Segmentation Bin Picking with Synthetic Data

Nick Allegretti[1,2], Kevin Xue[1,2]

[1]EECS, The University of Missouri
[2]Computer Graphics and Image Understanding Lab, The University of Missouri

Abstract

Segmentation bin picking is a critical task in computer vision, playing a vital role in various industrial applications, such as automation and robotics. The performance of convolutional neural networks (CNNs) on these tasks is highly dependent on the quality of training data. In this paper, we investigate the impact of synthetic data and stylized training images on the performance of a mask RCNN trained for segmentation bin picking, focusing on the accurate representation of LEGO objects.

Building on the findings of ImageNet-trained CNNs' bias towards texture, we aim to improve accuracy and robustness by increasing the shape bias in the training process. We create a novel dataset consisting of synthetic images that accurately depict the appearance of LEGO objects, as well as stylized versions of these images to emphasize shape information. We compare the performance of a mask RCNN trained on this dataset to that of a model trained on traditional datasets, demonstrating significant improvements in accuracy and generalization.

Our results reveal that incorporating synthetic data and stylized training images effectively reduces the texture bias in CNNs, enabling the model to better recognize and segment objects in challenging bin picking scenarios. Moreover, this approach enhances the model's robustness against varying lighting conditions and occlusions, demonstrating its potential for real-world applications in automation and robotics.

I. INTRODUCTION: MOTIVATION AND BACKGROUND

Segmentation bin picking is a fundamental challenge in computer vision and robotics, with a wide range of applications across numerous industries, including manufacturing, warehousing, and recycling. The task involves accurately identifying and segmenting individual objects within a cluttered bin, enabling robots to autonomously grasp and manipulate the items. Despite the advancements in deep learning and convolutional neural networks (CNNs), segmentation bin picking remains a difficult problem due to occlusions, varying lighting conditions, and the need for robust recognition of diverse object shapes and textures.

Recently, it has been shown that ImageNet-trained CNNs exhibit a bias towards texture, which can adversely impact their accuracy and robustness in real-world applications. To overcome this limitation, we propose the use of synthetic data and stylized training images to increase the shape bias in the model, ultimately improving its performance in segmentation bin picking tasks. Convolutional neural networks have achieved remarkable success in various computer vision tasks, including image classification, object detection, and semantic segmentation. However, these models are known to be sensitive to the quality and diversity of the training data. The ImageNet dataset, widely used for pre-training CNNs, has been found to induce a bias towards texture, which can lead to suboptimal performance in tasks that require recognition of object shape.

II. IMAGE PROCESSING MODULES AND PIPELINE

In this section, we present the image processing modules and pipeline that form the basis of our approach to segmentation bin picking with synthetic data and stylized training images. The pipeline consists of five main stages: Generation Instructions, Synthetic Image Construction, Model Training, and Segmentation and Testing.

Generation Instructions

The first stage of our pipeline involves creating detailed instructions that specify the objects used, their quantities, camera position, frames rendered, and other relevant parameters. These instructions guide the generation of synthetic images that accurately represent the desired scenarios for segmentation bin picking tasks. The selected objects are LEGO pieces, which are characterized by diverse shapes and high similarity, making them a suitable choice for evaluating the effectiveness of our approach.

Synthetic Image Construction

In the Synthetic Image Construction stage, we iterate through the generation instructions to create rigid

body simulations of LEGO objects in bins. Utilizing Blender, a powerful open-source 3D computer graphics software, we generate color, instance segmentation, and depth images for each simulated scene. This process allows us to create a diverse dataset of synthetic images that closely resemble real-world scenarios, while also providing precise ground truth labels for training and evaluation purposes.

Model Training

With the synthetic color and segmentation images, we proceed to train a Mask RCNN, a state-of-the-art model for object instance segmentation. The training process leverages both the accurate representation of LEGO objects and the stylized versions, which emphasize shape information over texture. This combination helps to reduce the texture bias commonly found in CNNs, resulting in improved accuracy and robustness for segmentation bin picking tasks.

Segmentation

After training the Mask RCNN, we evaluate its performance on both real and synthetic images of LEGO objects. The trained model uses the generated weights to perform instance segmentation, identifying and separating individual LEGO pieces within cluttered bins. It is important to note that there is no need to preprocess the synthetically generated color images, as the Blender rendering functionality enables us to denoise the images at render time, ensuring high-quality input data for model evaluation.

Testing

In this step of the pipeline, we describe the testing process to evaluate the pixel-wise accuracy of the masks generated by the Mask RCNN model in the Segmentation stage. The testing is performed on synthetic images of LEGO objects that are different from the ones used during training, ensuring a fair assessment of the model's generalization capabilities. Additionally, we test the model on real images of LEGO objects, with ground truth labels manually created by us.

Before testing, we preprocess the instance segmentation images generated in Step 2 (Synthetic Image Construction) into semantic segmentation images. This conversion enables us to compute pixel-wise accuracy by comparing the predicted masks with the ground truth semantic segmentation images.

III. APPROACH: ALGORITHMS AND MODULE IMPLEMENTATION DETAILS

In this section, we discuss the algorithms and module implementation details that form the foundation of our approach to segmentation bin picking with synthetic data and stylized training images. The implementation is divided into two main subsections: Blender Python for image generation and Pytorch with Mask RCNN for model training and testing.

Blender Python for Image Generation

The Blender Python section of our implementation focuses on generating synthetic images by reading information from a CSV file and creating the corresponding 3D scenes using Blender, an open-source 3D computer graphics software. The CSV file contains the generation instructions, specifying details including File Names, Spawn Positions, End Frame, Render Frames, Number of OBJs, Input DIR, Output DIR, Camera Location, Camera Rotations (Degrees, Start Frame, End Frame) and Material Type. The Blender Python code utilizes this information to generate color, instance segmentation, and depth images that closely resemble real-world scenarios.

The main functions in the Blender Python code include:

addObjects: This function takes a directory, the names of an OBJ file(s) in the directory, a tuple containing 2D bounds for the spawn locations of the objects and the amount of objects to simulates.

removeObjects: This function removes all objects from the project file, not just the scene. This is incredibly important for the automation of producing multiple batches of renders from CSV format because objects removed from the scene, but not the project file cause file bloat and slowdown the code.

createLegoMaterials/createLegoMaterials: These functions create the LEGO materials based on the Blender Principled BSDF Shader. the materials created by these functions use accurate IOR and RGB of LEGO bricks to accurately depict real LEGOs.

createRandomMaterials/createRandomMaterials: These functions create stylized materials also based on the Blender Principled BSDF Shader. These materials use multiple randomly generated maps including 'noise' and 'magic' textures which produce a random base color and differentiate the texture of the generated bricks. Additionally, random values of roughness are used to further distinguish the texture of the bricks from the realistic brick materials generated in the createLegoMaterials/createLegoMaterials functions.

createMetalMaterial: This function creates a metal material for the LEGO bricks. this again is done to create texture different than that of standard LEGOs to test on.

createCamera: This function either creates or ensures the presence of a Blender Camera as well as two 'empty' objects responsible for controlling the cameras orientation and oration.

keyframeCamera: This function animates the cameras rotation by keyframing the Camera Rotator to different rotations at desired frames in the timeline.

clearKeyframes: This function clears the keyframe information for the camera and is essential for receiving the desired orations when bulk rendering images using a CSV.

cacheSim: This function is responsible for generating a cache of the rigid body simulation. Because Blender does not simulate rigid bodies and other physics elements until the timeline is ran, we came up with a solution to iterate through each frame and evaluate the simulation manually.

renderImages: This function is responsible for rendering the images. Rendered images include an RGB image and a instance segmentation image.

renderFromCSVLego/bulkRenderLego: These functions are responsible for reading a CSV file and completing all of the functions listed above with the values found in each row of the CSV file.

These functions were written prior to this assignment, but were modified heavily during the assignment. The original functions they were based on are included in the code for comparison.

These functions work together to create a diverse dataset of synthetic images that both accurately represent real-world LEGO bin picking scenarios, as well as scenarios with stylized texture providing the foundation for training and testing our Mask RCNN model.

Pytorch with Mask RCNN for Model Training and Testing

Mask RCNN: Mask RCNN is a well-cited method known for its simple, but effective improvement made to Faster RCNN, published in 2017 []. It builds off of Faster R-CNN, a region-based convolutional neural network with a region proposal network comprising its first stage. Mask R-CNN adds an additional mask output for each region of interest, alongside class predictions and bounding box offsets.

Our implementation uses version 2 of the Mask R-CNN model included with torchvision library, which is pre-trained on the COCO dataset []. The backbone for this model is ResNet50, pre-trained on ImageNet, which is especially pertinent to the aforementioned previous work [] Primarily used in the implementation are the pytorch and opencv-python libraries for Python 3.

Training: Our implementation uses supervised learning, which is where training inputs are accompanied by a 'ground truth', the expected correct output of the model as determined by an annotator. The training dataset contains approximately 1,400 synthetic LEGO bin images using real LEGO style and another 1,400 using stylized texture. Each has a perfectly accurate masks produced alongside their actual renders. The synthetic data and masks are rendered out to a resolution of 1024x1024. With regards to memory limitations, we resize training data during loading to 512x512, and selected a batch size of 10 images per iteration. For each iteration, 10 images are randomly selected and loaded anew, to be evaluated by the model in its current state. The results of this evaluation are compared to the ground truths associated with those particular input images. We then use backpropagation and loss functions provided by pytorch to determine how the model should change its weights. This process repeats as many times as deemed necessary. In our case, we created a checkpoint every 100 iterations. Our final model trained on 10000 iterations took slightly over 5 hours to complete training.

Testing: Once the model is trained on a sufficient number of iterations, in our case 10000, the final checkpoint is used to evaluate input test data. Test data is generated the same as in testing, and at the time of the project consists of 200 images. Test data was again, natively sized at 1024x1024. We resize to match the training data input parameters of 512x512. Test data is fed into the model, and will return us a series of masks, one for each separate detected instance as determined by the model. We iterate through the masks, and mark all pixels with an appropriate confidence score above 0.8 for the current mask with a randomly generated RGB color.

Evaluation: For evaluation, the same process as testing occurs, but instead of randomly generated mask colors, all segmented pixels are masked to [255, 255, 255]. The ground-truth mask is then subject to a binary threshold, thresholding all non-background mask layers (intensity > 0) to the same [255, 255, 255]. The prediction and the ground truth are compared and a diff is taken. All values that differ between the two will be non-zero. Then, the percentage of pixels that matched (zero) is divided by the total number of pixels to obtain a similarity in percentage.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

In this section, we present the results of our experiments, demonstrating the impact of using both realistic and stylized images for training the Mask RCNN model on pixel-wise accuracy. We trained two different models using two sets of training data and evaluated their performance on a test set of synthetic images.

Model Trained with Realistic Texture

The first model was trained using 1400 images with realistic textures generated by our Blender Python implementation. We trained this model for 1000 iterations, after which it achieved a pixel-wise accuracy of 97.0696

Model Trained with Realistic and Stylized Images

The second model was trained on an extended dataset that included the initial 1400 images with realistic textures and an additional 1400 stylized images. The purpose of this experiment was to investigate the impact of incorporating stylized images on the segmentation performance of the model. We trained this model for the same number of iterations, 1000, and observed a slight improvement in pixel-wise accuracy. The model achieved an accuracy of 97.1127

Future Work

The foremost direction to expand this project will be on working with stylized and textured data. The main inspiration of the project was to experiment with the findings reported in the ImageNet shape-bias paper []. Further focus on creating training data with textural data to induce a greater bias towards shapes via expansion of the textures that can be generated would be the first step to accomplish in this direction. More advanced methods of random material generation could be implemented to further enhance the models performance.

Next, as this project currently lacks a good variety of performance metrics, or otherwise methodology to compare our results with other work, future work would seek to remedy that. This could include trying our method on objects that are contained in common benchmark datasets such as COCO. This way more direct comparisons could be tabulated to better inform how effective our method is. In addition, comparing results to more traditional image processing techniques such as generalized Hough transform [], a noise-tolerant template-matching method capable of handling occluded objects introduced in 1980, could also improve our understanding of where our method sits in a broader perspective.

Otherwise, some experiments on the segmentation accuracy of using different rendering engines within Blender, namely the Cycles engine vs. the EEVEE engine. Determining the capabilities and trade-offs between the two engines will give us a clearer picture of how best to optimize engine selection in the future, as a general improvement.

One of our goals when first designing the project was to adaptively generate synthetic data according to deficiencies detected in the instance segmentation results during training. This would effectively create a training process that would balance the training dataset to better cover object colors, rotations, camera angles, etc. that do not produce good results.

Finally, another concept that we could explore within the context of our project should we obtain reliability and stability with the aforementioned expanded capabilities would be to experiment with the concept of using knowledge distillation to create real-image segmentation networks that are trained with the assistance of our synthetically trained network. This idea would be akin to the work done in 2DPASS. by Xu Yan et al. [], which trains a 3D segmentation network in tandem with a 2D segmentation network, combining their features during training such that segmentation loss from the 2D network influence the weights of the 3D network.

1. 2.

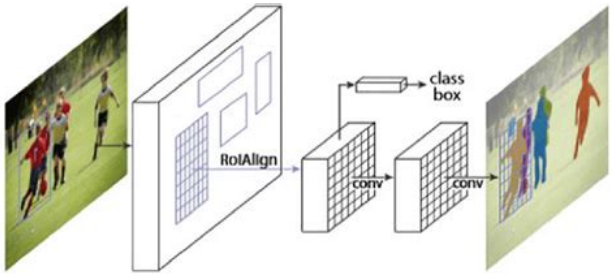

Figure 1: *Mask RCNN framework for instance segmentation.*

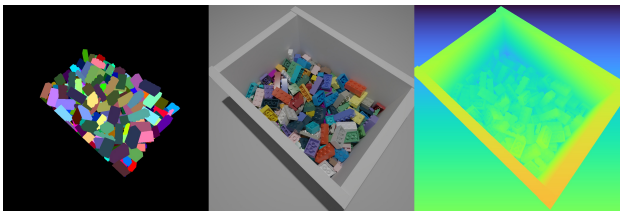

Figure 2: *Images generated by Blender code.*

REFERENCES

[1] D.h., Ballard. (1981). Generalizing the Hough transform to detect arbitrary shapes. Pattern Recognition, 13(2), 111–122. doi:10.1016/0031-3203(81)90009-1

[2] Feng, Y., Yang, B., Li, X., Fu, C.-W., Cao, R., Chen, K., ... Heng, P.-A. (2022). Towards Robust Part-aware Instance Segmentation for Industrial Bin Picking. ArXiv E-Prints, arXiv:2203.02767. doi:10.48550/arXiv.2203.02767

[3] Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., Brendel, W. (2018). ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. ArXiv E-Prints, arXiv:1811.12231. doi:10.48550/arXiv.1811.12231

[4] Hanh, L. D., Tu, H. B. (2020). Computer Vision for Industrial Robot in Planar Bin Picking Application. Advances in Science, Technology and Engineering Systems Journal, 5(6), 1244–1249. doi:10.25046/aj0506148

[5] He, K., Gkioxari, G., Dollár, P., Girshick, R. (2017). Mask R-CNN. ArXiv E-Prints, arXiv:1703.06870. doi:10.48550/arXiv.1703.06870

[6] Höfer, T., Shamsafar, F., Benbarka, N., Zell, A. (2021). Object detection and Autoencoder-based 6D pose estimation for highly cluttered Bin Picking. ArXiv E-Prints, arXiv:2106.08045. doi:10.48550/arXiv.2106.08045

[7] Li, Y., Xie, S., Chen, X., Dollar, P., He, K., Girshick, R. (2021). Benchmarking Detection Transfer Learning with Vision Transformers. ArXiv E-Prints, arXiv:2111.11429. doi:10.48550/arXiv.2111.11429

[8] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., ... Dollár, P. (2014). Microsoft COCO: Common Objects in Context. ArXiv E-Prints, arXiv:1405.0312. doi:10.48550/arXiv.1405.0312

[9] Ren, S., He, K., Girshick, R., Sun, J. (2015). Faster

R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. ArXiv E-Prints, arXiv:1506.01497. doi:10.48550/arXiv.1506.01497

[10] Wada, K., Murooka, M., Okada, K., Inaba, M. (2020). 3D Object Segmentation for Shelf Bin Picking by Humanoid with Deep Learning and Occupancy Voxel Grid Map. ArXiv E-Prints, arXiv:2001.05406. doi:10.48550/arXiv.2001.05406

[11] Yan, X., Gao, J., Zheng, C., Zheng, C., Zhang, R., Cui, S., Li, Z. (2022). 2DPASS: 2D Priors Assisted Semantic Segmentation on LiDAR Point Clouds. ArXiv E-Prints, arXiv:2207.04397. doi:10.48550/arXiv.2207.04397