The CoSTAR Block Stacking Dataset: Learning with Workspace Constraints

Andrew Hundt\textsuperscript{1}, Varun Jain\textsuperscript{1}, Chia-Hung Lin\textsuperscript{1}, Chris Paxton\textsuperscript{2}, Gregory D. Hager\textsuperscript{1}

\textbf{Abstract}—A robot can now grasp an object more effectively than ever before, but once it has the object what happens next? We show that a mild relaxation of the task and workspace constraints implicit in existing object grasping datasets can cause neural network based grasping algorithms to fail on even a simple block stacking task when executed under more realistic circumstances.

To address this, we introduce the JHU CoSTAR Block Stacking Dataset (BSD), where a robot interacts with 5.1 cm colored blocks to complete an order-fulfillment style block stacking task. It contains dynamic scenes and real time-series data in a less constrained environment than comparable datasets. There are nearly 12,000 stacking attempts and over 2 million frames of real data. We discuss the ways in which this dataset provides a valuable resource for a broad range of other topics of investigation.

We find that hand-designed neural networks that work on prior datasets do not generalize to this task. Thus, to establish a baseline for this dataset, we demonstrate an automated search of neural network based models using a novel multiple-input HyperTree MetaModel, and find a final model which makes reasonable 3D pose predictions for grasping and stacking on our dataset.

The CoSTAR BSD, code, and instructions are available at \url{sites.google.com/site/costardataset}.

\section{I. INTRODUCTION}

Existing task and motion planning algorithms are more than robust enough for a wide variety of impressive tasks, and the community is looking into environments that are even closer to truly unstructured scenes. In this context, the recent success of Deep Learning (DL) on challenging computer vision tasks has spurred efforts to develop DL systems that can be applied to perception-based robotics \cite{1, 2}. DL promises end-to-end training from representative data, to solve complex, perception-based robotics tasks in realistic environments with higher reliability and less programming effort than traditional programming methods. Data from existing planning methods can provide an excellent source of ground truth data against which we can evaluate new methods and compare the quality of model based algorithms against their unstructured peers.

Existing robotics datasets such as those outlined in Table II provide a good representation of certain aspects of manipulation, but fail to capture end-to-end task planning with obstacle avoidance. Capturing the interaction between the robot, objects, and obstacles is critical to ensure success in dynamic environments, as we show in Fig. 1. How can we investigate these dependencies within a dynamic scene?

Can an implicit understanding of physical dependencies be created from raw data? We introduce the CoSTAR Block Stacking Dataset (Fig. 2, 3, 4, and Sec. III) for the purpose of investigating these questions. It is designed as a benchmark for performing complex, multi-step manipulation tasks in challenging scenes. The target task is stacking 3 of 4 colored blocks in a specific order with simple target objects in a cluttered scene and variable surrounding environment.
Our block stacking task is constrained enough that one dataset might cover the task sufficiently, while still ensuring dynamics and physical dependencies are part of the environment. We show how, despite this simplicity, the task cannot be completed with the current design of existing grasping networks (Sec. III), nor by the trivial transfer of one example underlying architecture to a 3D control scheme (Sec. IV). Therefore, we apply Neural Architecture Search (NAS)\cite{4} to this dataset using our novel multiple-input HyperTree MetaModel (Fig. 6 and Sec. IV-C) to find a viable model. NAS is an approach to automatically optimize neural network based models to specific applications. In fact, we show that useful training progress is made with only a small subset of network models from across a broad selection of similar architectures (Fig 7). We hope that with specialization to other particular tasks, MetaModels based on HyperTrees might also serve to optimize other applications which incorporate multiple input data sources.

To summarize, we make the following contributions:

1) The CoSTAR Block Stacking Dataset: a valuable resource to researchers across a broad range of robotics and perception investigations.
2) The HyperTree MetaModel, which describes a space for automatically refining neural network based models with multiple input data streams.
3) Baseline architectures to predict 6 Degree of Freedom (DOF) end-effector goals for the grasping and placement of specific objects, as found via HyperTree search.

II. OVERVIEW AND RELATED WORK

Block stacking is itself already studied to improve scene understanding \cite{5}, and our videos include stacks standing, leaning, and tumbling. This pairs well with ShapeStacks\cite{6} a synthetic dataset for understanding how stacks of simple objects stand or fall. Example use cases for their dataset with our own includes the evaluation of model based methods’ ability to accurately predict future consequences and detect subtle collision scenarios with or without an object model.
Fig. 4: Row 1 is a successful and row 2 is a failed block stacking attempt. A sequence starts on the left with a clear view at frame $I_0$ then proceeds right showing the timesteps of the 5 goal poses $G$, (Eq. 2, Fig. 2, 3) at which the gripper may open or close. Notice the variation in bin position, gripper tilt, the challenging lighting conditions, the stack of 4 blocks, and the object wear. Viewing video and other details is highly recommended, see sites.google.com/site/costardataset.

The dataset provides the appearance of smooth actions (Eq. 2, Fig. 2, 3) at which the gripper may open or close. Notice the variation in bin position, gripper tilt, the challenging lighting conditions, the stack of 4 blocks, and the object wear. Viewing video and other details is highly recommended, see sites.google.com/site/costardataset.

might also be useful for developing, evaluating and comparing algorithms utilizing sim-to-real transfer, GANs, domain adaptation, and metalearning[26], [27]. These applications become particularly interesting when the 3D models we have available are added to a simulation, or when this dataset is combined with other real or synthetic robotics datasets.

Finally, Neural Architecture Search is an emerging way to automatically optimize neural network architectures to improve the generalization of an algorithm. Key examples include NASNet [28], and ENAS [29], but a broad overview becomes particularly interesting when the 3D models we have available are added to a simulation, or when this dataset is combined with other real or synthetic robotics datasets.

III. BLOCK STACKING DATASET

We define a block stacking task where a robot attempts to stack 3 of 4 colored blocks in a specified order. The robot can be seen in Fig. 3, and examples of key image frames for two stack attempts are shown in Fig. 4. A dataset summary can be found in Table I.

Data is collected utilizing our prior work on the collaborative manipulation system CoSTAR [3], [35]. CoSTAR is a system designed for end-user creation of robot task plans that offers a range of capabilities plus a rudimentary perception system based on ObjRecRANSAC. Motion is executed by first planning a direct jacobian pseudoinverse path, with an RRT-connect fallback if that path planning fails. In a single stack attempt the robot aims to complete a stack by performing 5 actions: 2 repetitions of the CoSTAR SmartGrasp and SmartPlace actions, plus a final move to the home position above the bin. The sequence pictured in Fig. 2 consists of the following 5 actions from top to bottom: grasp(red), place(red, on_blue), grasp(yellow), place(yellow, on_red_blue), and move(home). There are a total of 41 possible object-specific actions: grasp actions interact with each of the 4 colored blocks (4 actions), placement actions are defined for ordered stacks with up to height 2 (36 actions), and move(home).

The dataset provides the appearance of smooth actions with the gripper entering the frame, creating a stack in the scene, and finally exiting the frame at the end. During real time execution the robot (1) proceeds to a goal, (2) saves
the current robot pose, (3) stops recording data, (4) moves out of camera view to the home position, (5) estimates the block poses, (6) moves back to the saved pose, (7) resumes recording, (8) starts the next action. After moving to the final home position object poses are estimated and the maximum z height of a block determines stack success which is confirmed with human labeling. Some features, such as collision checks, are disabled so that a set of near-collision successes and failures may be recorded.

IV. PROBLEM AND APPROACH

We explore one example application on the CoSTAR dataset by demonstrating how high level pose goals might be set without object models. We assume that a higher level oracle has identified the next necessary action, and the purpose of the neural network is to learn to set 3D pose goals from data and an object-specific action identifier. The proposed goal can then be reached by a standard planning or inverse kinematics algorithm. The high level task and requirements placed on the network are outlined in Fig. 2.

A. Goals and Encodings

Each successful stacking attempt consists of 5 sequential actions (Fig. 2, 4) out of the 41 possible object-specific actions described in Sec. III. Stacking attempts and individual actions vary in duration and both are divided into separate 100 ms time steps t out of a total T. There is also a pose consisting of translation \( v \) and rotation \( r \) at each time step (Fig. 3), which are encoded between [0,1] for input into the neural network as follows: The translation vector encoding is \( v = (x, y, z)/d + 0.5 \), where \( d \) is the maximum workspace diameter in meters. The Rotation \( r \) axis-angle encoding is \( r = (ax, ay, az, \sin(\theta), \cos(\theta))/s + 0.5 \), where \( ax, ay, az \) is the axis vector for gripper rotation, \( \theta \) is the angle to rotate gripper in radians, and \( s \) is a weighting factor relative to translation. Example \( E \) is the input to the neural network:

\[
E_t = (I_0, I_t, v_t, r_t, a_t)
\]  

Where \( I_0 \) and \( I_t \) are the initial and current images, \( v_t, r_t \) are the respective base to gripper translation and rotation (Fig. 3), \( a_t \) is the object-specific one-hot encoding of 41 actions. Ground Truth Goal Pose \( G_t \) from Fig. 3 is the 3D pose at time \( g \) at which the gripper trigger to open or close, ending an action in a successful stacking attempt:

\[
G_t = (v^0_t, r^0_t) | t \leq g \leq T, e_g \neq e_{g-1}, a_g = a_t
\]  

where \( g \) is the first time the gripper moves after \( t \), \( e \) is the gripper open/closed position in [0, 1]. Finally, the Predicted Goal Pose \( P_t = (v^g_t, r^g_t) \) is a prediction of \( G_t \).

Each example \( E_t \) has a separate sub-goal \( G_t \) defined by (1) the current action \( a_t \), and (2) the robot’s 3D gripper pose relative to the robot base frame at the time step \( g \) when the gripper begins moving to either grasp or release an object. Motion of the gripper also signals the end of the current action, excluding the final move(home) action, which has a fixed goal pose.

B. Exploring the Block Stacking Dataset

We implemented several models similar to those found in existing work[1], [36], [21], [37]. We minimized our modifications to those necessary to accommodate our data encoding. Despite our best efforts, no baseline model we tried, and no hand-made neural network variation thereof could converge to reasonable values. Once we verified the CoSTAR dataset was itself correct, evaluated models on the Cornell Grasping Dataset[34] without issue, and tried a variety of learning rates, optimizers, models and various other parameters tuned by hand this complete lack of progress became very surprising. We analyze the underlying cause in Sec. V-A and include one reference model based on Kumra et. al.[36] in Fig. 5 for comparison. It quickly became clear that manually tweaking configurations would not be sufficient, so a more principled approach to network design would be essential. To this end, Neural Architecture Search and hyperparmeter search are well studied methods for automatically finding optimal parameters for a given problem, and we apply them here.

C. HyperTree MetaModel

Much like how Dr. Frankenstein’s creature was assembled from pieces before he came to life in the eponymous book, HyperTrees combine parts of other architectures to optimize for a new problem domain. Broadly, robotics networks often have inputs for images and/or vectors which are each processed by some number of neural network layers. These

| Robot Dataset | Real Data | Scene Varies | Human Demo | Open License | Grasp Place | Specific Objects | Scene Obstacle | Phys. Dep. | Robot Model | Val Set | Test Set | Code Incl. | Trials | Time Steps | Rate Hz |
|---------------|-----------|--------------|-------------|--------------|-------------|-----------------|----------------|------------|-------------|--------|---------|----------|-------|------------|--------|
| IHU CoSTAR Block Stacking | ☑ | ✗ | ✗ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | 11,977 | 186 | 10 |
| Google Grasping[1] | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ~800k | ~25 | 1 |
| MIME[30] | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | 8.260 | ~100 | 7 |
| BAIR Pushing[31] | ☑ | ✗ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | 45,000 | 30 | – |
| BAIR VisHard-solid/cloth[32] | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | 16k/31k | 30/20 | – |
| Jacquard[33] | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | 54,485 | 1 | – |
| Cornell[34] | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | 1,035 | 1 | – |
| Dex-Net 2.0[2] | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | 6.7M | 1 | – |

TABLE II: A comparison of robotics datasets. Our CoSTAR dataset also includes methods, documentation, examples, and the details to reproduce it. A dash indicates not available or not applicable. Physical dependencies are described in Fig. 1. The bin is our “Scene Obstacle”; forcible collision causes a security stop and the “Failure with errors” condition in Table I.
Fig. 5: (All) The best models’ predictions $P_t$ against ground truth $G_t$ at random times $t$. A high percentage of samples with low error is better. (Left) The importance of hyperparameter choice is visible in models 1-9 which were selected from the best of 1100 HyperTree candidates and then trained for 200 epochs. (Top) Distribution of angular error between predicted and actual 3D gripper rotations $\Delta \text{Rot}(r^P_t, r^G_t)$ (Eq. 2, and Fig. 3). (Bottom) Distribution of translation error $\|v^P_t - v^G_t\|$ (Eq. 2, and Fig. 3).

Fig. 6: A detailed view of the HyperTree MetaModel configured for predicting 3D ground truth goal poses, $G_t$, on the block stacking dataset. HyperTrees can accept an arbitrary number of image and vector inputs. Hyperparameter definitions are in Table III. “Blocks” are a sequence of layers. Components may then be concatenated to apply additional blocks of layers for data fusion. The output of these layers are subsequently split to one or more block sequences, typically dense layers. To search for viable architectures, the HyperTree MetaModel (Fig. 6) parameterizes these elements (Table III) so that models and their constituent parts might be defined, swapped, evaluated, and optimized in a fully automatic fashion. In fact, a HyperTree MetaModel’s search space can generalize many of the previously referenced architectures as a special case.

We explore and then optimize the models’ hyperparameter based configuration of the network structure using the standard optimization framework GPyOpt [42]. We (1) run HyperTree search for 1 epoch on between 500-5,000 models with augmentation, such as cutout[43], disabled depending on the available computing resources and dataset size. From this we (2) automatically construct a table of the best models, which we sort by a chosen metric, typically the average cartesian or angular validation error. We then (3) conduct a second automated training run proceeding down the top 1-10% of this sorted list for 10 epochs per model, which is added to our model table. In step (4) we repeat steps 2 and 3 for 200 epochs with 2-10 models and augmentation enabled, if appropriate. Step (5) is a 600 epoch training run initialized with the best model from step 4 resumed as needed until convergence, to reach a final model according to the chosen validation metric. An optional step (6) is to manually narrow the hyperparameter search space to ranges defined by the best image and trunk models and repeat steps 1-5.

Fig. 7: A cross-model comparison of average error with 1 epoch of training. Each dot represents a single HyperTree architecture which predicts both translation and orientation, $P_t$. Many models within the search space do not converge to useful predictions. The squares demonstrate how a selected pair of HyperTree architectures reduce error by predicting translation $v^P_t$ and rotation $r^P_t$ independently.
TABLE III: Architecture Search Parameters for the HyperTree MetaModel defined in Figure 6. Image Models: VGG16 [38], DN is DenseNet 121 [37], RN is ResNet 50 [21], [39], IRNv2 is Inception ResNetv2 [40], NAS is NASNet Mobile [28]. For Conv Trunk Block Model, NAS refers to the NASNet A Cell, DN refers to the DenseNet Dense Block, and ResNet refers to their Identity Block. The Activation hyperparameter applies to the Vector Model, the Conv3x3 Trunk Block, and the Dense Layers in the Dense Block. CoordConv [41] “Pre-Image” applies an initial CoordConv Layer to each input image and CoordConv “Pre-Trunk” applies a CoordConv layer after the vision and vector branches have been concatenated in the HyperTree Trunk. In Vector Block Model, “Dense” is a sequence of Dense Layers, while “DNBlock” is a DenseNet style block where Dense layers replace convolutions for the purpose of working with 1D input. Starred * parameters were searched then locked in manually for subsequent searches to ensure consistency across models.

V. RESULTS

Cornell Grasping Dataset: We first demonstrate that the HyperTree MetaModel with vector inputs generalizes reasonably well on the Cornell Grasping Dataset. Our pose classification model gets 96% object-wise 5-fold cross evaluation accuracy, compared with 93% for DexNet 2.0 [2]. State of the art is an image-only model at 98% [14].

Separation of translation and rotation models: In our initial search of the CoSTAR Block Stacking Dataset, a single model contained a final dense layer which output 8 sigmoid values encoding \( p_t \). The results of this search represent 1,229 models which are pictured as dots in Fig. 7. The figure demonstrates that we found no models which were effective for both translation \( v^p_t \) and rotation \( r^p_t \) simultaneously. This observation led us to conduct independent model searches with one producing 3 sigmoid values \( v^p_t \) (Eq. 2) encoding translations, and 5 sigmoid values predicting \( r^p_t \) (Eq.2) encoding rotations in \( p_t \) (Eq. 2). An example of the resulting improvement in performance plotted as squares is shown in Fig. 7.

CoSTAR Block Stacking Dataset: The hyperparameters of the best models resulting from the separate translation and rotation model searches are in Table III, while the performance of the top translation and rotation model is detailed in Fig. 5 for the training, validation, and test data.
Beyond models, several open questions remain before we can more fully leverage datasets: How can we assess accuracy with respect to successful or failed end-to-end trials without a physical robot? For example, is there not a trivial mapping from a given rotation and translation error to a trial’s success, so what metric will best generalize to real robot trials? Can we encode, embed, represent, and evaluate such information in a way that generalizes to new situations? The CoSTAR dataset can itself serve as a medium with which to tackle these objectives.

VI. Conclusion

We have presented the CoSTAR Block Stacking Dataset as a resource for researchers to investigate methods for perception-based manipulation tasks. This dataset supports a broad range of investigations including training off-policy models, the benchmarking of model based algorithms against data driven algorithms, scene understanding, semantic grasping, semantic placement of objects, sim-to-real transfer, GANs, and more. The CoSTAR BSD can serve to bridge the gap between basic skills and multi-step tasks, so we might explore the broader capabilities necessary to achieve generalized robotic object manipulation in complex environments.

To establish a baseline for this dataset we created the HyperTree MetaModel automated search method, which is designed for this problem and others in which existing architectures fail to generalize. Our final model from this search qualitatively demonstrates grasping of a specific object and can correctly avoid a scene’s boundaries, an essential capability for the full stacking task in a real-world environment.

VII. Acknowledgements

We thank Chunting Jiao for his assistance with data collection. This material is based upon work supported by the National Science Foundation under NSF NRI Grant Award No. 1637949.

REFERENCES

[1] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection,” The International Journal of Robotics Research, vol. 37, no. 4-5, pp. 421–436, 2018, dataset:https://sites.google.com/site/brainrobotdata/home. [Online]. Available: https://doi.org/10.1177/0278364917710318

[2] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, “Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics,” in Robotics: Science and Systems (RSS), 2017, dataset:berkeleyautomation.github.io/dex-net/.

[3] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, and G. D. Hager, “CoSTAR: Instructing collaborative robots with behavior trees and vision,” Robotics and Automation (ICRA), 2017 IEEE International Conference on, 2017. [Online]. Available: https://arxiv.org/abs/1611.06145

[4] T. Eikens, J. Hendrik Metzen, and F. Hutter, “Neural Architecture Search: A Survey,” arXiv preprint arXiv:1807.07226, Aug. 2018. [Online]. Available: http://arxiv.org/abs/1808.05377

[5] A. Lerer, S. Gross, and R. Fergus, “Learning physical intuition of block towers by example,” International Conference on Machine Learning, pp. 430–438, 2016.

[6] O. Groth, F. B. Fuchs, I. Posner, and A. Vedaldi, “Shapeshacks: Learning vision-based physical intuition for generalised object stacking,” in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 702–717.
[45] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” CoRR, vol. abs/1502.01852, 2015. [Online]. Available: http://arxiv.org/abs/1502.01852

[46] L. N. Smith, “No more pesky learning rate guessing games,” CoRR, vol. abs/1506.01186, 2015. [Online]. Available: http://arxiv.org/abs/1506.01186
A. Goals and Encoding Details, expanded

Each successful stacking attempt consists of 5 sequential actions (Fig. 2, 4) out of the 41 possible object-specific actions described in Sec. III. Stacking attempts and individual actions vary in duration and both are divided into separate 100 ms time steps \( t \) out of a total \( T \). There is also a pose consisting of translation \( v \) and rotation \( r \) at each time step (Fig. 3), which are encoded between [0,1] for use in a neural network as follows:

**Translation \( v \) vector encoding:**

\[
v = (x, y, z)/d + 0.5
\]

where: 
- \( d = 4 \), max workspace diameter (meters)
- \( x, y, z = \) robot base to gripper tip translation (meters)
- \( v = \) array of 3 float values with range [0,1]

**Rotation \( r \) axis-angle encoding:**

\[
r = (a_x, a_y, a_z, \sin(\theta), \cos(\theta))/s + 0.5
\]

where: 
- \( a_x, a_y, a_z = \) axis vector for gripper rotation
- \( \theta = \) angle to rotate gripper (radians)
- \( s = 1 \), scaling factor vs translation
- \( r = \) array of 5 float values with range [0,1]

Example \( E \) is the input to the neural network defined at a single time step \( t \) in a stacking attempt:

\[
E_t = (I_0, I_t, v_t, r_t, a_t)
\]

where: 
- \( T = \) Total time steps in one stack attempt
- \( t = \) A single 100ms time step index in \( T \)
- \( v_t = \) Base to gripper translation, see Eq. 3
- \( r_t = \) Base to gripper rotation, see Eq. 4
- \( h, w, c = \) Image height 224, width 224, channels 3
- \( I = \) RGB image tensor scaled from -1 to 1
- \( I_0 = \) First image, clear view of scene, \( t = 0 \)
- \( I_t = \) Current image, robot typically visible
- \( K, k = 41 \) possible actions, 1 action’s index
- \( a_t = \) action one-hot encoding

**Ground Truth Goal Pose** \( G_t \) from Fig. 3 is the 3D pose time \( g \) at which the gripper trigger to open or close, ending an action in a successful stacking attempt:

\[
G_t = (v_g, r_g)|t \leq g \leq T, e_g \neq e_{g-1}, a_g = a_t
\]

where: 
- \( g = \) First time the gripper moves after \( t \)
- \( e_g \neq e_{g-1} = \) gripper open/closed position in [0, 1]
- \( a_g = a_t = \) action at time \( t \) matches action at time \( g \)
- \( G_t = \) array of 8 float values with range [0,1]
- \( (v^g_t, r^g_t) = \) goal pose, same as \( (v_g, r_g) \)

**Predicted Goal Pose** \( P_t = (v^p_t, r^p_t) \) is a prediction of \( G_t \). Each example \( E_t \) has a separate sub-goal \( G_t = (v^g_t, r^g_t) \) defined by (1) the current action \( a_t \) and (2) the robot’s 3D gripper pose relative to the robot base frame at the time step \( t \) when the gripper begins moving to either grasp or release an object. Motion of the gripper also signals the end of the current action, excluding the final \text{move(home)} \) action, which has a fixed goal pose.

B. CoSTAR Block Stacking Dataset Details

We will outline a few additional CoSTAR BSD details here, and you can find our full documentation and links to both tensorflow and pytorch loading code at sites.google.com/site/costardataset. We include extensive notes with the dataset, explaining specific events and details of interest to researchers but outside the scope of this paper, such as where to obtain simulated robot models and dates when part failures occurred. We have included certain details regarding the approach, data channels, update frequency, time synchronization, and problems encountered throughout the data collection process in Fig. 1 that are not part of our approach to goal pose prediction, but may be useful for an approach to the stacking problem that is completely different from our own. We have also tried to ensure sufficient data is available for tackling other perception and vision related challenges. Attention to these details ensure a robotics dataset might prove useful as a benchmark for future research with methods that differ substantially from the original paper.

In between stack attempts the robot returns to its past saved poses in an attempt to unstack the blocks to automatically reset itself for the next attempt. If too many sequential errors are encountered the data collection application restarts itself which mitigates most system state and traditional motion planning errors. With this approach we find that we can automate the collection of stack attempts with approximately 1 human intervention per hour due to incidents such as security stops and failure cases in which all objects remain exactly where they started. A successful stack attempt typically takes on average about 2 minutes to collect and contains about 18 seconds of data logged at 10 Hz (100ms time steps), but this figure varies substantially across examples.

The AR tags on the robot is used to perform dual quaternion hand-eye calibration before the dataset was collected, and the AR tag in the bin was used to initialize the table surface for data collection as described in [3]. Object models and AR tags are not utilized in the neural network.

C. HyperTree Optimizer, losses, metrics, and preprocessing

HyperTree search repeatedly runs a sampling from 100 random architectures and then estimates 10 additional random architectures by optimizing the Expected Improvement (EI) in training loss with a predictive Sparse Gaussian Process Model (SGPM). These limits were chosen due to the practical tradeoff between processing time and memory utilization when evaluating the SGPM, since we found GPyOpt prediction time outstripped model evaluation time with large sample sizes. During training we perform optimization with Stochastic Gradient Descent (SGD). We also evaluated the Adam optimizer but we found it converged to a solution less
Fig. 9: Final training of the HyperTree rotation model in Table III and Fig. 5. Higher accuracy, lower error, and lower loss is better. Training was restarted on epoch 238, with a corresponding increase in learning rate. The final result is average angular errors of 16.0° (val) and 18.3° (test) at epoch 411. The horizontal axis represents performance at each training epoch on a linear scale while the vertical axis is log scale.

reliably. Mini-batches consist of a random example sampled at a random time step. Input to the network includes the initial image plus the image, encoded pose, and one-hot encoded action ID at the randomly chosen time step. The input gripper pose was encoded as described in Fig. 4 at that time step as an input to the network. The output of the neural network is a prediction of either the x, y, z coordinate at the goal time step encoded as $v^g_t$ (Eq. 3, 6), or the angle-axis encoded rotation $r^g_t$ (Eq. 4, 6) at the goal time step $g$.

We initialize the network by loading the pretrained ImageNet weights when available, and otherwise weights are trained from scratch utilizing He et al. initialization [45]. During HyperTree architecture search, we evaluate each model after a single epoch of training, so we either utilize a fixed reasonable initial learning rate such as 1.0, and for longer final training runs we utilized either a triangular or exp range (gamma=0.999998) cyclical learning rate [46] with a cycle period of 8 epochs, maximum of 2.0, and minimum of $10^{-5}$.

Translation training is augmented with cutout [43] plus random input pose changes of up to 0.5cm and 5 degrees. Each colored block is a 5.1 cm cube. An example of training for the final rotation model is shown in Fig. 9.

D. HyperTree Search Heuristics

We incorporated several heuristics to improve HyperTree search efficiency. We found that the best models would quickly make progress towards predicting goal poses, so if models did not improve beyond 1m accuracy within 300 batches, we would abort training of that model early. We also found that some generated models would stretch, but not break the limits of our hardware, leading to batches that can take up to a minute to run and a single epoch training time of several hours, so we incorporated slow model stopping where after 30 batches the average batch time took longer than 1 second we would abort the training run. In each case where the heuristic limits are triggered we return an infinite loss for that model to the Bayesian search algorithm.
Fig. 10: Examples from the dataset from top to bottom with key time steps in each example from left to right. All rows except the bottom represent successful stacking attempts. Also see the description in Fig. 4. Viewing video and other details is highly recommended, see sites.google.com/site/costardataset.