Karlsruher Institut für Technologie

Stuttgart, 27.09.2021

INTERNSHIP REPORT

Ping-Cheng Wei (B.Sc.)

Period: 01.05.2021 – 30.09.2021

Supervisor: Muhammad Usman Khalid

Table of Contents

1.Company

Fraunhofer-Gesellschaft is one of the worldwide leading german research organization with 72 institutes spread throughout Germany, each focusing on different fields of applied science. With some 29,000 employees, mainly scientists and engineers, it is the biggest organization for applied research and development services in Europe. Fraunhofer IPA – one of the Fraunhofer-Gesellschaft's largest institutes – was founded in 1959 in Stuttgart and employs nearly 1000 workers. The focus of their research and development work is on organizational and technological issues related to the manufacturing industry. With 15 specialist departments, Fraunhofer IPA covers the entire field of manufacturing engineering, working on an interdisciplinary basis with industrial enterprises from the following sectors: automotive, machinery & equipment, electronics & microsystems, energy, medical engineering & biotechnology, and the process industry.

1.1. Department

The department "Robot and Assistive Systems" is concerned with the development of robot systems and automation solutions for industry. The department develops and implements key technologies in innovative industrial robots, service robots and intelligent machines. The Center for Cognitive Robotics – one of the subarea where my internship mainly focuses on – extends the research and innovation expertise of Fraunhofer IPA by implementing Artificial Intelligence (AI) in the research fields of robotics and human-machine-interaction. In combination with AI methods, robotics in real applications is used to enhance the performance as well as autonomy of robot systems within the sense of "automating automation" and master complex tasks. The Center develops solutions that enable the technology to be "universally applied" and the industrial robots gain the cognitive ability to perceive their environments.

1.2. Tasks

The main goal of my internship is to realize and implement a universally applied method based on machine learning for our robotic grasping task so that the robotic arm could detect not only all kinds of objects but also more effectively and efficiently in comparison with traditional methods. Moreover, in order to fit into the whole grasping process pipeline, which is totally automated, I have to implement an end-to-end solution and make sure it could run in real-time without too high latency.

2.Project 1 / Week 1 – 9

The goal of the project 1 is to implement a contour detection algorithm also known as edge detection at first to filter our images and then try to find a way to combine those contours to decide where our objects and instances are and separate them from back-ground by marking them with different colors and labels for each instance, which is so called instance segmentation.

The idea behind this approach is that we want our model could be universally applied in any grasping tasks. That means our model has to have the capacity to detect all kinds of objects. However, this leads to an unfeasible task to be solved by using the DNN for classification, which is the most typical way to realize object detection, since there are more than 1 million of different objects in the world and you can't really train a model to classify 1 million different things, either due to the lack of the training images, which you already need like 1 million of images for just 1000 classes to train a good model, or the insufficient complexity of the model. Therefore, in order to counter this problem we come up with this idea of exploiting contour detection plus combinatorial algorithm to achieve our goal. Instead of letting our model really understand or classify what the objects are, we actually just need to let the model know there are objects. Moreover, because all objects with respect to each other and background have contours in between in common, a well-trained contour detection ML model could be suited and speed up the detection speed as the time-consuming part for classification is no longer needed. That's why we decided to utilize a contour detection ML model plus combinatorial algorithm to realize the instance level segmentation.

Inside this project the following tasks have to be done:

- Go through the research papers about contour detection and combinatorial algorithm
    - Object Contour Detection with Encoder-Decoder Network (CEDN) [1]
    - Dense Extreme Inception Network (DexiNed) [2]
    - Multiscale Combinatorial Grouping (MCG) [3] [4]
- Implement different contour detection models and compare the results
- Test the combinatorial algorithm to propose the position of objects

2.1. Dense Extreme Inception Network (DexiNed)

2.1.1. Theory

The main goal of Object Contour Detection with an Encoder-Decoder Network is to develop an end-to-end edge detection system that automatically learn the type of rich hierarchical features. The network model has multi-scale and multi-level feature learning neural network structure as shown in below:

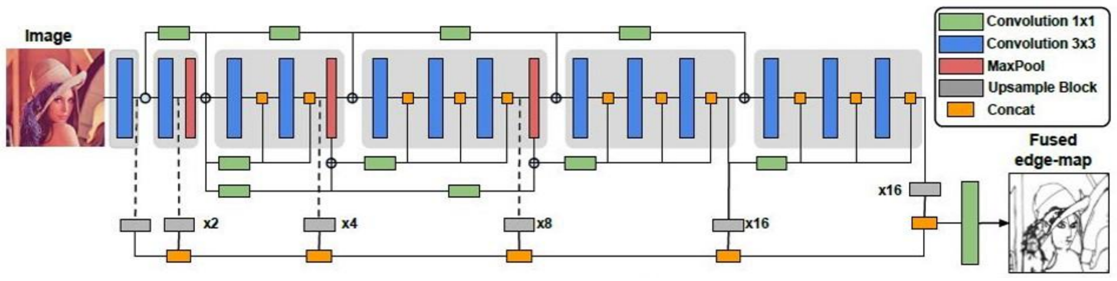


Figure 2.1.1-1: Dense Extreme Inception Network, consists of an encoder composed by six main blocks (showed in light gray). The main blocks are connected between them through 1x1 convolutional blocks. Each of the main blocks is composed by sub-blocks that are densely interconnected by the output of the previous main block. The output from each of the main blocks is fed to an upsampling block that produces an intermediate edge-map in order to build a Scale Space Volume, which is used to compose a final fused edge-map.

As we could directly notice form the structure, DexiNed is based on VGG16 architecture, which was proposed by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab of Oxford University in 2014 in the paper [5], to extract the features out of the original image but without FC-layer and softmax at the end of the architecture. DexiNed ameliorate this structure by expending the complexity, such as outputting of feature map from each stack and producing intermediate edge map by upsampling outputted feature maps. The reason for this is that some important features have lost during the downsampling process in the backbone, which were necessary if we want to increase the model accuracy. Therefore, in order to make sure that all the features are effectively utilized, DexiNed branches over each stack and reuses each outputted feature map to generate an intermediate edge map. Finally, all intermediate edge maps would then be concatenated and produce a final fused edge-map as it is shown in Figure 2.1.1-1.

Each pixel inside of the fused edge map represents the probability of whether it is an edge or not. Therefore, in the final step, we only need to set up the threshold to filter the result and we get a binary image, where 1 corresponds to edge and 0 to non-edge. The value of the threshold is usually around 0.5.

As for the Loss Function of the model for the back propagation, DexiNed exploits the “weighted cross-entropy [6]” since it is a deep supervised model which could be summarized as a regression function ð, that is, $\hat{Y} = ð(X, Y)$, where X is an input image, Y is its respective ground truth, and $\hat{Y}$ is a set of predicted edge maps. $\hat{Y} = [\hat{y}_1, \hat{y}_2, \dots, \hat{y}_N]$, where $\hat{y}_i$ has the same size as Y, and N number of outputs from each upsampling block; $\hat{y}_N$ is the result from the last fusion layer $f(\hat{y}_N = \hat{y}_f)$. The overall loss function could be tackled as flowed:

$$\begin{aligned} \ell^n(W, w^n) = -\beta \sum_{j \in Y^+} \log \sigma\left(y_j = 1 \middle| X; W, w^n\right) \\ -(1-\beta) \sum_{j \in Y^-} \log \sigma\left(y_j = 0 \middle| X; W, w^n\right) \end{aligned} \tag{1}$$

$$\mathcal{L}(W, w) = \sum_{n=1}^{N} \delta^n \times \ell^n(W, w^n) \tag{2}$$

, where W is the collection of all network parameters and w is the n corresponding parameter, δ is a weight for each scale level. $\beta = |Y^-|/|Y^+ + Y^-|$ and $(1-\beta) = |Y^+|/|Y^+ + Y^-|$ ($|Y^+|$, $|Y^-|$ denote the edge and non-edge in the ground truth).

2.1.2. Implementation and Results

Since the source code of the DexiNed is published as open source in GitHub, I downloaded it and configured the python environment for it. The environment for DexiNed is based on python 3.7 and TensorFlow 2 version. Other libraries and modules are also needed to install but to keep it short, these two are the most important. After finishing all the installation processes and fixing some buges, I tested the model with most updated model weights, which is provided by the author. The results could be find in Figure 2.1.2-1 in the next page.

In this figure there are three different scenarios arranged from top to bottom, representing easy to challenge cases. we notice that as the scenarios become more and more complex, namely filling with enormous amount of objects and even all identical objects, so-called instances, the model could not clearly decide where the contours are and therefore lots of noises were included (see the third row of images). However, if the condition were not too complicated, the model had indeed achieved a great job. The reason for the noises might be caused by the similarity of each object as well as lots of identical instances like in the third image. Since they are already difficult for human perception, there is no doubt that they would also be difficult for a computer. Furthermore, the texture on the surface of the object also plays a role that it acts as deception for the model, as you could tell from the chess plaid in the first image or the sock in second image.

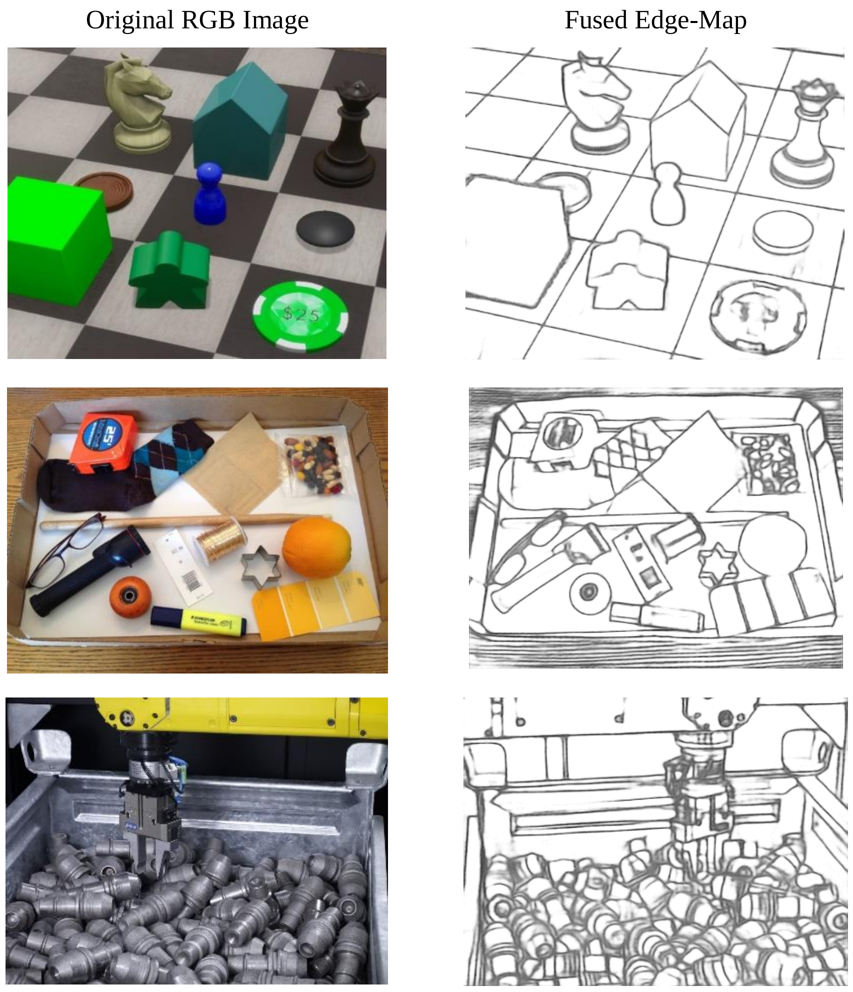


Figure 2.1.2-1: The results of DexiNed model on different level of scenarios from top to bottom

2.2. Object Contour Detection with Encoder-Decoder Network (CEDN)

2.2.1. Theory

Object contour detection with fully convolutional encoder-decoder network is a model based on a “convolutional encoder” and a “convolutional decoder”, as you could tell from the name. CEDN is designed to focus on detecting higher-level object contours. The overall architecture of CEDN is shown in Figure 2.2.1-1:

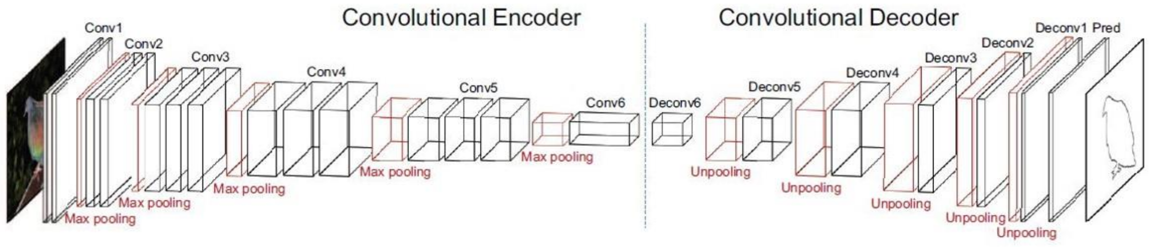


Figure 2.2.1-1: Architecture of the proposed fully convolutional encoder-decoder network

As shown in Figure 2.2.1-1, CEND initializes the encoder with VGG-16 [5] up to FC-6 layer as a pre-trained model to extract the features and convert the FC-6 to a convolutional layer, named "conv6", and it builds the decoder part symmetrically to encoder with 6 upsampling blocks, which are initialized with random value and optimized during the training. The Input image would be resize to (224, 224, 3) and downsampling with the following size: 224 -> 112 -> 56 -> 28 -> 14 -> 7. The output would be in size (7, 7, 512). After encoding, decoder was used to deconvolute the encoder output to get a (224, 224, 3) image result. Moreover, it is important to mention that CEDN develops a special system called "Penalty" by calculating the loss for the backpropagation since "contour" & "non-contour" pixels are extremely imbalanced. For example, penalty for being a "contour" is set 10 times larger than for being a "non-contour". By doing so, we could avoid our model prediction to be skewed due to the imbalance. Furthermore, CEDN uses Adam method [7] – a method for stochastic optimization – to optimize the network parameters, which is more efficient than standard stochastic gradient descent.

2.2.2. Implementation and Results

The process of implementation of CEDN is similar with DexiNed since the source code is also publish as open sources in GitHub. Thus, I downloaded it, configured the environment, and tested it with the pre-trained model. The results of the contour maps are shown in the Figure 2.2.2-1 in the following page. At here the results are also binary images with a threshold set to 0.5. We can see that the model had done a decent job deciding clear boundaries between contour and non-contour. Moreover, in comparison with DexiNed, CEDN seems to have a better performance on reducing the noises, such as textures of the clothes in the second image. But it also overdoes it a little bit because some objects in the first image were not shown.

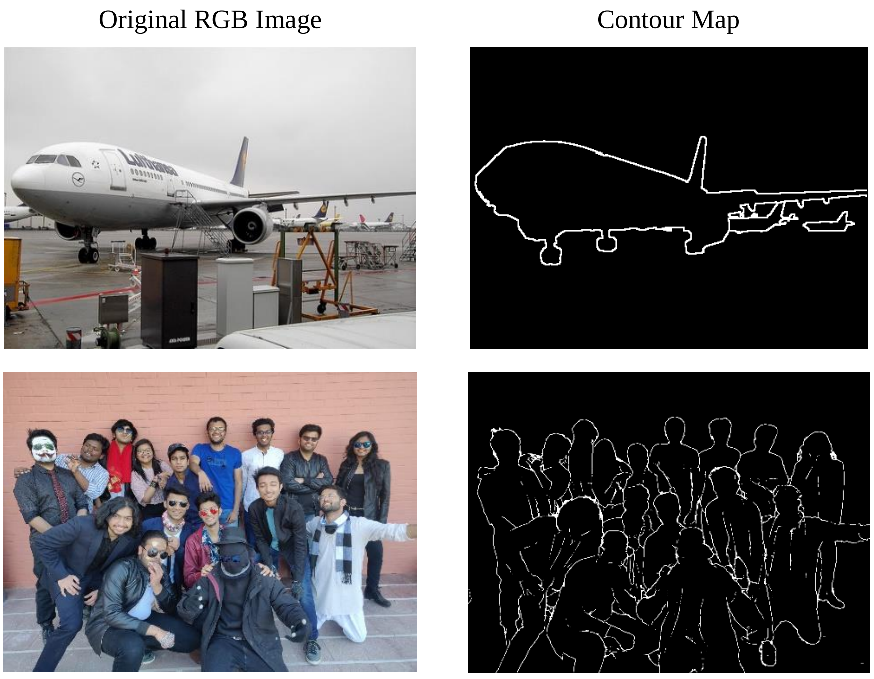


Figure 2.2.2-1: The results of CEDN model on different scenarios

Besides, in order to see how fast the model could make the prediction and run in real-time so that it could fit in the pipeline, I wrote a script to let the model be able to run on webcam in real-time with OpenCV and PyTorch, which is what the model based on, because the original source code is solely written for the model to run on a single image. The code of the real-time detection script is shown in Script 2.2.2-1 in the next page. The main idea behind the code is to use OpenCV to keep capturing or reading the image from the webcam and each read image is then passed to the model as input image. The model will run the detection on this image, namely, finding the probability of each pixel to be a contour, and after filtering the results with threshold value, we receive the final binary image, which will then be output on the screen shown like real-time video.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from net import get_model
import torch
import torchvision
import torch.nn as nn
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

if __name__ == "__main__":

    # Given the stored model path
    model_name = "best_model.pth"
    model = get_model(model_name)
    model = model.to(device)

    # Run in real time to show contour detection
    cap = cv2.VideoCapture(0)
    while 1:
        # capture the image with cv
        _, img_original = cap.read()

        # convert the color series back to RGB
        img = cv2.cvtColor(img_original, cv2.COLOR_BGR2RGB)

        # convert into a numpy array
        img = np.array(img)
        img = np.rollaxis(img, 2)

        # The input needs to be a pytorch tensor
        # The model expects a batch of images, so add an axis
        img = torch.tensor(img).unsqueeze(0).to(device).float()/255

        # input the image into the model
        with torch.no_grad():
            final_img = model(img)

        final_img = final_img.cpu().numpy()[0][0]

        # filter the result
        final_img[final_img >= .5] = 1
        final_img[final_img < .5] = 0
        # show the results in realtime and terminate it with 'q' key
        cv2.imshow('Edge Detection', final_img)
        if cv2.waitKey(1) == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
```

Script 2.2.2-1: Code for real-time detection based on CEDN model

2.3. Multiscale Combinatorial Grouping (MCG)

2.3.1. Theory

Multiscale Combinatorial Grouping or MCG is a unified approach to solve instance segmentation problem designed by Arbel´aez et al [3][4]. It is mainly composed of two parts, namely "multiscale hierarchical segmentation" and "object proposal generation". The main idea behind multiscale hierarchical segmentation is that it first rescale the input image into 3 different scales [0.5, 1.0, 2] and then these rescaled images would be passed through a complex algorithm based on the concept of spectral clustering (2.3.2) to generate a contour map, which resembles to the results of DexiNed as well as CEDN mentioned above. These contour maps will then be rescaled back to same size and recombined together into a multiscale hierarchical image or so called ultrametric contour map (UCM). In the second stage "object proposal generation", the UCM would be set as an input and a complex algorithm based on the idea of watershed 2.3.3 would then create a dendrogram also known as region tree to generate the segmentation proposal. The overall structures of MCG and dendrogram could be found in the images below:

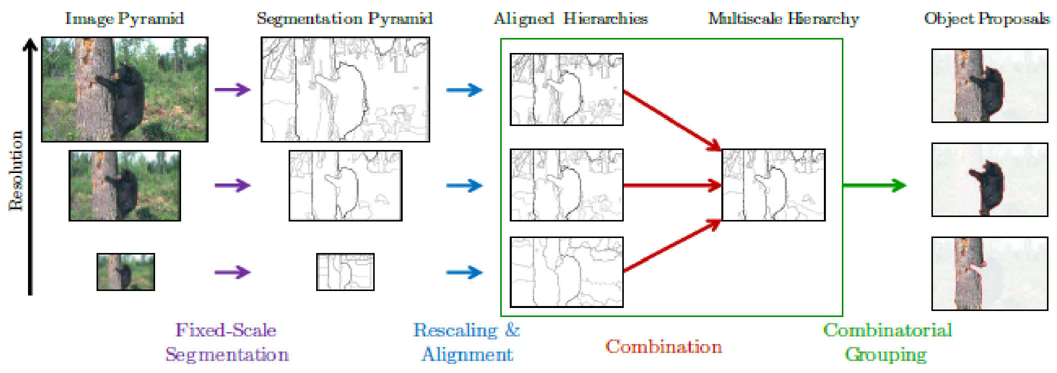


Figure 2.3.1-1: Overall Structure of Multiscale Combinatorial Grouping

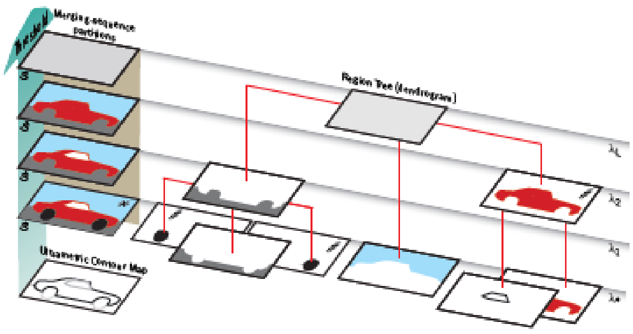
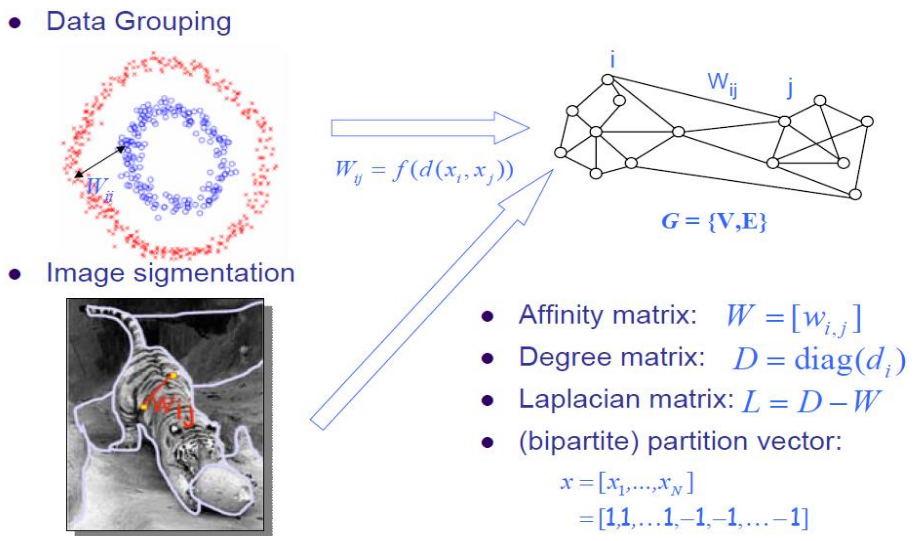


Figure 2.3.1-2: Duality between a UCM and a region tree: Schematic view of the dual representation of a segmentation hierarchy as a region dendrogram and as an ultrametric contour map.

The reason why the input image has to be rescaled is that large scaled images have the advantage of detecting details inside the image since the details or small objects would also be enlarged and small scaled images bring the benefit of focusing on the main objects as well as contour. Therefore, by combining all these images together, we could make sure that all the objects and details in the image could be equally represented in the final UCM.

2.3.2. Spectral Clustering

In multivariate statistics, spectral clustering techniques make use of the spectrum (eigenvalues) of the similarity matrix, which may be defined as a symmetric matrix A, of the data to perform dimensionality reduction before clustering in fewer dimensions. The similarity matrix is provided as an input and consists of a quantitative assessment of the relative similarity of each pair of points in the dataset. Not only could we apply spectral clustering on a graph, which is conspicuous, but also on an image to cluster the segmentations. The illustration of this complicated concepts of graph-based clustering is shown below:



• Data Grouping
W_{ij}
$W_{ij} = f(d(x_i, x_j))$
i
W_{ij}
j
$G = \{V,E\}$
• Image sigmentation
W_{ij}
• Affinity matrix: $W = [w_{i,j}]$
• Degree matrix: $D = \text{diag}(d_i)$
• Laplacian matrix: $L = D - W$
• (bipartite) partition vector:
$x = [x_1, \ldots, x_N]$
$= [1, 1, \ldots 1, -1, -1, \ldots -1]$

Figure 2.3.2-1: Illustration of Graph-based Clustering

Therefore, once we convert either a graph or an image into the structure like G, which means we calculate the Affinity matrix, Degree matrix, and the corresponding Laplacian matrix. We could use the technic of spectral clustering to generate partitions and separate each cluster in a graph as well as each segmentation in an image. A result of spectral clustering in comparison with “Kmeans” algorithm is shown in Figure 2.3.2-2. From this example we notice that with the help of spectral clustering we conquer the problem that Kmeans would have and successfully divide almost all data points in the correct cluster.

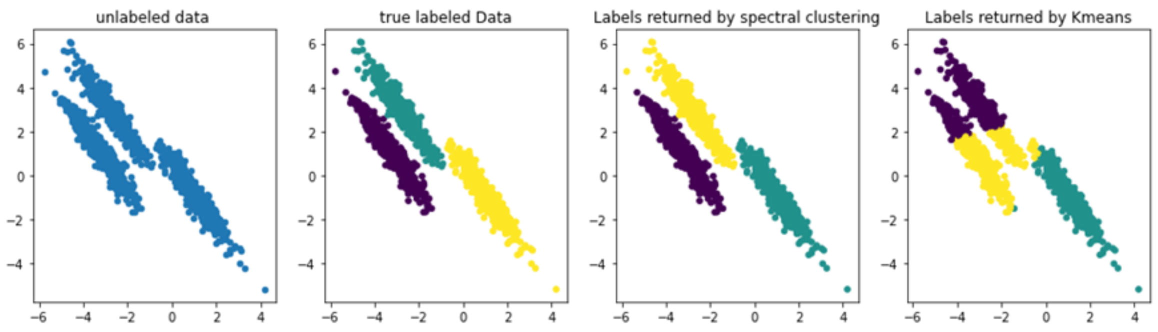


Figure 2.3.2-2: Result of spectral clustering in comparison with Kmeans on randomly generated data points.

The walkthrough process of spectral clustering algorithm is the following:

1. Calculate the adjacency matrix and diagonal matrix of the graph
2. Calculate the graph Laplacian & eigenvalues/vectors
3. Find the needed eigenvectors, there are two options
    - Find Fiedler value & vector, which accord to the second smallest eigenvalue and corresponding eigenvector
    - Calculate the first k eigenvectors (The eigenvectors corresponding to the k smallest eigenvalues of L)
4. Consider the matrix formed by the first k eigenvectors; the l-th row defines the features of graph node l
5. Cluster the graph nodes based on these features (e.g., using k-means clustering)

2.3.3. Watershed

In the study of image processing, a watershed is a classical algorithm used for segmentation defined on a grayscale image. The watershed transformation treats the image it operates upon like a topographic map, with the brightness of each point representing its height, and finds the lines that run along the tops of ridges. In graphs, watershed lines may be defined on the nodes, on the edges, or hybrid lines on both nodes and edges. To be more specific, watershed algorithm is based on extracting sure background and foreground and then using markers will make watershed run and detect the exact boundaries. This algorithm generally helps in detecting touching and overlapping objects in image, which is illustrated in Figure 2.3.3-1 below:

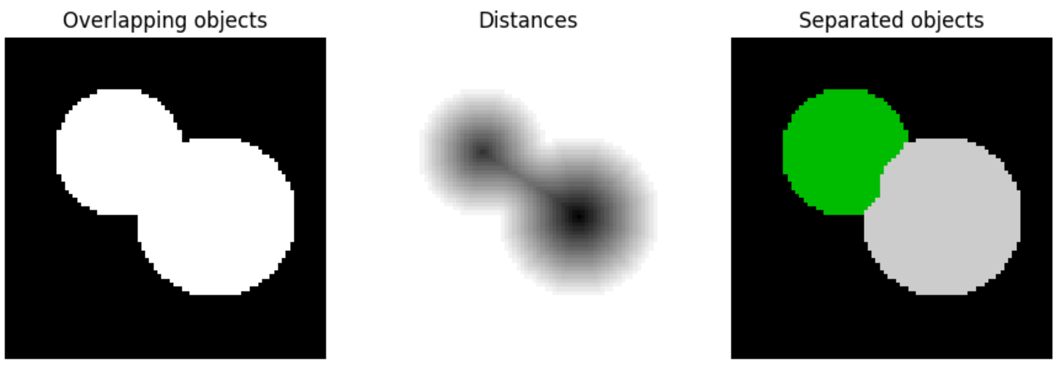


Figure 2.3.3-1: Illustration of the concept of Watershed

In the following Script 2.3.3-1 you could find the step-by-step process of code for Watershed segmentation and in Figure 2.3.3-2 you could find the outputs of each step and final result to understand how the image is processed thorough the algorithm.

```
# import all the needed libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from scipy import ndimage
from skimage import measure, color, io

# read the image and convert to gray image
img_orig = cv2.imread('cell_01.jpg')
img = cv2.cvtColor(img_orig, cv2.COLOR_BGR2GRAY)

# Filtering the gray image with threshold to generate a binary image
ret1, thresh = cv2.threshold(img, 0, 255,
cv2.THRESH_BINARY+cv2.THRESH_TRIANGLE)

# clean up the noises and holes in binary image
k = np.ones((10,10), np.uint8)
clean_img = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,k,iterations=1)

# identifying sure background area
sure_bg = cv2.dilate(clean_img, kernel, iterations=2)

# Finding sure foreground area using distance transform and threshold
dist_tran = cv2.distanceTransform(clean_img, cv2.DIST_L2, 3)
ret2, sure_fg = cv2.threshold(dist_tran, 0.1*dist_tran.max(), 255, 0)
sure_fg = np.uint8(sure_fg)

# get the unknown boundary region
unknown = cv2.subtract(sure_bg, sure_fg)

# Now we create a marker for the regions inside
ret3, labels = cv2.connectedComponents(sure_fg)
```

```
# to avoid conflict with background we ad 10 to each pixel label
# we want to label the unknown region (edge) as 0
labels = labels + 10
labels[unknown == 255] = 0

# Using watershed to get the final boundary and assign the label
watershed_label = cv2.watershed(img_orig, labels)

# mark the unknown boundary in original image in red
img_orig_bd = img_orig.copy()
img_orig_bd[watershed_label==-1] = [255,0,0]
# covert label image into color image
final_img = color.label2rgb(watershed_label, bg_label=0)
```

Script 2.3.3-1: Code for Watershed segmentation

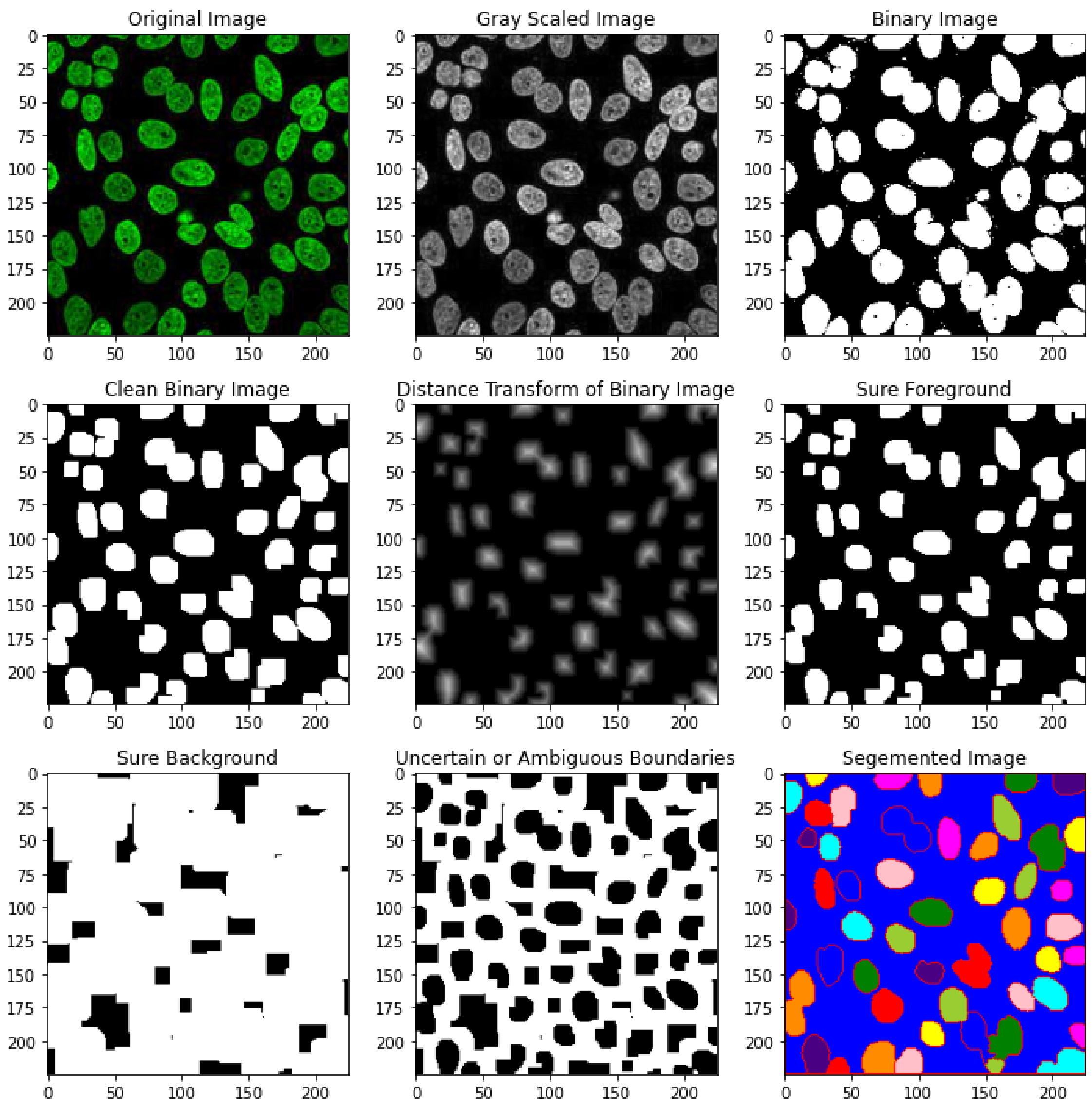


Figure 2.3.3-2: Outputs of each step starting from top left and the final result at bottom right

2.3.4. Implementation and Results

The implementation of MCG is based on MATLAB since the source code is written in MATLAB. Therefore, I have installed MATLAB in Linux environment as well as all the needed libraries. In the Figure 2.3.4-1 is the output of MCG algorithm at each step including the contour map as well as binary proposes and the final result with instance masks, which are created thorough the color labels according to the results of Watershed.

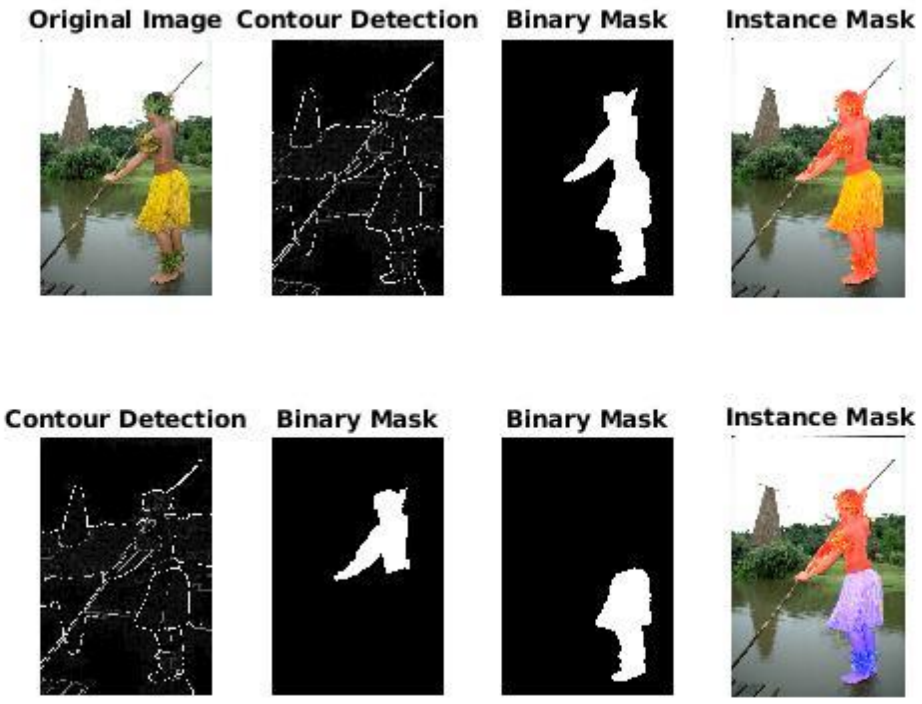


Figure 2.3.4-1: The result of MCG algorithm with the instance masks

2.4. Summary

The concept of project 1 is indeed an innovative solution to achieve a universal approach for instance segmentation for the bin picking task. However, during the implementation of MCG, I do face some difficulties.

The main problem with MCG is that the code is written in MATLAB and thus it is too slow to be able to run in the real-time. One image alone like in Figure 2.3.4-1 takes already 5 second to process the contour map and another 25 seconds for binary proposals. Therefore, in order to ameliorate the efficiency, we have to convert all the codes into either python or C++, which is genuinely a big task to finish as the scale of MCG algorithm is over 10,000 lines of code and some functional black box as well as libraries aren't available in other language.

Furthermore, MCG algorithm is in fact not an end-to-end algorithm. It stops after finishing the binary proposals, which could aggregate over 2000 depending on the original image size and quality. As you can see in the Figure 2.3.4-1, there are proposals about the whole person, the upper body of the person, and lower body of the person. There are even other proposals such as the skirt, the stick, the sculpture behind, and the bush, which are not shown at here. Therefore, it leads to two questions: What is the definition of our instance? And how could we find them out among all proposals? Should a person be deemed as a person or be separated into his body parts?

After several discussions and experiments with different approaches, we could not figure a way to filter and automatically output the best proposals. Besides, in our main pipeline of bin picking project each part of functionality should be an end-to-end solution otherwise the robotic arm can't operate automatically and need human intervene, which is against our goal. So, ultimately we cease to work on this method and turn to the second approach which is my second project.

3.Project 2 / Week 10 – 22

The goal of the project 2 resemble the goal of project 1 in figuring out a universal approach to realize the instance segmentation for the robotic bin picking task. However, this time we found another different method to solve this task, namely using “SD Mask R-CNN” model.

“SD Mask R-CNN” is a variant model of “Mask R-CNN”, especially developed for bin picking or robotic grasping tasks. Therefore, in order to successfully grasp the concepts of SD Mask R-CNN and implement the model, I have to dive into Mask R-CNN first, going through all the codes und understand how it works. Moreover, since the SD Mask R-CNN model works genuinely well on benchmark datasets, I start to retrain the model with our own simulation datasets which are already provided from my colleague, but with some adjustments so that it could be utilized in the SD Mask R-CNN model. Thus, I have create some scripts to automatically adjust the dataset.

After making sure that the benchmark model works well and the datasets is also well prepared in the right format, I have created and conducted two different experiments based on the difficulty of the objects. We name them experiment 1 and experiment 2. All the objects in experiment 1 are all included in Experiment 2, but experiment 2 contains other much more different shapes and forms of objects. For each experiment, I have created a big datasets which collects lots of different simulation objects (CAD models) from small datasets under 2 different conditions, that is, whether the container is cover with packaging or not. The Packaging variable at here acts like the noise for our model, which has to learn how to avoid it through the training process. In addition, I retain the model weights based on these two big dataset and then test the performance of newly trained model weights to see how it performs.

Last but not least, I have written a new detection script to enable the SD Mask R-CNN model to run on single image, a whole dataset, video, and webcam detection. The original code of the model only allows to run on a dataset. Furthermore, we wanted to create more simulation data in blender, especially for customized workpieces. So I have learned the Blender software and how the overall pipeline for automatic data generation works. After finishing with that and fixing some bugs, I generated over 1000 of simulation data images in blender, among of which include different scale of bin box and each box contain 0 to 100 objects.

3.1. Mask R-CNN

3.1.1. Theory

Mask R-CNN [8] is a CNN and state-of-the-art in terms of image segmentation. This variant of a DNN detects objects in an image and generates a high-quality segmentation mask for each instance, which is so called “Instance Segmentation”. Instance segmentation is the function of pixel-level recognition of object outlines. It’s one of the hardest possible vision tasks relative to equivalent computer vision tasks. Furthermore, Mask R-CNN extends Faster R-CNN [10] to solve instance segmentation tasks. It achieves this by adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition. In principle, Mask R-CNN is an intuitive extension of Faster R-CNN, but constructing the mask branch properly is critical for good results.

Finally, the overall architecture of Mask R-CNN could be separated into the following parts: “Backbone (ResNet 35 or ResNet 50 or ResNet 101)”, “Feature Pyramid Network (FPN)”, “Region Proposal Network (RPN)”, and “Detection Layer”, which includes class, bonding box, mask prediction. In addition, an illustration of the Mask R-CNN architecture is shown in Figure 3.1.1-1:

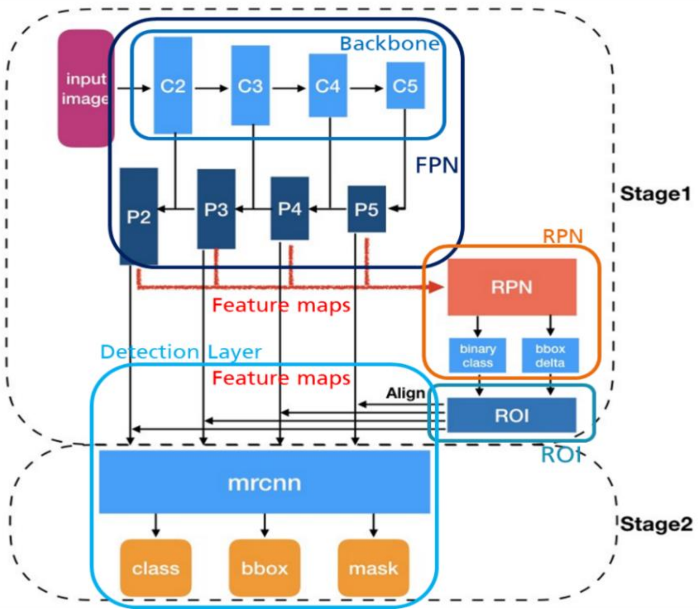


Figure 3.1.1-1: Overall architecture of Mask RCNN

During the training process, Mask R-CNN first extracts features of the input image through backbone and FPN. It then generates anchors and make binary as well as bonding box proposals in RPN by assuming the coordinate where the objects could possibly be. These proposals will then be filtered, namely picking the top k amount of proposals based on the higher accuracy scores and generate final proposals in ROI, which will then combine with the feature maps from each output of backbone branches. Finally, these new feature maps will run through detection layer to make class, bonding box, mask predictions.

3.1.1.1 Backbone and Feature Pyramid Network

The main idea of using backbone structure like ResNet 101 is to extract the features from the image through multiple CNN layers. However, instead of just a single backbone Mask RCNN prefers to use FPN to strengthen the feature extraction because there might have some feature losses during the downsampling in CNN layers. In Figure 3.1.1-2 is the architecture of FPN and in Figure 3.1.1-3 is the architecture of ResNet 101 shown. For each block in each identity block, it has a Figure 3.1.1-4 like CNN structure.

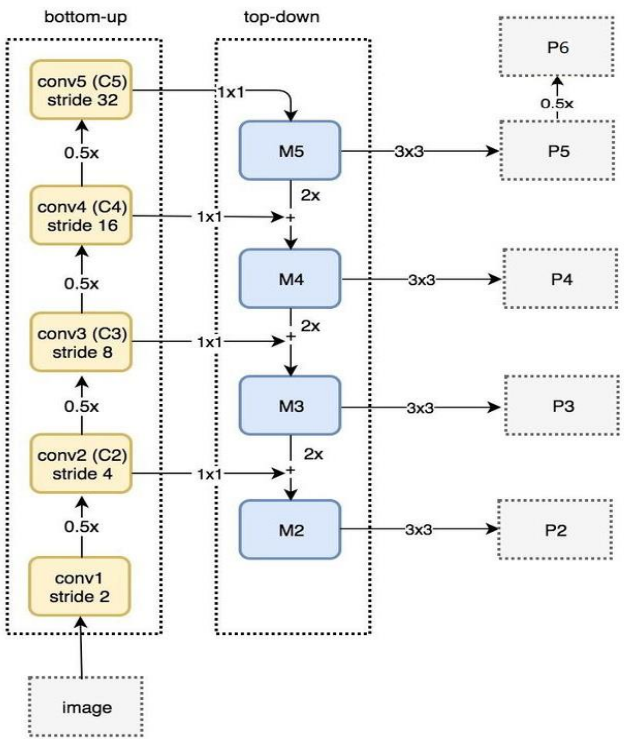
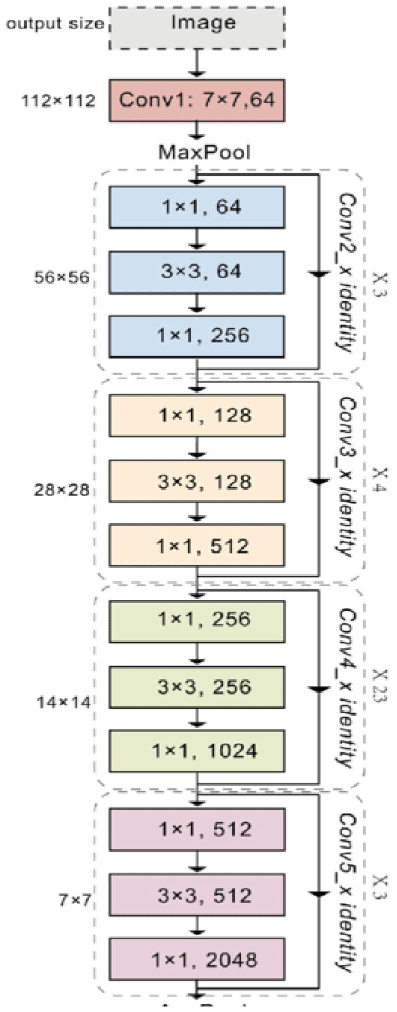


Figure 3.1.1-2: Overall architecture of FPN



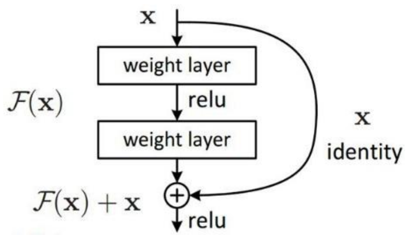

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
x
identity

Figure 3.1.1-4: Structure of each block in identity block

Figure 3.1.1-3: ResNet 101 Architecture without FC layer at the end

3.1.1.2 Region Proposal Network and Region of Interest

The RPN is a lightweight neural network that scans the input feature maps from previous FPN through a sliding-window and create anchors and finds areas that contain objects. Therefore, the RPN outputs two thing: the predictions of object coordinate (proposals) and the probabilities of existence of object (accuracy scores). After RPN, the ROI filters the generated proposals depending on the accuracy scores and picks the top k predicted coordinates. These predictions are then filtered by non-maximum suppression, NMS, to generate the final proposed coordinates (default 2000). The coordinate at here means the coordinate of the feature maps where object could exist.

More importantly, Faster R-CNN [10] was not designed for pixel-to-pixel alignment between network inputs and outputs. This is evident in how "RoIPool" performs coarse spatial quantization for feature extraction. To fix the misalignment, Mask R-CNN utilizes a simple, quantization-free layer, called "RoIAlign" that faithfully preserves exact spatial locations.

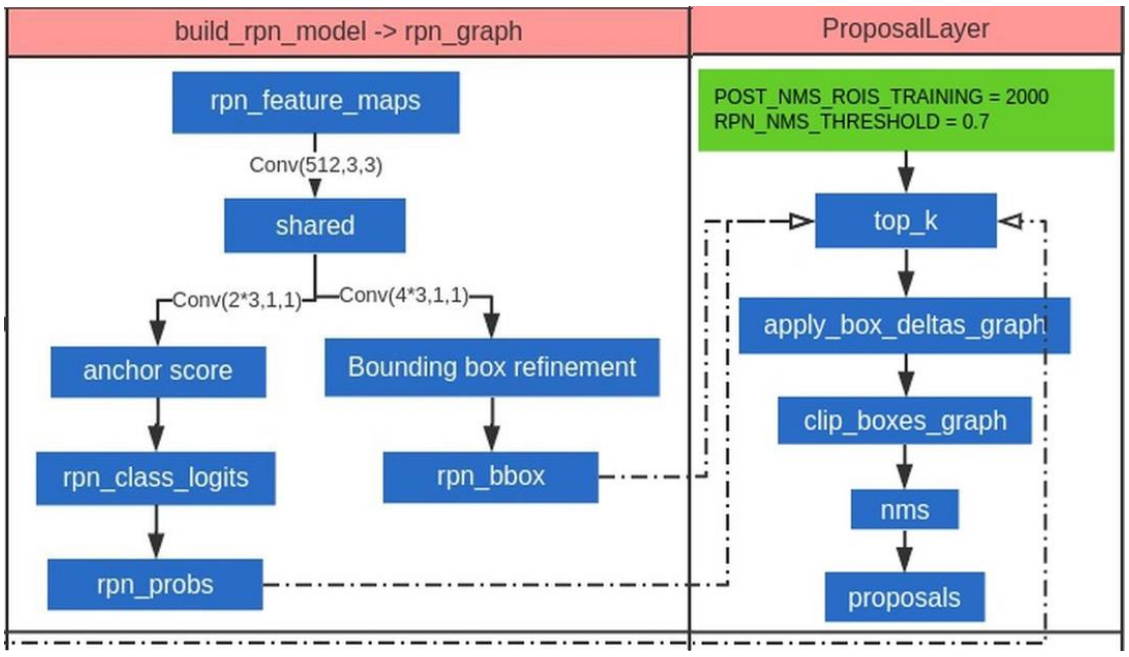


Figure 3.1.1-5: Overall architecture of RPN and ROI

3.1.1.3 Detection Layer

By combining the predicted coordinates from ROIs and feature maps from FPN, detection layer then runs through different convolutional layers, fully connected layer to generate the final class, bonding box and mask predictions of original input image. The architecture of the detection layer is shown in Figure 3.1.1-6.

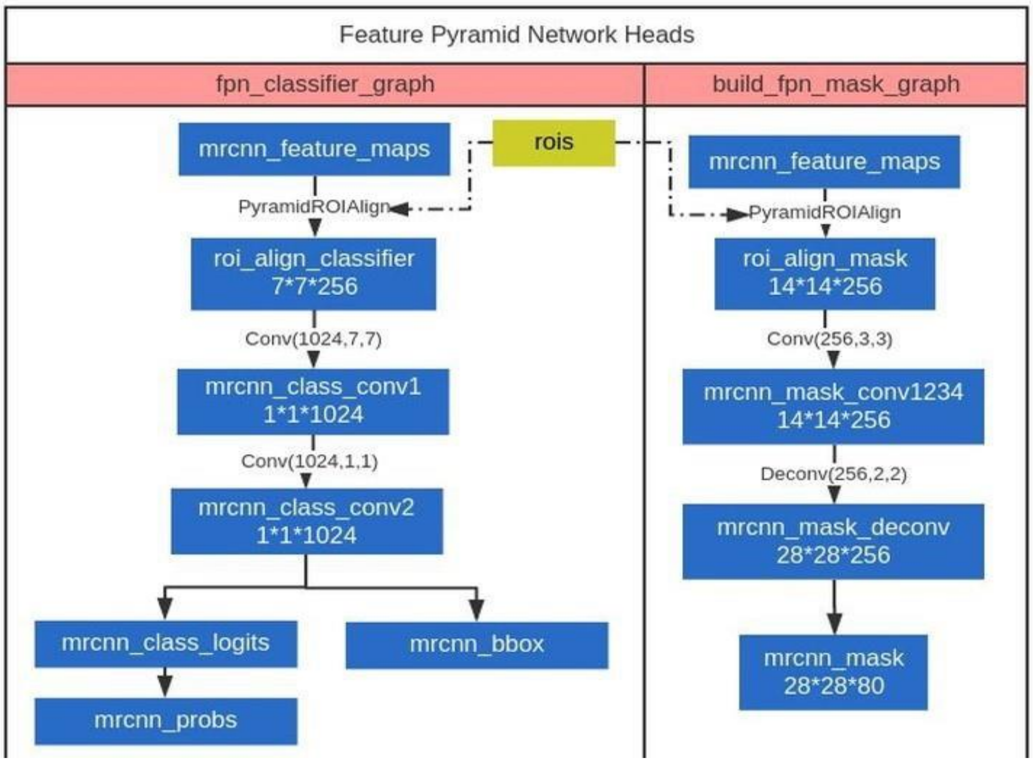


Figure 3.1.1-6: Overall architecture of detection layer

3.1.2. Benchmark Results

After successfully configuring the environment for Mask R-CNN and fixing all the bugs of converting Mask R-CNN model form TensorFlow version 1 to TensorFlow version 2 under Linux system, I tested the Mask R-CNN model on balloon dataset, which is one of the benchmark to run and make sure the model is working correctly. The results is shown below. We notice that the model has correctly detect the balloons and marks them with different colors.

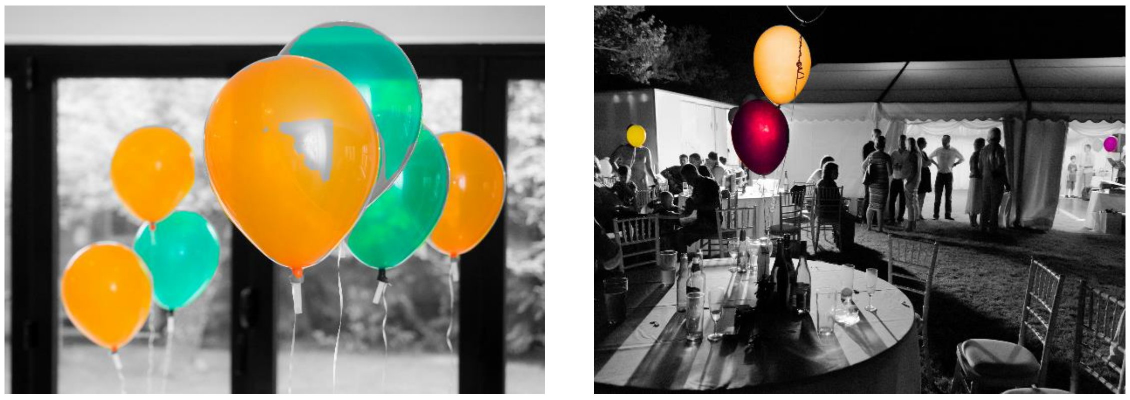


Figure 3.1.2-1: Benchmark Results of Mask R-CNN

3.2. SD Mask R-CNN

3.2.1. Theory

Basically, the model of SD Mask R-CNN is based on Mask R-CNN, but is particularly designed for bin picking or robotic grasping tasks. There are main two following changes:

- Instead of training on RBG image, SD Mask R-CNN trains on depth images and triplicate the depth values to treat them as grayscale images
- Output classes set to 2, which means background and object

The idea behind using the depth images is that not only it could help reducing the noises such as textures on the object surface and background or the ambiguous boundaries between object due to the similarity of colors but also provides our robotic arm extra information about the distance between each object and the sensor, which could have a huge benefit for decision making of picking priority and motion plaining. As a result, it could speed up the overall time of completion.

Furthermore, contrary to what I have mentioned above in project 1 about the problem of classification, the author of SD Mask R-CNN comes up with an innovative idea. Instead of classifying all the different objects around the world, we actually just need two classes, namely object and non-object (background), since our robotic arm does not need to know what kind of objects they are, but whether there are still objects out there to pick. Therefore, by reforming the query of bin picking task, we could in fact exploit classification model again. Moreover, due to the ability of instance segmentation of Mask R-CNN and output 2 classes feature, SD Mask R-CNN is capable to separate the foreground successfully from the background and mark each different object as well as instances in same object with different labels and colors.

Other contributions from SD Mask R-CNN:

- Provide a better way to set all the parameters for training and testing by setting a configuration file as input to the model
- Create a function to be able to run the inference on a whole testing dataset
- Provide augmentation, resize script for better training results
- Create a function to filter the BG in visualization if semantic masks are provided

3.2.2. Benchmark Results

Similar to the installation process and implementation of Mask R-CNN, we first downloaded the source code and configured the environment, but this time we also need to integrate the Mask R-CNN model into it. After that all finished, I tested the model with the benchmark dataset, which are real world images rather than simulation images. The results is shown below:

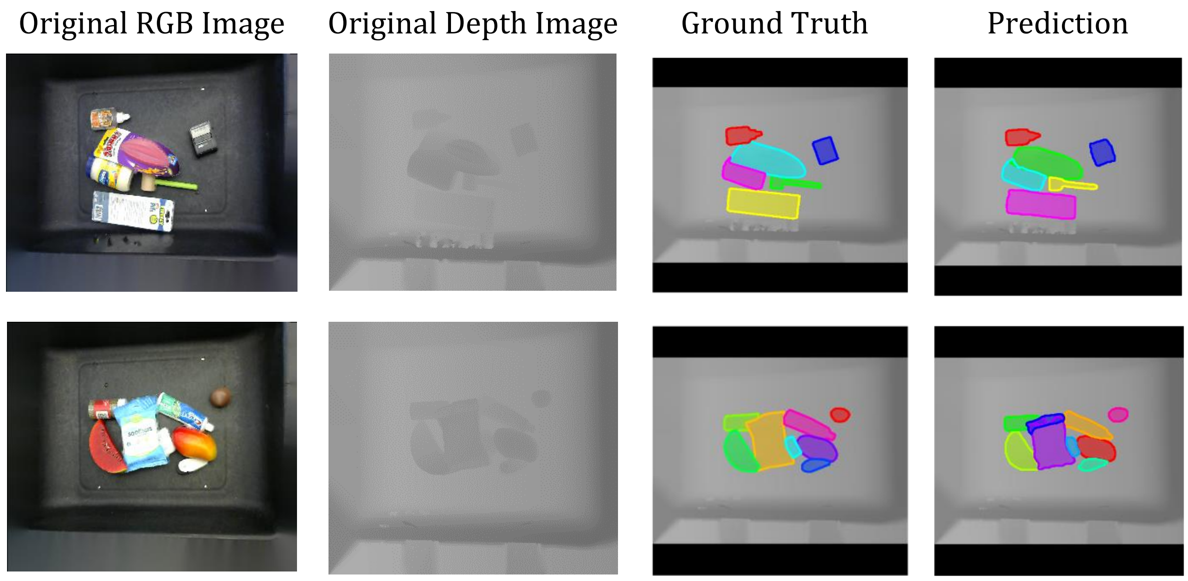


Figure 3.2.2-1: Benchmark Results of SD Mask R-CNN

We could conclude that the model has a good performance despite the overlapping of some objects. Now what I then have to do is to fit the model in our use case.

3.3. Datasets preparation

In order to retrain our model, the first thing to finish is to prepare the datasets. In Fraunhofer IPA there are already some simulation datasets of different objects based on CAD object models available. Thus, I took all these simulations datasets and adjust their format to match the demand of SD Mask R-CNN in training on dataset. According to the author of SD Mask R-CNN, all datasets are assumed to be in the following format: All ground truth instance segmentation masks must be single-layer .pngs with 0 corresponding to the background and 1, 2, ..., corresponding to a particular instance. Additionally, depth images and the ground truth instance segmentation masks must be the same size. We could use “resize.py” script in the pipeline to accomplish this. If we wish to filter out the Background to have a better visualization in results, the semantic masks should be provided and stored in “segmasks_filled” directory. These should be binary (0 if bin, 255 if object). Also each image file should be named in “image_{image index}” format since the model utilize these indices to run through the for-loop.

3.3.1. Big Dataset Generation from each object dataset

I have created a script, called “Data_genertation.py”. By running this script with given root directory, which includes all the object datasets, it will automatically iterate through all the directories and subdirectories and collect the RGB images, depth images, and instance segmentation masks, which will then be stored in the corresponding folders. During the process, the algorithm will also automatically rename all the images in to correct format and generate the indices files in numpy format for either training or testing utilization

3.3.2. Renaming All Images

As mentioned above, SD Mask R-CNN require all images named in correct format like this “image_{image index}”. Thus, I created a script to automatically rename all images by providing the root directory.

3.3.3. Semantic Mask Generation

Since there are only RGB images, depth images, and ground truth instance segmentation mask provided in the exited simulation datasets, I have written a script to automatically generate the semantic masks. The concept behind the script is to compare the labels in instance segmentation masks which are already available. Background, such as bin picking box, packaging, truth background, have their own certain labels in simulation datasets. Therefore, by matching the label of each pixel of an image to certain label values, we could get binary results, which corresponds to semantic mask.

3.4. Experiment 1

This is the first implementation of SD Mask R-CNN, which is an instance segmentation model based on Mask R-CNN. We name this implementation as Experiment 1. The idea behind instance segmentation is to let the model achieve higher pixel-level detection instead of just traditional object detection with bonding box. This will not only benefit us with higher accuracy and precision of detection of each instance/object but also enable in our task, Bin Picking, much further development such as center point detection, better decision of grasping point and better grasping angle.

3.4.1. Environment and Dataset

This Experiment 1 SD Mask R-CNN model is tested and trained against the following environment:

- GPU GeForce GTX 1080 Ti
- Ubuntu 18.04.5 LTS
- Nvidia CUDA 11.2 & cuDNN 8.1
- Python 3.7.10
- Tensorflow 2.5.0
- Keras 2.5.0

For this experiment 1, we exploited all depth images and instance segmentation masks of “Dexnet objects” and separate them into training and testing dataset. For training dataset we randomly pick images and generate the “train_indices.npy” file for train and “test_indices.npy” file for validation. The training dataset aggregate 6526 depth images, among of which 5520 for training and 1006 for validation, including the following objects: Dexnet Bar Clamp, Climbing Hold, Mount 2, Nozzle, Pipe Connector, U2, Part 1. The testing data aggregate 918 depth images with object, Dexnet Vase. Besides, each depth image has shape (480, 840, 4).

3.4.2. Training Process

Since we want our model to be capable to detect most of the objects and could be applied in general cases, which means it has to learn lots of features, we decide to use ResNet 101 for our model‘s backbone. Although it might slow down the detection speed, we gain higher accuracy. Moreover, in order to speed up the whole training process, we use “transfer learning” based on the COCO pre-trained weights. We also set learning rate at 0.001 and 4 for image per GPU, which is the maximum capability for our GPU. For each epoch, it takes around 1 hour and 15 minutes to finish. Finally, after 80 epochs, the final loss is around 0.2 and final validation loss is also around 0.2.

3.4.3. Instance Segmentation detection

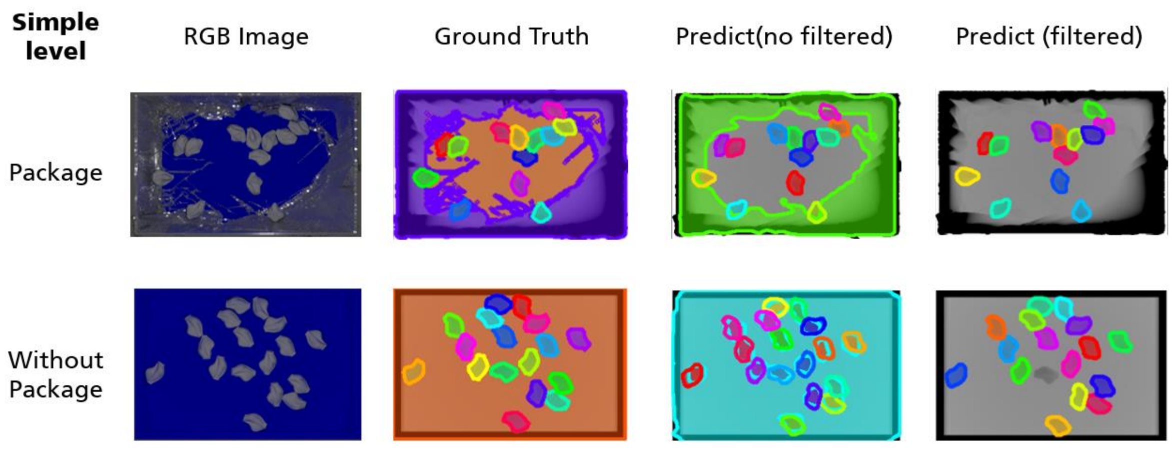
We test our retrained model on 3 different objects, namely Dexnet Vase, the original testing dataset, Dexnet U2 and Part PW1. Each object has a respectively meaning to test and evaluate our model, namely unseen object, seen but still challenging object, unseen as well as Challenging object. For each test object we do 4 different detection, that is with packaging, without packaging, filtered BG, unfiltered BG. Background (BG) at here means true background, package and box.

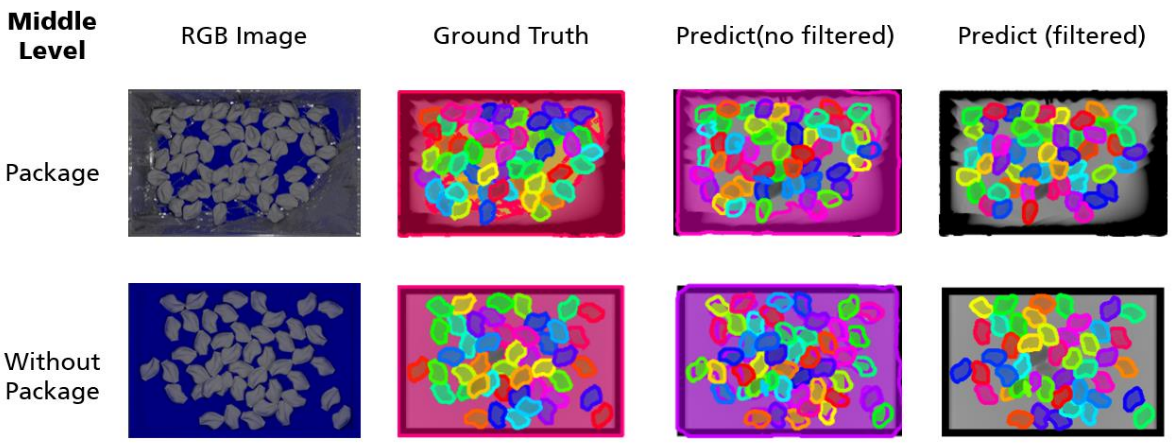
The different between filtering BG and unfiltering BG is whether the binary masks (semantic masks) are provided. Since our original dataset only include the instance segmentation masks, we have written a new script “SemanticMask_generation.py” to enable all semantic masks generation of a dataset. Once semantic masks are provided, our SD Mask R-CNN model could then filter out the BG and show better visualization of the results.

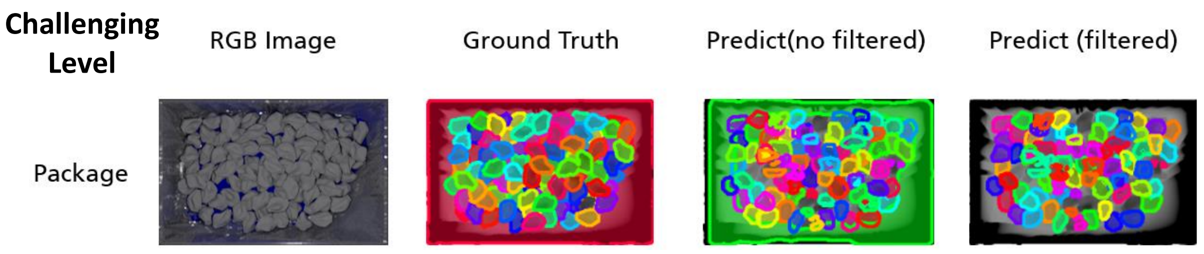
It takes the model around 140 milliseconds to detect the Instance segmentation mask per image and for visualization around 4 seconds

3.4.4. Results

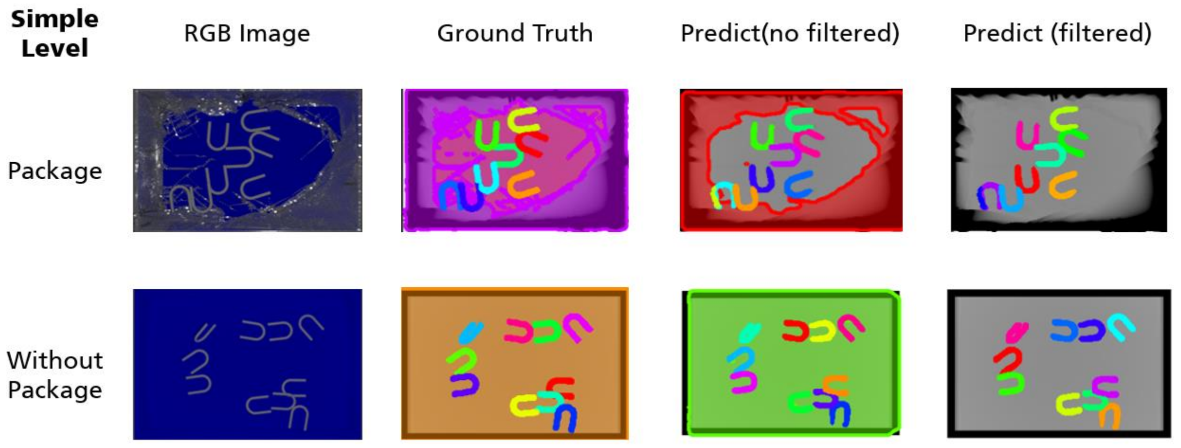
- Dexnet Vase (unseen object)

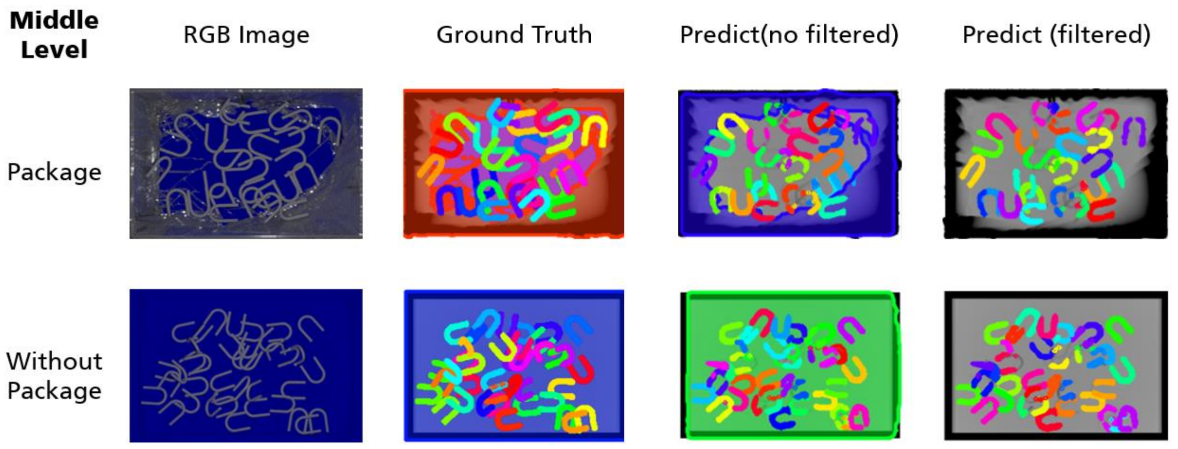


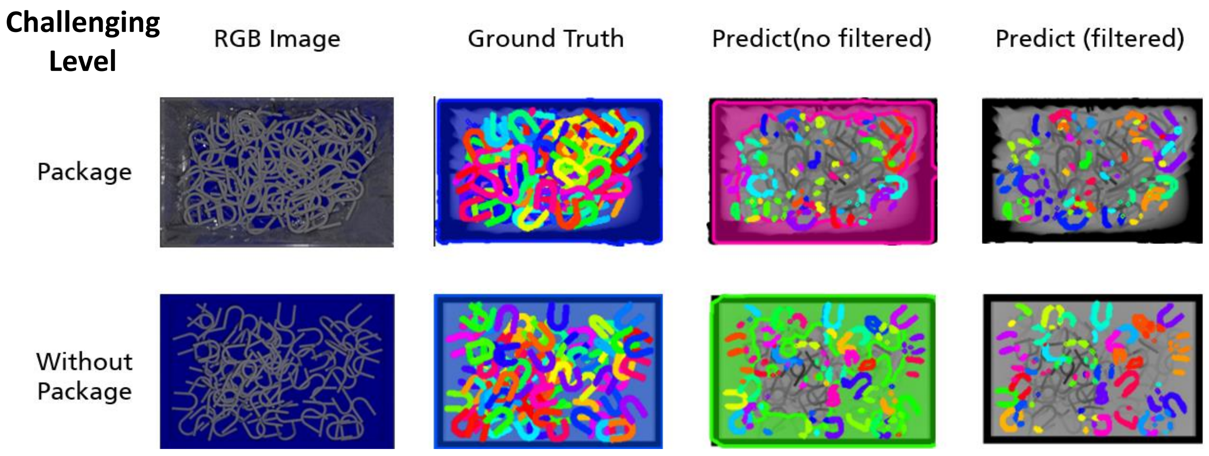




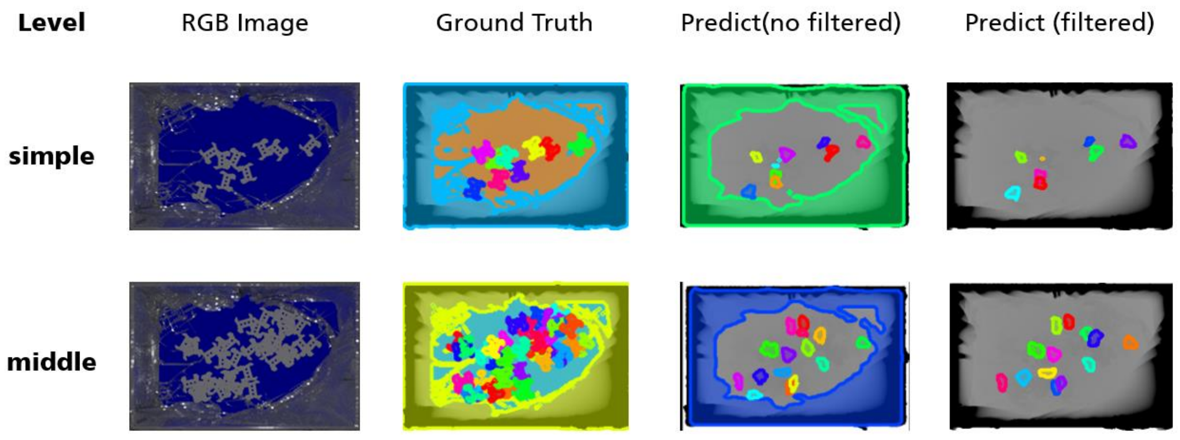
- Dexnet U2 (seen, but challenging object)







- Part PW1 (unseen and challenging object)



3.5. Experiment 2

This is the second implementation of SD Mask R-CNN. The overall process resembles the Experiment 1 but with much larger datasets include more various and difficult objects.

3.5.1. Environment and Dataset

Same hardware and software environment have been applied but with larger datasets with more objects, including also all objects in Experiment 1. The total amount of depth images in training datasets aggregates 11940, among of which 9552 for training and 2388 for validation, which include the following objects: Dexnet - Bar Clamp, Mount 2, Nozzle, Pipe Connector, U2, Vase, Part 1, Cardboard_Reversing_camera, Part PW1, 3, 4, 5, 7, 9, 11, 12. Moreover, 2495 depth images in testing datasets contained objects: Dexnet Climbing Hold, Part_PW2, 12b.

3.5.2. Training Process

Likewise in Experiment 1, all parameters setting stays same but this time I exploited transfer learning technic based on the retrained model weights from Experiment 1. For each epoch it takes around 2 hour and 30 minutes to finish. The final loss after 120 epochs is around 0.176 and validation loss 0.310.

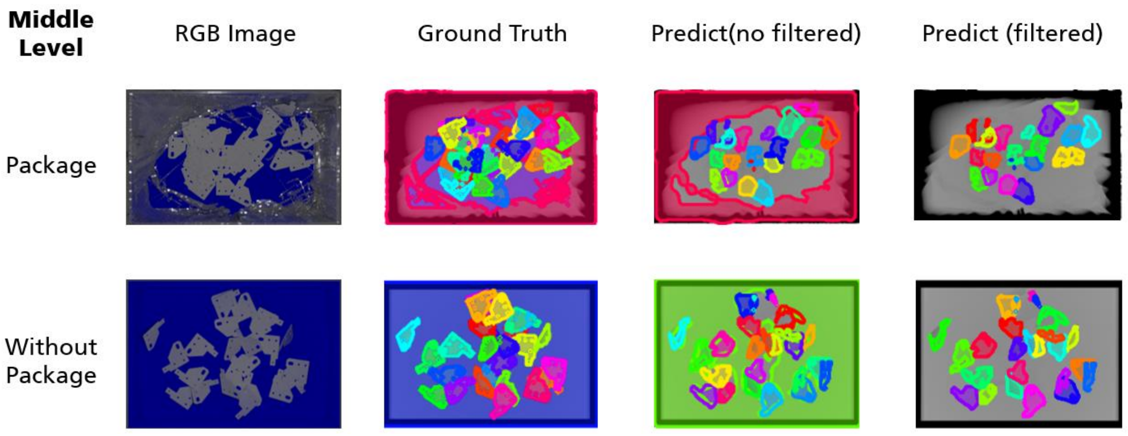
3.5.3. Instance Segmentation detection

I tested our retrained model on 3 different objects, namely Dexnet Climbing Hold, Part PW2, Part PW12b and for personal interest Dexnet U2, Part PW11.

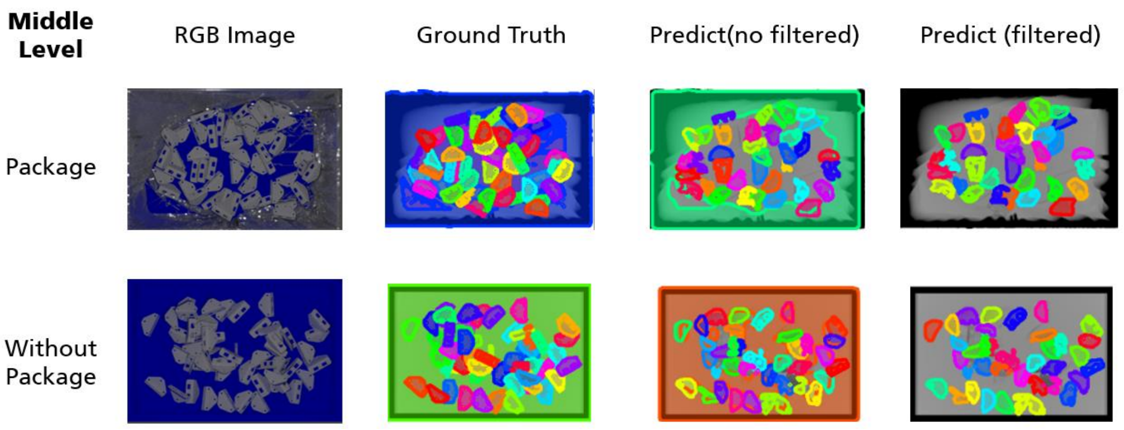
3.5.4. Results

Because there are too many results, only a few are shown here.

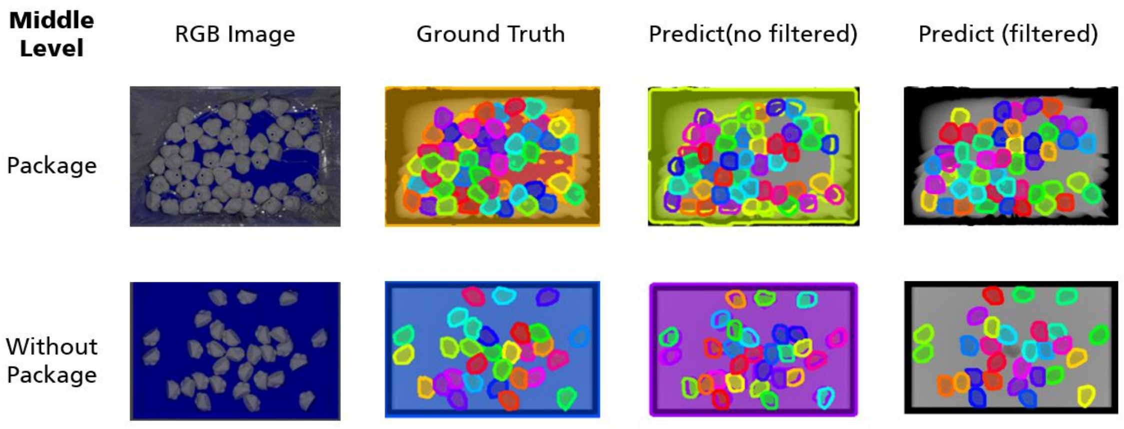
- Part PW 2, unseen object



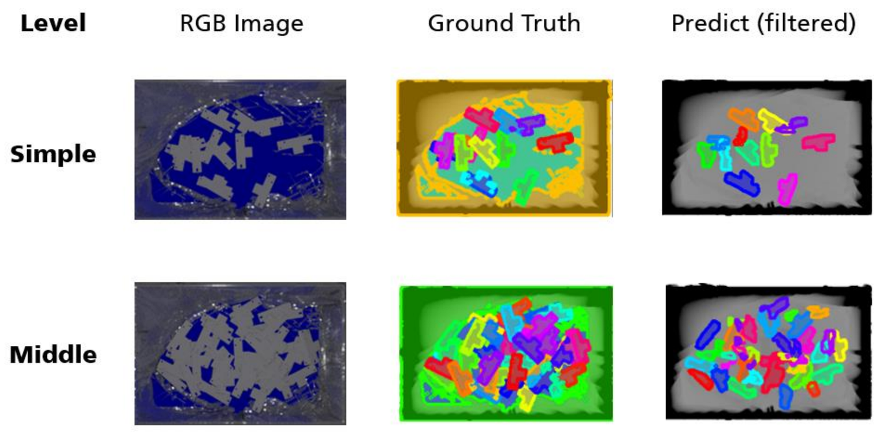
- Part PW 12b, unseen object



- Dexnet Climbing Hold, unseen object



- Part PW 12, seen but challenging object



3.6. Summary

As we could tell from the Results 3.5.4, SD Mask R-CNN model indeed works really well on implementation of instance segmentation, even with some really challenging shapes and scenarios. It also successfully learn to separate the package noise from object. However, we do also notice that when there are too many objects pile up in the Bin Box, the model does face some difficulties to clearly detection the instances, which is actually reasonable since this kind of scenarios are already pretty challenging for human perception not to mention for computer. We had tried to solve this problem by expending the training time, but the loss seems did not really improve nevertheless. Instead, it just kept oscillating around 0.17 and thus we stopped the training process.

Despite the overwhelming scenarios, the model could literally detect any kind of objects, which could be proven from the result of unseen objects, indicating that we do reach a universal solution for instance segmentation for Bin Picking.

3.7. Blender Data Generation

In order to train our model to have higher accuracy in customized workpieces, I need to generate more simulation data with customized workpieces by exploiting Blender for data generation. The scripts for simulation data generation included either with packaging or without packaging are already provided by my colleague. Thus, my first task is to understand the whole pipeline of simulation data generation.

After understanding the process, I started generating more datasets with customized workpieces from CAD models, especially without packaging since it is currently more relevant for our need. I have also scaled up and down the Bin Box size with different scale [0.5, 0.8, 0.9, 1.0, 1.1, 1.2] so that we could have more different scenarios and inside each box were 100 workpieces in total. In the end, I have generated over 5000 data images, including depth images, RGB images, and instance segmentation masks. Because there are too many data, only a few are shown here below:

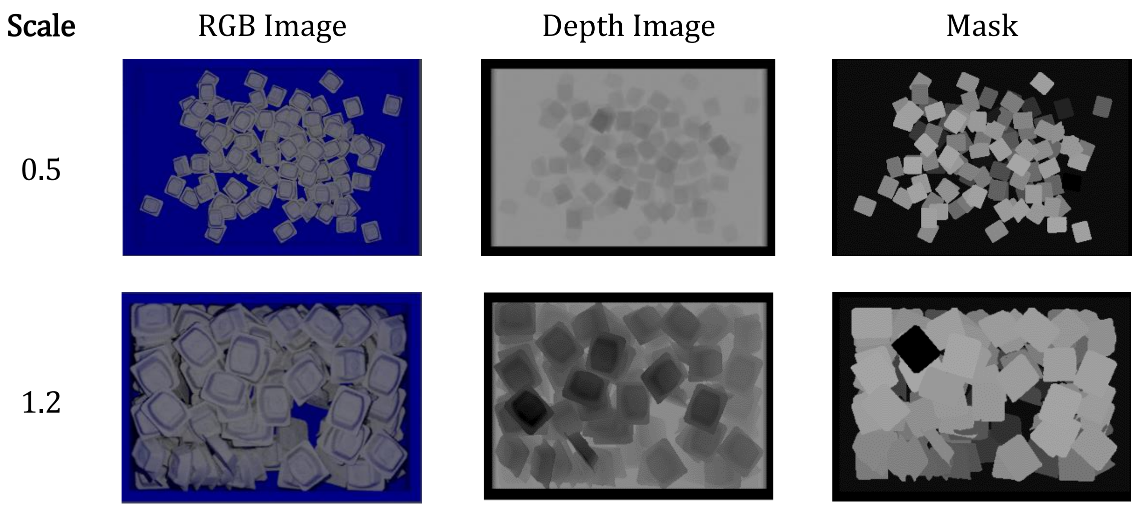


Figure 3.5.4-1: Simulation Data Generation Results of customized workpieces

In Figure 3.5.4-2 , you could see how well our model performs on unseen customized workpieces. In a word, we could draw the conclusion that the model has in fact genuinely good performance even on unknown objects.

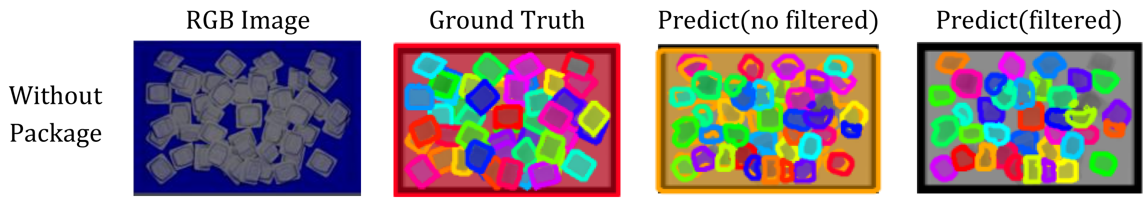


Figure 3.5.4-2: Instance Segmentation on unseen customized workpieces

A. List of Abbreviations

ML	Machine Learing
CNN	Convolutional Neural Networks
DNN	Deep Neural Networks
DexiNed	Dense Extreme Inception Network: Towards a Robust CN Model for Edge Detection
MCG	Multiscale Combinatorial Grouping
CEND	Object Contour Detection with Encoder-Decoder Network
FC	Fully Connected Layer
UCM	Ultrametric Contour Map
R-CNN	Region-Based Convolutional Neural Network
SD	Synthetic Data
ROI	Region of Interest
RoIPool	Region of Interest Pooling
RoIAlign	Region of Interest Align
FCN	Fully Convolutional Networks
FPN	Feature Pyramid Network
RPN	Region Proposal Network
NMS	Non-Maximum Suppression

References

[1]	J. Yang, B. Price, S. Cohen, H. Lee, and M.-H. Yang. Object contour detection with a fully convolutional encoder-decoder network. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 193–202, 2016.
[2]	X. Soria, E. Riba, A. D. Sappa. Dense Extreme Inception Network: Towards a Robust CNN Model for Edge Detection. IEEE Conference on Computer Vision and Pattern Recognition. arXiv:1909.01955
[3]	P. Arbel´aez, J. Pont-Tuset, J. Barron, F. Marques, and J. Malik. Multiscale combinatorial grouping. In CVPR, 2014.
[4]	P. Arbel´aez, J. Pont-Tuset, J. Barron, F. Marques, and J. Malik. Multiscale Combinatorial Grouping for Image Segmentation and Object Proposal Generation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. arXiv:1503.00848
[5]	K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. IEEE Conference on Computer Vision and Pattern Recognition. arXiv:1409.1556
[6]	S. Xie and Z. Tu. Holistically-nested edge detection. International Journal of Computer Vision, 125(1-3):3–18, 2017.
[7]	D. P. Kingma, J. Ba. Adam: A Method for Stochastic Optimization. Machine Learning. arXiv:1412.6980
[8]	K. He, G. Gkioxari, P. Doll´ar, and R. Girshick, “Mask R-CNN. arxiv preprint arxiv: 1703.06870,” 2017
[9]	M. Danielczuk, M. Matl, S. Gupta, A. Li, A. Lee, J. Mahler, and K. Goldberg, “Segmenting Unknown 3D Objects from Real Depth Images using Mask R-CNN Trained on Synthetic Data” arxiv: 1809.05825, 2019
[10]	S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In TPAMI, 2017.
[11]	R. Girshick. Fast R-CNN. In ICCV, 2015