The CoSTAR Block Stacking Dataset: Learning with Workspace Constraints

Andrew Hundt[1], Varun Jain[1], Chia-Hung Lin[1], Chris Paxton[2], Gregory D. Hager[1]

***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 `sites.google.com/site/costardataset`.

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 ever 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 [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?

[1] Johns Hopkins University Department of Computer Science. {ahundt, vjain, ch.lin, cpaxton, ghager1}@jhu.edu

[2] Chris Paxton is with NVIDIA, USA

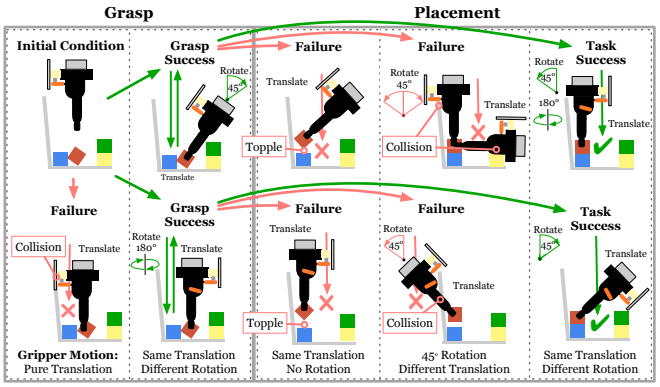

Fig. 1: A simplified 2-step `grasp(red)`, `place(red, on_blue)` stacking task with a side wall and asymmetric gripper. Arrows indicate possible sequences of actions. Task success is affected by the shape of the gripper, the obstacles, and the relationship between the pose of the gripper and the block it is holding.

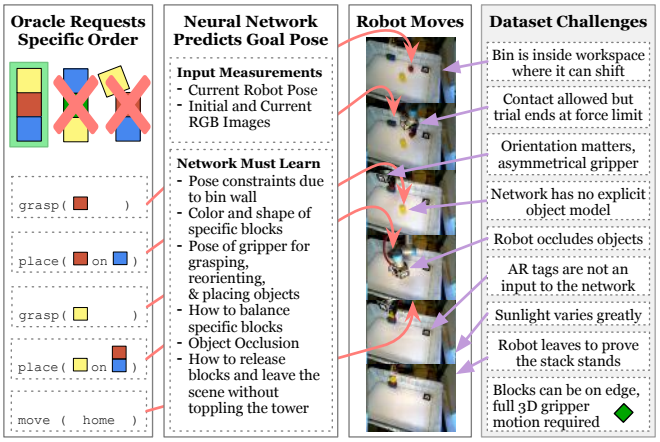

Fig. 2: An overview of the CoSTAR Block Stacking Dataset task and the requirements placed on our example neural network. In each example of stacking, the oracle requests a random specific order of colored blocks to simulate different customer choices.

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.

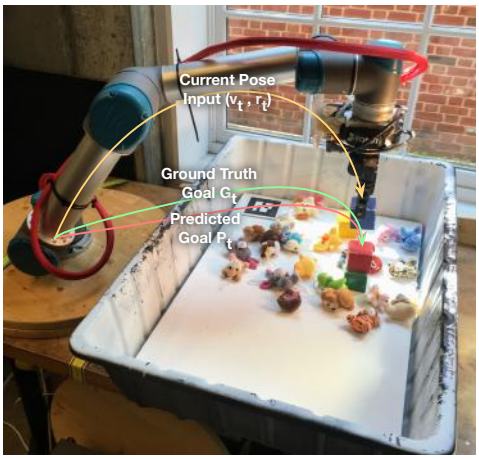

Fig. 3: The CoSTAR system [3] collecting the block stacking dataset.

CoSTAR Block Stacking Dataset Summary	
Calibrated Images	color, depth
Joint Data	angle, velocity
Labels	action, success/failure/error.failure
Blocks	red, green, yellow, blue
Block Actions	grasp(block), place(block, on_block(s))
Location Action	move_to(home)
Typical Example Timeline	18.6s duration, 186 frames, 10Hz

3D Coordinate Poses Recorded

gripper base and center	rgb camera	depth camera
robot joints	AR tags + ID#	colored blocks

Examples by Category out of the Combined Total of 11,977

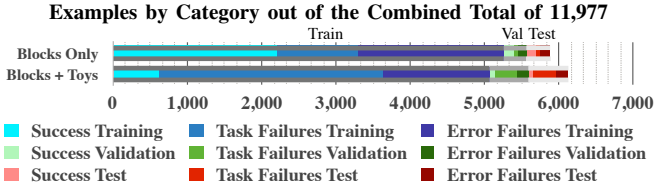

TABLE I: Stacking was conducted under 2 conditions: (1) blocks only and (2) blocks with plush toy distractors. Task Failures complete 5 actions but are unsuccessful at stacking. Causes of failures with errors include security stops from collisions, user emergency stops, planning failures, and software runtime errors.

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)[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 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 [5], and our videos include stacks standing, leaning, and tumbling. This pairs well with ShapeStacks[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.

Intuitively, block stacking might appear to be trivially solved by existing grasping or 3D object pose estimation algorithms alone. Recent advances in deep learning have revolutionized robotic grasping with perception based methods learned from big data [1], [2], [7], [8], [9], [10], [11]. One notable limitation of past approaches to robotic manipulation is the restriction of end effector poses to be either vertical and facing down or normal to local depth values, with only an angular parameter available to define orientation changes [1], [2], [12], [13]. Also common is the use of depth-only data [2] which precludes the possibility of object discrimination based on color. Progress towards semantic grasping of specific objects [9] is substantial, but it remains an open problem.

We demonstrate one specific starting condition for the block stacking task, visualized in Fig. 1, where obstacles and task requirements imply these methods are not sufficient on their own. We can reach two conclusions from the physical shape, translation, and orientation dependencies in this figure: (1) The 4 DOF (x, y, z, θ) available for gripper motion in current grasping networks [14], [12], [15] is not sufficient for precisely grasping and then placing one specific block on another in the general case, so at least one additional axis of rotation is necessary. (2) A neural network which predicts a 6 DOF object pose alone is not sufficient to overcome obstacles because a one to one mapping between 3D object poses and a sequence of successful gripper grasp and placement poses does not exist. In this example, no single definition of object poses will work because the required 45°rotation for precise placement will not match any square object pose. This means that even if an oracle provides both high level task instructions and perfect 3D object poses, an agent must discern the sequence of gripper poses, the shape of

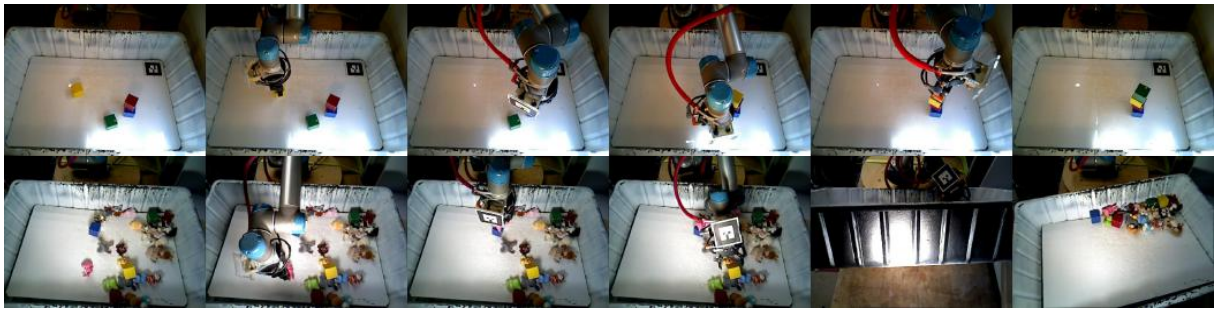

Fig. 4: Row 1 is a successful and row 2 is a failed block stacking attempt. A sequence starts on the left with a clear view at frame I_0 then proceeds right showing the timesteps of the 5 goal poses G_t (Eq. 2, Fig. 2, 3) at which the gripper may open or close. Notice the variation in bin position, gripper 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 objects, and control the robot correctly without a fatal collision for all time steps in between. We describe several of these requirements and challenges in Fig. 2.

These principles extend trivially to 3D if we consider all possible positions and orientations of 4 blocks and all 4 side walls, with rounded wall intersections. Consideration of other initial states will reveal other clear counter-examples, but we leave this exercise to the reader. It quickly becomes clear that in the general 3D case of this scenario at least 5 DOF are necessary to successfully grasp and then place a specific block on another. Our goal is to eventually generalize to more complex tasks than block stacking, so we design our example algorithm for full 6 DOF gripper motions.

Other work has investigated learning from simulation and then incorporating those models into robotic control [16]. Authors have explored simulation [17], [18], [19] and image composition [20] to generate training data that transfers to physical scenarios. Our approach to motion learning is inspired by [1], [9], with additional extensions based on [21], [20], [22], [23]. Others have used reinforcement learning for generating API calls for pre-programmed actions [24].

An investigation into the multi-step retrieval of an occluded object called Mechanical Search [15] specifically states: "[The] performance gap [between our method and a human supervisor] suggests a number of open questions, such as: Can better perception algorithms improve performance? Can we formulate different sets of low level policies to increase the diversity of manipulation capability? Can we model Mechanical Search as a long-horizon POMDP and apply deep reinforcement learning in simulation and/or in physical trials to learn more efficient policies?" Our CoSTAR dataset is specifically designed as a resource for exploring, addressing, pre-training, and benchmarking solutions to questions like these.

We can also imagine many additional topics for which the JHU CoSTAR Block Stacking Dataset might be utilized, such as the investigation and validation of task and motion planning algorithms offline on real data, imitation learning and off policy training of Reinforcement Learning[12], [25] algorithms for complete tasks with a sparse reward, model based algorithms which aim to complete assembly tasks in cartesian or joint space. Both the dataset and HyperTrees 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 is outside the scope of this paper, so we refer to a recent survey [4].

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

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
JHU 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 VisInt-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"; forceful collision causes a security stop and the "Failure with errors" condition in Table I.

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 $r = (a_x, a_y, a_z, sin(\theta), cos(\theta))/s + 0.5$, where a_x, a_y, a_z is the axis vector for gripper rotation, θ 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**:

$$E_t = (I_0, I_t, v_t, r_t, a_t) \tag{1}$$

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:

$$G_t = (v_t^g, r_t^g) | t \leq g \leq T, e_g \neq e_{g-1}, a_g == a_t \tag{2}$$

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** $P_t = (v_t^p, r_t^p)$ 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

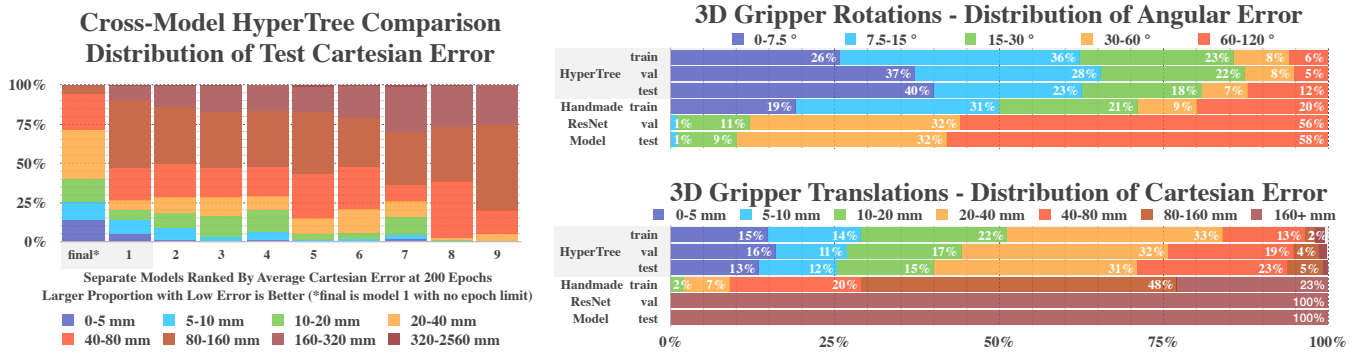

Fig. 5: **(All)** The best models' predictions P_t against ground truth G_t at random times t. A high percentage of samples with low error is better. **(Left)** The importance of hyperparameter choice is visible in models 1-9 which were selected from the best of 1100 HyperTree candidates and then trained for 200 epochs. **(Top)** Distribution of angular error between predicted and actual 3D gripper rotations $\Delta Rot(r_t^p, r_t^g)$ (Eq. 2, and Fig. 3). **(Bottom)** Distribution of translation error $\|v_t^p - v_t^g\|$ (Eq. 2, and Fig. 3).

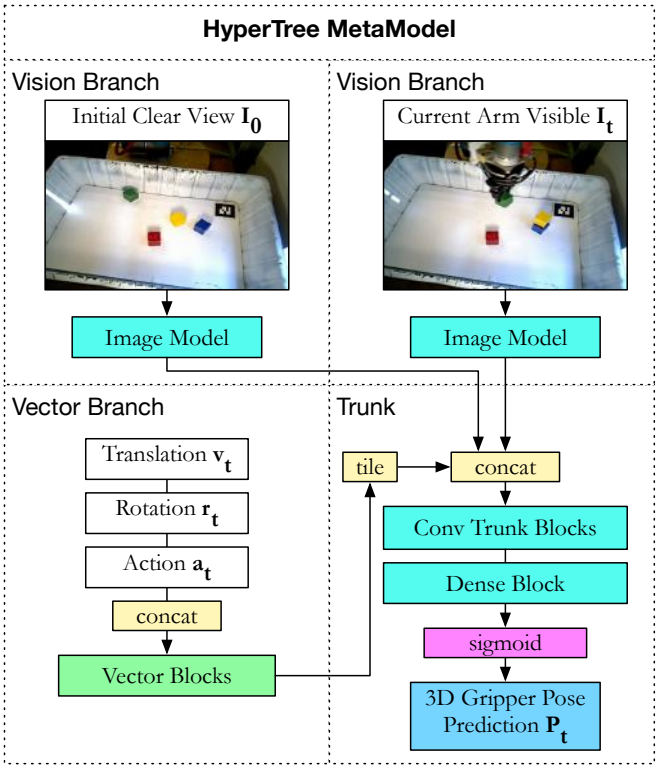

Fig. 6: A detailed view of the HyperTree MetaModel configured for predicting 3D ground truth goal poses, G_t, on the block stacking dataset. HyperTrees can accept an arbitrary number of image and vector inputs. Hyperparameter definitions are in Table III. "Blocks" are a sequence of layers.

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

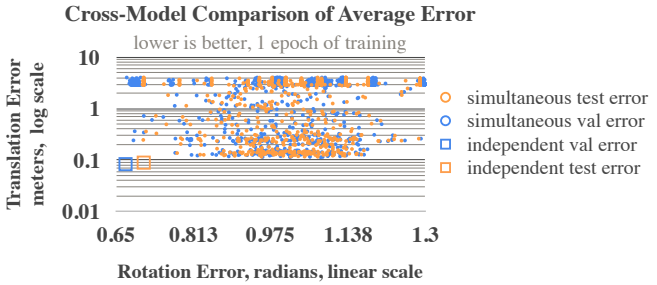

Fig. 7: A cross-model comparison of average error with 1 epoch of training. Each dot represents a single HyperTree architecture which predicts both translation and orientation, P_t. Many models within the search space do not converge to useful predictions. The squares demonstrate how a selected pair of HyperTree architectures reduce error by predicting translation v_t^p and rotation r_t^p independently.

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.

Variables, dimensions and inputs above (as in Sec. IV-A and IV-C) are parameterized. For example, HyperTrees

Hyperparameter	Search Space	Translation Model	Rotation Model
Image Model	[VGG, DN, RN, IRNv2, NAS]	NAS	VGG16
Trainable Image Model Weights*	[True, False]	True	True
CoordConv Layer Location	[None, Pre-Trunk, Pre-Image]	None	Pre-Trunk
Loss Function*	[mse, mae, msle]	mse	msle
Activation (Conv3x3, Vector Block, Dense Block)	[relu, elu, linear]	relu, relu, relu	N/A, relu, relu
Vector Block Model	[Dense, DN]	Dense	DN
Vector Block Layer Count	$n \in [0..5)$	2	1
Conv Trunk Block Model	[Conv3x3, NAS, DN, RN]	Conv3x3	NAS
Conv Trunk Block Count	$n \in [0..11)$	8	8
Filters (Vector, Trunk, Dense Block)	$2^n \vert n \in [6..13), [6..12), [6..14)$	2048, 1024, 512	256, 32, 2048
Dense Block Layer Count	$n \in [0..5)$	2	3
Normalization (Vector, Trunk)	[Batch, Group, None]	Batch, None	Batch, Batch
Optimizer*	[SGD, Adam]	SGD	SGD
Initial Learning Rate*	$0.9^n \vert n \in [0.0..100.0]$ continuous	1.0	1.0
Dropout rate*	$[0, 1/8, 1/5, 1/4, 1/2, 3/4]$	$1/5$	$1/5$

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.

accept zero or more vector and image inputs. The Cornell Grasping Dataset provides one image, and we utilize two on the block stacking dataset. Block stacking results are described in Fig. 5, Table III, and Section V.

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_t^p and rotation r_t^p simultaneously. This observation led us to conduct independent model searches with one producing 3 sigmoid values v_t^p (Eq. 2) encoding translations, and 5 sigmoid values predicting r_t^p (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. Results are presented on the success-only non-plush subset because the plush subset was being prepared during these experiments. For translations on the HyperTree network, 67% of test pose predictions are within 4 cm and the average error is 3.3 cm. For comparison, the colored blocks are 5.1 cm on a side. HyperTrees have 81% of rotation predictions within 30°and an average test angular error of 18.3°.

A. Ablation Study

In essence, HyperTree search is itself an automated ablative study on the usefulness of each component in its own structure. This is because a hyperparameter value of 0 or None in Table III represents the case where that component is removed. For this reason, the best HyperTree models will or will not have these components depending solely on the ranking of validation performance (Fig. 7). For example, a ResNetv2[36] based grasping model like the manually defined one in Fig. 5 is a special case which would rise to the top of the ranking if it were particularly effective.

As we look back to our initial hand-designed models (Sec. IV-B), recall that these did not converge to useful levels of error. Fig. 7 reveals why this might be. Only a select few of the HyperTree models make substantial progress even after training for 1 full epoch of more than 1 million time steps. Essentially, this means the hand-designed models are simply not converging due to the choice of hyperparameters. For this reason we can conclude that an automated search of a well designed search space can improve outcomes dramatically.

An additional HyperTree search of 1100 cartesian models confirms that differences in model quality persist with

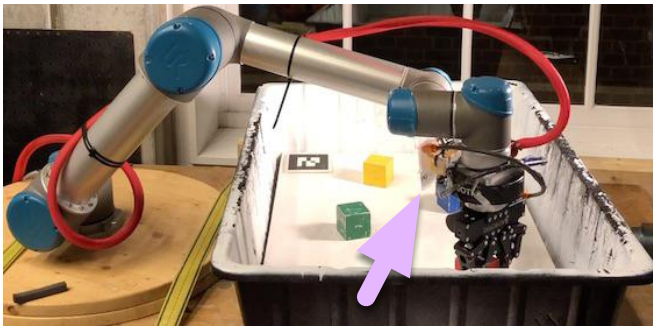

Fig. 8: A successful execution of the `grasp(red)` action with our final model. The predicted gripper orientation keeps the attached AR tag facing away from the walls.

additional training (Fig. 5). This search specified a NASNet-Mobile [28] image model and either a Conv3x3 or NASNet model A cell trunk, selected to explore the space around our final cartesian model. We conducted an initial 1 epoch run, a second 40 epoch run, and then a final 200 epoch run on the 9 best models with respect to validation cartesian error. The hyperparameters of the top 9 models vary widely within the search space. Examples of variation include: 0-3 vector branch layers, both vector block models, 0-4 dense block layers, 2-10 trunk layers, 512-8192 vector filters, all 3 CoordConv options, and both trunk options. This dramatic variation is very counter-intuitive. Indeed, we found the selection of 8 separate 32 filter NASNet A cells in our own best rotation HyperTree model (Table III) to be a very surprising choice. We would be unlikely to select this by hand. This unpredictability implies that there are many local minima among different possible architectures. Therefore, the broader conclusion we draw here is that researchers applying neural networks to new methods should perform broad hyperparameter sweeps and disclose their search method before reaching a firm conclusion regarding the strength of one method over another.

B. Physical Implications and Future Work

An example of the physical behavior of our final model (Fig. 8) shows initial progress towards an understanding of the obstacles in the scene, because the protruding side of the gripper faces away from the wall. Our accompanying video shows several qualitative test grasps and the motion of the predicted pose as a block moves around the scene. However, these qualitative tests also indicate the current model is not yet accurate enough for end-to-end execution, which we leave to future work.

Several clear avenues for improvement exist. Predictions might be made on a pixel-wise basis[13] to improve spatial accuracy, and pose binning[44] might improve accuracy. The Cross Entropy Method could sample around proposals for assessment with a Q function[12]. In turn, a well defined MetaModel based on HyperTrees might improve the accuracy of the networks underlying these other methods.

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, there is 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. Elsken, 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, "Shapestacks: Learning vision-based physical intuition for generalised object stacking," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 702–717.

[7] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[8] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 3406–3413. [Online]. Available: http://arxiv.org/abs/1509.06825

[9] E. Jang, S. Vijayanarasimhan, P. Pastor, J. Ibarz, and S. Levine, "End-to-end learning of semantic grasping," in *Conference on Robot Learning*, 2017, pp. 119–132. [Online]. Available: http://arxiv.org/abs/1707.01932

[10] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," in *Advances in Neural Information Processing Systems*, 2016, pp. 5074–5082.

[11] J. Redmon and A. Angelova, "Real-time grasp detection using convolutional neural networks," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1316–1322.

[12] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation," *ArXiv e-prints*, June 2018. [Online]. Available: https://arxiv.org/abs/1806.10293

[13] D. Morrison, J. Leitner, and P. Corke, "Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach," *Robotics: Science and Systems XIV*, Jun 2018. [Online]. Available: http://dx.doi.org/10.15607/RSS.2018.XIV.021

[14] H. Zhang, X. Zhou, X. Lan, J. Li, Z. Tian, and N. Zheng, "A real-time robotic grasp approach with oriented anchor box," *arXiv preprint arXiv:1809.03873*, 2018. [Online]. Available: http://arxiv.org/abs/1809.03873

[15] M. Danielczuk, A. Kurenkov, A. Balakrishna, M. Matl, R. Martin-Martin, A. Garg, S. Savarese, and K. Goldberg, "Mechanical search: Mutli-step retrieval of a target object occluded by clutter," 2019. [Online]. Available: https://abalakrishna123.github.io/files/2018-mech-search.pdf

[16] I. Mordatch, N. Mishra, C. Eppner, and P. Abbeel, "Combining model-based policy search with online model learning for control of physical humanoids," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 242–248.

[17] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.

[18] F. Zhang, J. Leitner, M. Milford, and P. Corke, "Modular deep q networks for sim-to-real transfer of visuo-motor policies," *arXiv preprint arXiv:1610.06781*, 2016. [Online]. Available: http://arxiv.org/abs/1610.06781

[19] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 4243–4250.

[20] C. Li, M. Zeeshan Zia, Q.-H. Tran, X. Yu, G. D. Hager, and M. Chandraker, "Deep supervision with shape concepts for occlusion-aware 3d object parsing," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *European Conference on Computer Vision*, pp. 630–645, 2016. [Online]. Available: http://arxiv.org/abs/1603.05027

[22] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 2169–2176.

[23] L. Wang, C.-Y. Lee, Z. Tu, and S. Lazebnik, "Training deeper convolutional networks with deep supervision," *arXiv preprint arXiv:1505.02496*, 2015. [Online]. Available: http://arxiv.org/abs/1505.02496

[24] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese, "Neural task programming: Learning to generalize across hierarchical tasks," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[25] Y. Aytar, T. Pfaff, D. Budden, T. Paine, Z. Wang, and N. de Freitas, "Playing hard exploration games by watching youtube," in *Advances in Neural Information Processing Systems*, 2018, pp. 2935–2945.

[26] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," *arXiv preprint arXiv:1812.07252*, 2018. [Online]. Available: http://arxiv.org/abs/1812.07252

[27] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks." *International Conference on Machine Learning*, pp. 1126–1135, 2017.

[28] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. [Online]. Available: http://arxiv.org/abs/1707.07012

[29] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," *International Conference on Machine Learning*, pp. 4092–4101, 2018. [Online]. Available: http://arxiv.org/abs/1802.03268

[30] P. Sharma, L. Mohan, L. Pinto, and A. Gupta, "Multiple interactions made easy (mime): Large scale demonstrations data for imitation," in *Conference on Robot Learning*, 2018, pp. 906–915.

[31] F. Ebert, C. Finn, A. X. Lee, and S. Levine, "Self-supervised visual planning with temporal skip connections," in *Conference on Robot Learning*, 2017, pp. 344–356, dataset:https://sites.google.com/berkeley.edu/robotic-interaction-datasets/home.

[32] F. Ebert, S. Dasari, A. X. Lee, S. Levine, and C. Finn, "Robustness via retrying: Closed-loop robotic manipulation with self-supervised learning," in *Conference on Robot Learning*, 2018, pp. 983–993, dataset:https://sites.google.com/berkeley.edu/robotic-interaction-datasets/home.

[33] A. Depierre, E. Dellandréa, and L. Chen, "Jacquard: A large scale dataset for robotic grasp detection," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3511–3516.

[34] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015, dataset:http://pr.cs.cornell.edu/grasping/rect. [Online]. Available: https://doi.org/10.1177/0278364914549607

[35] C. Paxton, F. Jonathan, A. Hundt, B. Mutlu, and G. D. Hager, "Evaluating methods for end-user creation of robot task plans," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6086–6092.

[36] S. Kumra and C. Kanan, "Robotic grasp detection using deep convolutional neural networks," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep 2017. [Online]. Available: http://dx.doi.org/10.1109/IROS.2017.8202237

[37] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[38] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *International Conference on Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. [Online]. Available: http://arxiv.org/abs/1512.03385

[40] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI Press, 2017, pp. 4278–4284.

[41] R. Liu, J. Lehman, P. Molino, F. P. Such, E. Frank, A. Sergeev, and J. Yosinski, "An intriguing failing of convolutional neural networks and the coordconv solution," in *Advances in Neural Information Processing Systems*, 2018, pp. 9628–9639.

[42] T. G. authors, "GPyOpt: A bayesian optimization framework in python," http://github.com/SheffieldML/GPyOpt, 2016.

[43] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1708.04552*, 2017. [Online]. Available: http://arxiv.org/abs/1708.04552

[44] S. Mahendran, M. Y. Lu, H. Ali, and R. Vidal, "Monocular object orientation estimation using riemannian regression and classification networks," *arXiv preprint arXiv:1807.07226*, 2018. [Online]. Available: http://arxiv.org/abs/1807.07226

[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

APPENDIX

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:

$$v = (x, y, z)/d + 0.5 \tag{3}$$

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:

$$r = (a_x, a_y, a_z, sin(\theta), cos(\theta))/s + 0.5 \tag{4}$$

where: a_x, a_y, a_z = axis vector for gripper rotation
θ = 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:

$$E_t = (I_0, I_t, v_t, r_t, a_t) \tag{5}$$

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
$\underset{h \times w \times c}{I}$ = 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
$\underset{1 \times K}{a_t}$ = 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:

$$G_t = (v_g, r_g) | t \leq g \leq T, e_g \neq e_{g-1}, a_g == a_t \tag{6}$$

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

Predicted Goal Pose $P_t = (v_t^p, r_t^p)$ is a prediction of G_t.

Each example E_t has a separate sub-goal $G_t = (v_t^g, r_t^g)$ 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 `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. I 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

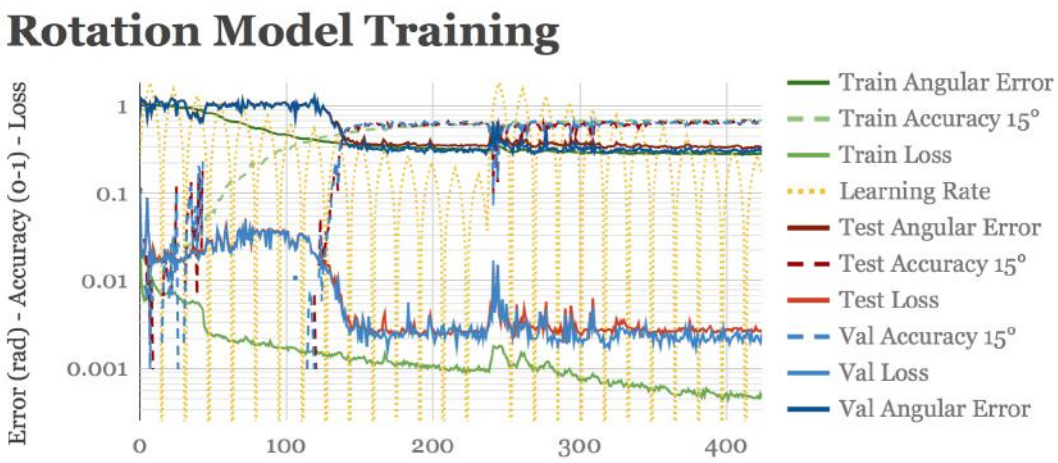

Fig. 9: Final training of the HyperTree rotation model in Table III and Fig. 5. Higher accuracy, lower error, and lower loss is better. Training was restarted on epoch 238, with a corresponding increase in learning rate. The final result is average angular errors of $16.0°$(val) and $18.3°$(test) at 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_t^g (Eq. 3, 6), or the angle-axis encoded rotation r_t^g (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.