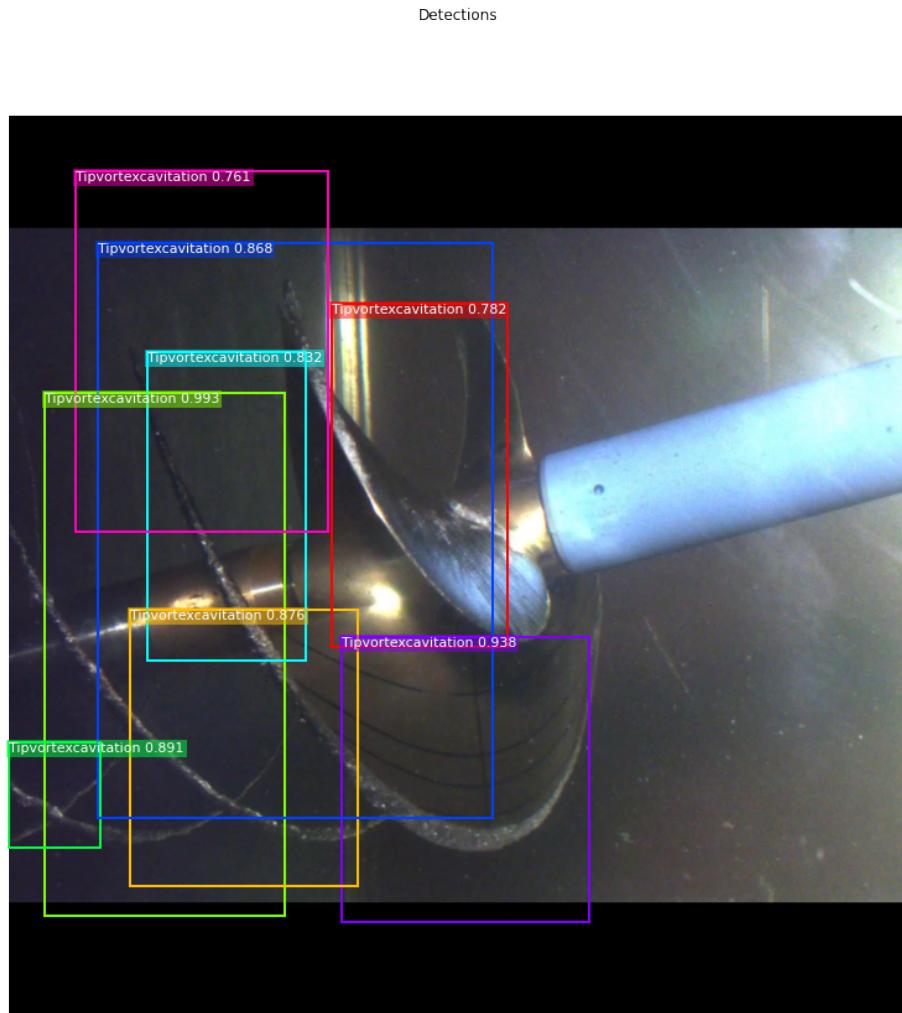
```

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

```
deltas           shape: (1, 1000, 2, 4)      min: -3.16166 max:  
4.34006 float32  
masks           shape: (1, 100, 28, 28, 2)   min: 0.00001 max:  
0.64014 float32  
detections     shape: (1, 100, 6)          min: 0.00000 max:  
1.00000 float32
```

```
[97]: #  
#  
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())
```

8 detections: ['Tipvortexcavitation' 'Tipvortexcavitation' 'Tipvortexcavitation'
'Tipvortexcavitation' 'Tipvortexcavitation' 'Tipvortexcavitation'
'Tipvortexcavitation' 'Tipvortexcavitation']



```
[98]: # Vorschläge sind normalisierte Koordinaten. Skaliere die Vorschläge 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
163 Positive ROIs
[('BG', 837), ('Tipvortexexcavitation', 163)]

```

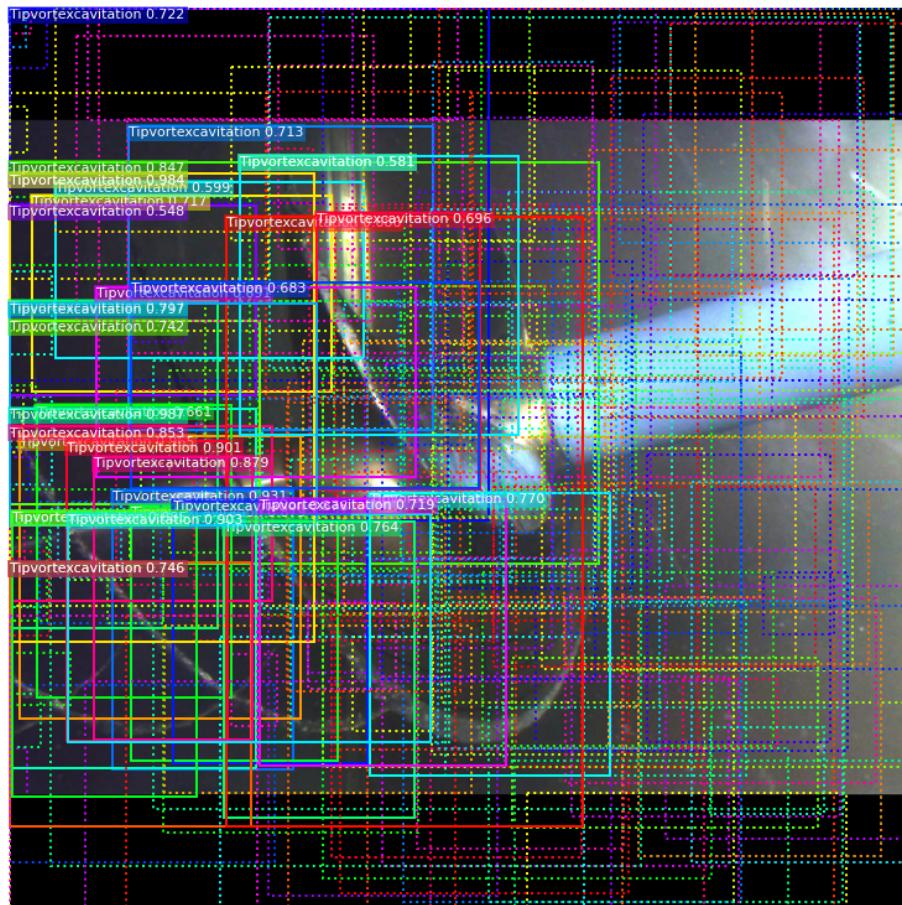
[99]:

```

# 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 ↴
# Tipvortexexcavitation
# 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



```
[100]: # 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)
```

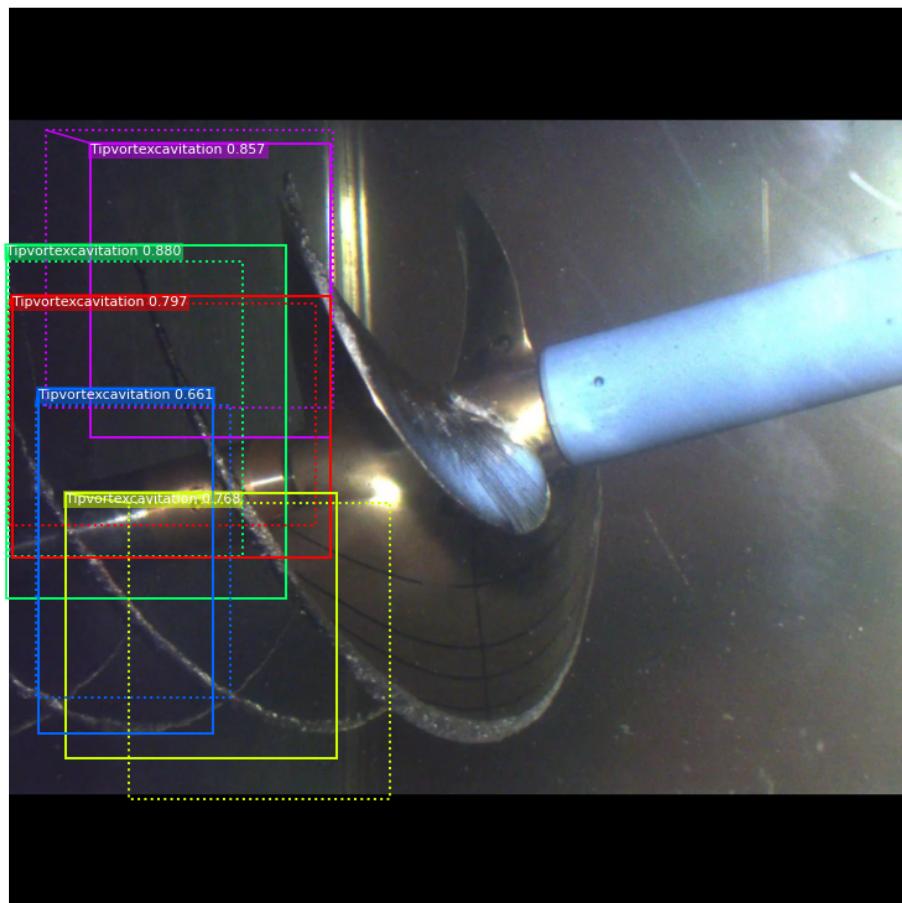
```

# 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: -2.65246 max:
3.35129 float32		
refined_proposals	shape: (1000, 4)	min: -147.00000 max:
1236.00000 int32		

ROIs After Refinement



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

```
Keep 163 detections:
[ 0   1   2   3   5   6   8   13  15  16  17  19  21  23  24  25  26  27
 28  30  31  32  33  34  35  36  40  45  47  48  50  52  53  55  56  58
 60  61  63  64  66  68  70  71  72  73  75  77  78  82  83  84  85  88
 91  92  93  94  96  98  99 103 105 106 108 109 116 120 127 128 131 135
140 141 146 153 154 155 156 157 158 159 160 164 166 176 177 179 181 184
187 191 192 194 202 203 206 221 223 224 229 230 232 236 238 241 243 252
258 260 273 274 282 285 288 297 304 308 315 316 347 349 358 363 364 373
390 391 408 418 427 428 432 447 454 456 470 476 482 493 494 497 522 541
```

```
562 572 602 607 698 709 730 747 748 757 764 774 775 779 780 787 876 961  
989]
```

```
[102]: # 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 114:  
[ 0   1   3   5   6   8   13  16  19  21  23  24  26  27  28  30  31  32  
 34  35  36  40  45  47  52  53  56  58  60  61  63  64  66  68  70  71  
 75  78  83  84  88  91  92  93  94  96  98  99 103 105 106 108 109 116  
127 128 131 140 146 153 154 155 156 158 160 164 176 177 184 187 191 202  
203 206 221 223 224 229 230 236 241 243 258 260 274 308 316 347 363 364  
390 391 408 418 427 428 454 493 494 522 562 572 602 607 698 709 730 747  
748 779 780 787 876 989]
```

```
[103]: # 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 indices  
    # 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))
```

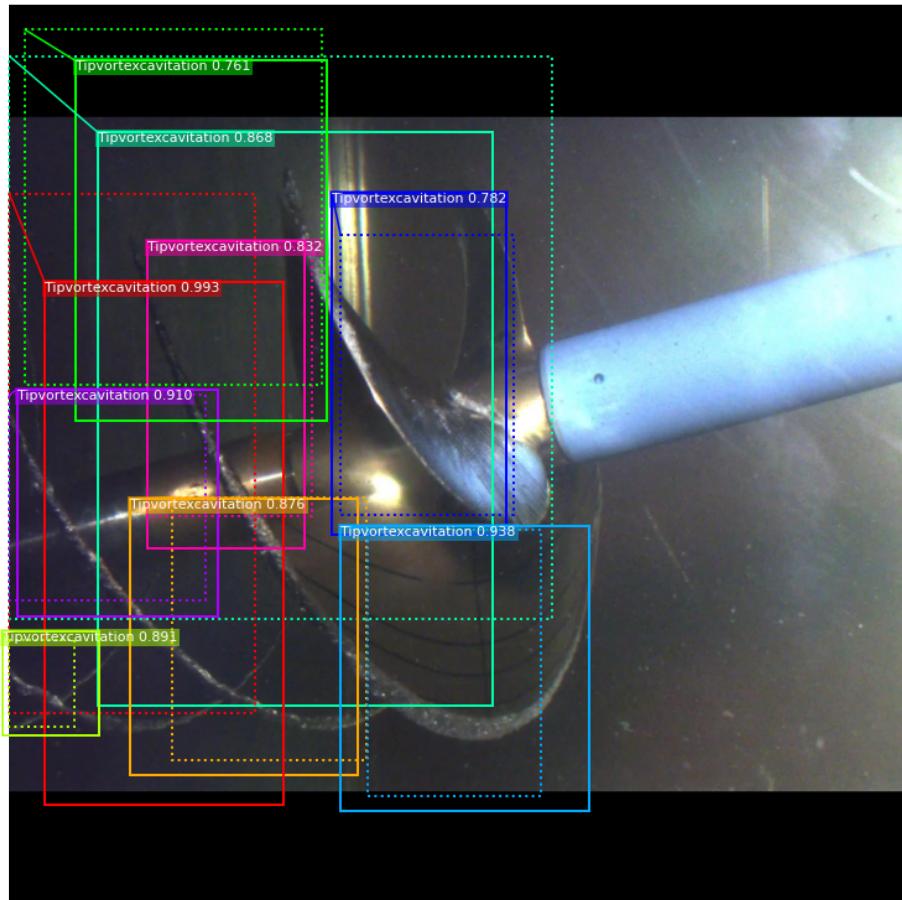
```
Tipvortexcavitation : [ 0   1   3   5   6   8   13  16  19  21  23  24  26  27  
28  30  31  32  
 34  35  36  40  45  47  52  53  56  58  60  61  63  64  66  68  70  71  
 75  78  83  84  88  91  92  93  94  96  98  99 103 105 106 108 109 116  
127 128 131 140 146 153 154 155 156 158 160 164 176 177 184 187 191 202  
203 206 221 223 224 229 230 236 241 243 258 260 274 308 316 347 363 364
```

```
390 391 408 418 427 428 454 493 494 522 562 572 602 607 698 709 730 747  
748 779 780 787 876 989] -> [347 223 493 989 202 32 106 203 747]
```

```
Kept after per-class NMS: 9  
[ 32 106 202 203 223 347 493 747 989]
```

```
[104]: # 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



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



```
[106]: # 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),
])

# 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]

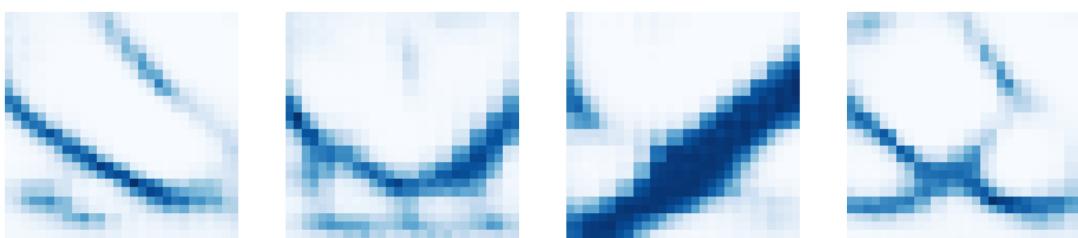
print("{} detections: {}".format(
    det_count, np.array(dataset.class_names)[det_class_ids]))
```

detections shape: (1, 100, 6) min: 0.00000 max:
1.00000 float32
masks shape: (1, 100, 28, 28, 2) min: 0.00001 max:
0.64014 float32
8 detections: ['Tipvortexcavitation' 'Tipvortexcavitation' 'Tipvortexcavitation'
'Tipvortexcavitation' 'Tipvortexcavitation' 'Tipvortexcavitation'
'Tipvortexcavitation' 'Tipvortexcavitation']

```
[107]: # 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: (8, 28, 28) min: 0.00001 max:
0.64014 float32
det_masks shape: (8, 1024, 1024) min: 0.00000 max:
1.00000 bool

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



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



```
[ ]:
```

10.5.3 resnet101

Modellvisualisierung

April 22, 2022

```
[47]: 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 System 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")
```

```
[48]: 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

```
[49]: # config wird die Funktionsweise der TipvortexcavitationConfig funktion
    ↪übernommen und
# ein paar Werte überschrieben
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	0.001
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	Tipvortexexcavitation
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                  50
WEIGHT_DECAY                      0.0001
```

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

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

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

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

Images: 11
Classes: ['BG', 'Tipvortexcavitation']

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

```
[54]: #weights_path = model.find_last()
weights_path = "C:/Users/majd4/Desktop/Bachelorarbeit/Bachelor-Arbeit-Daten/
             MaskRCNNProjekt/MaskRCNN_2/Mask_RCNN/mask_rcnn_tipvortexcavitation_0010.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_0010.h5

```
[55]: 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: {}.\n{} ({}).".format(info["source"], info["id"], image_id,
```

```

        dataset.image_reference(image_id))

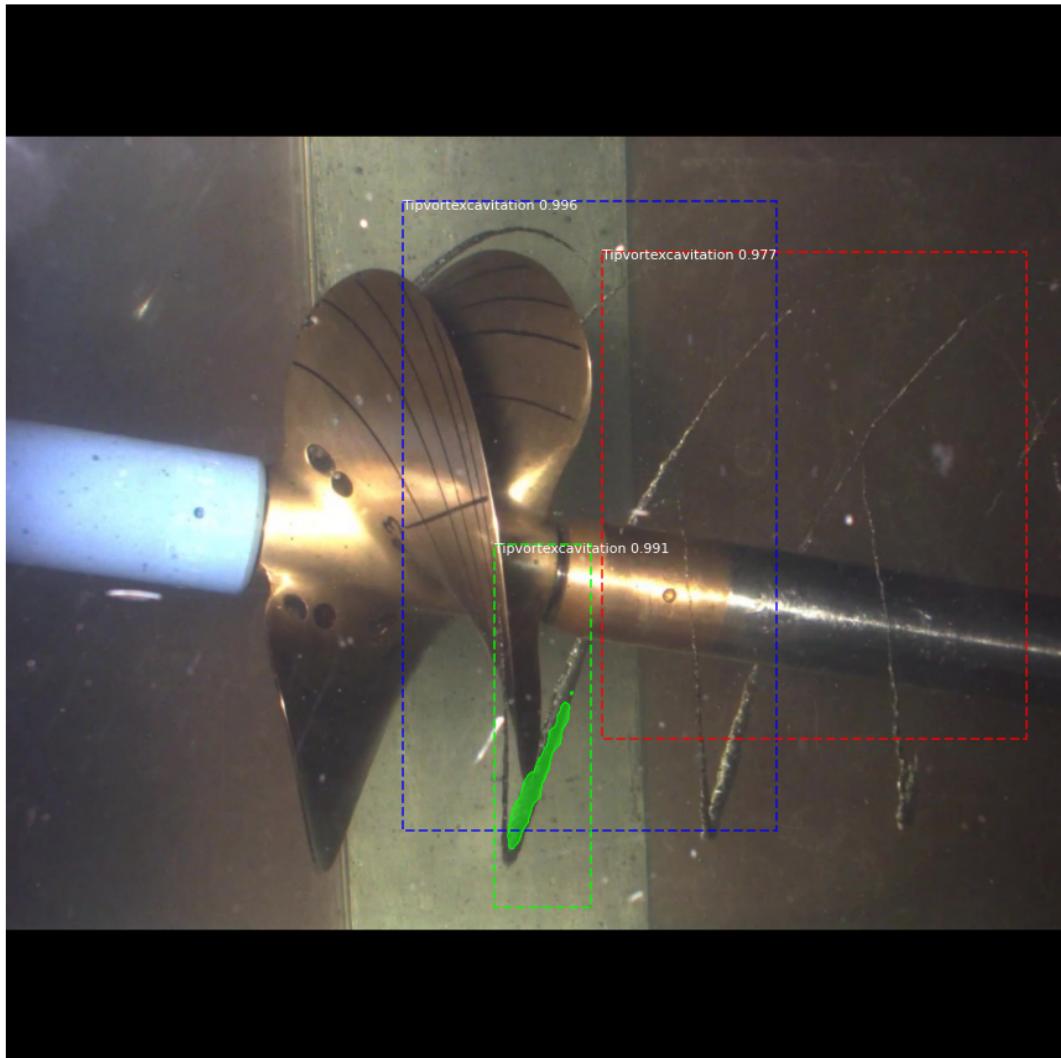
# Objekterkennung ausführen
results = model.detect([image], verbose=1)
# Ergebnisse anzeigen
ax = get_ax(1)
r = results[0]
visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'],
                             dataset.class_names, r['scores'], ax=ax,
                             title="Predictions")
log("gt_class_id", gt_class_id)
log("gt_bbox", gt_bbox)
log("gt_mask", gt_mask)

```

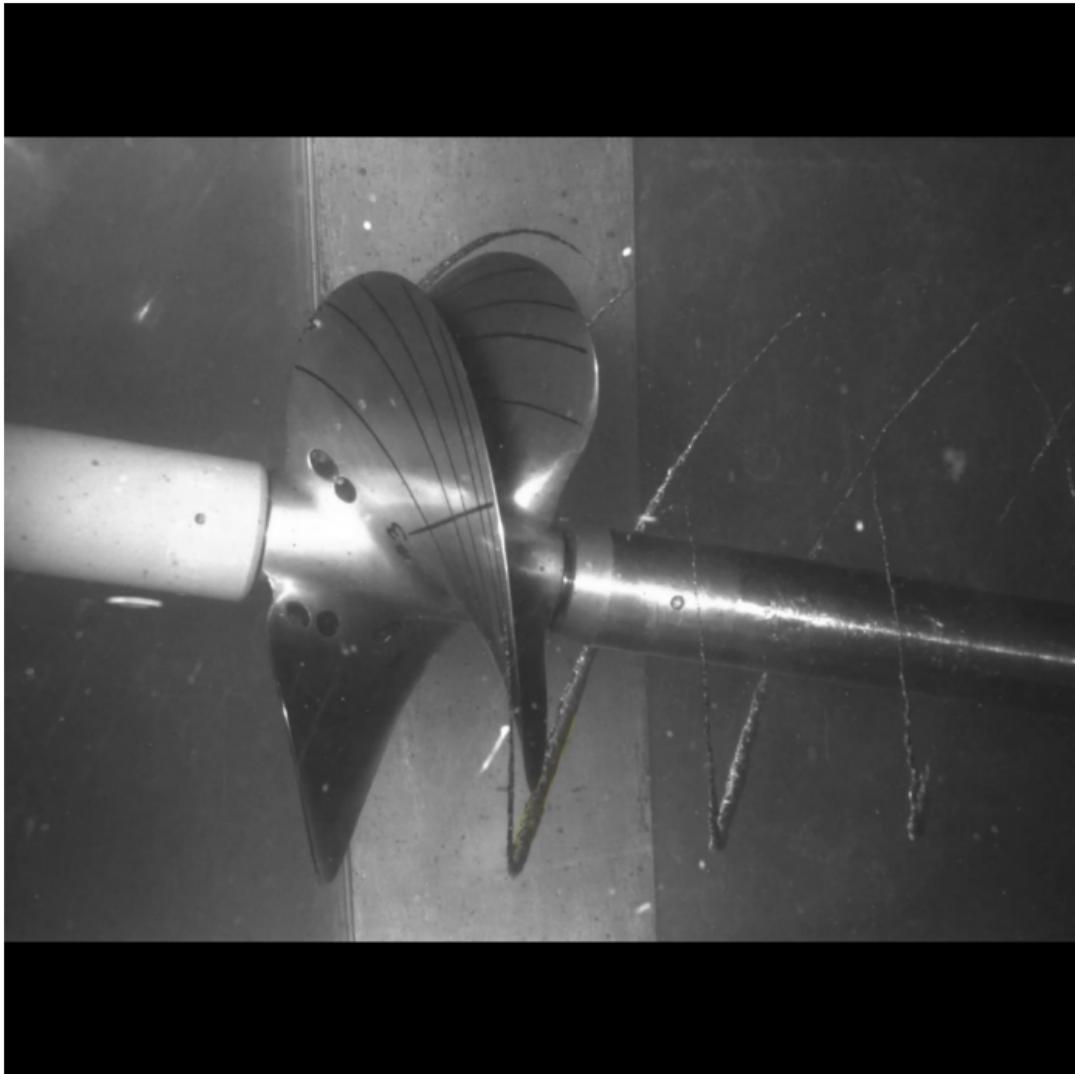
image ID: Tipvortexexcavation.Bb Gesamt0001 13-09-27 08-55-31-1 10.jpg (7)
 C:\Users\majd4\Desktop\Bachelorarbeit\Bachelor-Arbeit-
 Daten\MaskRCNNProjekt\MaskRCNN_2\Mask_RCNN\datasets\Tipvortexexcavation\val\Bb
 Gesamt0001 13-09-27 08-55-31-1 10.jpg
 Processing 1 images

	shape:	min:	max:
image	(1024, 1024, 3)	0.00000	
255.00000 uint8			
molded_images	(1, 1024, 1024, 3)	-123.70000	
151.10000 float64			
image_metas	(1, 14)	0.00000	
1024.00000 int32			
anchors	(1, 261888, 4)	-0.35390	
1.29134 float32			
gt_class_id	(3,)	1.00000	
1.00000 int32			
gt_bbox	(3, 4)	208.00000	
1022.00000 int32			
gt_mask	(1024, 1024, 3)	0.00000	
1.00000 bool			

Predictions



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



```
[57]: # Region Prposal Network Schlägt Regionen vor, wo das gewollte oder erkannte Objekt zu sehen ist
# Dieses Matchcing, 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 boxes 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]
```

```

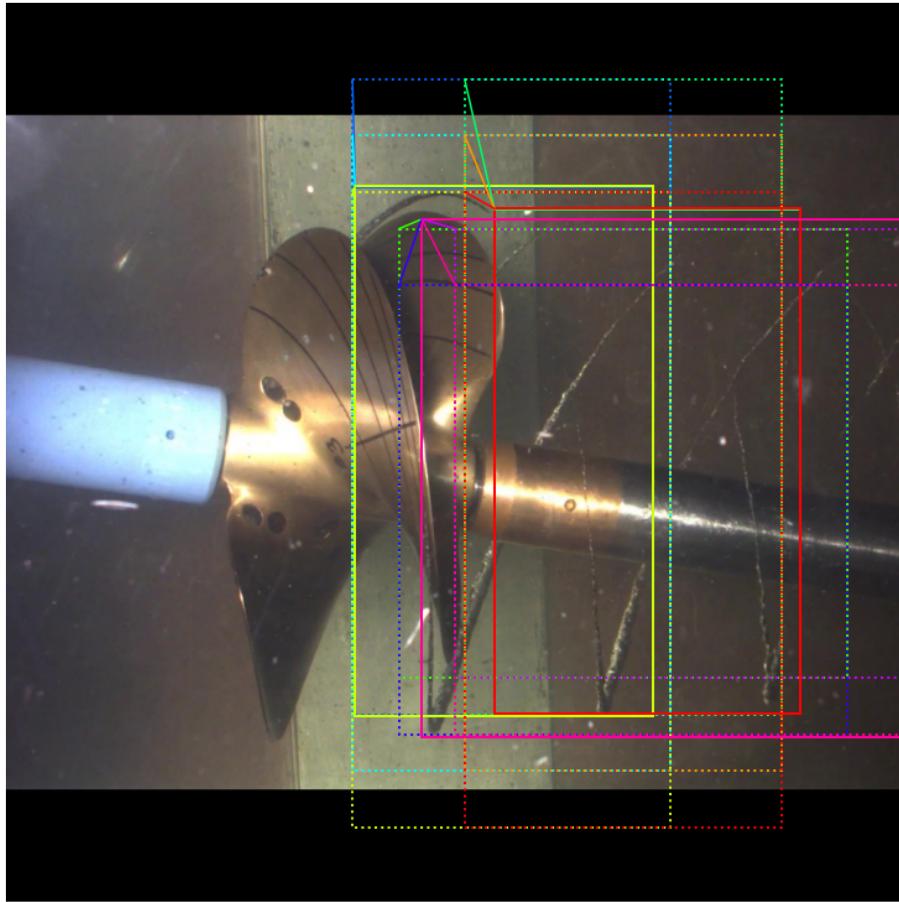
negative_anchor_ix = np.where(target_rpn_match[:] == -1)[0]
neutral_anchor_ix = np.where(target_rpn_match[:] == 0)[0]
positive_anchors = model.anchors[positive_anchor_ix]
negative_anchors = model.anchors[negative_anchor_ix]
neutral_anchors = model.anchors[neutral_anchor_ix]
log("positive_anchors", positive_anchors)
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: -1.15264	max:
1.01508	float64		
positive_anchors	shape: (10, 4)	min: 85.96133	max:
1024.00000	float64		
negative_anchors	shape: (246, 4)	min: -64.00000	max:
1120.00000	float64		
neutral anchors	shape: (261632, 4)	min: -362.03867	max:
1322.03867	float64		
refined_anchors	shape: (10, 4)	min: 207.99997	max:
1022.00000	float32		

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



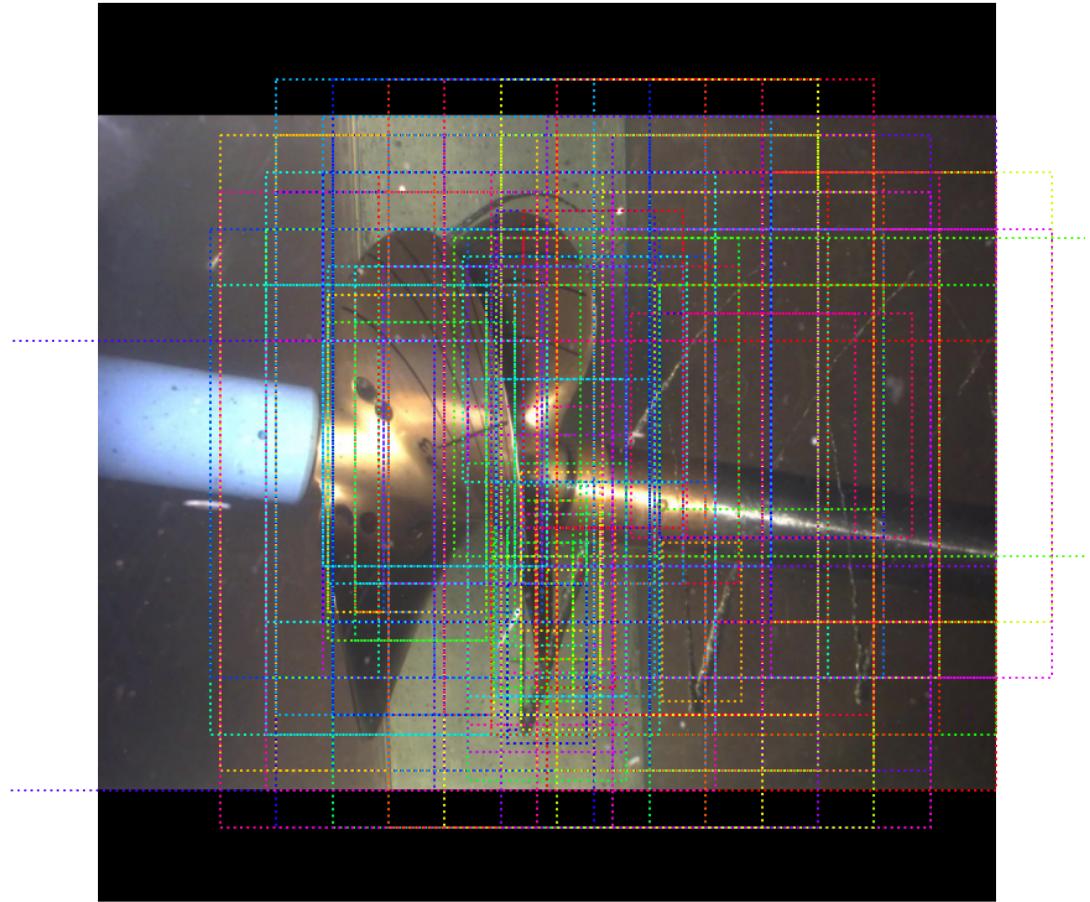
```
[59]: # das RPN-Unterdiagramm ausführen
pillar = model.keras_model.get_layer("ROI").output # node to start searching
from

# 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")
```

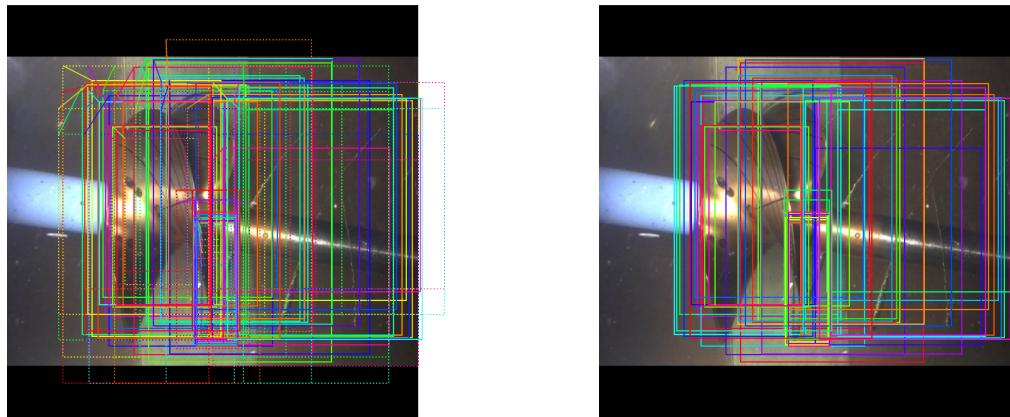
```
rpn = model.run_graph([image], [
    ("rpn_class", model.keras_model.get_layer("rpn_class").output),
    ("pre_nms_anchors", model.ancestor(pillar, "ROI/pre_nms_anchors:0")),
    ("refined_anchors", model.ancestor(pillar, "ROI/refined_anchors:0")),
    ("refined_anchors_clipped", model.ancestor(pillar, "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:	-0.35390	max:
1.29134	float32			
refined_anchors	shape: (1, 6000, 4)	min:	-8.93050	max:
9.96493	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:
4804.00000	int32			
proposals	shape: (1, 1000, 4)	min:	0.00000	max:
1.00000	float32			

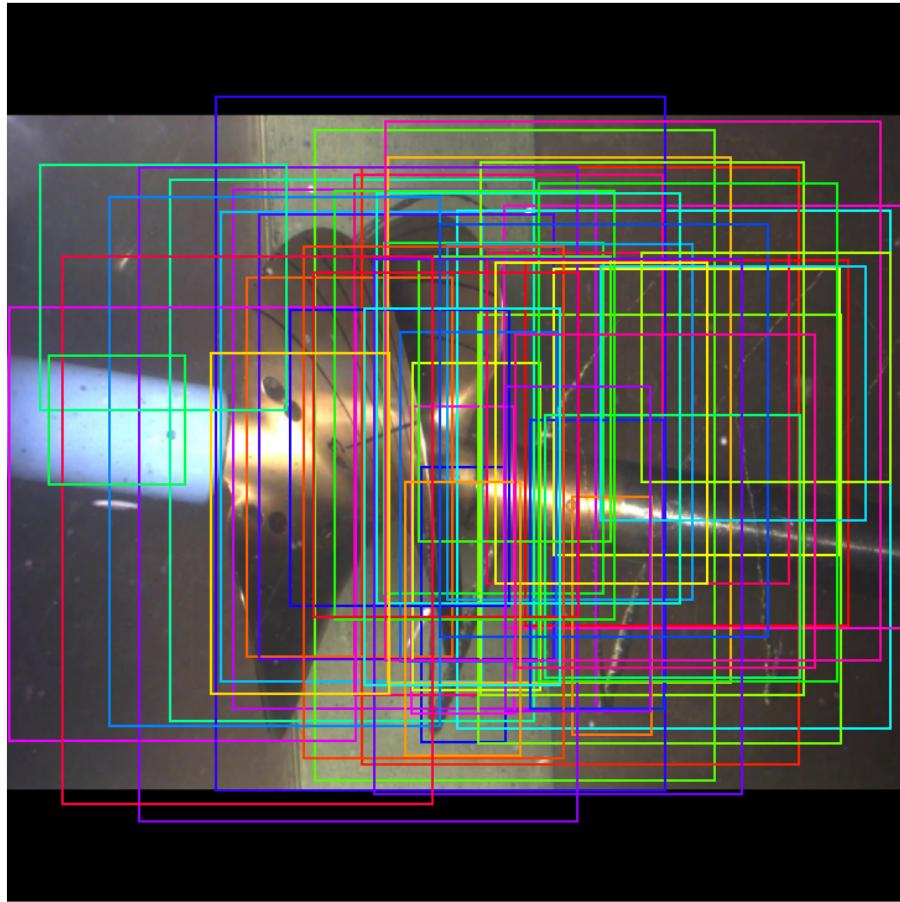
```
[60]: # Top-Anker nach Punktzahl anzeigen (vor der Verfeinerung)
limit = 100
sorted_anchor_ids = np.argsort(rpn['rpn_class'][:, :, 1].flatten())[::-1]
visualize.draw_boxes(image, boxes=model.anchors[sorted_anchor_ids[:limit]], ↪
    ↪ax=get_ax())
```



```
[61]: #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])
```



```
[62]: # 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())
```



```
[63]: #
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),
])

```

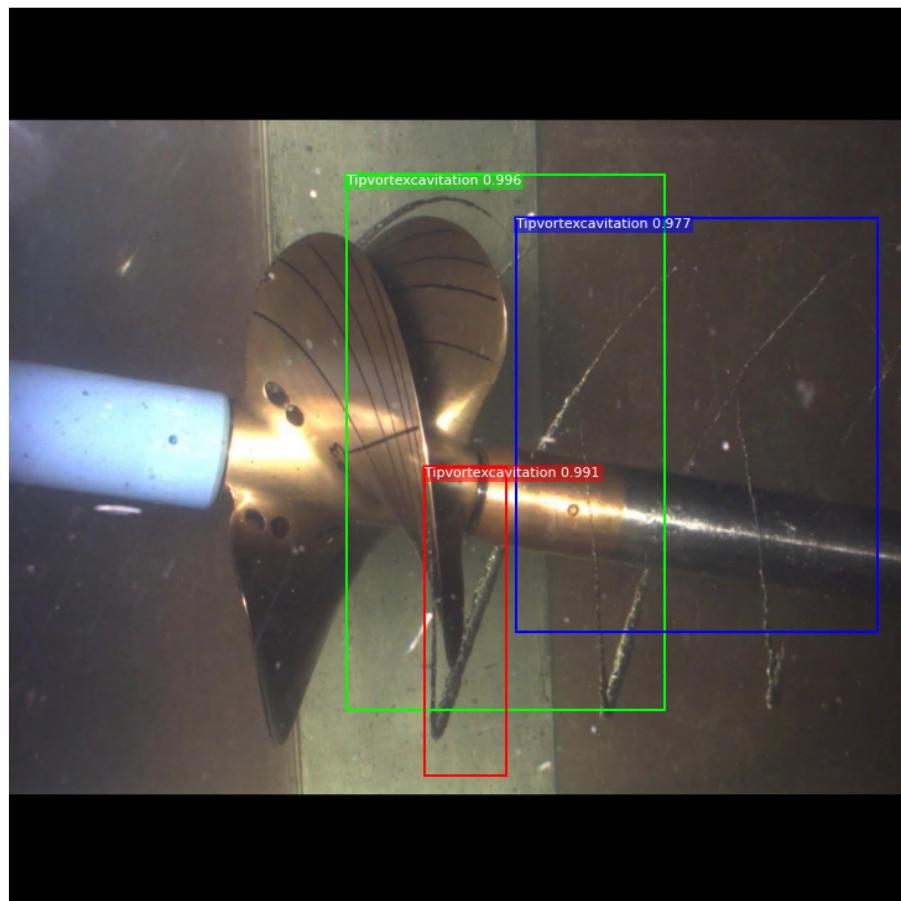
	shape:	min:	max:
proposals	(1, 1000, 4)	0.00000	1.00000
	float32		
probs	(1, 1000, 2)	0.00000	1.00000
	float32		

```
deltas           shape: (1, 1000, 2, 4)      min: -5.00139 max:  
5.09969 float32  
masks           shape: (1, 100, 28, 28, 2)   min: 0.00002 max:  
0.99542 float32  
detections      shape: (1, 100, 6)        min: 0.00000 max:  
1.00000 float32
```

```
[64]: #  
#  
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())
```

3 detections: ['Tipvortexexcavation' 'Tipvortexexcavation'
'Tipvortexexcavation']

Detections



```
[65]: # Vorschläge sind normalisierte Koordinaten. Skaliere die Vorschläge 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
70 Positive ROIs
[('BG', 930), ('Tipvortexexcavitation', 70)]

```

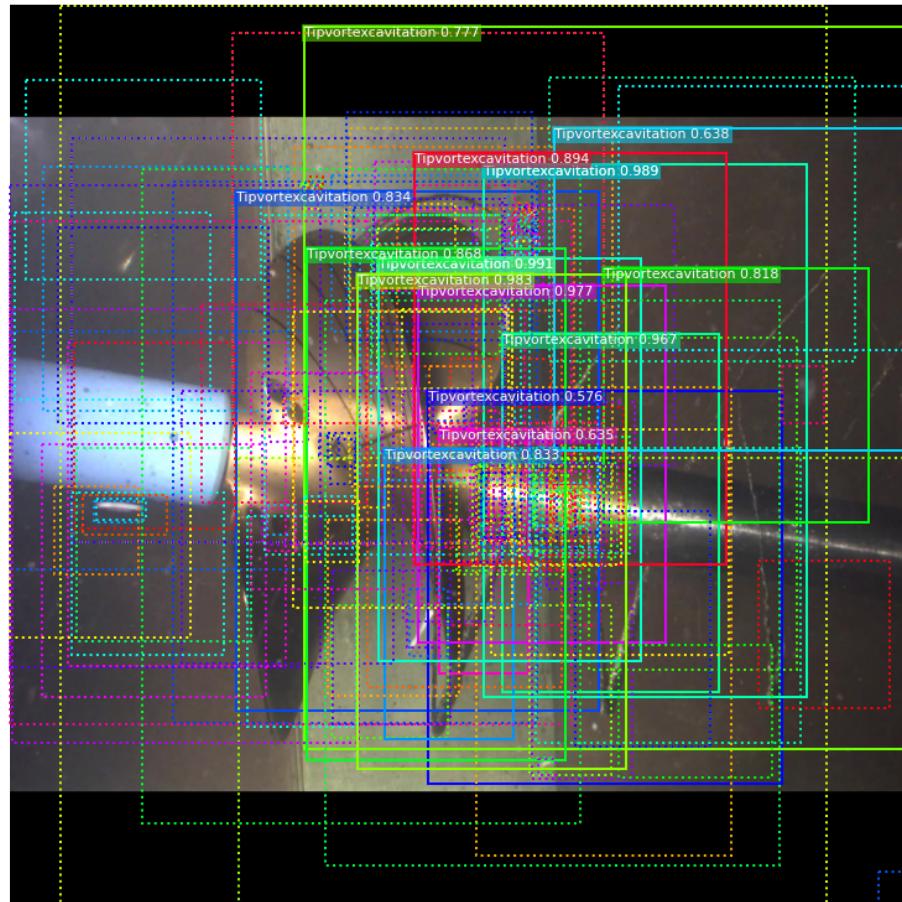
[66]:

```

# 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 ↴
# Tipvortexexcavitation
# 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



```
[67]: # 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)
```

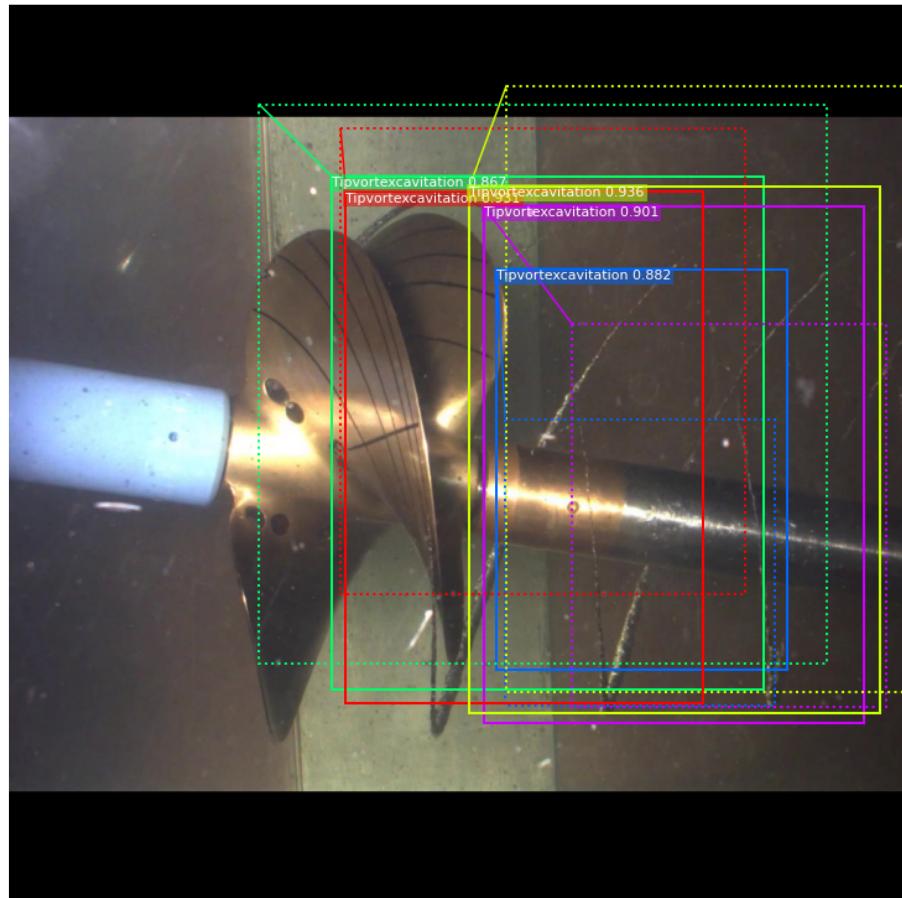
```

# 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: -4.05317 max:
5.09969 float32		
refined_proposals	shape: (1000, 4)	min: -333.00000 max:
1110.00000 int32		

ROIs After Refinement



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

```
Keep 70 detections:
[ 0   1   4   5   6   7   8   9   11  12  13  15  17  18  19  21  23  28
 31  32  33  34  35  36  39  40  42  43  47  50  52  54  61  65  69  70
 74  77  80  90  94  104 108 115 118 122 124 125 135 137 143 146 155 166
 230 247 257 276 296 298 301 302 318 324 338 404 476 523 574 906]
```

```
[69]: # 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 61:

[0	1	4	5	6	7	8	9	12	13	15	17	18	19	21	23	28	31
32	33	34	35	36	39	40	42	43	47	50	52	54	61	65	69	74	77
80	90	94	108	115	118	122	125	135	137	143	146	166	230	257	296	298	302
318	324	338	404	523	574	906]											

```

[70]: # 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("{}: {} -> {}".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))

```

Tipvortexexcavation : [0 1 4 5 6 7 8 9 12 13 15 17 18 19
21 23 28 31
32 33 34 35 36 39 40 42 43 47 50 52 54 61 65 69 74 77
80 90 94 108 115 118 122 125 135 137 143 146 166 230 257 296 298 302
318 324 338 404 523 574 906] -> [6 0 21]

Kept after per-class NMS: 3
[0 6 21]

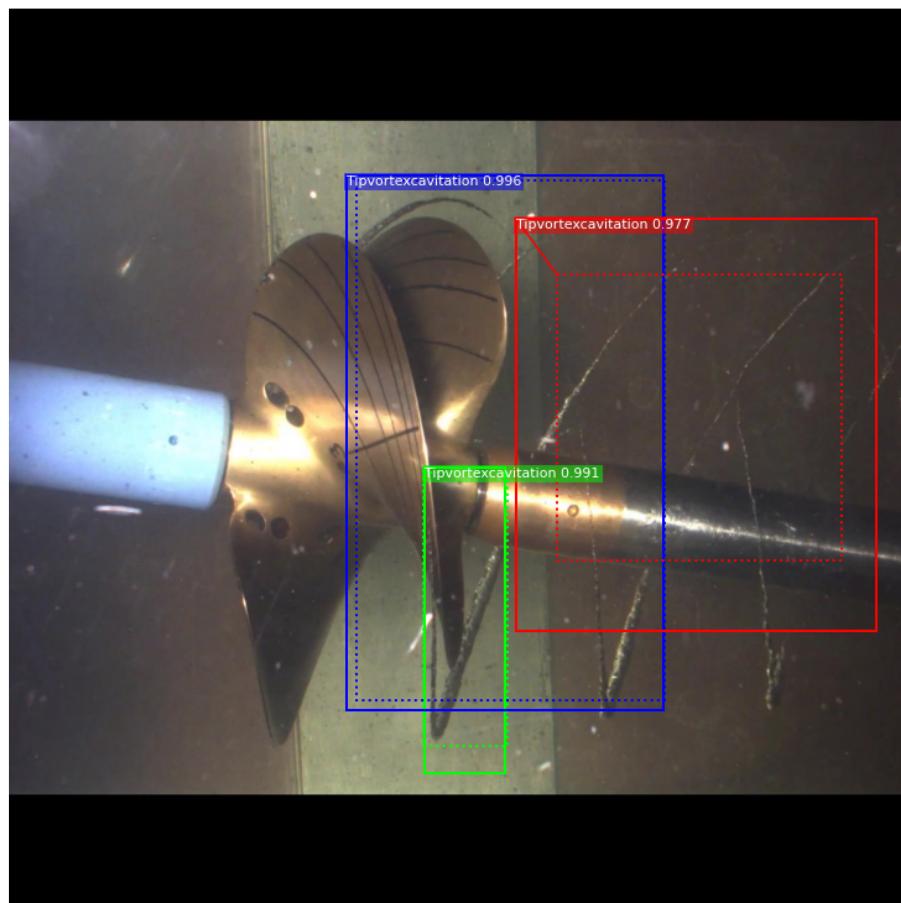
```

[71]: # 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



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



```
[73]: # 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),
])

# 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]

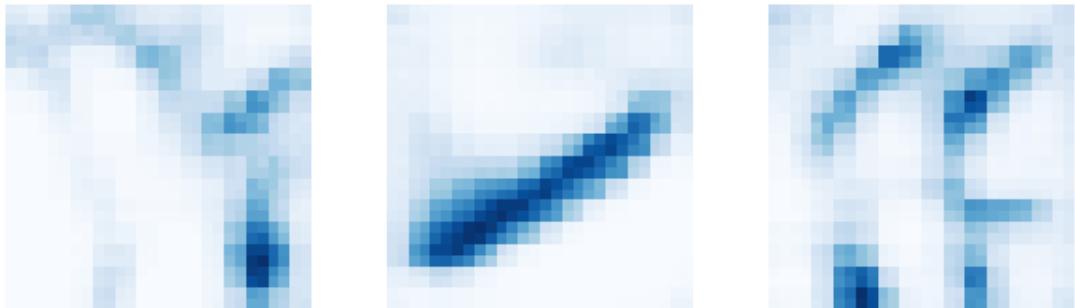
print("{} detections: {}".format(
    det_count, np.array(dataset.class_names)[det_class_ids]))
```

```
detections           shape: (1, 100, 6)      min: 0.00000  max:
1.00000  float32
masks                shape: (1, 100, 28, 28, 2)  min: 0.00002  max:
0.99542  float32
3 detections: ['Tipvortexexcavation' 'Tipvortexexcavation'
'Tipvortexexcavation']
```

```
[74]: # 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: (3, 28, 28)      min: 0.00002  max:
0.63248  float32
det_masks              shape: (3, 1024, 1024)  min: 0.00000  max:
1.00000  bool
```

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



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

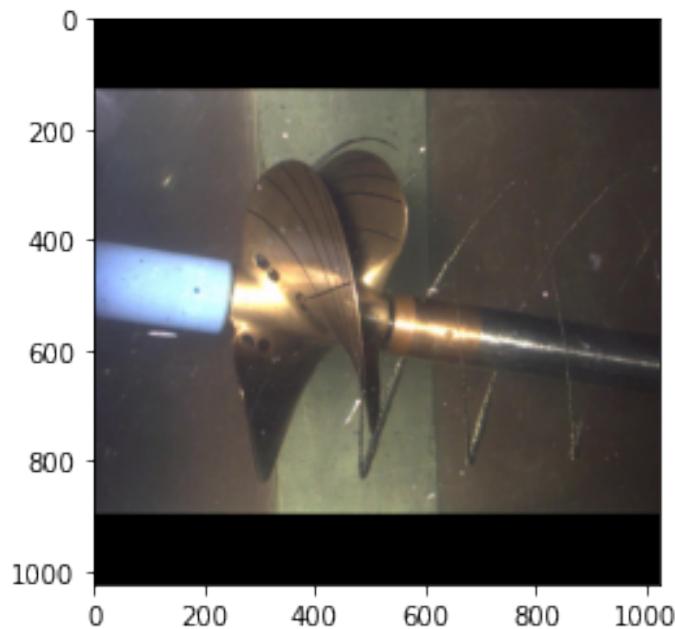


```
[77]: # die aktivierungsfunctionen für die layers erhalten
activations = model.run_graph([image], [
    ("input_image", tf.identity(model.keras_model.
    ↪get_layer("input_image").output)),
    ("res2c_out", model.keras_model.get_layer("res2c_out").output),
    ("res3c_out", model.keras_model.get_layer("res3c_out").output),
    ("res4w_out", model.keras_model.get_layer("res4w_out").output), # ↪
    ↪for resnet100
    ("rpn_bbox", model.keras_model.get_layer("rpn_bbox").output),
    ("roi", model.keras_model.get_layer("ROI").output),
])
```

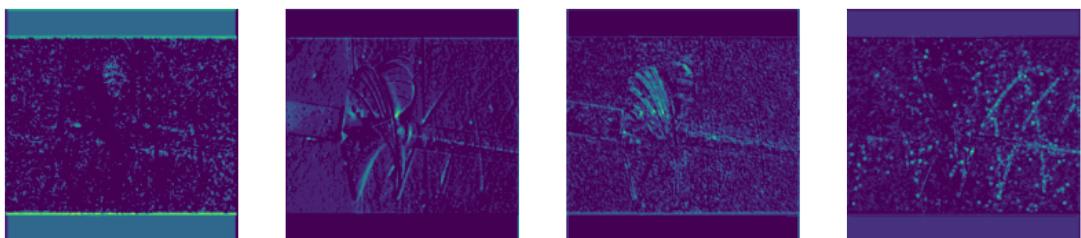
```
input_image           shape: (1, 1024, 1024, 3)   min: -123.70000  max:
151.10001  float32
res2c_out            shape: (1, 256, 256, 256)   min:      0.00000  max:
19.73085  float32
res3c_out            shape: (1, 128, 128, 512)   min:      0.00000  max:
29.74792  float32
```

```
res4w_out           shape: (1, 64, 64, 1024)      min: 0.00000 max:  
59.76387 float32  
rpn_bbox            shape: (1, 261888, 4)       min: -7.81913 max:  
103.89145 float32  
roi                 shape: (1, 1000, 4)        min: 0.00000 max:  
1.00000 float32
```

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



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



11 Eidesstattliche Versicherung

Hiermit erkläre ich Majd Alyan, Matrikelnummer [217203553] an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solchen kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Rostock, den 04.04.2022