Human Assisted Few-Shot Object Sorting

Gokul Swamy*
gokul.swamy@berkeley.edu

Andrew Li*
andrewyli@berkeley.edu

Abstract

For industrial as well as personal robotics applications, sorting objects is a problem with wide applicability. In this project, we focus on performing sorting where we do not know a priori what features a person is separating objects by and must discover this online based on a few examples, allowing us to complete the rest of the sorting task without need for assistance from the person. We perform this task from vision to make it intuitive for a person to specify what they want to be done, without them needing to be trained in the use of a controller or programming language. To simplify the implementation, we focus on binary sorting where the robot has to move the object to either side of the table based on the objects that are already there. On a hand-collected dataset of 135 RGB object images, which capture various poses of the 22 objects, we train a CNN-based autoencoder to reconstruct each instance and thereby obtain information-rich embeddings. This performs with perfect classification rate as compared to our 0.5 rate for the random embedding baseline. Our code is available at this link.

1. Introduction

Few-shot object classification, or the ability to categorize a set of objects based on a limited dataset, can improve human robot interaction in the workplace, where after a certain number of human actions the robot can automatically finish the job of sorting a collection of objects. To accomplish this task successfully, the robot must be able to infer what features a person was using to sort the objects. Because we use a vision-based sensing module, some more functional features might be hard to infer (for example, a person could put a match and a lighter in the same category and unless they were both producing fire, the robot might have difficulty understanding why they were grouped together). Therefore, we focus on examples where the sorting is done based on visual features like shape or color.

Figure 1: The setup we used for our experiments. The USB camera on the left is used to take a picture of the objects on the table, and one computer performs the template matching algorithm on it. On the right the Jaco Arm pushes objects with the "wrist" joint either left (into the picture) or right (towards the viewer).

1.1. Key Contributions

We propose a pipeline to accomplish the above task. Our pipeline can be divided into a few key components:

- A comprehensive pose dataset (n=135) of the 22 objects used in training.
- A convolutional autoencoder which produces an embedding suitable for categorization into arbitrary classes.

- A geometric method to map from image-space pixel coordinates of detected objects to robot coordinate frame 3D methods.
- A basic motion planner that gets behind objects and pushes them towards one of two pre-defined goals.

1.2. Related Work

Other recent approaches to classification in the context of computer vision have used convolutional networks to approach target datasets with few available examples [3][6] We apply similar methods but in the context of robotic sorting to produce a prototype that goes from camera image to physically sorted objects.

Our work is also connected to the problem of intent inference in human-robot interaction [2]. Here, we attempt to infer which of two classes a person wants an object to be in. As mentioned in our introduction, we assume the features a person uses to make this classification are purely visual and require no knowledge of the functionality of an object. We perform this inference examining compressed visual features of these objects. The compression is performed via a standard CNN-based autoencoder [1]. We discuss why we choose this approach in our methods section.

Our method for sorting objects based on embeddings was partially inspired by similar work in natural language processing [7].

2. Methods

2.1. Overview

We used a Logitech webcam to take photos through the library ffmpeg, which produced three channel color images with spatial dimensions of 720 by 540 pixels. Once we took in the scene image, we passed over the image with our poses (template matching) to find the positions and bounding boxes of all the objects on the table. From there, the program calculated the coordinates for the dynamics of the arm from the pixel positions of the object bounding boxes. To decide which direction to push the object, a snapshot of the object was fed through the autoencoder and the middle layer was compared against the embeddings of either side's objects and the new object was assigned to the side with the closer mean embedding. Finally, the coordinates and direction were sent to a machine connected to the robot, which upon receiving the file solved the motion planning problem for the robot to push the object to a side. This process was repeated with the set of 100 untrained random embeddings.

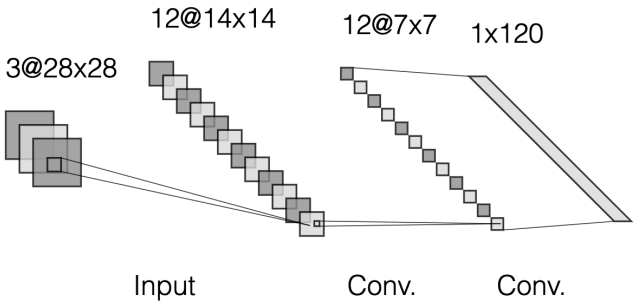

Figure 2: A rough sketch of the encoder architecture we used. The decoder architecture performed the same operations but in reverse.

2.2. Dataset Generation

We took 10 pictures of our 22 objects, cycling them in rows and columns around the table and rotating them 30 degrees per shot, as well as choosing different faces to balance the object on if it was a prism. This gave us 10 poses per image, which we filtered down to 135 images if two poses were too similar to both be of value. Our images gave us a wide variety of poses and sizes to work with, which vastly improved the object detector.

2.3. Object Detection

Because of our limited dataset, we used our poses in a simple image template matching algorithm. We took each pose image and convolved it over the input scene, producing a Euclidean inner product per location. The top left corner of an object, (i, j), was taken to be

$$\text{argmax}_{i,j} X_{[i:i+H,j:j+W]} \cdot P$$

for image X and pose image P.

2.4. Object Classification

The network we used to train the embeddings was a convolutional autoencoder with two convolutional layers with kernel size 3 and 12 output channels each. The input images were fed in batches of 2, and were standardized to 28 by 28 pixel dimensions. Following

these it had one fully connected layer that produced the embedding in the middle, and then one linear and two convolutional transpose layers to attempt to recreate the input image. Between corresponding encoder and decoder layers, we used additive skip connections to preserve spatial structure, which greatly improved reconstruction performance. The main loss function we used was the Euclidean L2 Loss over all pixels between the input and output images, defined as

$$L = \frac{1}{2}||X - g(f(X))||_2^2$$

We trained for 200 epochs with an initial learning rate of 0.002, using the AdamOptimizer method. We tried using KL divergence and a variational autoencoder instead of a simple autoencoder, but they did not improve reconstruction performance [5].

Ultimately, the layer we were interested in is the middle embedding layer, which contained 120 features.

We took the objects moved by the human to the left and those to the right, and computed the average embedding for each side. This allows the features common to each side to remain the same value. For each object not already on a side, we computed its embedding and assigned it to either the left or the right based on which embedding vector was closer to it in the embedding space by L2 distance (equivalent to cosine distance because vectors were normalized).

For the case of random embeddings, we took 100 randomly initialized copies of the encoder-decoder network we designed and picked the embedding that maximized the L2 distance between the left and right average embeddings. This allowed us to quickly generate a reasonable baseline that would have enough difference between its two classes to allow for more robust classification.

2.5. Converting to Robot Work Space

We used several similar triangles to convert between the image space of the camera and the 3D space passed to the robot. We assumed a pinhole camera model which was accurate enough because our camera was very close to the objects and the table was flat. We measured several calibration parameters:

- The height of the camera to the table h.

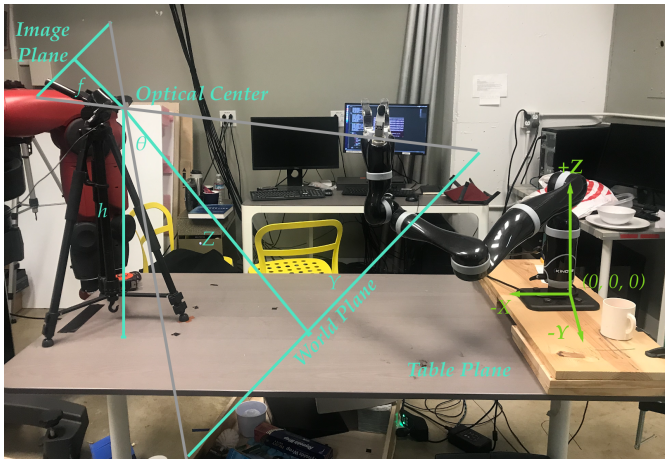

Figure 3: A visual of some of the parameters we measured for calibration along with some of the transformations used. At a high level, we find the point along a world plane that intersects the table plane corresponding to the detected object center on the image plane and then shift the coordinate center to the robot's base. We refer the reader to our implementation for the actual values of various parameters and equations used to perform the transformations.

- The angle between the optical axis of the camera and the table θ.

- The position of objects of known size in image space and the distance to them from the camera.

First, we estimated the focal length of our camera. To do this, we placed a piece of letter paper at the base of the camera and used the coordinates in image space of two corners of the paper to solve the pinhole camera equation:

$$X = \frac{fx}{Z}$$

where Z is the distance along the optical axis, X is the known width of the piece of paper, and x is the difference from image center of the pixel coordinate.

Then, we performed a series of transformations to get from the camera's point of view to the coordinate frame of the robot. Our code implementation has the details of the math used but at a high level, we first find the 3D position in the camera's point of view that corresponds to the 2D position by using the above equation with the constraint that the object lie on the table (which is h away from the camera), rotate the frame

by θ degrees to get one aligned with the table, and the shift the frame such that it is centered at the base of the robot.

2.6. Motion Planning

We used the MoveIt package (built on top of ROS [8]) to move from a position behind the object towards one of two pre-defined goal positions [10]. We initially tried using the fingers of the robot to push the objects but because of how small they were, they would often slip through and we were worried about damaging the delicate fingers of the robot. Therefore, we switched to a prehensile rather than a dextrous manipulation approach where the robot would execute a sweeping motion from behind an object and towards a goal position for the end effector. To have the robot execute a sweeping motion, we fixed the z coordinate and told the robot to minimize the distance travelled while moving from start to finish, the solution of which is usually a pushing motion. This was implemented by first converting from the positions to poses in the robot's configuration space by using an inverse kinematics module provided as part of MoveIt. Because the robot has seven degrees of freedoms, there are many points in configuration space that correspond to the same position for the end effector in work space. Therefore, we reset the robot between pushes and used the first solution found by the inverse kinematics module, which corresponded to the configuration space pose closest in norm to the reset position. This worked well unless the object was beyond the reach of the joint we were pushing with (in terms of radius from the robot's base), in which case the arm would contort itself and try to go under the table to get extra reach (see results). Given more time, we would include a collision constraint with the table and potentially adaptively choose which point to push by (including potentially the fingers) by segmenting by radius.

To perform the motion planning for this project, we used the Open Motion Planning Library (OMPL) [11] which defaults to a bi-directional tree-based planner, similar to the RRT approaches we covered in class. We tried other avaiable options including PRM based planners and optimization-based planners like TrajOpt [9] but did not find any significant difference in performance, likely due to the lack of collision constraints and simple cost function.

Figure 4: One of ten images used to collect poses from each object. To make the detector more robust to changes in size, lighting, and angle, for each picture every object is shuffled around systematically and rotated at random.

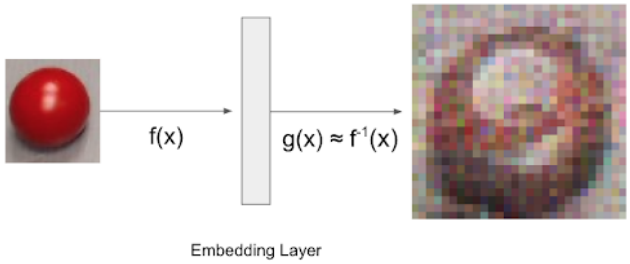

Figure 5: Embedding network output.

3. Results

The full video results can be found at https://www.youtube.com/watch?v=oxmoNlZqDwM.

We present a series of illustrations that demonstrate the physical motions of the robot during a trial, in Figures 4 through 7.

If we include the failure to reach the yellow hexagon, we achieve 0.75 sorting accuracy. Programmatically, our trained embedding still sorts it into the correct (yellow) category and we have a classification accuracy of 1. Our random embedding (as shown in the video) is able to move the objects but fails to classify 2 out of 4 of them for an accuracy of 0.5.

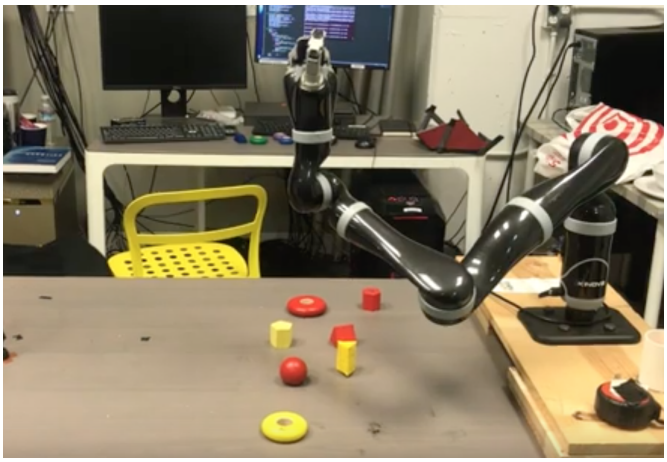

Figure 6: The starting position after the human moves the red ring and red hexagonal prism to the far side and the yellow ring to the close side is as follows:

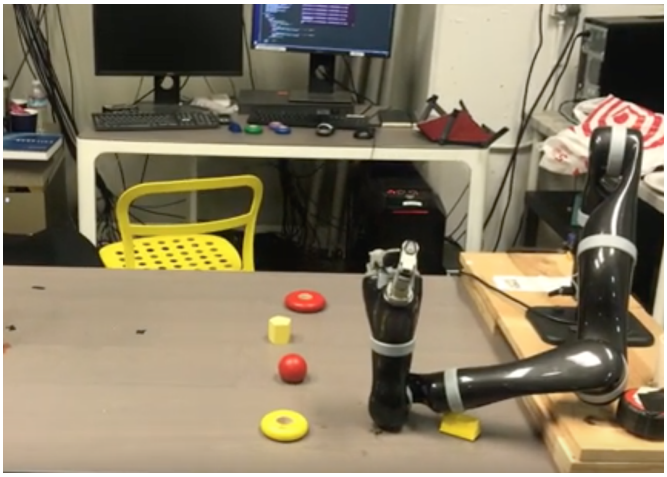

Figure 7: After all steps have been taken in the pipeline (see Methods, Overview), the robot chooses to move the yellow triangle and successfully moves it towards the side with the yellow ring already there.

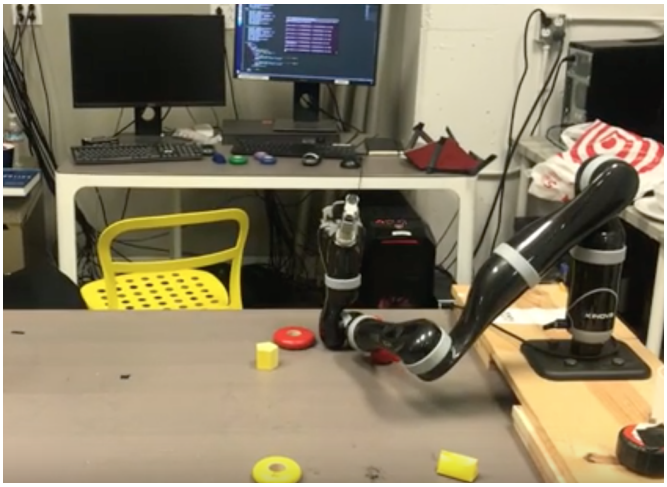

Figure 8: Next, it moves the red ball towards the red ring and red hexagon.

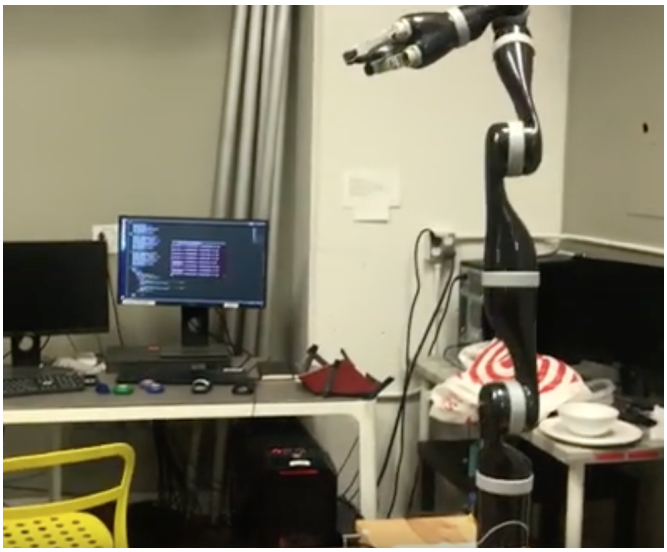

Figure 9: However, due to solving the motion planning problem incorrectly, the robot fails to reach the yellow hexagon.

4. Conclusions and Future Work

Our results seem to indicate that simply generating many random embeddings and using the best fit to sort objects does not work as well as a trained encoder or requires generating an extremely large number of embeddings. These trained embeddings also appear to be more robust to changes in lighting, as demonstrated by the fact that the robot moved an object it had already sorted correctly to the incorrect pile after a change in view when using random embeddings (see video).

As noted in the introduction of our paper, we focused on sorting objects purely based on their visual characteristics. In future work, we would like to extend this to broader sources of information about an object. One way to do this would be to use a standard object detection pipeline to classify an object, look up a definition or related words, take the average across words of the word2vec embedding of the associated text, and concatenate this with the visual features to generate a more comprehensive embedding. In this

work, we did not consider the intricacies of grasping, dexterous manipulation, and deformable object modeling because they were orthogonal to the core goal of our project. For this to be truly general purpose, a more advanced pipeline for performing these tasks should be used, like that of [4]. We also assumed a fixed set of objects so a simple template matching approach was sufficient for detection. In parallel to the preceding, we would switch to a CNN-based object detector to help remove this restriction. We would also then need to provide a larger set of images as input to our autoencoder, which we could collect manually or source from an online dataset of images.

In conclusion, we believe our approach, when scaled up to work with more complex situations, presents a feasible path forward to removing some of the burden required for doing sorting tasks by enabling a robot to learn from a few examples how to complete the rest of the task.

5. Acknowledgements

We would like to thank the InterACT Lab under Professor Anca Dragan for both use of the robot and the space, and Andreea Bobu, Ellis Ratner, Andrea Bajcsy, and Rohin Shah for help with configuring the motion planning library.

References

[1] Pierre Baldi. "Autoencoders, unsupervised learning, and deep architectures". In: *Proceedings of ICML workshop on unsupervised and transfer learning*. 2012, pp. 37–49.

[2] Tirthankar Bandyopadhyay et al. "Intention-aware motion planning". In: *Algorithmic foundations of robotics X*. Springer, 2013, pp. 475–491.

[3] Wei-Yu Chen et al. "A CLOSER LOOK AT FEW-SHOT CLASSIFICATION". In: *ICLR* (2019).

[4] Michael Danielczuk et al. "Mechanical Search: Multi-Step Retrieval of a Target Object Occluded by Clutter". In: *ICRA* (2019). URL: https://arxiv.org/abs/1903.01588.

[5] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *Foundations and TrendsR in Machine Learning* (2019).

[6] Yann Lifchitz et al. "Dense classification and implanting for few-shot learning". In: *IEEE* (2019).

[7] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[8] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In:

[9] John Schulman et al. "Motion planning with sequential convex optimization and convex collision checking". In: *The International Journal of Robotics Research* 33.9 (2014), pp. 1251–1270.

[10] Ioan A. Sucan and Sachin Chitta. "MoveIt". In: *Online* (). URL: http://moveit.ros.org.

[11] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. "The open motion planning library". In: *IEEE Robotics & Automation Magazine* 19.4 (2012), pp. 72–82.