Object localization of texture less cylindrical objects from 2D Images based on a Deep Learning approach using Mask-RCNN and Blender

Master's Project Report

Srini Prakash Maiya (923123)
ING-M, TH Rosenheim

Guide:
Prof. Dr.-Ing. Michael Wagner

Acknowledgement

This report was generated at the Technical University of Applied Sciences, Rosenheim in the Summer Semester of 2020 (March 2020 – August 2020)

At this point, I want to wholeheartedly thank my Academic and Project guide, Prof. Dr.-Ing Michael Wagner for providing me with the opportunity to work on this project. I am also thankful for his guidance and technical support. I would also like to thank my colleagues and fellow students who supported me both technically and personally.

I am also much obliged and thankful to my parents, who have raised, guided and supported me throughout my life.

Declaration

Hereby I, Srini Prakash Maiya declare corresponding to § 35 passage 7 of Rahmenprüfungsordnung for advanced technical colleges in Bavaria, that the project report with the title

Object localization of texture less cylindrical objects from 2D Images based on a Deep Learning approach using Mask-RCNN and Blender

Is compiled by myself and is not exhibited for other examination purposes. Equally, all the resources used are referenced and have not used anything undeclared. I also confirm that the content of the digital version is entirely identical to the printed version.

Abstract

In the era of data-driven engineering, Deep Learning models are becoming prominent in all fields of Engineering and Medicine. Deep Learning surpasses Machine Learning Algorithms in terms of performance and feature recognition when there is enough data. By building complex multi-layer abstract representations rather than trying to fit the data to an algorithm like in Machine Learning, Deep Neural Networks (DNN) can recognize different features in different layers of their Network. Due to this advantage, DNNs are great problem solvers in the field of Computer Vision, where they can solve plenty of problems with the assistance of image processing techniques.

In this project, a similar effort has been made to use a Mask RCNN (a convolutional DNN) to automate a mostly manual process of picking one of the most used Mechanical components in the machines, a shiny, contour-less metallic shaft from a box by detecting the location and orientation of the objects which are most suitable for picking among a pile of them.

This network is driven by data generated from open-source software, Blender which reduces the efforts of data generation and annotation by a great amount by exploiting the embedded Python interpreter to generate images with a random orientation for each image.

The efforts to annotate the images are fully borne by Blender and inbuilt Python Interpreter, therefore raking off the repetitive human labour of manual annotation and gathering of data meanwhile reducing the annotation errors to a very bare minimum.

Table of Contents

Table of figures

1 Introduction

For a decade and a half, Industrial Automation Technology has seen an immense transformation and change. This technology can be used to build flexible, independent and reliable systems for the automation of many tasks in various industrial sectors. The automated systems are usually faster, less error-prone and usually are in synch with other machines or tasks. To be able to achieve these behaviours, the system must understand its surroundings well and be able to address the uncertainty up to a good extent.

With the emergence of Deep Learning systems (DLS) and dedicated computer hardware, Automation Systems are getting more intelligent and sophisticated. The availability and open-sourcing of a large quantity of data, near-to-real synthetic data creation, clever crowdsourcing methods have amassed a wide variety of datasets which in turn help to find and address new kinds of barriers in the DLS which are being actively overcome by researchers, a vast community of enthusiasts and commercial companies.

One of the most common non-automated tasks in an assembly line is bin picking. As the objects lie in a container in the form of a pile, it is pretty much impossible to pick one out all, as there is not a clear-cut boundary between the objects. The traditional edge detection algorithms fail to detect the contours or edges, especially when the objects are shiny and textureless as they induce false edges under the influence of variable light.

Deep Learning Systems tend to work very effectively also under various misleading information and outliers, given the network is shown enough examples as seen in figure(1).

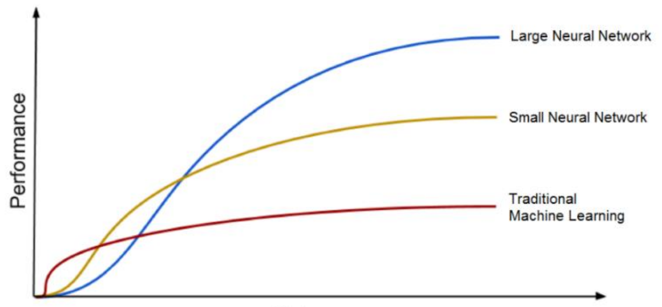


Figure 1. Comparison of Deep Learning to Machine Learning [16]

Inspired by many implementations of instance segmentation networks on various circumstances which are capable to solve the problem quite efficiently, in this project, I have tried to implement one of such DLS, a TensorFlow implementation [1]of the original Mask RCNN [2]to perform an instance segmentation of the pickable shafts from a box of shafts. Blender [3] is used to generate the images with a resolution of 256x256 for training and the corresponding annotations required are extracted using OpenCV [4] in the Blender Python Interpreter.

2 Literature Review

2.1 Deep Neural Networks (DNN)

The terms 'Machine Learning' and 'Artificial Intelligence' have been in use for a long time although they are in the spotlight for a decade. The term 'Machine Learning' was coined by A.L. Samuel in the year 1959 [5]. The notion of a machine learning the complex patterns and along with time be as good or even better than a human at performing a task has been there since the dawn of computers. With the exponential rise in the processing speeds of computers and the availability of faster Graphics Processing Units deeper Deep Learning models and complex Machine Learning models are possible.

The simplest unit of a DNN is a perceptron, which can take multiple inputs, x_i, multiply with the corresponding weight w to produce an output y. A neural network is a collection of these perceptrons, stacked in a layer along with an input and an output layer to solve nonlinear problems. A deep neural network has multiple layers of stacked perceptrons along with the input and output layers, giving them the ability to build abstractions to recognize features in an image and more.

The term *training* means feeding data to the model. During the process of training, the model starts to fit the data by adjusting the weights for each perceptron to reduce the error in the next training cycle. In other words, the model builds a notion or abstraction for the input, which is used to predict the output at a later stage. This is done with the help of backpropagation, a noble method introduced by Rumelhart in 1986 [6]. In the DNNs an activation function checks if the signal is strong enough to propagate forward in the network.

2.1.1 Semantic segmentation using Convolutional Neural Networks (CNN)

In contrast to Machine Learning and DNNs, the loss of Spatial information can be avoided by using CNNs [7]. As a DNN treats each pixel as an individual input vector, the network loses spatial information on how the pixels are related to each other as depicted in figure(2)

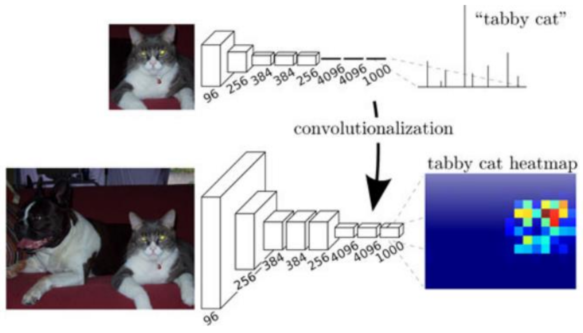


Figure 2. DNNs vs. CNNs [7]

On the other hand, CNNs use the whole images as inputs, thus making it possible to perform instance and semantic segmentation by retaining spatial information. By convolution, the kernel makes sure to take the surrounding features and pixels into account while training.

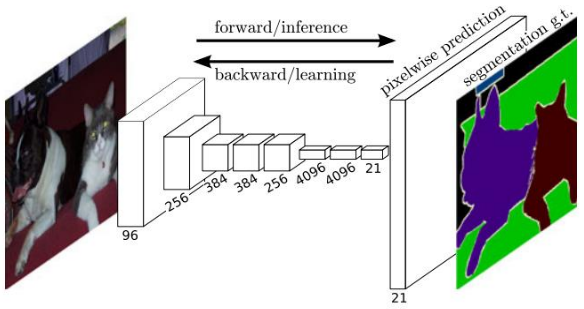


Figure 3. Fully convolutional networks for instance Segmentation [7]

Instead of doing an Image level classification, Fully Convolutional Networks (FCN) can create a pixel-wise label map by implementing a 1x1 convolutional layer in the end. At this layer, the network has down-sampled the pixel-wise output to a very small size, which does not match the input image. This abstraction is again expanded to original image size with the help of deconvolutional layers, which interpolates the feature output, thus making it possible to predict multiple class objects in a single image.

2.1.2 Object detection with Region-CNN(R-CNN)

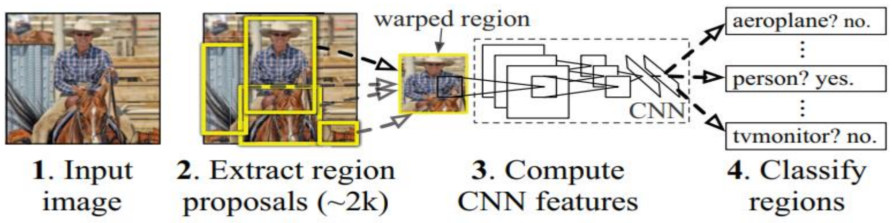


Figure 4. R-CNN overview [8]

In an image consisting of different objects of varying amounts, positions and sizes, a R-CNN [8] operates much effectively than a CNN network. Around 2000 Regions are proposed by using a *selective search* algorithm and are converted into a fixed size (227x227 in the article). A CNN extracts the important features, and the extracted features are appended to a 4096-dimensional vector. This vector is used along with a state vector machine (SVM) to classify if the object is present in the region. As the network had to propose 2000 regions and classify them in an image, this network was slow for both training and predictions.

This bottleneck was addressed by Grishik in2015 by introducing, Fast R-CNN [9] thus propelling the deep learning community further.

In Fast R-CNN, the image is directly fed into the convolutional layer, which in turn output the feature maps. From these maps, the region proposals are created. The proposed regions are pooled in the image to create a region of interest (RoI) through a RoIPool layer. These shrunk RoI s are fed to an SVM to determine the presence of an object in the region.

In 2016 further developments on Fast R-CNNs were seen. By proposing a dedicated network called region proposal network (RPN) instead of selective search, the speed of the Fast R-CNN was furthermore increased, thus terming it Faster R-CNN [10].

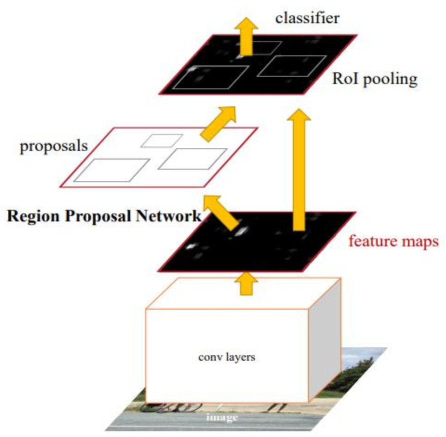


Figure 5. Faster RCNN for object detection [10]

The RPN network is used to place several specified anchors on the image depending on the size of the image. Bounding boxes of different sizes and ratios are created which act as the candidate regions. These anchors act as input to the convolutional layers with filters. The extracted features from the CNN are sent to the classification network, which classifies the presence of the object in that proposed region.

2.1.3 Instance segmentation with Mask R-CNN

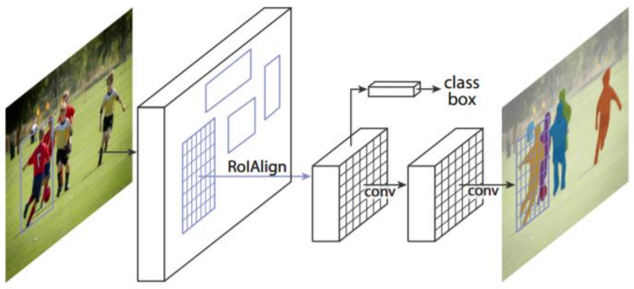


Figure 6. Mask R-CNN framework for instance segmentation

In 2017 Kaiming Ha et.al published a network built on Faster R-CNN. This network was called Mask Region Convolutional Neural Network (Mask R-CNN) which is today's modern Computer Vision benchmark. The most renowned feature of this network was its ability to perform instance segmentation among the overlapping objects of the same class with high accuracy.

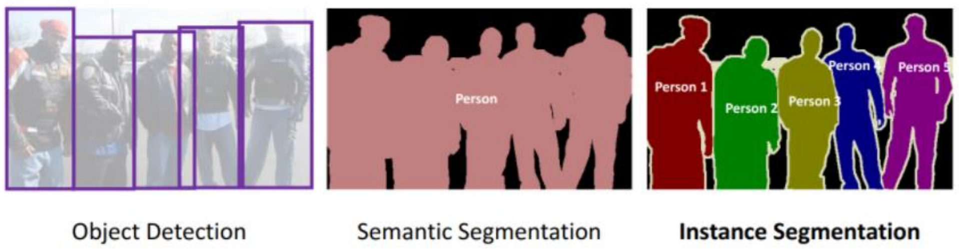


Figure 7. Difference between Semantic and Instance Segmentations [17]

To do such a precise instance segmentation; the network must solve two problems. The first being detection of a varying number of objects with corresponding bounding boxes for each of the objects and the second is to perform semantic segmentation within each bounding box to classify the area within the bounding box pixel-wise without overlapping the pixels from the other object.

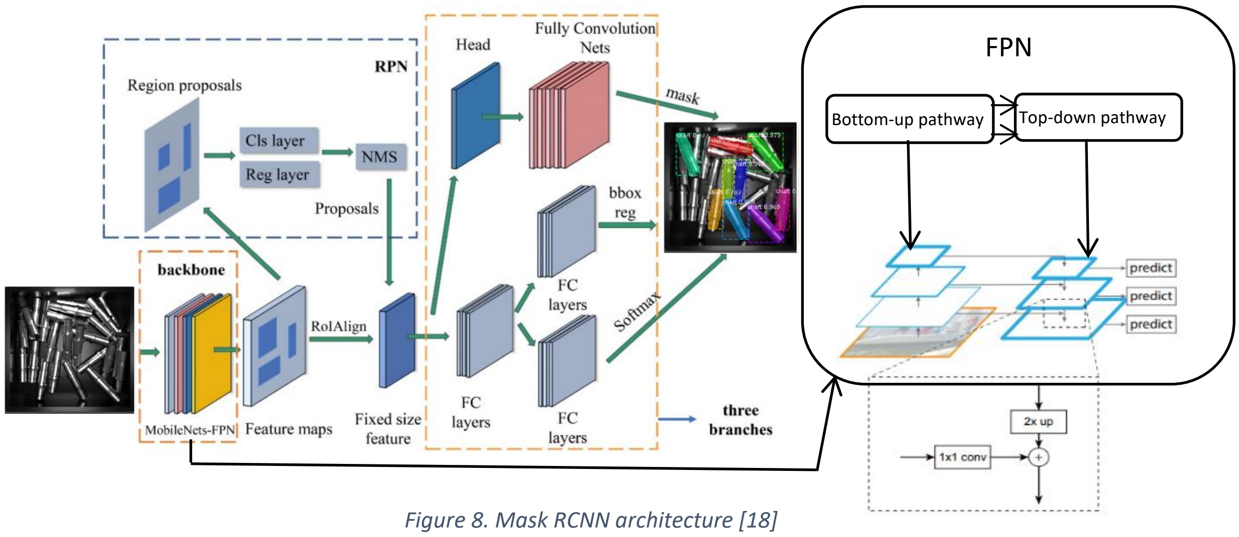


Figure 8. Mask RCNN architecture [18]

The backbone (right image, figure(8)) of the network is usually a ResNet50 or ResNet101 network, with a feature pyramid network (FPN) [11] which can efficiently extract the features of the image. First, the image is fed to the bottom-up pathway which does a feed-forward computation among its stages. The last layer of the stage represents the reference feature map with the richest features and is fed to the top-down pathway

The top-down pathway expands the high-resolution feature map built by the bottom-up pathway creating a spatially coarser but semantically stronger feature map from higher stages. These maps are enhanced by the lateral connections. These connections convolute maps of the stages which have the same sample size, thus enriching the features furthermore.

The RPN scans the top-down pathway and proposes the regions which may contain objects. The anchors are the proposed regions which are nothing but bounding boxes with various shapes and width to height ratios depending on the stage. As different size of anchors binds to different stages of the feature map, RPN tries to understand the location of the object on the feature map stack and the size of the bounding box.

These proposed regions are fed to another neural network with various fully convolutional layers, to predict the mask, bounding box and the class of the detected object.

Another major improvement in Mask R-CNN was the Region of Interest (RoI) layer quantization issue persisted in Faster R-CNN. In short, it is the problem of misalignment between the RoI layer and the feature maps when the RoI layer is pooled. Even though this is not a great deal when doing classification, it is one when the segmentation is pixel-wise, like in instance segmentation.

This problem was addressed by the addition of a new layer named 'RoI Align' which uses bilinear interpolation to calculate the exact alignment values of the RoI layer thus aligning each region of interest to the image.

2.2 Blender

Blender is a free and open-source 3D computer graphics software toolset used to create a wide range of realistic-looking motion graphics, 3D applications and visual images and movies and much more. The laws of physics, gravity, friction and aerodynamics can be applied to the objects to make the render and videos look near to reality. This powerful software packs with particle, cloth and soft body simulation, UV unwrapping, sculpting, rendering, skinning, fluid and smoke simulations, video and photo editing, compositing, scripting along with 3D modelling.

2.2.1 History

Blender was created by Ton Roosendaal a Dutch art director and a self-taught software developer. The first source files were written by him on 2nd January 1994 and this day is still considered Blender's official birthday.

The usage of Blender started booming in the year 2002 when it was declared to be forever open source under GNU General Public Licence and a successful crowdfunding campaign which raised 1,00,000€ within a span of a week.

Blender is seeing an active user community, active development from developers and independent creators. In years Blender has gained a vast fondness among the artists due to its powerful render engines with GPU support such as Eevee and Cycles, its easy-to-use User Interface, and inbuilt Blender Python Interpreter used to write scripts and add-ons to automate the workflow.

2.2.2 Blender Python (bpy)

The high-level programming language, Python was introduced in Blender 2.8 and onwards with a dedicated tab for scripting using bpy.

Along with the default 'bpy' and 'mathutils' modules, other Python libraries and modules can be installed in Blender, 'Open CV' to name one.

Using bpy custom add-ons can be built. These addons can further be used to automate the process of 3D modelling, by providing templates. There are many 3D modelling add-ons, which can be downloaded and run directly into the scene.

2.3 Installation procedure

2.3.1 Blender

Blender can be installed by downloading the executable file from the homepage of Blender. For this project, Blender 2.82 was used.

2.3.1.1 Open CV for Blender Python

Open the command prompt with administration privileges.

1. Navigate to the root folder of Blender python using the command, in this case, it was:
   ```
   cd "C:\Program Files\Blender Foundation\Blender 2.82\2.82\python\bin"
   ```
2. Execute the following command to install pip
   ```
   python -m pip install --upgrade pip
   ```
3. Install OpenCV in the Blender Python by executing the following command
   ```
   python -m pip install opencv-contrib-python numpy
   ```

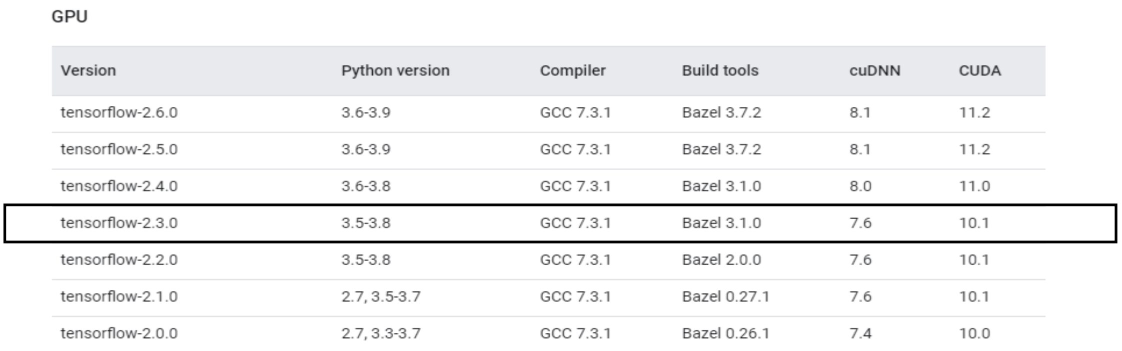
2.3.2 Mask R-CNN

The project was executed using anaconda, a distribution of Python and other programming languages. More information at https://docs.anaconda.com/anaconda/install/windows/

Different environments can be created for different projects, making them easily manageable to install various libraries exclusive and belonging to individual projects, thus eliminating the version dependencies and conflicts for and among different projects.

2.3.2.1 Tensorflow with GPU support:

Proceed to the link https://www.tensorflow.org/install/source#gpu to see the list of CUDA, cuDNN and TensorFlow GPU versions.



GPU

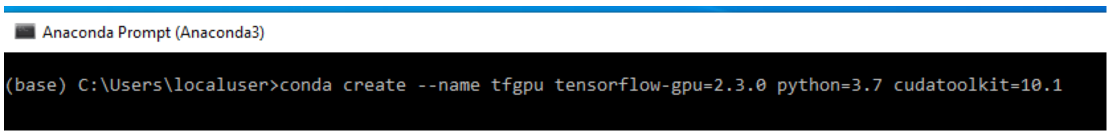
Version	Python version	Compiler	Build tools	cuDNN	CUDA
tensorflow-2.6.0	3.6-3.9	GCC 7.3.1	Bazel 3.7.2	8.1	11.2
tensorflow-2.5.0	3.6-3.9	GCC 7.3.1	Bazel 3.7.2	8.1	11.2
tensorflow-2.4.0	3.6-3.8	GCC 7.3.1	Bazel 3.1.0	8.0	11.0
tensorflow-2.3.0	3.5-3.8	GCC 7.3.1	Bazel 3.1.0	7.6	10.1
tensorflow-2.2.0	3.5-3.8	GCC 7.3.1	Bazel 2.0.0	7.6	10.1
tensorflow-2.1.0	2.7, 3.5-3.7	GCC 7.3.1	Bazel 0.27.1	7.6	10.1
tensorflow-2.0.0	2.7, 3.3-3.7	GCC 7.3.1	Bazel 0.26.1	7.4	10.0

Figure 9. Tensorflow GPU version selection

For this project, a stable build of TensorFlow-GPU (2.3.0), corresponding CUDA (10.1) and cuDNN (7.6) was selected.

2.3.2.2 Anaconda environment creation

1. Launch Anaconda prompt from the start menu.
2. Type and execute the command,

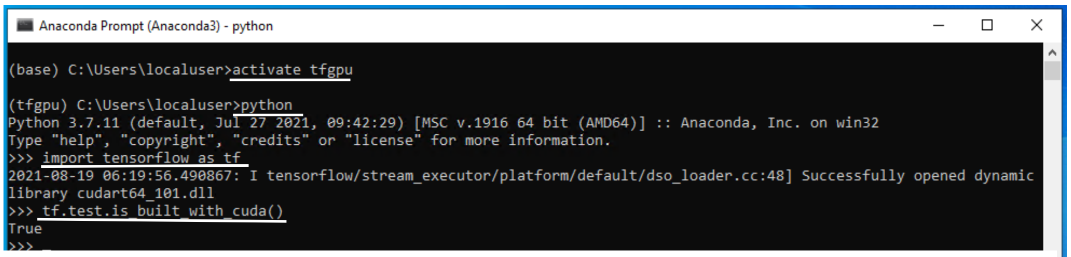


```
(base) C:\Users\localuser>conda create --name tfgpu tensorflow-gpu=2.3.0 python=3.7 cudatoolkit=10.1
```

Figure 10. Anaconda environment creation

Type 'y' to proceed.

3. Upon successful installation, activate the environment and test for successful GPU detection by TensorFlow.



Anaconda Prompt (Anaconda3) - python

```
(base) C:\Users\localuser>activate tfgpu

(tfgpu) C:\Users\localuser>python
Python 3.7.11 (default, Jul 27 2021, 09:42:29) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
2021-08-19 06:19:56.490867: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic
library cudart64_101.dll
>>> tf.test.is_built_with_cuda()
True
>>> _
```

Figure 11. TensorFlow GPU Support

2.3.2.3 Install Matterport Mask R-CNN

The link https://github.com/matterport/Mask_RCNN leads to the TensorFlow implementation of Mask R-CNN.

By clicking https://github.com/matterport/Mask_RCNN#installation the instructions to install the requirements can be found.

The link https://github.com/matterport/Mask_RCNN#getting-started has a few interactive Jupiter notebooks.

The link https://github.com/matterport/Mask_RCNN#training-on-your-own-dataset guides the user for training a custom dataset using the network.

Thorough documentation and an active user forum are present with already addressed queries and solutions.

3 Methodology

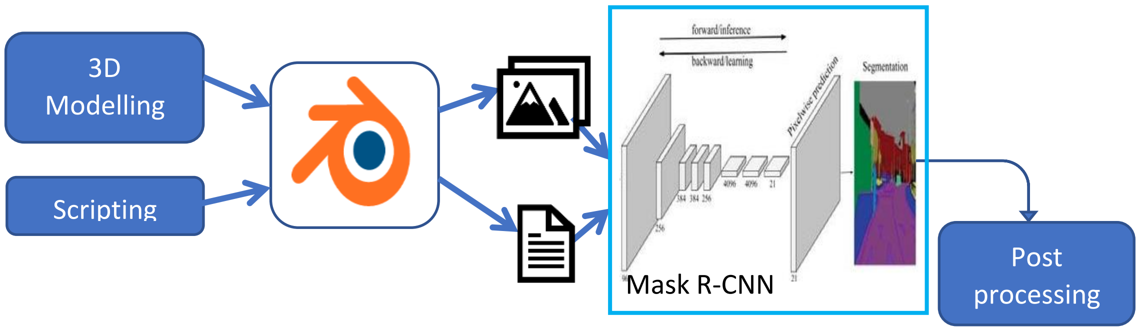


Figure 12. Methodology

The methodology of this project is drawn above in figure(12) and can be classified into following points.

Data generation: The blender file consists of an initial 3D model, which is later automated to generate random orientations with the help of a python script. The annotated data for the whole image is stored in a single JSON file, in COCO dataset [12] format which is fed to the neural network along with the generated images.

Training: During training, the neural network is monitored. The training is aimed at decreasing both the training and validation losses consistently, avoiding the overfitting problem. Various performance measures are calculated on the validation images.

Testing: Real camera images are used to test the model's ability to perform in the real life. Prediction is done using the saved weights. On the predicted masks, a PCA analysis is performed to get the orientation and center of the detected shafts.

Post processing: The analyzed shafts are visualized by drawing center and orientations on the image and saving them separately. The generated PCA annotations are saved in a separate notepad file for each predicted image.

4 Blender

Building upon the synthetic data generation project [13] of the fellow students of TH Rosenheim, the blender model and code were adopted and improvised accordingly to generate the required annotations and annotations.

4.1 Design changes

The shafts were increased in the length by 10% to match the length of the actual images.

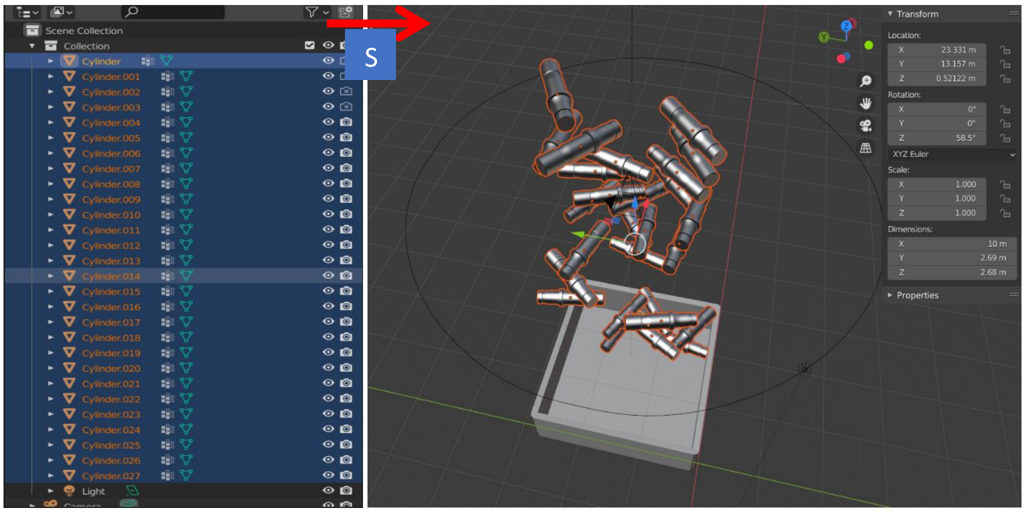


Figure 13. Scaling of the shafts

- The frame of the scene is set to the value '0' to bring the shafts to their initial position.
- In the 'Outliner window', which can be found at the right of the Blender layout UI, the first shaft is selected, and the 'Shift' button is held while clicking on the last shaft to select all the shafts.
- With the mouse cursor in the '3D window', the shortcut key 'S' is pressed to bring all the shafts into scaling mode. With moving the cursor to right or left, the scale of the shaft can either be increased or decreased.
- 'Shift' is held to make small increments in the scale and the scale of all the shafts in all X, Y, and Z directions are incremented from 1.000 to 1.1000.

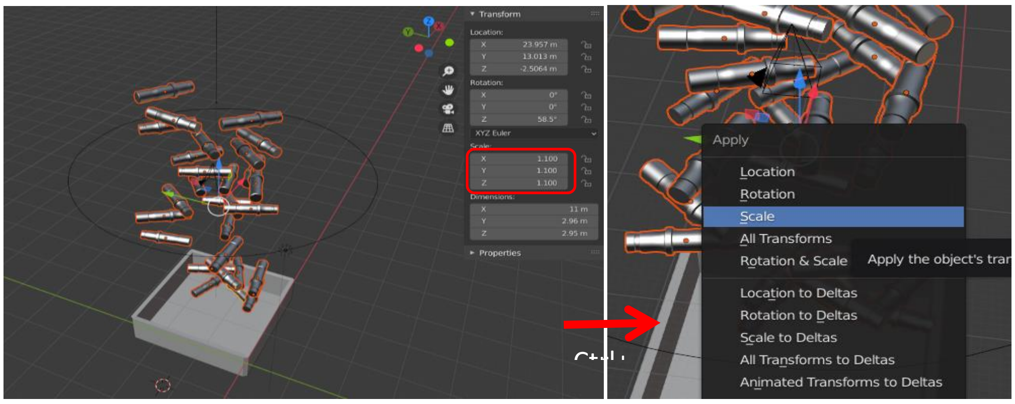


Figure 14. Applying the transforms

- With the shafts still selected, ‘Ctrl+A’ is pressed to activate the ‘Apply Menu’. The option ‘Scale’ is selected to apply the scale as shown in the above image, thus ensuring proper application of Physics on the scaled shafts

4.2 Main features

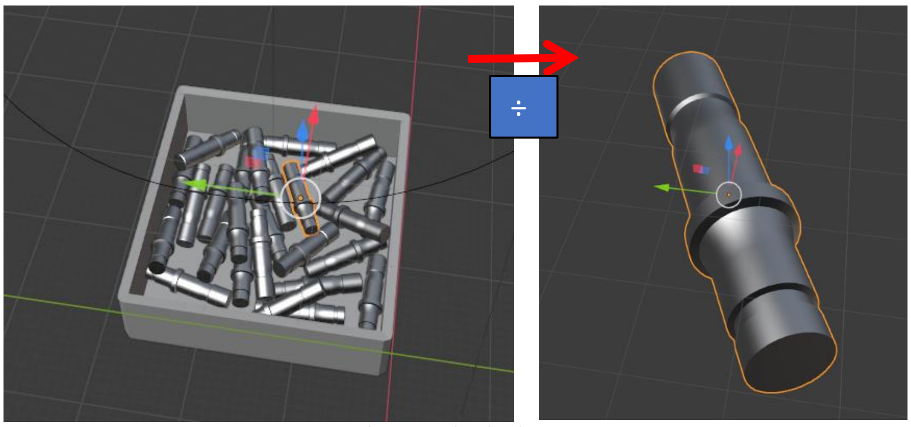


Figure 15. Isolating individual components

- A single shaft can be isolated from the box of the shaft in the viewport by selecting the shaft and pressing the ‘/’ or ‘÷’ on the numberpad.

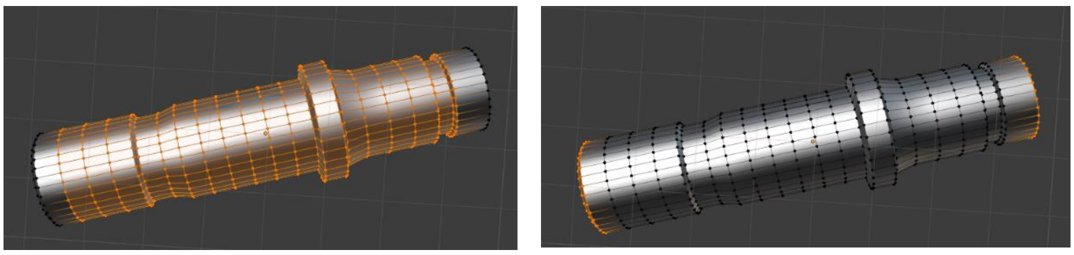


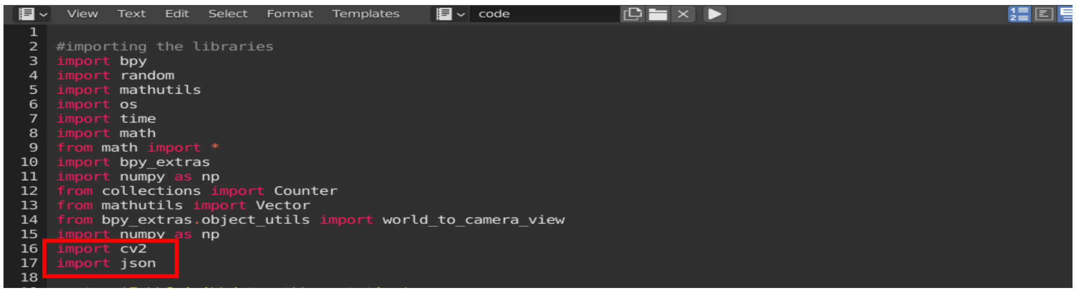
Figure 16. Visualization of edge vertices

- The vertices of the large edges and the small edges of the shafts are grouped in the ‘vertex groups’ tab of ‘Editor Menu’. The vertices can be visualized in the Edit mode, which can be activated by pressing ‘Tab’. These vertex groups can be used later in the script to select the shafts which are in the top-most layer.

4.3 Scripting changes

4.3.1 Importing installed OpenCV library

The main changes were made to the Blender script to generate the necessary annotations in COCO format along with the training images.

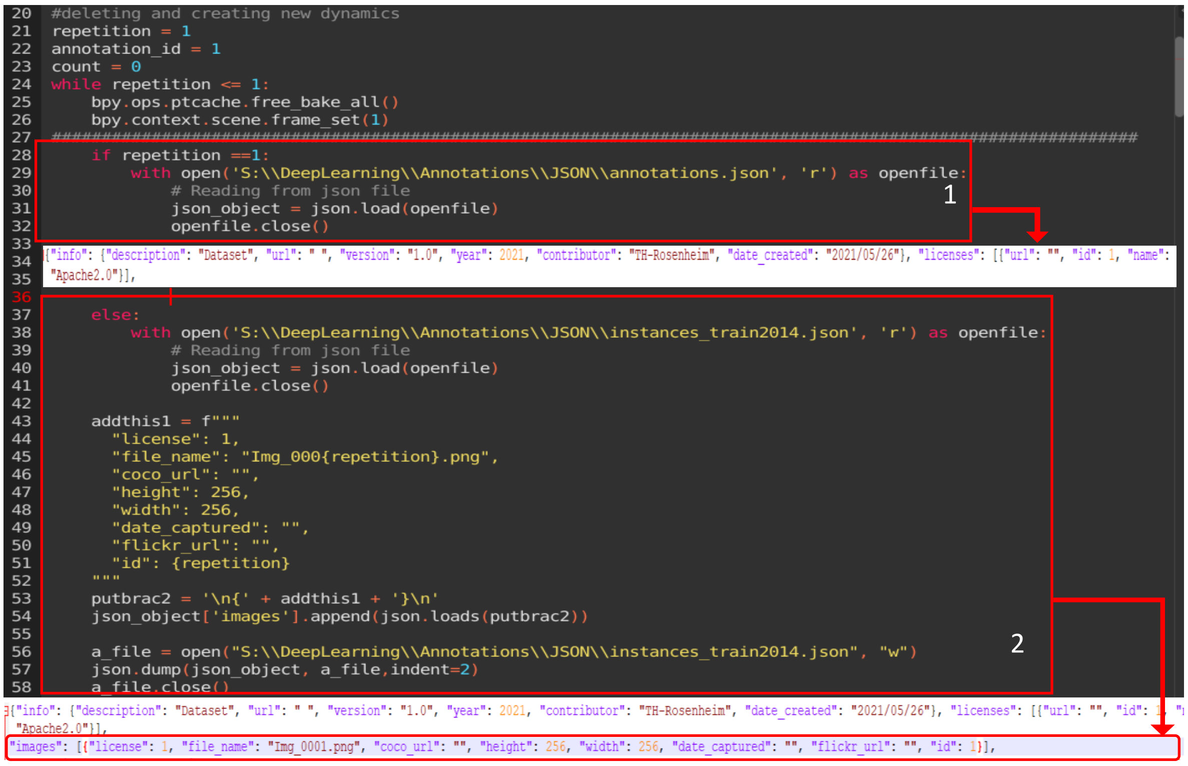


```

#importing the libraries
import bpy
import random
import mathutils
import os
import time
import math
from math import *
import bpy_extras
import numpy as np
from collections import Counter
from mathutils import Vector
from bpy_extras.object_utils import world_to_camera_view
import numpy as np
import cv2
import json

```

Figure 17. Importing libraries

- The installed OpenCV library is imported along with JSON and other necessary libraries.

4.3.2 JSON file creation and data logging

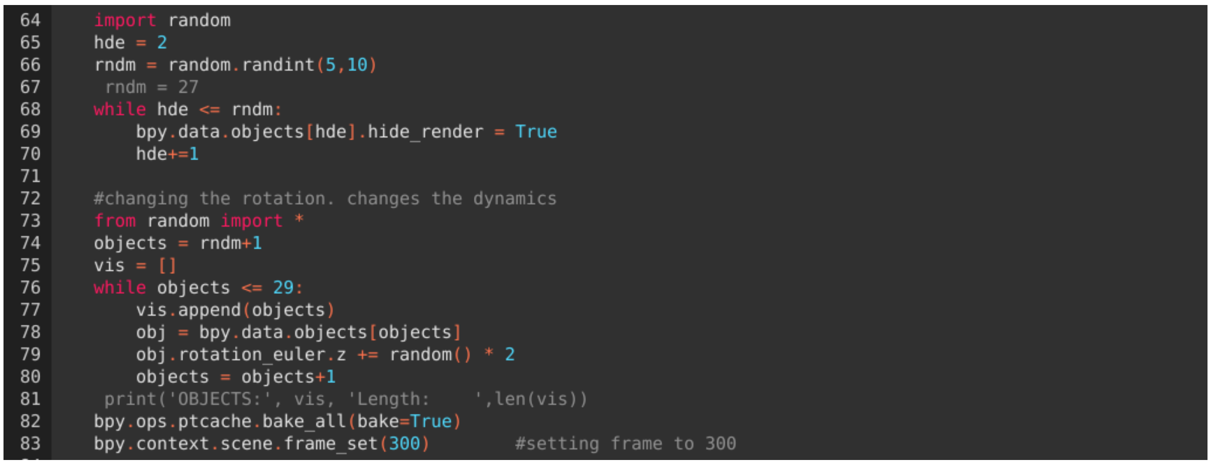


```
#deleting and creating new dynamics
repetition = 1
annotation_id = 1
count = 0
while repetition <= 1:
    bpy.ops.ptcache.free_bake_all()
    bpy.context.scene.frame_set(1)
################################################################################################################
    if repetition ==1:
        with open('S:\\DeepLearning\\Annotations\\JSON\\annotations.json', 'r') as openfile:
            # Reading from json file
            json_object = json.load(openfile)
            openfile.close()

{"info": {"description": "Dataset", "url": " ", "version": "1.0", "year": 2021, "contributor": "TH-Rosenheim", "date_created": "2021/05/26"}, "licenses": [{"url": "", "id": 1, "name":
 "Apache2.0"}],

    else:
        with open('S:\\DeepLearning\\Annotations\\JSON\\instances_train2014.json', 'r') as openfile:
            # Reading from json file
            json_object = json.load(openfile)
            openfile.close()

    addthis1 = f"""
      "license": 1,
      "file_name": "Img_000{repetition}.png",
      "coco_url": "",
      "height": 256,
      "width": 256,
      "date_captured": "",
      "flickr_url": "",
      "id": {repetition}
    """
    putbrac2 = '\n{' + addthis1 + '}\n'
    json_object['images'].append(json.loads(putbrac2))

    a_file = open("S:\\DeepLearning\\Annotations\\JSON\\instances_train2014.json", "w")
    json.dump(json_object, a_file,indent=2)
    a_file.close()
{"info": {"description": "Dataset", "url": " ", "version": "1.0", "year": 2021, "contributor": "TH-Rosenheim", "date_created": "2021/05/26"}, "licenses": [{"url": "", "id": 1, "
 "Apache2.0"}],
"images": [{"license": 1, "file_name": "Img_0001.png", "coco_url": "", "height": 256, "width": 256, "date_captured": "", "flickr_url": "", "id": 1}],
```

Figure 18. JSON file creation for data logging

- In the above figure, the non-iterable information, info and licenses are written to the JSON file in section 1.
- In section 2, the information of each image is written to the append able JSON file.

4.3.3 Random orientation

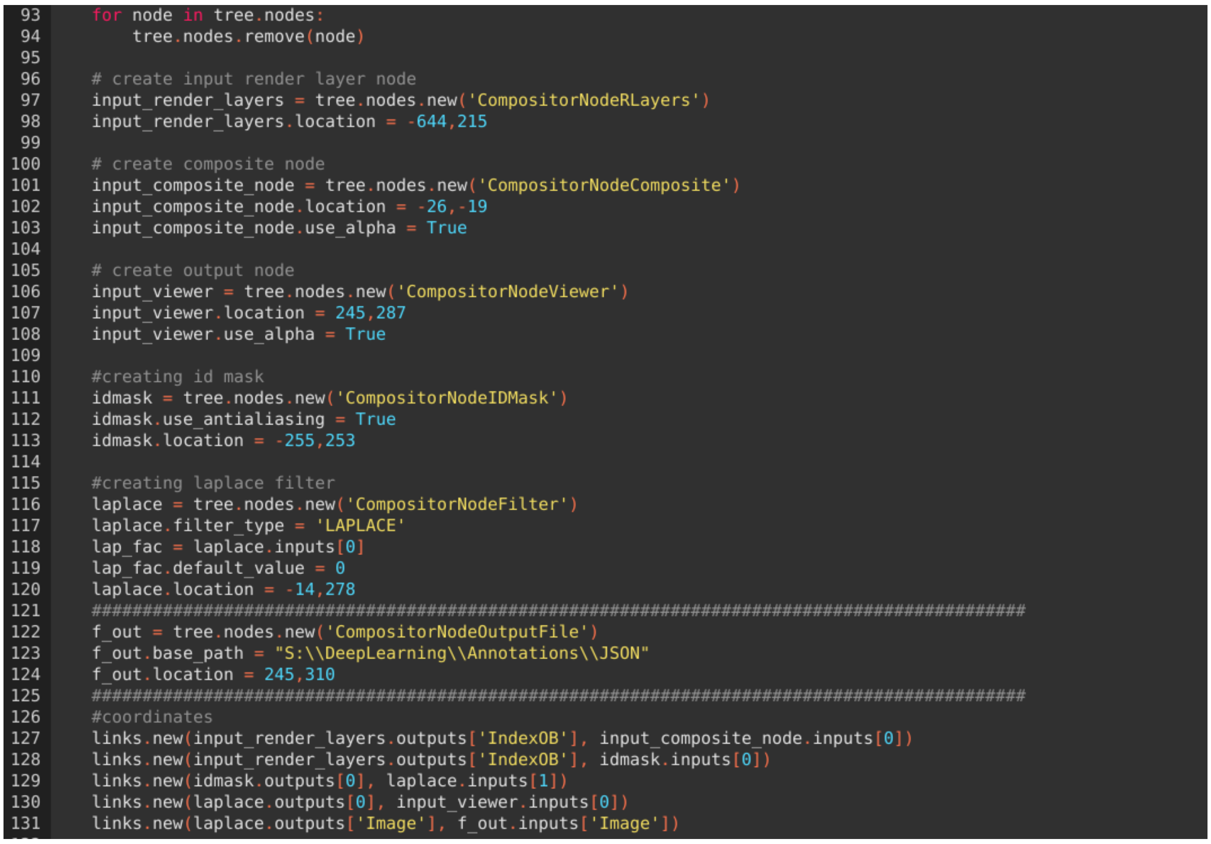


```
import random
hde = 2
rndm = random.randint(5,10)
 rndm = 27
while hde <= rndm:
    bpy.data.objects[hde].hide_render = True
    hde+=1

#changing the rotation. changes the dynamics
from random import *
objects = rndm+1
vis = []
while objects <= 29:
    vis.append(objects)
    obj = bpy.data.objects[objects]
    obj.rotation_euler.z += random() * 2
    objects = objects+1
 print('OBJECTS:', vis, 'Length:    ',len(vis))
bpy.ops.ptcache.bake_all(bake=True)
bpy.context.scene.frame_set(300)        #setting frame to 300
```

Figure 19. Random orientation creation

In each cycle of image generation, the shafts are set to random orientation within the while loop of the above figure in line 76, thus ensuring different randomized orientations in each of the generated images.

4.3.4 Node creation changes



```
 93        for node in tree.nodes:
 94            tree.nodes.remove(node)
 95
 96        # create input render layer node
 97        input_render_layers = tree.nodes.new('CompositorNodeRLayers')
 98        input_render_layers.location = -644,215
 99
100        # create composite node
101        input_composite_node = tree.nodes.new('CompositorNodeComposite')
102        input_composite_node.location = -26,-19
103        input_composite_node.use_alpha = True
104
105        # create output node
106        input_viewer = tree.nodes.new('CompositorNodeViewer')
107        input_viewer.location = 245,287
108        input_viewer.use_alpha = True
109
110        #creating id mask
111        idmask = tree.nodes.new('CompositorNodeIDMask')
112        idmask.use_antialiasing = True
113        idmask.location = -255,253
114
115        #creating laplace filter
116        laplace = tree.nodes.new('CompositorNodeFilter')
117        laplace.filter_type = 'LAPLACE'
118        lap_fac = laplace.inputs[0]
119        lap_fac.default_value = 0
120        laplace.location = -14,278
121        ###########################################################################################
122        f_out = tree.nodes.new('CompositorNodeOutputFile')
123        f_out.base_path = "S:\\DeepLearning\\Annotations\\JSON"
124        f_out.location = 245,310
125        ###########################################################################################
126        #coordinates
127        links.new(input_render_layers.outputs['IndexOB'], input_composite_node.inputs[0])
128        links.new(input_render_layers.outputs['IndexOB'], idmask.inputs[0])
129        links.new(idmask.outputs[0], laplace.inputs[1])
130        links.new(laplace.outputs[0], input_viewer.inputs[0])
131        links.new(laplace.outputs['Image'], f_out.inputs['Image'])
```

Figure 20. Node creation

- Figure(20) illustrates the process of node creation. At each image, the corresponding nodes are created at each image generation cycle and are deleted once the image is rendered.

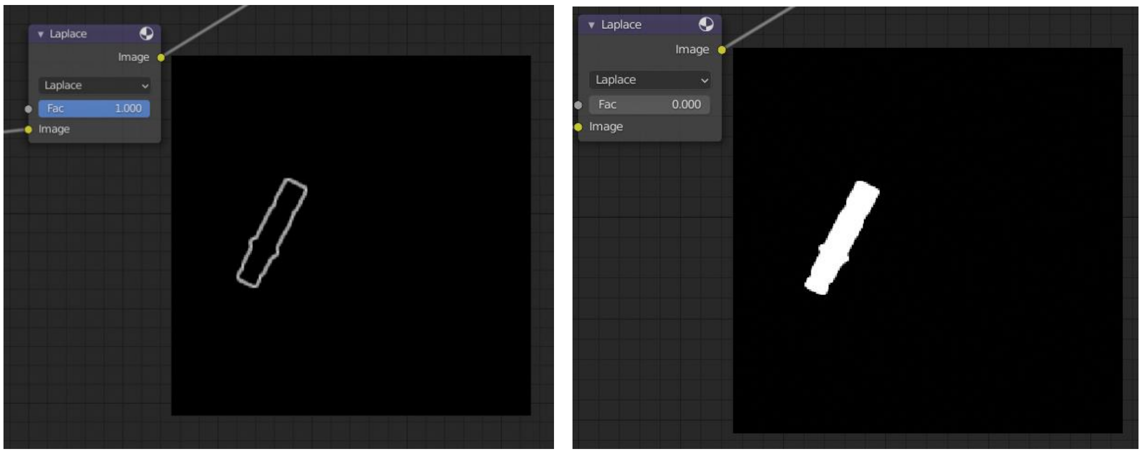
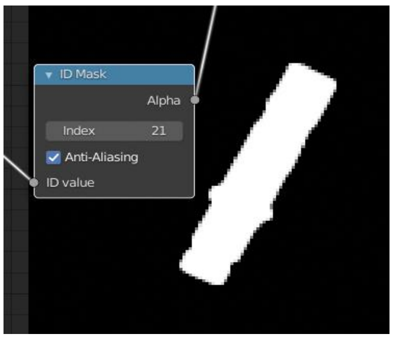


Figure 21. Laplace node changes

- The factor value in the Laplace node is reduced to zero in line 119 of the figure(20) to get the full mask of the shaft, instead of the outline as visualized in figure(21). This ensures error-free contour detection, which is applied in the latter stage of the script.



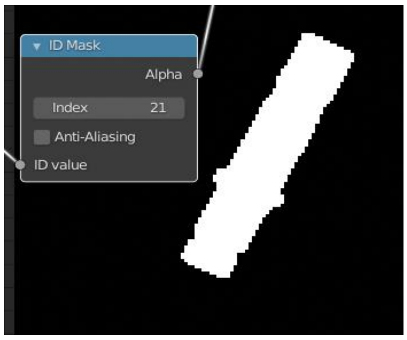
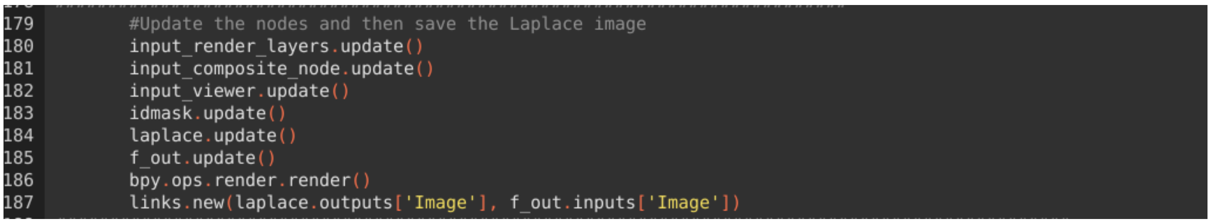


Figure 22. Anti-aliasing effects on the contour

- Furthermore, in line 112 of figure(20) Antialiasing has been ticked on in the 'ID Mask' node, to get a smoother mask in the Laplace node which can be seen in figure(22).

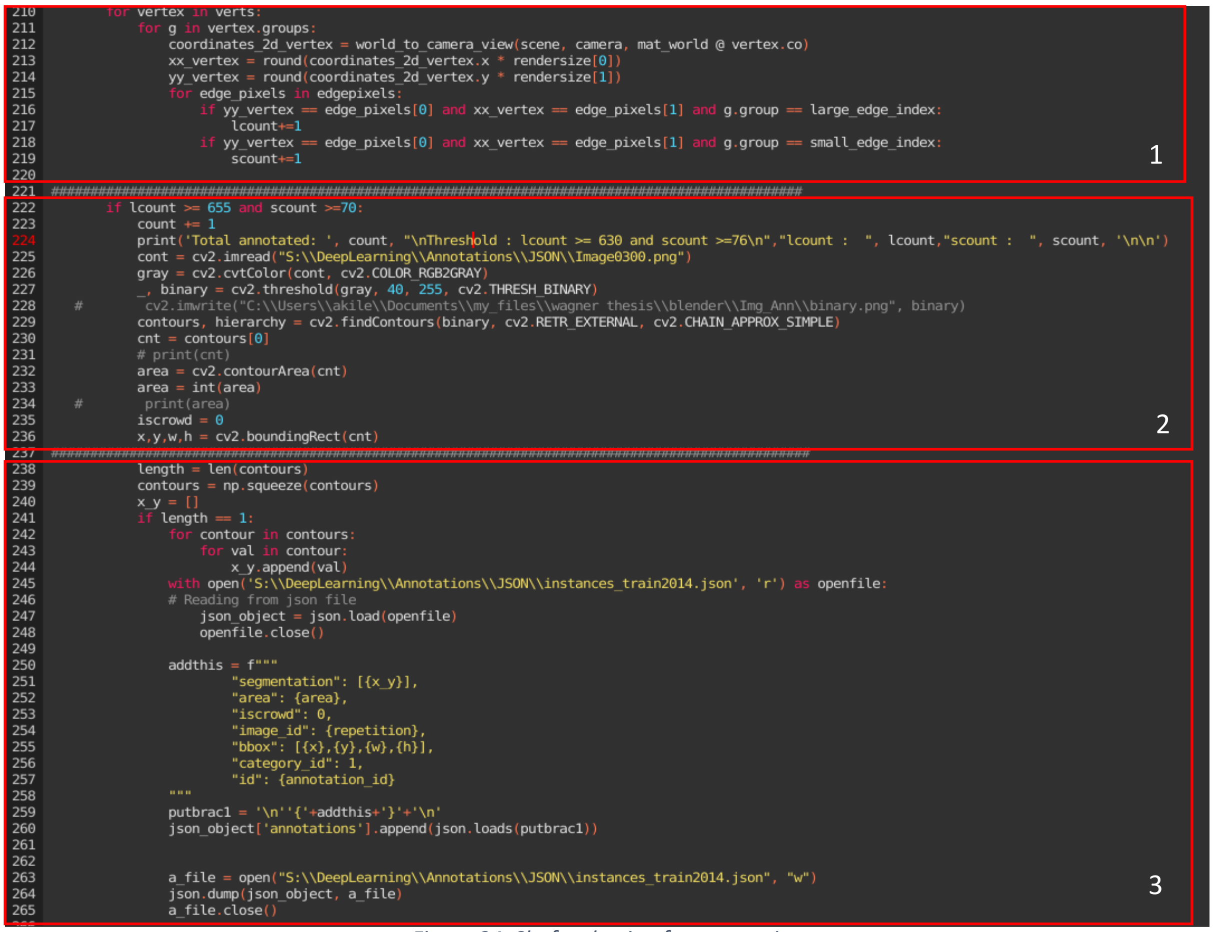


```
        #Update the nodes and then save the Laplace image
        input_render_layers.update()
        input_composite_node.update()
        input_viewer.update()
        idmask.update()
        laplace.update()
        f_out.update()
        bpy.ops.render.render()
        links.new(laplace.outputs['Image'], f_out.inputs['Image'])
```

Figure 23. Creation of node update block

- In figure(23) a set of node update functions are called, to ensures nodes are updated and have the actual information of the current iteration.

4.3.5 Thresholding of vertices and coordinates logging



```
    for vertex in verts:
        for g in vertex.groups:
            coordinates_2d_vertex = world_to_camera_view(scene, camera, mat_world @ vertex.co)
            xx_vertex = round(coordinates_2d_vertex.x * rendersize[0])
            yy_vertex = round(coordinates_2d_vertex.y * rendersize[1])
            for edge_pixels in edgepixels:
                if yy_vertex == edge_pixels[0] and xx_vertex == edge_pixels[1] and g.group == large_edge_index:
                    lcount+=1
                if yy_vertex == edge_pixels[0] and xx_vertex == edge_pixels[1] and g.group == small_edge_index:
                    scount+=1

################################################################################################
    if lcount >= 655 and scount >=70:
        count += 1
        print('Total annotated: ', count, "\nThreshold : lcount >= 630 and scount >=76\n","lcount :  ", lcount,"scount :  ", scount, '\n\n')
        cont = cv2.imread("S:\\DeepLearning\\Annotations\\JSON\\Image0300.png")
        gray = cv2.cvtColor(cont, cv2.COLOR_RGB2GRAY)
        _, binary = cv2.threshold(gray, 40, 255, cv2.THRESH_BINARY)
#        cv2.imwrite("C:\\Users\\akile\\Documents\\my_files\\wagner thesis\\blender\\Img_Ann\\binary.png", binary)
        contours, hierarchy = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnt = contours[0]
        # print(cnt)
        area = cv2.contourArea(cnt)
        area = int(area)
#        print(area)
        iscrowd = 0
        x,y,w,h = cv2.boundingRect(cnt)
################################################################################################
        length = len(contours)
        contours = np.squeeze(contours)
        x_y = []
        if length == 1:
            for contour in contours:
                for val in contour:
                    x_y.append(val)
            with open('S:\\DeepLearning\\Annotations\\JSON\\instances_train2014.json', 'r') as openfile:
            # Reading from json file
                json_object = json.load(openfile)
                openfile.close()

            addthis = f"""
                    "segmentation": [{x_y}],
                    "area": {area},
                    "iscrowd": 0,
                    "image_id": {repetition},
                    "bbox": [{x},{y},{w},{h}],
                    "category_id": 1,
                    "id": {annotation_id}
            """
            putbrac1 = '\n''{'+addthis+'}'+'\n'
            json_object['annotations'].append(json.loads(putbrac1))


            a_file = open("S:\\DeepLearning\\Annotations\\JSON\\instances_train2014.json", "w")
            json.dump(json_object, a_file)
            a_file.close()
```

Figure 24. Shaft selection for annotation

- In section 1 of figure(24), the calculation of the number of large and small vertices visible to the camera is carried out in a loop.
- Section 2 concentrates on thresholding the visibility of vertices to isolate the top layer shafts from the heap. If a shaft has more than the specified number of large and small edge vertices visible to the camera, contour detection is done on the Laplace image of that specific shaft using OpenCV. Along with the contour, the area of the contour, bounding box vertices is calculated with the help of OpenCV.
- The code of section 3 of the image does a job of cleaning up the contours. It reformats the list of arrays of contours generated from contour detection algorithms into a single list of x-y point pairs, as per the COCO annotation standards. Along with the point pairs, the area of the contour, bounding box vertices, category id, segmentation id are written to the respective contour to fulfil the COCO dataset standards.

In the end, a random selection among four types of light i.e., 'Area', 'Sun', 'Point' and 'Spot' lights is made with random intensity, to simulate the random lighting conditions. The final image is rendered and saved, all the existing nodes are deleted, and a new cycle of image generation is started.

4.3.6 Result

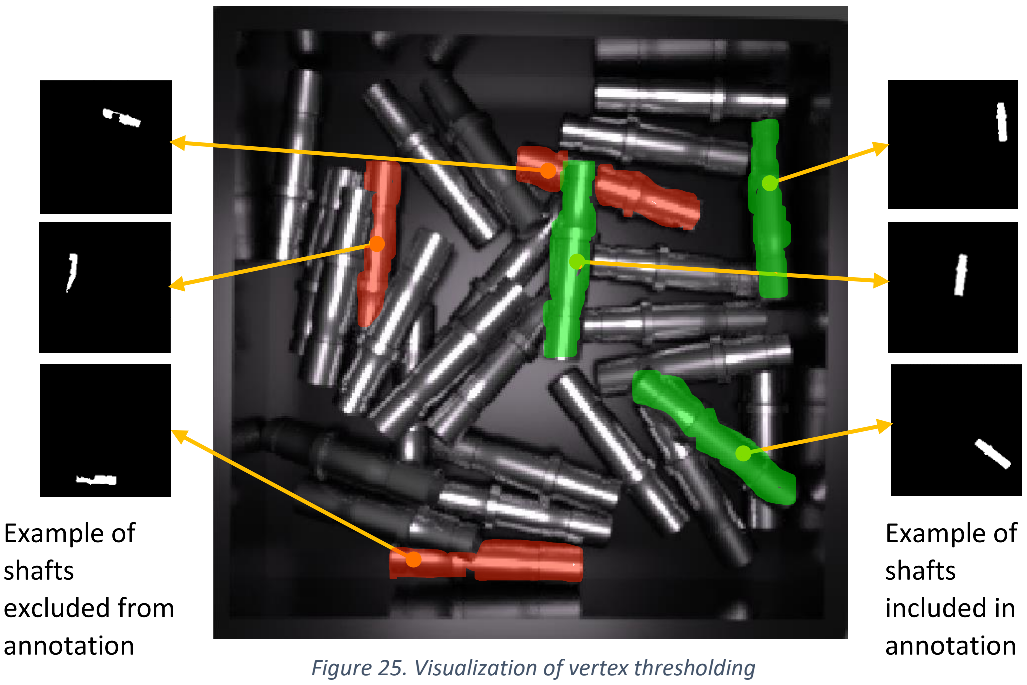

Example of shafts excluded from annotation
Example of shafts included in annotation

Figure 25. Visualization of vertex thresholding

In figure(25) instances a few examples of the classification of shafts into annotatable and non-annotatable categories. The shafts which are oriented like the ones highlighted in red are not annotated, as the camera sees lesser vertices than the threshold. These shafts are present in the bottom layers of the image, and it is not desirable to include them in the pickable category. The shafts marked in green have visible vertices higher than or equal to the vertex threshold and are included in the process of annotation generation.

5 Mask R-CNN

5.1 Mask R-CNN workflow

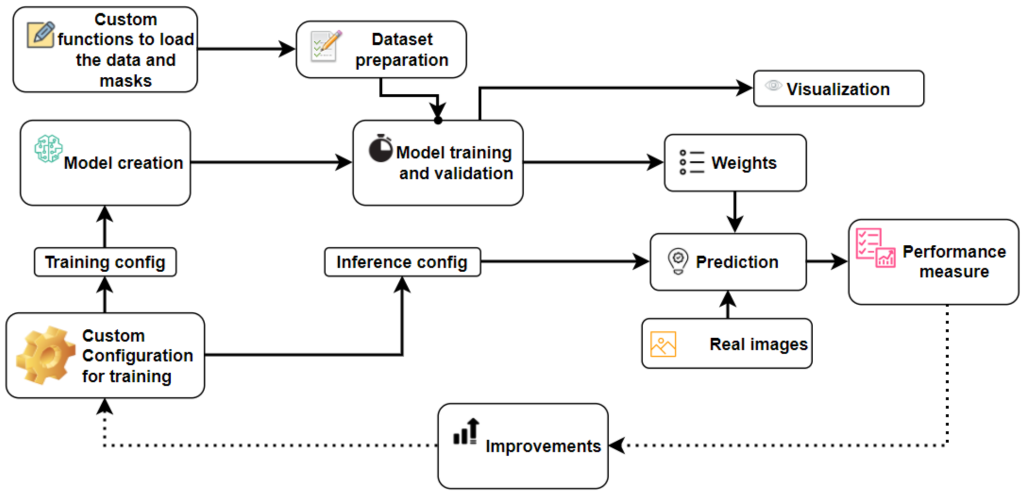


Figure 26. Mask R-CNN workflow

- The above figure illustrates the workflow cycle of training and improving the Mask R-CNN network. Custom configuration files for both training and prediction are created. Custom data preparation functions are created to load data as well as initial visualization of data.
- A model is created using training configuration and the same model is trained using the prepared dataset. Losses and accuracies are monitored, and predictions are made using the trained weights.
- A performance measure function is written which calculates the model's average precision, average recall and F1 score.
- With the help of visualization and performance measures, further improvements are made to the model.

5.2 Visualization

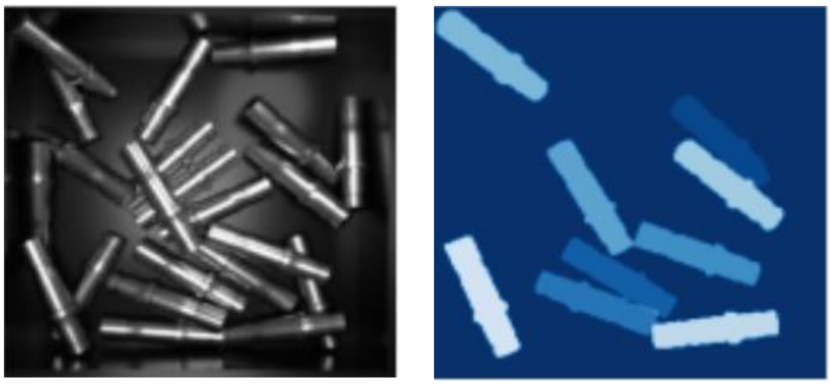
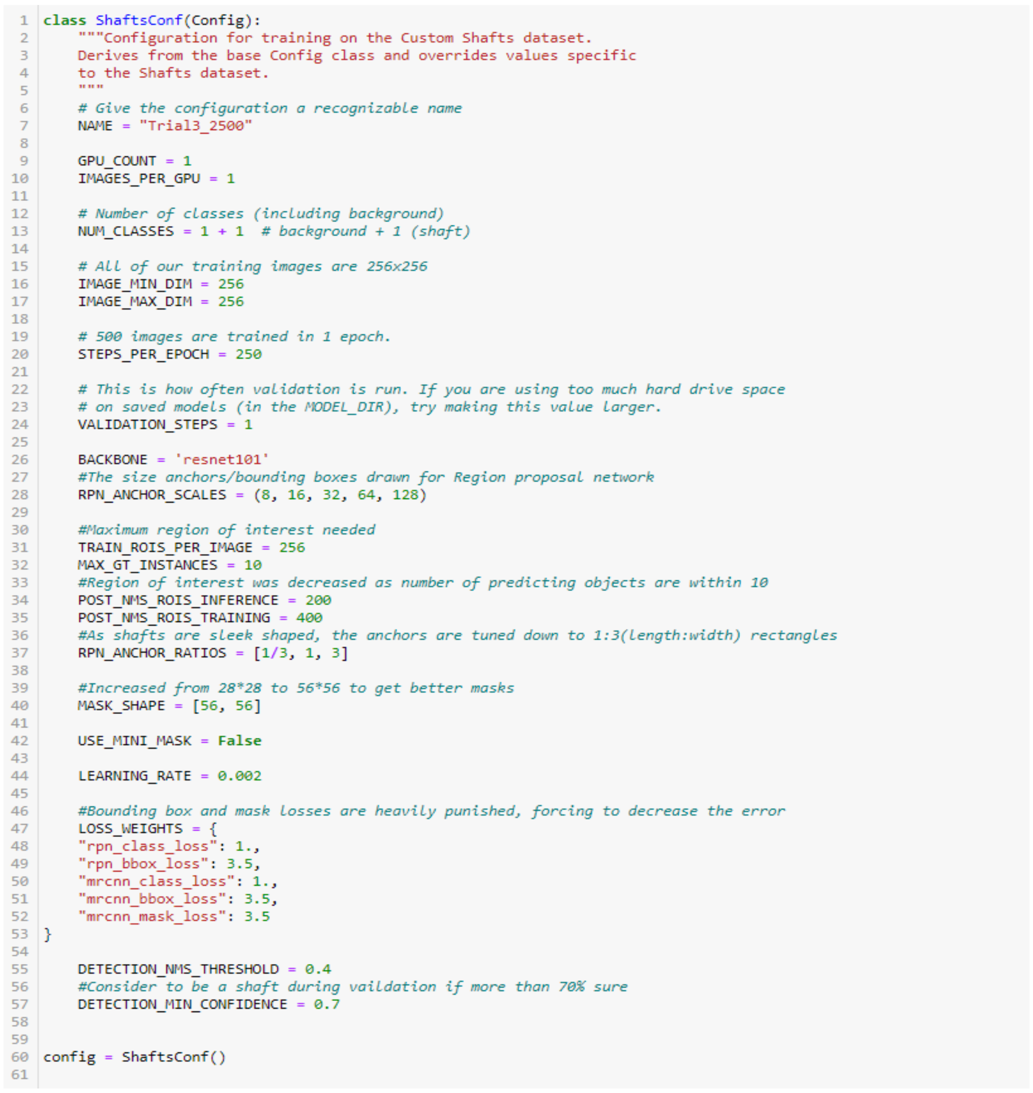


Figure 27. Mask visualization

With the help of visualization tools included in Mask R-CNN, an example image has been visualized in figure(27). The function visualizes only the regions of the input image present in the annotation file. Only the shafts present in the top layer are annotated using blender scripting, thus verifying the correctness of the input data of the neural network.

5.3 Optimum configuration settings



```
 1 class ShaftsConf(Config):
 2     """Configuration for training on the Custom Shafts dataset.
 3     Derives from the base Config class and overrides values specific
 4     to the Shafts dataset.
 5     """
 6     # Give the configuration a recognizable name
 7     NAME = "Trial3_2500"
 8 
 9     GPU_COUNT = 1
10     IMAGES_PER_GPU = 1
11 
12     # Number of classes (including background)
13     NUM_CLASSES = 1 + 1  # background + 1 (shaft)
14 
15     # All of our training images are 256x256
16     IMAGE_MIN_DIM = 256
17     IMAGE_MAX_DIM = 256
18 
19     # 500 images are trained in 1 epoch.
20     STEPS_PER_EPOCH = 250
21 
22     # This is how often validation is run. If you are using too much hard drive space
23     # on saved models (in the MODEL_DIR), try making this value larger.
24     VALIDATION_STEPS = 1
25 
26     BACKBONE = 'resnet101'
27     #The size anchors/bounding boxes drawn for Region proposal network
28     RPN_ANCHOR_SCALES = (8, 16, 32, 64, 128)
29 
30     #Maximum region of interest needed
31     TRAIN_ROIS_PER_IMAGE = 256
32     MAX_GT_INSTANCES = 10
33     #Region of interest was decreased as number of predicting objects are within 10
34     POST_NMS_ROIS_INFERENCE = 200
35     POST_NMS_ROIS_TRAINING = 400
36     #As shafts are sleek shaped, the anchors are tuned down to 1:3(length:width) rectangles
37     RPN_ANCHOR_RATIOS = [1/3, 1, 3]
38 
39     #Increased from 28*28 to 56*56 to get better masks
40     MASK_SHAPE = [56, 56]
41 
42     USE_MINI_MASK = False
43 
44     LEARNING_RATE = 0.002
45 
46     #Bounding box and mask losses are heavily punished, forcing to decrease the error
47     LOSS_WEIGHTS = {
48     "rpn_class_loss": 1.,
49     "rpn_bbox_loss": 3.5,
50     "mrcnn_class_loss": 1.,
51     "mrcnn_bbox_loss": 3.5,
52     "mrcnn_mask_loss": 3.5
53 }
54 
55     DETECTION_NMS_THRESHOLD = 0.4
56     #Consider to be a shaft during vaildation if more than 70% sure
57     DETECTION_MIN_CONFIDENCE = 0.7
58 
59 
60 config = ShaftsConf()
61 
```

Figure 28. Final configuration parameters

After many experiments and trial and error, I was able to find the configuration parameters which produced optimum results and are shown in figure(28).

- A deeper FPN, ResNet101 is chosen as the backbone as it performed superior to ResNet50 in the real camera images at initial trials on the model.
- An initial learning rate of 0.002 was chosen to train the head layers of the network. The learning rate was decreased to 0.001 during training of all the layers of the model.
- A tuning of the RPN bounding boxes by changing anchor scales in line 28 and anchor ratios line 37 as shown in the above image.
- A single image is passed to the GPU to avoid memory overload problems.
- 250 images were passed per epoch to train the head layers and 400 images per epoch while training all the layers of the model.

- The loss weights were tweaked and increased to the values in line 47 to punish the losses occurring in bounding boxes and masks.
- The mask shape was increased to 56x56 pixels as shown in line 40 and a deconvolutional layer was added to the model to match the increased mask size.

5.4 Model training and results

The model was trained under different configuration parameters and was fine-tuned to achieve the best results. It could be observed that the ResNet101 backbone outperformed ResNet50 as the deeper FPN, ResNet101 was able to capture more features than its predecessor ResNet50. Each of the models was trained for 53 epochs, with 400 images per epoch.

A comparison between two models using backbones, ResNet101 and ResNet50 respectively has been made with other configuration parameters being the same. The important results and comparisons of prediction of the models on both synthetic as well as real images of the shafts are presented below.

5.4.1 Prediction on Synthetic images

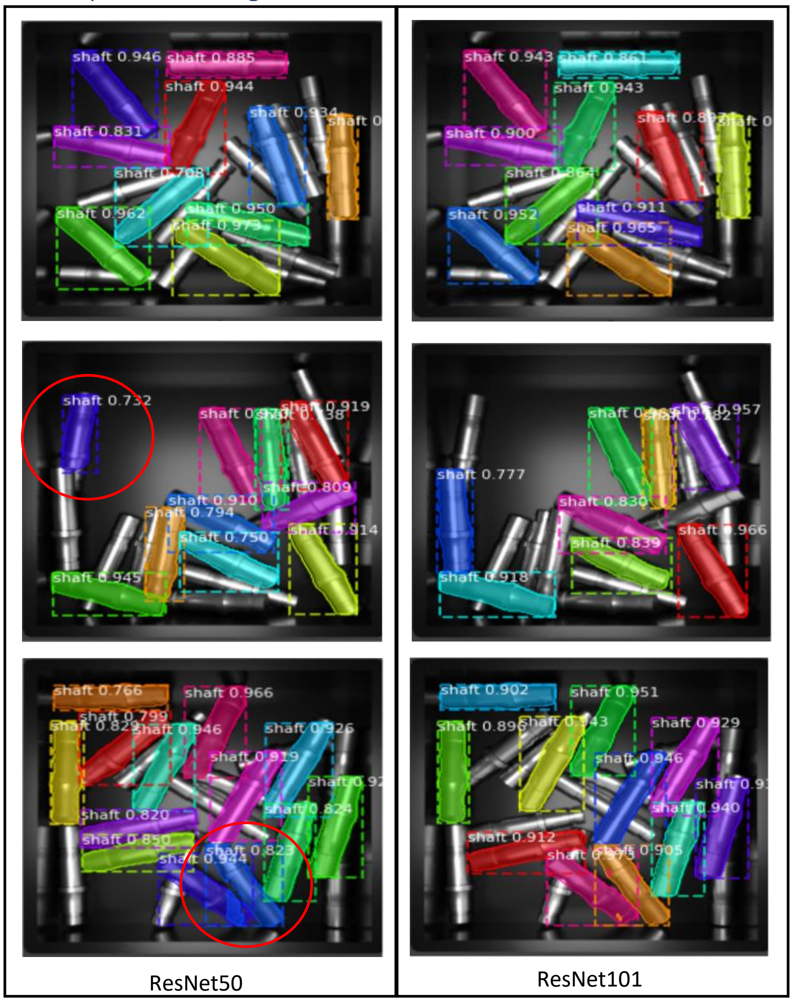


Figure 29. Prediction of Mask R-CNN on synthetic images

The above figure(29) compares the backbone ResNet50 architecture to ResNet101 used in Mask R-CNN while performing prediction on Synthetic images. The minimum confidence of prediction has been set to 60% in both models.

Although at first sight there is no striking difference between the two backbones, the ResNet101 provides better masks (row 3) along with more accurate top layer detection (row 2) in comparison with ResNet50.

5.4.2 Prediction on Real images of Shafts

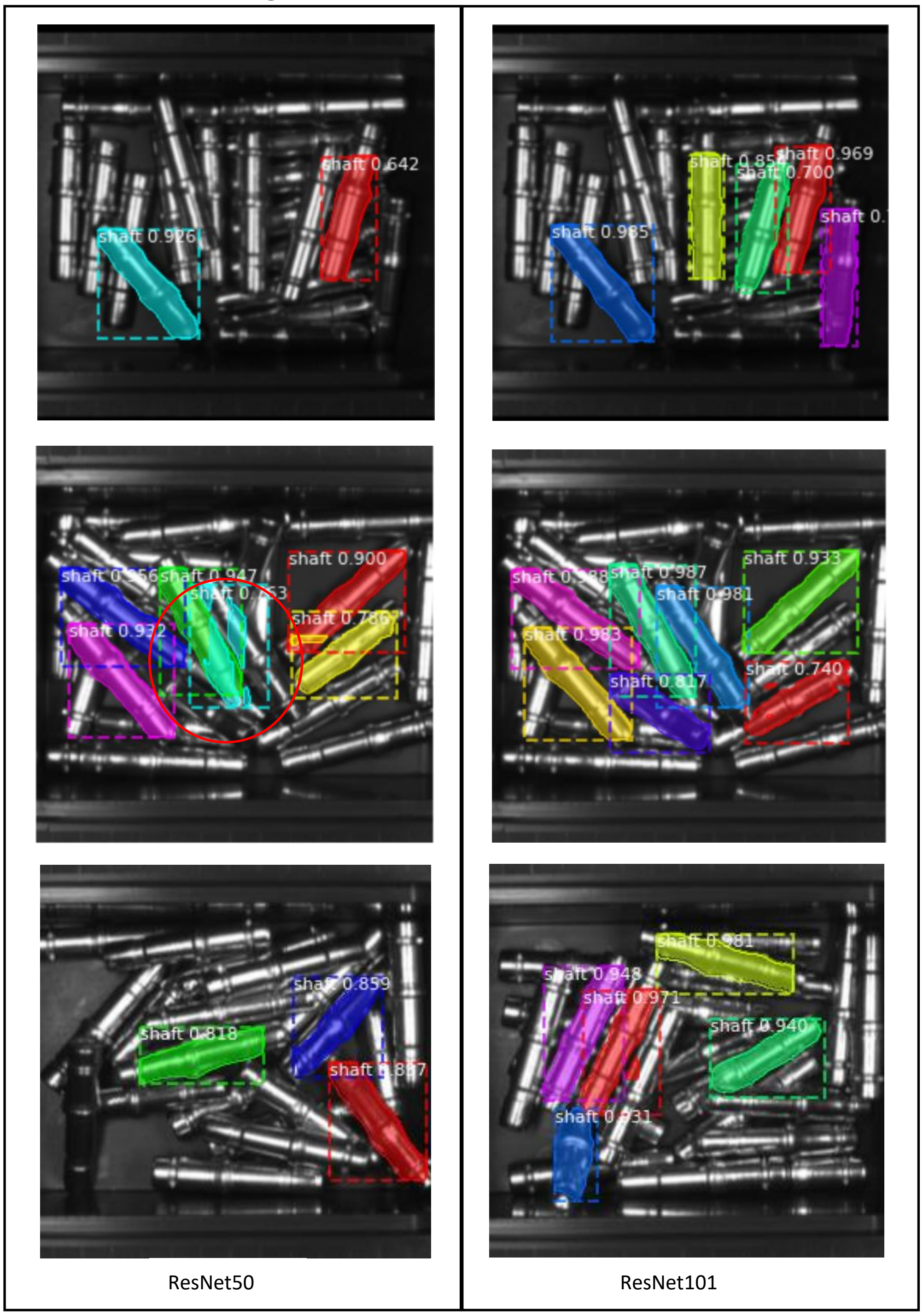


Figure 30. Prediction of Mask R-CNN on real images

When the models with two backbones are asked to make predictions, the model with ResNet101 outperforms ResNet50 with a considerable margin which could be seen above in figure(30).

The model with ResNet50 backbone struggles hard to do a decent prediction on the real images of the shafts. Incomplete masks are seen much often while predicting the real-life camera image and much often the masks are broken, thus falling behind the other model.

Even though the ResNet101 model also struggles with performance drop compared to its performance on the synthetic images, it still does a good job of predicting the topmost layer of the shafts. It performs much superiorly compared to its predecessor ResNet50 due to its deeper FPN network. The performance drops are discussed in the later section.

6 Principal Component Analysis

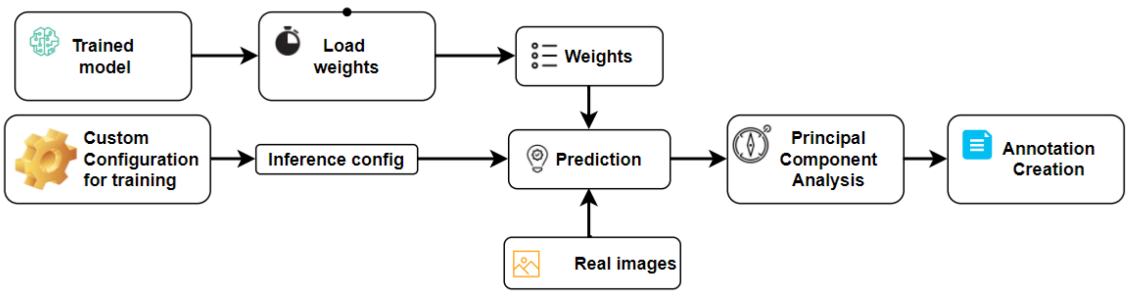


Figure 31. PCA analysis workflow

The finalized model with the best configuration is loaded in inference mode and the saved weights are called. These weights are used to make predictions on the real images. The predicted masks are subjected to Principal Component Analysis (PCA) to find the center and alignment of each predicted shaft.

Obtained center and orientations are drawn on the image for visualization purposes and the generated PCA data is saved to a text file.

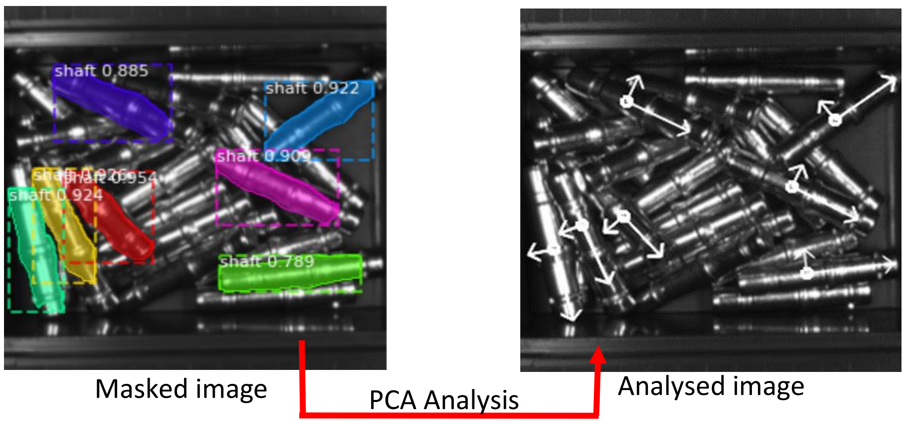


Figure 32. Results of PCA analysis

The right image of the figure(32) is the final output of the network, indicating the top-most shafts which are pickable by the robotic gripper.

6.1 Brief intuition of PCA

Principal Component Analysis is a statistical procedure to find the prominent features of datasets.

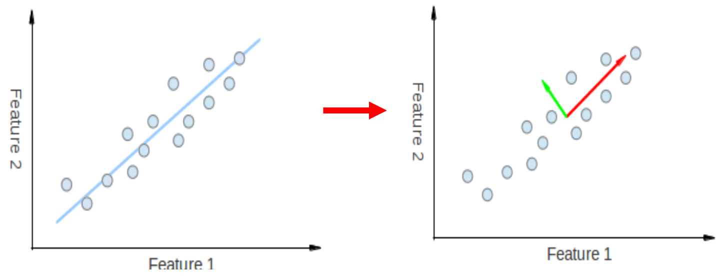


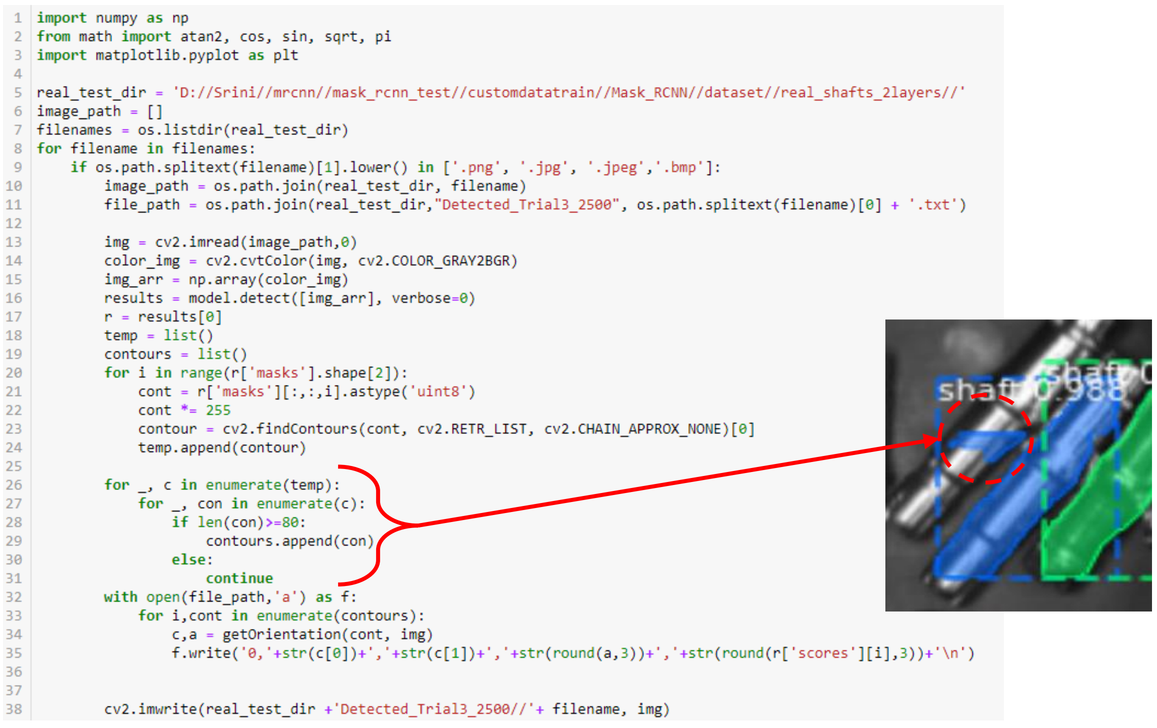
Figure 33. PCA analysis [14]

PCA finds the linear pattern hidden in the data points. The key point of this analysis is dimensionality reduction. In the left image of figure(33), the 2D data points are interpolated to a 1D blue line, to find the variation of the data points along the dimension reduced line. It can also be seen that the points in the right image vary more along the Feature 1 axis than Feature 2 axis, thus drawing a longer axis on the side of variation.

Performing PCA using OpenCV results in 2 eigenvectors and the center of the data points, which can be seen in the right image of the above figure. The red and green axes are principal components of the data points which have the origin at the center of the data points.

The source code from OpenCV documentation [14] is modified as functions for ease of understanding. Detailed step by step explanation of both theory of PCA analysis and code can be found in the documentation.

6.1.1 Implementation:



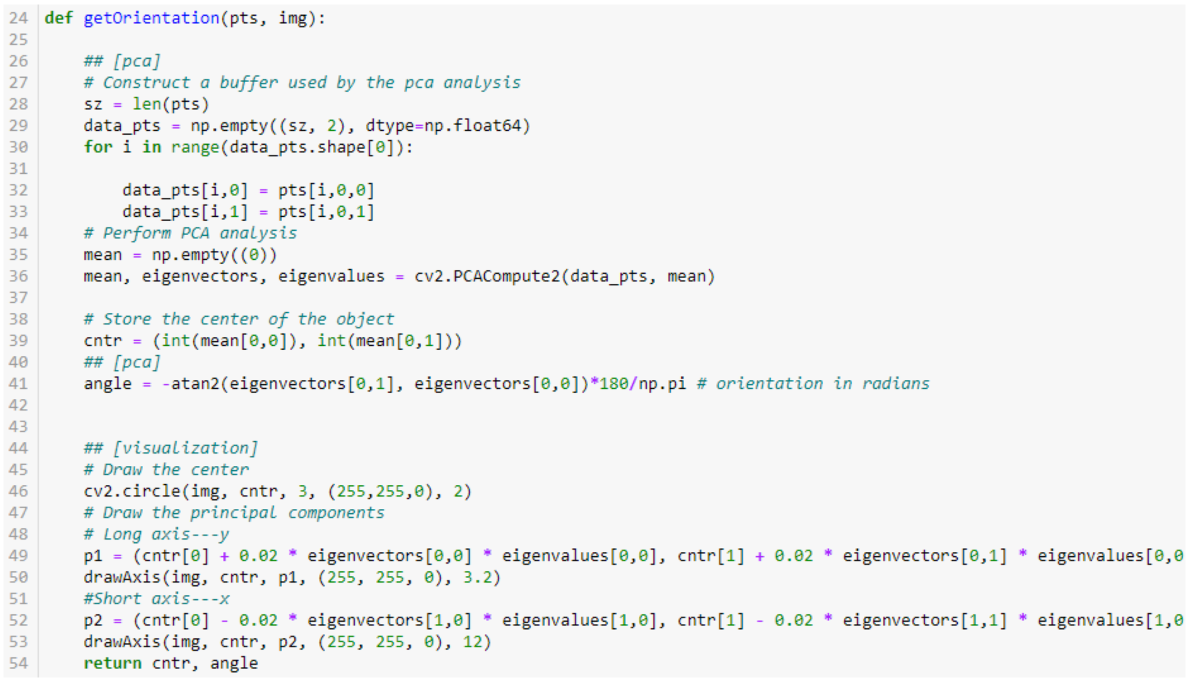
```
 1 import numpy as np
 2 from math import atan2, cos, sin, sqrt, pi
 3 import matplotlib.pyplot as plt
 4
 5 real_test_dir = 'D://Srini//mrcnn//mask_rcnn_test//customdatatrain//Mask_RCNN//dataset//real_shafts_2layers//'
 6 image_path = []
 7 filenames = os.listdir(real_test_dir)
 8 for filename in filenames:
 9     if os.path.splitext(filename)[1].lower() in ['.png', '.jpg', '.jpeg','.bmp']:
10         image_path = os.path.join(real_test_dir, filename)
11         file_path = os.path.join(real_test_dir,"Detected_Trial3_2500", os.path.splitext(filename)[0] + '.txt')
12
13         img = cv2.imread(image_path,0)
14         color_img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
15         img_arr = np.array(color_img)
16         results = model.detect([img_arr], verbose=0)
17         r = results[0]
18         temp = list()
19         contours = list()
20         for i in range(r['masks'].shape[2]):
21             cont = r['masks'][:,:,i].astype('uint8')
22             cont *= 255
23             contour = cv2.findContours(cont, cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)[0]
24             temp.append(contour)
25
26         for _, c in enumerate(temp):
27             for _, con in enumerate(c):
28                 if len(con)>=80:
29                     contours.append(con)
30                 else:
31                     continue
32         with open(file_path,'a') as f:
33             for i,cont in enumerate(contours):
34                 c,a = getOrientation(cont, img)
35                 f.write('0,'+str(c[0])+','+str(c[1])+','+str(round(a,3))+','+str(round(r['scores'][i],3))+'\n')
36
37
38         cv2.imwrite(real_test_dir +'Detected_Trial3_2500//'+ filename, img)
```

Figure 34. Implementation of PCA analysis

- In the above figure, Principal Component Analysis has been carried out. The file names of real camera images are read and stored into a variable in the for loop of line 8. Each image is predicted one after the other, simultaneously finding the orientation and center of each shaft.
- In line 14 OpenCV is used to convert the grayscale image into 3 channel images to make it compatible with Mask R-CNN.
- Prediction of the image is made in line 16 and the results are stored in variable 'r'.
- A for loop iterates through each mask of the image and the mask is further processed to run a contour detection code at line 23.
- Due to seldom occurring broken masks stored as a single nested list, the contour detection algorithm creates more than one contour for these masks. In line 26, a for

loop is implemented which thresholds the length of the masks, thus preventing double PCA on a single shaft. Small chunks of broken masks as shown in the red circle are ignored, thus concentrating on larger masks to get a single center per shaft.

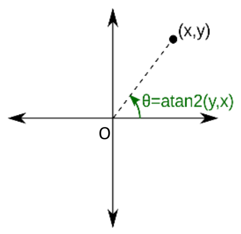
- Within the getOrientation() function, further processing and PCA analysis occur.
- The analyzed image is saved into the specified directory in line 33.
- Per image, the corresponding extracted annotations are saved in a text file as shown in with loop of the above figure in line 32.



```
24 def getOrientation(pts, img):
25 
26     ## [pca]
27     # Construct a buffer used by the pca analysis
28     sz = len(pts)
29     data_pts = np.empty((sz, 2), dtype=np.float64)
30     for i in range(data_pts.shape[0]):
31 
32         data_pts[i,0] = pts[i,0,0]
33         data_pts[i,1] = pts[i,0,1]
34     # Perform PCA analysis
35     mean = np.empty((0))
36     mean, eigenvectors, eigenvalues = cv2.PCACompute2(data_pts, mean)
37 
38     # Store the center of the object
39     cntr = (int(mean[0,0]), int(mean[0,1]))
40     ## [pca]
41     angle = -atan2(eigenvectors[0,1], eigenvectors[0,0])*180/np.pi # orientation in radians
42 
43 
44     ## [visualization]
45     # Draw the center
46     cv2.circle(img, cntr, 3, (255,255,0), 2)
47     # Draw the principal components
48     # Long axis---y
49     p1 = (cntr[0] + 0.02 * eigenvectors[0,0] * eigenvalues[0,0], cntr[1] + 0.02 * eigenvectors[0,1] * eigenvalues[0,0
50     drawAxis(img, cntr, p1, (255, 255, 0), 3.2)
51     #Short axis---x
52     p2 = (cntr[0] - 0.02 * eigenvectors[1,0] * eigenvalues[1,0], cntr[1] - 0.02 * eigenvectors[1,1] * eigenvalues[1,0
53     drawAxis(img, cntr, p2, (255, 255, 0), 12)
54     return cntr, angle
```

Figure 35. Orientation and center extraction using Principal Component Analysis

- The getOrientation() function takes the masks i.e., the XY coordinates along with the image as input. In lines 29-32 in figure(35), each mask is converted into an nx2 sized matrix with x and y coordinates as the columns.
- On the matrix of the current mask, a PCA analysis is performed in line 36 to calculate the center of the mask, and the corresponding eigenvectors which are used later for calculating the angle and to draw the axis for visualization.
- The calculated center is drawn as a circle in line 39 of the above figure and the angle is converted to degrees in line 41.
- p1 and p2 in lines 49 and 50 are the eigenvectors multiplied by their eigenvalue and are translated to the center of the contour. These variables are solely used for visualization purposes.
- The function drawaxis() draws the axes onto the image for visualization purposes. The eigenvectors are aligned with the contour center. These aligned vectors are then scaled, and the linewidth of the axes is set for better visibility. With the knowledge of the eigenvector, the angle of orientation can be calculated by using the arctangent as illustrated in figure(36).



Comparing with the code,

O = `cntr`

x = `eigenvectors[0,0]`

y = `eigenvectors[0,1]`

θ = `angle`

Figure 36. angle calculation with eigenvectors

6.2 Visualizations

Some of the PCA analyzed images are shown in the below figure, along with the masked figure and the original image.

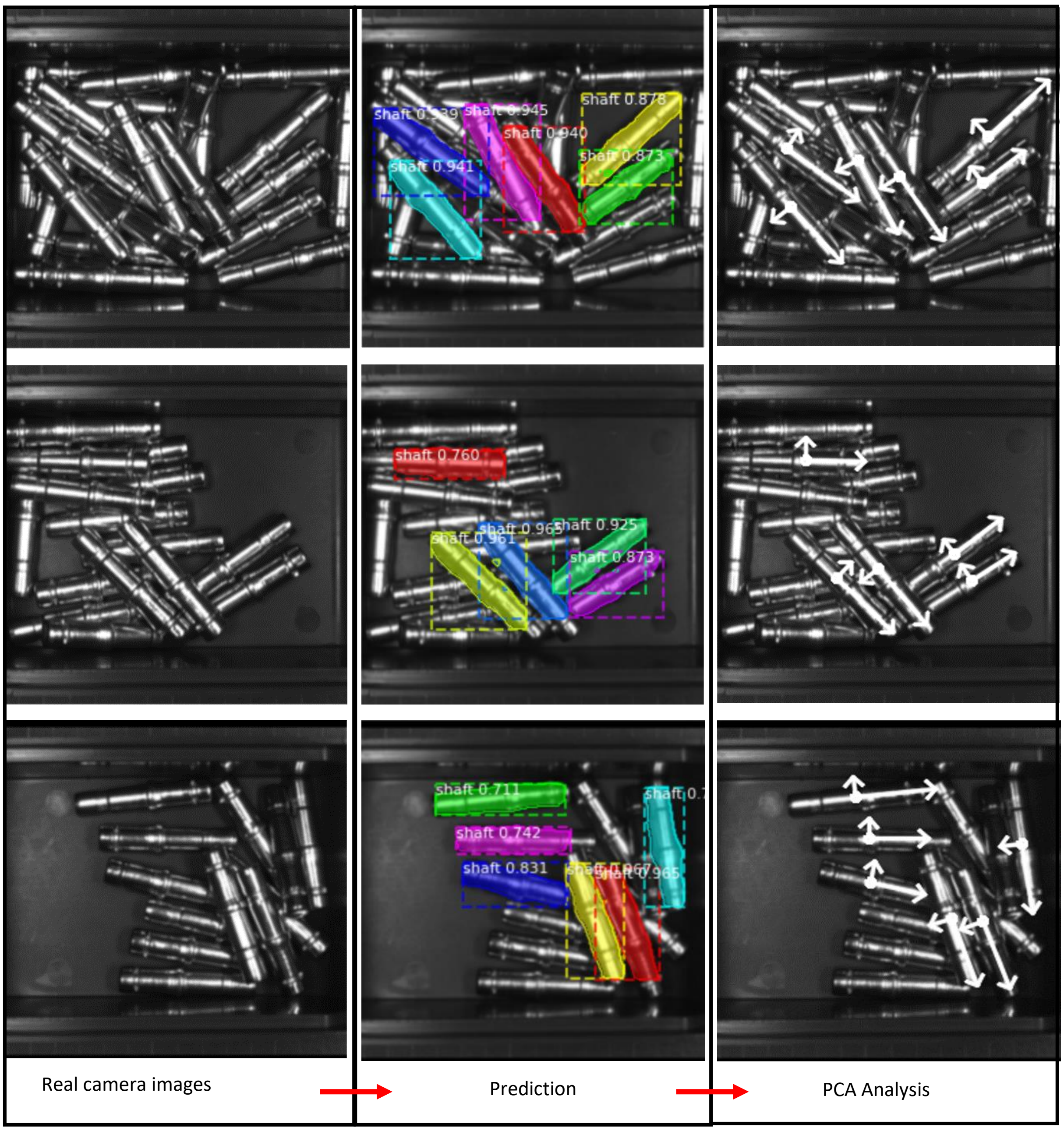


Figure 37. PCA analysis on real images with visualization

6.4 Annotation generation

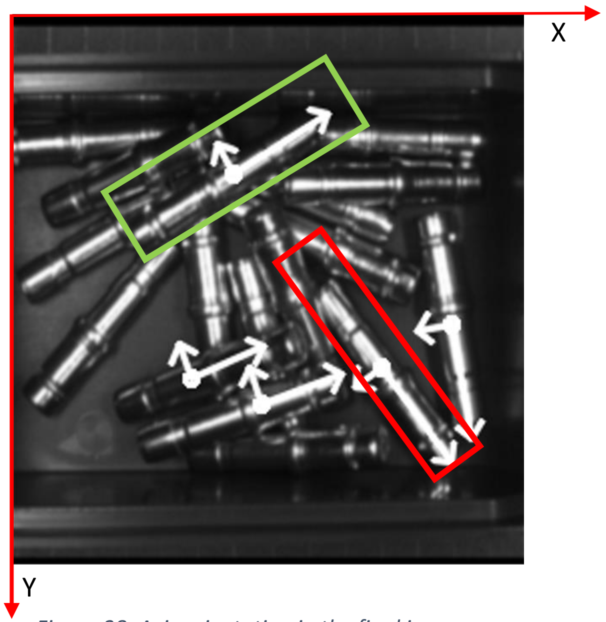
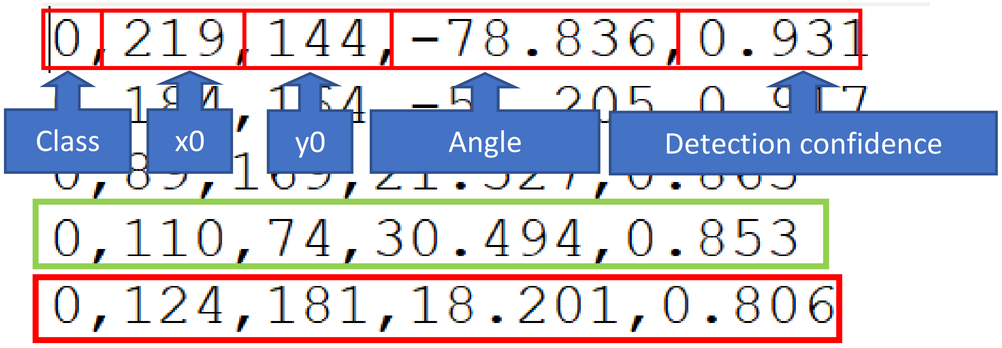


Figure 38. Axis orientation in the final image

The corresponding annotations generated to the image in figure(38) are displayed in figure(39) below. The detected shafts are ranked according to their detection confidence.



```
0,219,144,-78.836,0.931
0,18[illegible],1[illegible]4,-5[illegible].205,0.9[illegible]7
0,89,16[illegible],21.[illegible]27,0.[illegible]
0,110,74,30.494,0.853
0,124,181,18.201,0.806
```

Figure 39. Annotations generated from PCA analysis

The shafts with boxes of the figure(38) are indicated in figure(39) by drawing boxes of the same colour.

- Class: The ID number of the type of object. In this project, the class always is zero, as there is only one type of object.
- x0, y0: 'X' and 'Y' coordinates of the center of the detected object.
- Angle: The angle of orientation of the object. Angle is calculated in degrees, with reference to the x-axis of the first quadrant.
    - Ranges between +180° to -180°.
    - Positive, when the long axis of the PCA is above the x-axis
    - Negative, when the long axis of the PCA is below the x-axis.
- Detection Confidence: The confidence of the neural network, that the detected object belongs to its category.

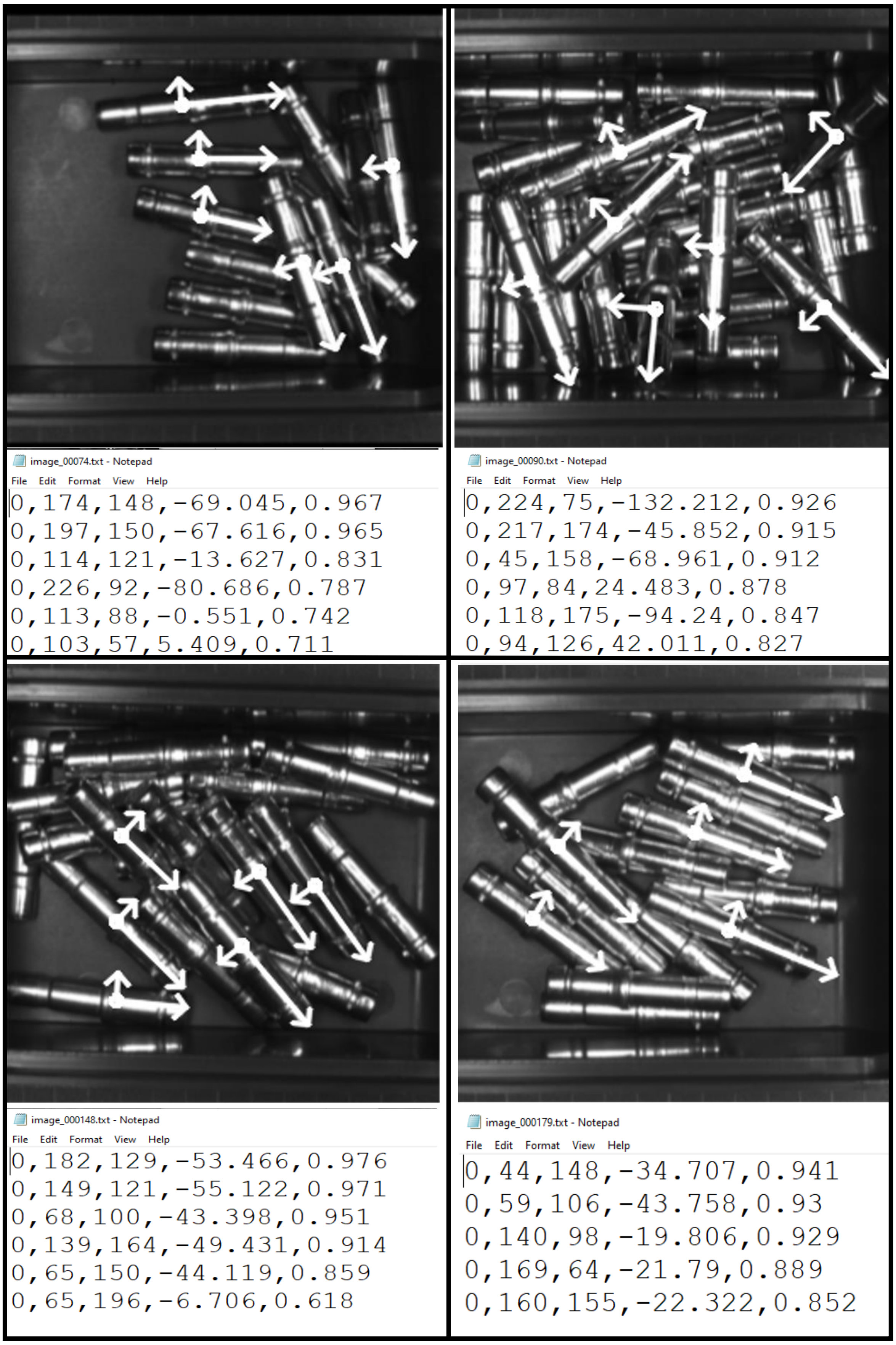


image_00074.txt - Notepad
File Edit Format View Help

```
0,174,148,-69.045,0.967
0,197,150,-67.616,0.965
0,114,121,-13.627,0.831
0,226,92,-80.686,0.787
0,113,88,-0.551,0.742
0,103,57,5.409,0.711
```

image_00090.txt - Notepad
File Edit Format View Help

```
0,224,75,-132.212,0.926
0,217,174,-45.852,0.915
0,45,158,-68.961,0.912
0,97,84,24.483,0.878
0,118,175,-94.24,0.847
0,94,126,42.011,0.827
```

image_000148.txt - Notepad
File Edit Format View Help

```
0,182,129,-53.466,0.976
0,149,121,-55.122,0.971
0,68,100,-43.398,0.951
0,139,164,-49.431,0.914
0,65,150,-44.119,0.859
0,65,196,-6.706,0.618
```

image_000179.txt - Notepad
File Edit Format View Help

```
0,44,148,-34.707,0.941
0,59,106,-43.758,0.93
0,140,98,-19.806,0.929
0,169,64,-21.79,0.889
0,160,155,-22.322,0.852
```

Figure 40. Visualization of some PCA shafts with annotations

7 Evaluation metrics

The model was evaluated by calculating mean average precision(mAP). The mAP denotes the precision of the model in determining the ability of the model to detect the images in comparison to the ground truth of the test set.

The find mAP we need to calculate precision and recall. Precision is the measure of finding an object. The recall is the measure of correctly classifying the found object. These terms are defined as,

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

where,

$$TP = True\ Positve$$

$$FP = False\ Positve$$

$$FN = False\ Positve$$

The terms TP, FP and FN are calculated with the help of Intersection over Union(IOU) [15] of an object. IoU compares the ground truth to the detected area as shown in figure(40). The values of IoU threshold can be varied from 50% to 95% thus creating a challenging benchmark. If the intersection of the detected area is more than the IOU threshold, then the detection is considered as TP otherwise FP. If there is no detection at a ground truth object or the predicted label is wrong, then it is considered as FN.

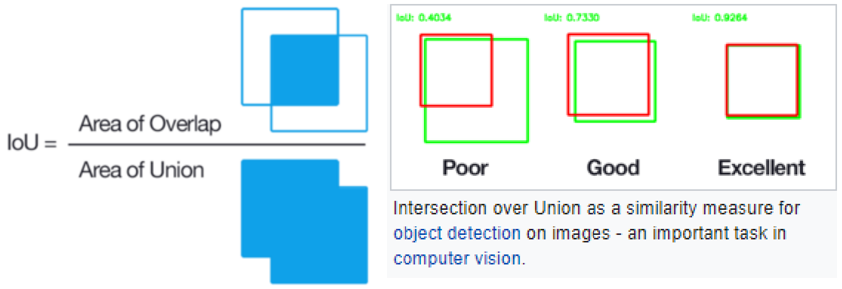
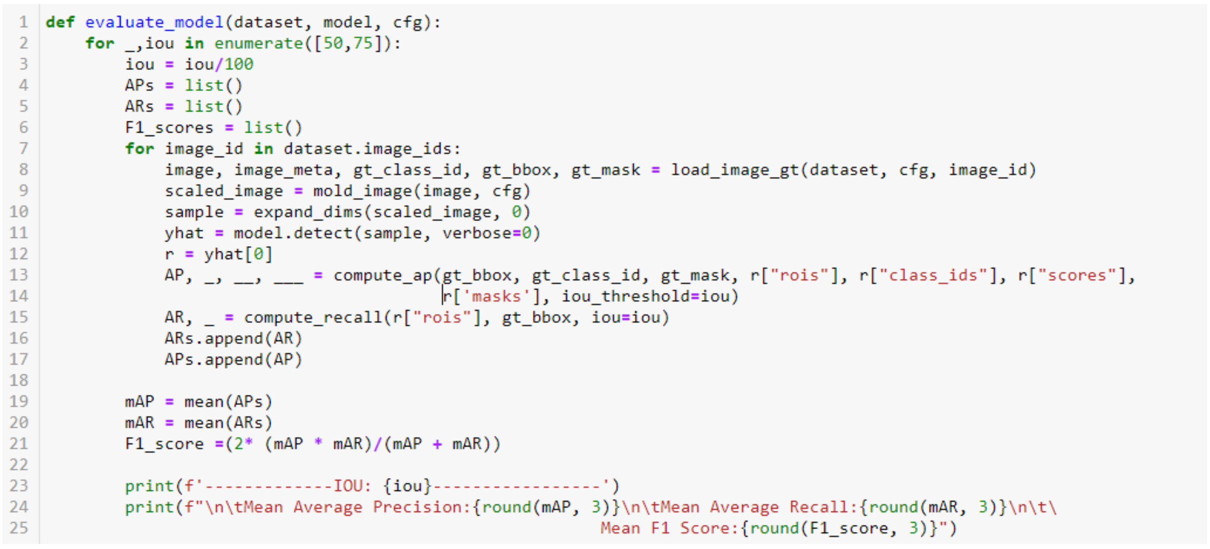

IoU =
Area of Overlap
Area of Union
IoU: 0.4034
IoU: 0.7330
IoU: 0.9264
Poor
Good
Excellent
Intersection over Union as a similarity measure for object detection on images - an important task in computer vision.

Figure 41. Intersection over Union [15]

If the model can detect or perform a perfect instance segmentation in comparison to the ground truth, will get a score of 100% and if it can't detect any the model gets a score of 0%. The benchmark score for Mask R-CNN on the COCO dataset is 36.7%.

As evaluation metrics, mean average precision (mAP), mean average recall (mAR) and F1 scored are used.

7.1 Quantitative results



```python
1  def evaluate_model(dataset, model, cfg):
2      for _,iou in enumerate([50,75]):
3          iou = iou/100
4          APs = list()
5          ARs = list()
6          F1_scores = list()
7          for image_id in dataset.image_ids:
8              image, image_meta, gt_class_id, gt_bbox, gt_mask = load_image_gt(dataset, cfg, image_id)
9              scaled_image = mold_image(image, cfg)
10             sample = expand_dims(scaled_image, 0)
11             yhat = model.detect(sample, verbose=0)
12             r = yhat[0]
13             AP, _, __, ___ = compute_ap(gt_bbox, gt_class_id, gt_mask, r["rois"], r["class_ids"], r["scores"],
14                                         r['masks'], iou_threshold=iou)
15             AR, _ = compute_recall(r["rois"], gt_bbox, iou=iou)
16             ARs.append(AR)
17             APs.append(AP)
18
19         mAP = mean(APs)
20         mAR = mean(ARs)
21         F1_score =(2* (mAP * mAR)/(mAP + mAR))
22
23         print(f'-------------IOU: {iou}-----------------')
24         print(f"\n\tMean Average Precision:{round(mAP, 3)}\n\tMean Average Recall:{round(mAR, 3)}\n\t\
25                                                            Mean F1 Score:{round(F1_score, 3)}")
```

Figure 42. Performance measure calculation

Using the function evaluate_model() in figure(41) mAP, mAR and F1 score are calculated. A comparison between two types of models is done with backbones of ResNet50 and ResNet101 for IoU of 50% and 75% as per the standards.

The average precisions and average recalls are calculated for each image by using the inbuilt functions compute_ap() in line 13 and compute_recall() in line 15 of the above image.

A test set of 1000 images is used to check for the stable performance of the model. The inferences of both the models are listed in figure(42) below.

Model	AP_{50}			AP_{75}		
	mAP	mAR	F1	mAP	mAR	F1
Mask R-CNN + ResNet50	0.448	0.494	0.47	0.425	0.458	0.441
Mask R-CNN + ResNet101	0.508	0.558	0.532	0.478	0.499	0.488

Figure 43. Comparison of results

Even though the performance is not strikingly huge, Mask R-CNN performs way better on the real dataset as seen in visualization before.

7.2 Best predictions

Some of the best predictions where most of the pickable shafts are detected are visualized in this section.

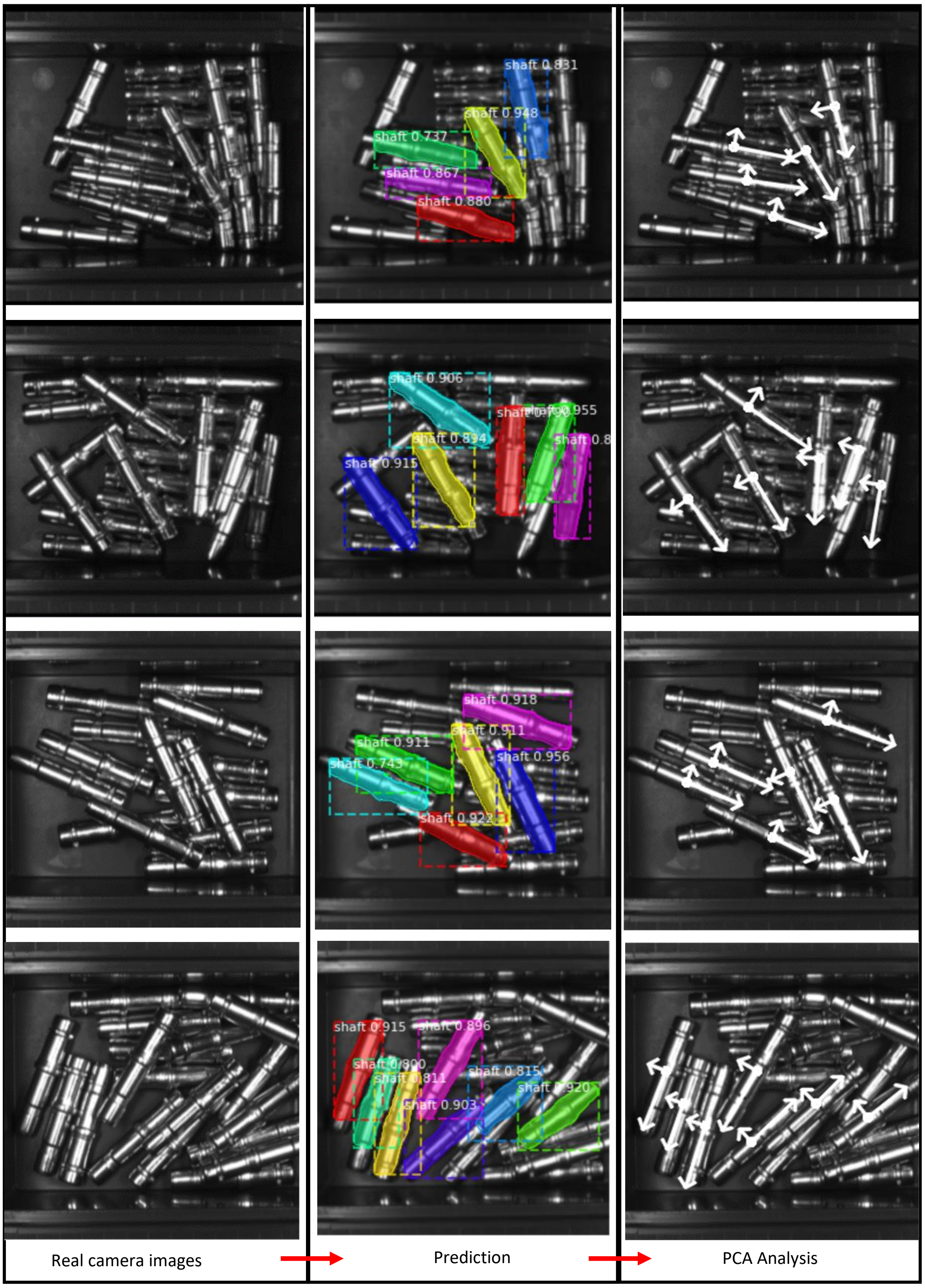


Figure 44. Best obtained predictions

8 Discussions

From the visualizations of the Mask R-CNN + ResNet101 model on prediction of real shaft images, it can be concluded that the instance segmentation network does a fine job of detecting the shafts in the top layer of the box which are suitable for picking.

In this section, some still existing problems of this network along with their probable causes are discussed.

8.1 Incomplete masks on real images

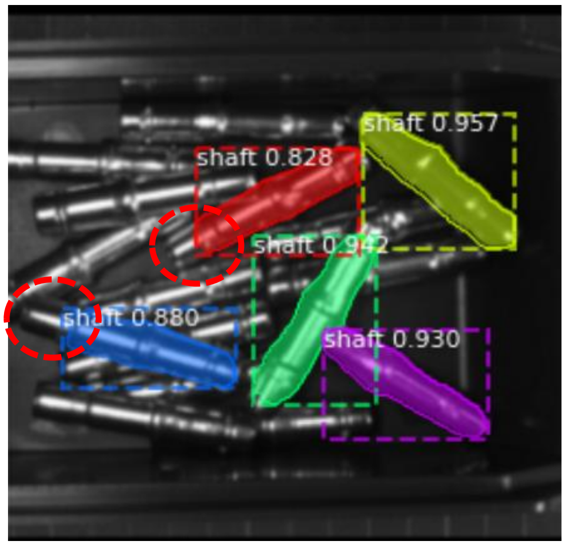


Figure 45. Problem of incomplete masks

There still exists the problem of incomplete masks while predicting the camera captured images of the shafts as seen in figure(44). This problem can be associated with the dissimilarity of the synthetic images in comparison with the actual images. After the increase in the length of the shafts of synthetic images by 10%, a significant improvement could be achieved, but not eliminating the problem. This problem causes the PCA analysis to output a slightly shifted center.

8.2 Distorted masks

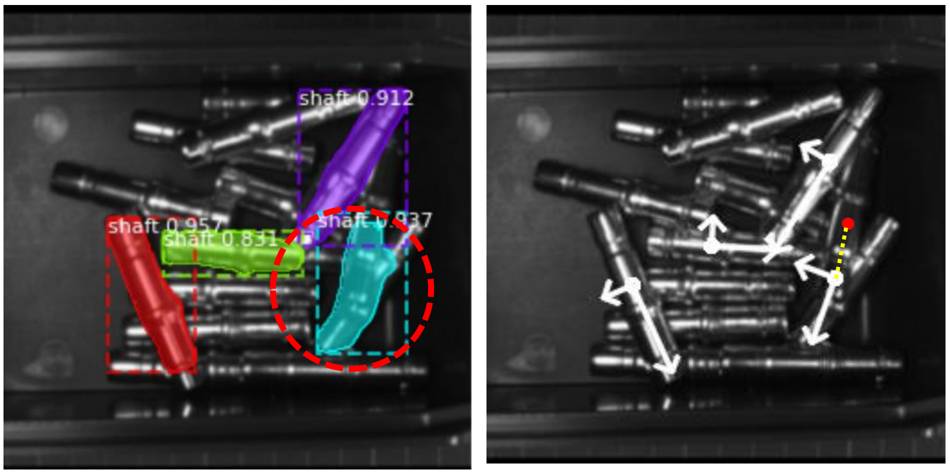


Figure 46. Problem of distorted masks

In very rare cases the masks are severely distorted are predicted by considering two shafts as one like the left image of the figure(45). This problem causes the PCA analysis to output incorrect center and orientation of the shafts as shown by the yellow line in the right image.

Red being the approximate true center, we see a false calculation of center and orientation in the right image due to the distorted blue mask in the left image.

8.3 Broken masks and invisible shafts

In some cases, the masks are broken into more than one contour. Even though the problem of double PCA analysis has been overcome by using a thresholding loop, this problem causes PCA analysis to output a slightly offset result.

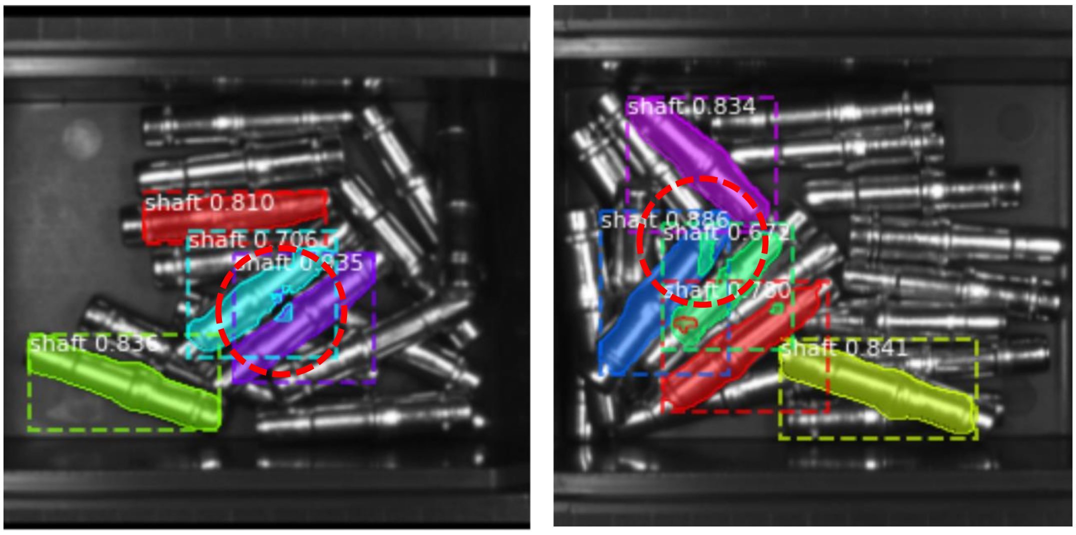


Figure 47. Problem of broken masks

When such a case occurs, the shafts are predicted with lower confidence, as seen in the figure(46) above. The sky-blue mask prediction has been made with a confidence of 70.6% on the left image and the parrot green mask on the right with a confidence of 67.2%, meanwhile, other shafts are predicted with a minimum confidence of 80%.

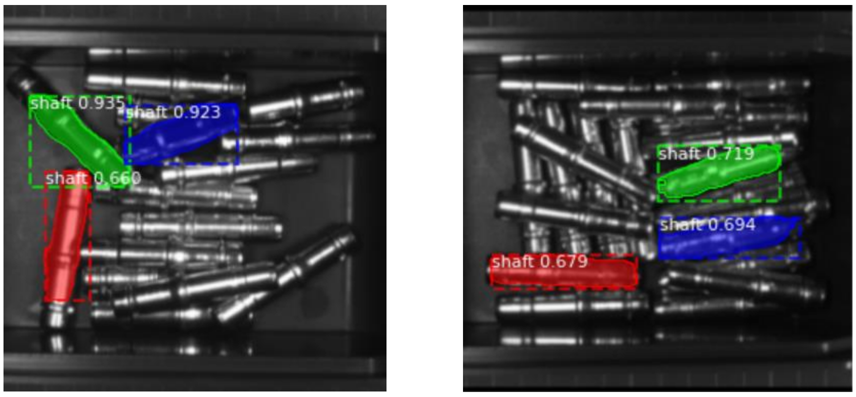


Figure 48. Problem of invisible shafts

It also sometimes occurs that the model is not able to predict all the pickable shafts in the image as indicated in the figure(47). Better hyperparameter tuning and the creation of a larger synthetic dataset could overcome these problems.

9 Conclusion and future work

9.1 Conclusion

The objective of this project was to explore the domains of synthetic data generation and how well an instance segmentation model performs in real when trained on synthetic data. It was found that the model performs almost up to expectations for the initial tuning with some imperfections. It can also be possible that some hyperparameters were not optimally tuned or were left out from tuning.

With an optimum hyperparameter tuning and a sufficient and more sophisticated dataset, I am optimistic that the model would do a good job of segmenting the real shaft images and would be suitable to be deployed for industrial purposes. Even though there are many approaches to bin picking, these approaches use images with depth information. This project is an honest approach to the implementation of bin picking using 2D images.

Overall, the approach to utilize near to real synthetic data for instance segmentation appears to be promising. This workflow can be easily generalized for different kinds of objects. Due to the ability of Mask R-CNN to perform both semantic and instance segmentations, this project can also be extended for further applications of bin picking with objects of multiple classes in the bin rather than only one.

9.2 Future work

- Better hyperparameter tuning is needed to achieve better results in this model. It could have occurred such that, some hyperparameter has not been optimally tuned or would have been missed out of tuning.
- With training the model on a larger dataset for longer epochs and with more steps per epoch, the model would generalize better.
- If better instance segmentation models are invented, it is much likely that those will perform better on this task. Thus, it seems to be practical to try this approach on other models.
- Further addition of a key point detection layer or usage of depth-sensing cameras would increase the performance of this network furthermore.

10 References

[1] I. Matterport, "Github-Matterport Mask-RCNN," 20 March 2018. [Online]. Available: https://github.com/matterport/Mask_RCNN.

[2] Kaiming He, Georgia Gkioxari, Piotr Dollár and Ross Girshick, "Mask R-CNN," p. 12, 24 January 2018.

[3] "Free and Open 3D creation software-Blender," Blender, [Online]. Available: https://www.blender.org/.

[4] O. CV, "opencv-Python," PyPi, [Online]. Available: https://pypi.org/project/opencv-python/.

[5] "Arthur_Samuel-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Arthur_Samuel.

[6] D. H. G. &. W. R. Rumelhart, "Learning representations by back-propagating errors.," *Nature,* no. https://doi.org/10.1038/323533a0, p. 533–536, 1986.

[7] J. L. a. T. D. Evan Shelhamer, "Fully Convolutional Networks for Semantic Segmentation," *arXiv.org,* vol. 39, no. 4, 2017.

[8] R. G. J. D. T. D. J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *2014 IEEE Conference on Computer Vision and Pattern Recognition,* June 2014.

[9] R. Girshick, "Fast R-CNN," *arXiv.org,* April 2015.

[10] K. H. R. G. a. J. S. Shaoqing Ren, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," 6 January 2016. [Online]. Available: https://arxiv.org/abs/1506.01497v3.

[11] P. D. G. K. H. B. H. Tsung-Yi Lin, "Feature Pyramid Networks for Object Detection," *arXiv.org, Cornell University, Facebook AI Research (FAIR).*

[12] "Create COCO annotations from scratch," 10 January 2019. [Online]. Available: https://www.immersivelimit.com/tutorials/create-coco-annotations-from-scratch/#coco-dataset-format.

[13] S. C. Desapogu Sankeerth, "Design and generation of synthetic data for Optical localization of objects with machine learning using open source tool - Blender," TH Rosenheim, Rosenheim, Bayern, April 2020.

[14] OpenCV, "OpenCV_ Introduction to Principal Component Analysis (PCA)," [Online]. Available: https://docs.opencv.org/3.4/d1/dee/tutorial_introduction_to_pca.html.

[15] Wikipedia, "Jaccard index - Wikipedia," Wikipedia, 27 August 2021. [Online]. Available: https://en.wikipedia.org/wiki/Jaccard_index.

[16] A. Opperman, "What is Deep Learning and How does it work?," 12 Nov 2019. [Online]. Available: https://towardsdatascience.com/what-is-deep-learning-and-how-does-it-work-2ce44bb692ac.

[17] P. L. Liu, "Towards data science," Medium, 29 April 2020. [Online]. Available: https://towardsdatascience.com/single-stage-instance-segmentation-a-review-1eeb66e0cc49.

[18] Z. Yang, R. Dong, H. Xu and J. Gu, "Instance Segmentation Method Based on Improved Mask R-CNN for the Stacked Electronic Components," *Electronics 2020,* pp. 9, 886.