Why Are CNNs The Model of Choice for Simulated Robotic Picking?

Josh Citron
Stanford University
450 Jane Stanford Way, CA
jcitron@stanford.edu

Doug Fulop
Stanford University
450 Jane Stanford Way, CA
fulop@stanford.edu

Olivia Taylor
Stanford University
450 Jane Stanford Way, CA
otaylor@stanford.edu

Abstract

Picking tasks are a staple of robotic task demonstrations, whether that be with dexterous hands or parallel-jaw grippers. We observed a variety of recent robotics simulation papers including MuJoCo Playground [11] and Maniskill3 [7] both implemented simple CNN architectures as the vision backbone for the task, even with state of the art models slowly taking the place of vanilla-CNNs in most other domains - and despite the marketing hype of robotics foundation models. To this end, we investigate the usage of CNN architectures for picking tasks by evaluating the performance of a simple CNN architecture against 5 more complex models: ResNets, ViTs, DINOv1, DINOv2, Depth-Anythingv2, and Theia. We find that simple CNNs with randomly initialized weights outperform other model architectures, achieving the highest reward across both simulated environments, with a fraction of the training time (at least 21x faster in our experiments).

1. Introduction

Robots promise to revolutionize physical labor and accelerate the economy, yet their real world applications are restricted to highly constrained tasks. Solving a task requires engineers to assemble physical hardware to make the task solvable by a particular robotic device (e.g. a set of cameras, robotic arms, mounts for components, etc.) then to carefully write a program to solve the particular task (primarily using traditional computer vision and robotics control techniques). Pixels to action control promises to enable robots to adapt to new tasks and change the paradigm from task engineering to task training. In this paper we explore the opportunity of training pixels to action policies that help robots solve tasks in simulation.

While robots can use a variety of sensors to understand the world, such as LiDAR (Light Detection And Ranging) scanners that generate point clouds and RGB-D depth cameras that produce an RGB image as well as a proceed depth map, many robot reinforcement-learning experiments and papers use standard RGB images captured from one or multiple cameras. End-to-end pixels-to-actions frameworks are especially desirable as they allow for lower cost imaging sensors to be deployed in the real-world and researchers hope the policies that are learned can handle the complexity of the real world.

The ability to train a robot to pick up an item remains a valuable task for industrial applications as well as blocking capability for robots to enter new environments such as the home and healthcare environments. As real-world experimentation is expensive due to hardware costs, challenges with resetting the environment to a consistent configuration, and an inability to scale environments to parallelize learning, sim-to-real transfer learning proposes a solution of leveraging high-speed simulation environments with physics engines and rendering capabilities.

Compounding a significant challenge in the pixels-to-action training scheme of the cost associated with collecting accurate data (both in simulation and in the real-world), many successful demonstrations are required to train both the vision backbone of a policy as well as the policy itself. When less data is available, it is common to use a lightweight vision model end-to-end, or a pretrained vision encoder that is finetuned. To this end, we want to investigate these two approaches: using a lightweight vision encoder and training end-to-end versus finetuning an existing state of the art model.

In this work, we use MuJoCo Playground [11] and Maniskill3 [7], two high-speed parallelized robotics simulation frameworks that allows for fast GPU training on various robotic manipulation tasks. We specifically investigate five models as the visual backbone for picking and pushing tasks: CNN, ViT, ResNet, DINO, and Depth-Anything-V2. From this investigation, we found that simple CNNs outperform even state of the art models, training faster and achieving higher average reward.

Through this exploration we hope to further the goal of training robotics policies that can transfer to the real world.

2. Related Work

2.1. Vision in Robotics

Lightweight models such as the canonical deep CNN presented in [4] have been used to achieve impressive performance on both simulated and real-world tasks [11]. Such models have the advantage of having relatively few parameters, and thus can be trained end-to-end.

More complicated models have also been shown to work well when data is available and training time is not a concern. Specifically, pixels-to-action robotics tasks have proven to be achievable zero-shot through Visual-Language-Action (VLA) models where multi-modal Large Language Models are fine-tuned on robotics demonstrations data to accept camera images and robot joint states as input and outputs action sequences. NORA [3] is a new open-weight VLA model that may have a sufficiently small memory footprint for inference on a consumer GPU such as the Nvidia 4090 in our gaming PC. The primary advantage of a VLA is that they can expand to arbitrarily complex sequence of tasks during test time that are instructed by natural language, rather than hand-crafted reward models and long simulation runs in task-specific simulation environments. However the fine-tuning process to adapt an open-weight model to a new robot form factor is unclear in computational cost, as the original VLA was pre-trained on over 4,000 H100 hours and the authors did not share the compute resources required to fine-tune for one robotics hardware platform.

Recent work such as Theia [6] has explored a mixture of experts combination of different off the shelf vision models into one foundation model specific to robotics. Specifically, Theia trains an encoder and backbone that can map RGB camera imagery to a latent space that can then be accurately decoded into an estimate of the outputs produced by various CLIP, SAM, DINOv2, Depth-Anything, and ViT models, at a fraction of the cost of computing each model individually.

2.2. Simulation Frameworks

To enable easy evaluation of robot policies without setting up real-world scenes, simulation environments have been developed. Recently, MuJoCo Playground [11] has allowed single GPU training for robot policies in MuJoCo. MuJoCo Playground also offers select environments with vision as an input to the policy using Madrona batch rendering [5]. While the vision environment is accessible, it is difficult to import pretrained weights into the Brax framework which uses Jax instead of Pytorch.

Alternative PyTorch-based environments such as Maniskill3 [7] are compatible with a wide-variety of well-tested model architectures with pre-trained weights, with similar training performance on a single GPU.

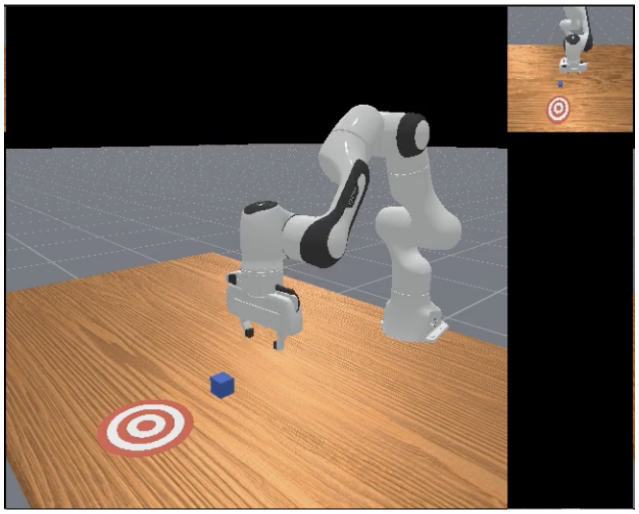
Figure 1. Maniskill PushCube Environment

3. Data

For this project we used two robot simulation environments: MuJoCo Playground and Maniskill3. Both simulation environments accept XML-based scene descriptions that include visual and dynamics models for robot arm, specifications for virtual cameras, the objects for the robot to interact with (e.g. tables and blocks), and visual markers for the task goal. The environments simulate the dynamics of the robot arm and other objects in the scene, and provide simulated virtual camera feeds in the form of RGB images. The simulated RGB camera images were rescaled from 128 height x 128 width by 3 channels to 224 height x 224 width by 3 channels to match the expected inputs of existing pre-trained models. Pixel values were normalized within the range 0 to 1 for all models except Theia [6] which provides its own normalization method.

In a block pushing task in Maniskill3 the table includes a bullseye graphic that marks the target block position. See Figure 1 for a rendered view of the environment as well a simulated camera feed in the top right corner of the image. A block picking task in MuJoCo Playground describes a scene with a table, a strip of bright colored tape, and a block that is placed on top of the tape. A similar block picking task in Maniskill identifies the target picking pose with a floating green dot.

Both environments are wrapped as an OpenAI Gym [1] environment that waits for a policy to provide an action for the current environment state for which the gym environment returns the next state. Both simulation frameworks are able to simultaneously render multiple, domain-randomized environments to enable policies to learn from multiple roll-outs each step of policy iteration. In our experiments we used between 64 and 256 simultaneous environments based on GPU VRAM limitations.

The final results in this paper were generated in approximately 20 hours of *g6e.2xlarge* instance time on Amazon Web Services which includes an Nvidia 48GB L40S GPU. Previous iterations of our experiments ran for over 50 hours on these GPUs as we rolled out several policies for 1 million steps to understand their robustness to over-fitting during training as well as providing larger models sufficient training time.

In MuJoCo Playground, we use the PandaPickCube-Cartesian environment, which can be seen in Figure 2. The goal of the task is to locate the red cube and lift it above a certain height.

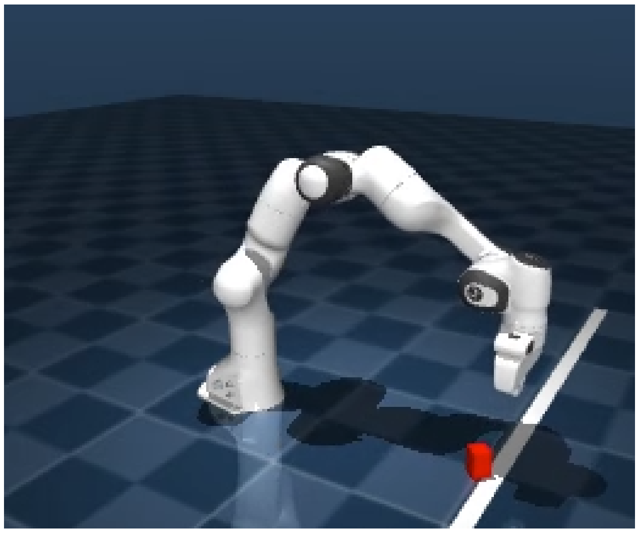
Figure 2. PandaPickCube environment in MuJoCo Playground.

4. Methods

We explored the baseline CNN and five alternative model architectures across the two simulation environments which we initially defined in Section 3 and further describe in Section 5. These alternative models scale in complexity from ResNets through to the robotics foundation model Theia. Across both simulation environments we maintain the same PPO training policy. Within the Maniskill environment we train a 2-linear layer with ReLu activation for each of the actors and critic. For consistency the models are all provided the same format of simulated RGB camera images, and both the actor and critic use the same model.

4.1. Convolutional Neural Networks (CNNs)

We utilized CNNs as a baseline to benchmark the our other models against, as CNNs are the primary backbone for nearly all vision-based systems. CNNs generally perform well given their built-in locality and translation invariance. However, CNNs with ImageNet pretraining struggle to generalize to robotic simulations [2]. Additionally, unusual viewpoints or highly textured environments not seen in training can cause CNNs to underperform and have poor feature extraction unless retrained, thus removing the advantage of CNN pretrained weights. Additionally, heavy CNNs can be too slow on low-power hardware systems without pruning or quantization.

In our experiments, we used two different variations of the same NatureCNN [4] in MuJoCo Playground and Maniskill. Neither baseline model used pretrained weights. We hypothesized that the relative simplicity of CNNs led to strong performance.

4.2. ResNet

Although CNNs generally perform well, they can struggle to learn complex tasks as their size increases. This can partially be attributed to vanishing gradients and loss of context of earlier inputs, a problem ResNets were invented to solve. ResNet is a specific CNN architecture that utilizes skip connections to mitigate vanishing gradients and enable deeper networks. By learning residual functions instead of direct mappings, ResNets are often able to achieve better feature representation and generalization when compared to plain CNNs of dimilar depth. ResNets pretrained on ImageNet allow for rapid transfer learning to new domains, thus yielding strong performance even with limted robotic data. Even without pretraining, the residual connections help ResNets train end to end and converge more quickly.

In MuJoCo Playground, we tested training an end-to-end ResNet with an architecture consisting of:

1. Convolutional stem:
   - `Conv2D(16, 7×7, stride=2)`
   - Layer normalization
   - ReLU activation
   - MaxPool(3×3, stride=2)
2. Two residual blocks:
   - Block 1: 16 channels, stride=2
   - Block 2: 32 channels, stride=2
   - Each block contains:
     - Two 3×3 convolutions with layer normalization
     - Skip connection with 1×1 projection when needed
     - ReLU activations
3. Task-specific head:
   - Global average pooling
   - Single-layer MLP (64 units)
   - Linear output layer

The architecture uses aggressive downsampling (total stride=16) and minimal channels to maintain real-time performance. Layer normalization is used throughout instead of batch normalization for RL stability. Channel normalization is applied to the input.

4.3. Vision Transformers (ViTs)

ViTs have demonstrated the ability to outperform CNNs on classification and segmentation benchmarks, but still remain less common in robotic applications. Specifically, ViTs have seen rapid adoption in academic robotic labs over the past few years, but the large amount of computation required to compute attention limits the ability of ViTs to run on embedded platforms. However, ViTs have the advantage of solving the CNN's limited receptive field by allowing each patch to attend to every other patch, thus solving long-range dependencies. ViTs are often used in simulation or offline learning, then distilled to lighter CNNs for on-robot inference, as even smaller version of ViTs are still slower than optimized CNNs as we discovered.

ViTs generally require sigificant pre-training in order to perform well, even more than CNNs. As a result, we wanted to test the performance of a small ViT on a robotic manipulation task.

We developed a custom ViT model in Jax which is intentionally lightweight for real-time robotics control. The network consists of:

1. Hybrid convolutional stem:
   - `Conv2D(32, 7×7, stride=4)`
   - `Conv2D(32, 3×3, stride=2)`
   - GELU activation
2. Minimal transformer:
   - Single transformer block
   - Single attention head (dim=32)
   - MLP hidden dimension = 64
   - Pre-norm architecture (LayerNorm)
   - No dropout
3. Task-specific head:
   - Global average pooling
   - Single-layer MLP (64 units)
   - Linear output layer

The architecture uses large patch sizes (effectively created by the conv stem's aggressive downsampling) and minimal layers to maintain real-time performance while still capturing spatial relationships through self-attention. Channel normalization is applied to the input for training stability.

4.4. DINO

DINO is a self-supervised distillation architecture that allows for large-scale pretraining without labels. While DINO is widely used in academic labs, it has yet to find widespread use in production robots end-to-end. However, DINO has found use in offline dataset preparation in applications such as semantic clustering and pseudo-label generation [8]. Its self-supervised architecture allows for training on millions of unlabeled frames to result in semantically meaningful features as well as ones which might not be captured in labeled data. Generally, either ResNets or ViTs have been used as a backbone for DINO. Training from scratch on ViT can require long training times and be computationally expensive, but models with pretrained weights can be used for transfer learning and fine-tuning.

Given DINO's limited use in robotics, we wanted to test various transfer learning methods to see if it could be viably used. We trained a variety of policies using DINO. In addition to a trainable policy, we tested a frozen DINO encoder, an unfrozen DINO encoder, and a DINO encoder with the final 3 layers unfrozen.

4.5. Depth-AnythingV2

Depth-AnythingV2 [10] is a monocular depth estimation model that builds on the original Depth-Anything model [9]. Both models make use of a teacher and student model, with the teacher model trained on labeled data, which predicts depths for unlabeled data that the student then uses in combination with the labeled data to learn. These models use an encoder-decoder structure, with the encoder taking the form of DINOv2 with a DPT decoder architecture. The simulated depth image is fed as an input into the Critic and Actor in our Maniskill3 PPO agent.

4.6. Theia

Theia [6] is a recent work that distills knowledge from across different off-the-shelf vision models such as ViTs, Depth-Anything v2, DINOv2, SAM, and CLIP. Model distillation can be formulated as training on a distillation objective that encourages matching between ground truth representations and predicted representations. Theia specifically distills the spatial tokens and not the CLS token, training a set of feature translators which are then used to produce a latent representation during inference, then decoded for a variety of outputs. We experiment with using the latents directly as well as concatenating the decoded simulated outputs of a ViT, Depth-Anthing v2, and CLIP as inputs to our critic and actor in our Maniskill3 agent.

5. Experiments

5.1. MuJoCo Playground Experimentation

We originally chose a pick cube task in MuJoCo Playground, where a robot is rewarded for lifting a cube 10cm above its starting position on a table, as its authors at Deepmind demonstrated the sim-to-real transfer of trained policies to a Franka Emika Panda 7-Degree of Freedom Robot. We originally intended to benchmark all of our policies on our own 6-Degree of Freedom Aubo i5 Arm to ensure that any insights on CNNs could transfer to real-world robotics. We modified the existing simulation environment to use an Aubo i5 model 3 instead of a Franka Emika Panda and

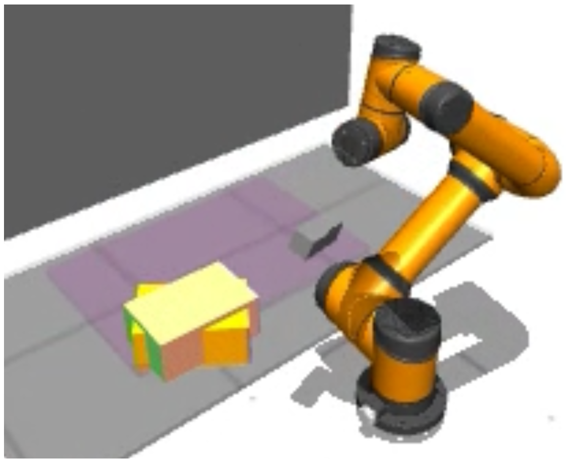

Figure 3. Aubo in MuJoCo Playground

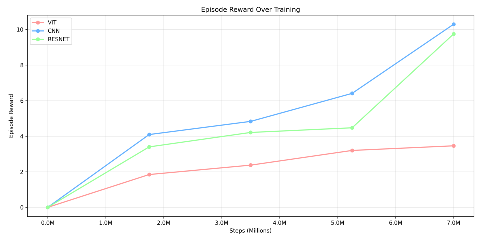

Figure 4. Episode Eval Reward for ResNet, ViT, and CNN on MuJoCo Playground

found that the model could not learn any non-random control policy.

We turned to simulation and benchmarked the baseline CNN model then discovered the challenge of importing pretrained models and weights for architectures such as ResNet and ViT. We implemented simplified versions of these architectures in Jax but they failed to solve the robotic picking task 4.

5.2. Migrating to Maniskill

Once we transitioned to the Maniskill and attempted to re-create the vision-guided reinforcement learning results provided by the authors in the paper we noticed that their baseline CNN (Nature CNN [4]) learned to achieve the goal state at a high level of accuracy within a surprisingly short 200k steps then maintained a high level of performance. We discovered that the authors were concatenating three elements as inputs to their vision-guided model: the objective ground truth state of the environment (including all known object positions) which would only be available within a simulator and are unrealistic for any real-world environment, along with simulated RGB camera images, as well as a simulated Depth Camera RGB-D image. We removed the objective ground truth data as well as the depth images from the model inputs and trained a new baseline model (see Figure 5 using the author's original training configuration, which we see takes approximately 700k steps to achieve similar performance to what was achieved by the privileged-information model within the first 250k steps, and the model loses its robustness to over-fitting in later stages (see the drop near step 1M).

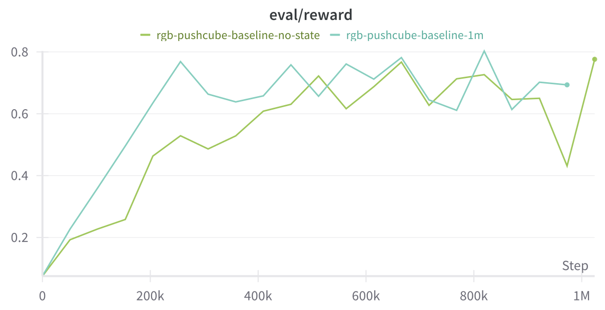

Figure 5. Maniskill Baseline Performance - Push Cube

5.3. Training Variations of DINO

We then attempted to train DINO. We made around 30 different attempts to train a policy that on the Push Cube task, but were unable to find a policy that would converge. We attempted to train ViT-Base with patch size 8 (largest model), ResNet-50, ViT-Small with patch size 16 (smallest model), attempting each with the entire network frozen and with the final 3 layers unfrozen. In all variations, the model did not make significant progress towards learning the task after between 30m to 6 hours each. We also experimented with the policy network size as well as various hyper-parameters, but were unsuccessful. To go through a few key examples, we began by attempting to train DINO on ViT-small pretrained frozen weights. This model trained for 5 hours and failed to converge, so instead we attempted DINO on ViT-small with last 3 layers unfrozen. However, this network also failed to converge. The policy was able to move the robot end effector towards the block and occasionally touch it, but could not move it in a meaningful way.

We then tried ViT-small with larger policy network and frozen network, which flailed backwards without touching the block. Next, we trained a larger policy network and last 3 layers unfrozen. This network was able to push the block every time, but was not able to move it in the direction of the goal. We then experimented with different backbones for DINO including ResNet-50 which performed no better than the others. Memory limitations prevented us from training a fully unfrozen network, but we hypothesize that this would not have performed significantly better. Ultimately, in over 1.5 hours of training, our best DINO model (See Figure 6) failed to learn a meaningful policy.

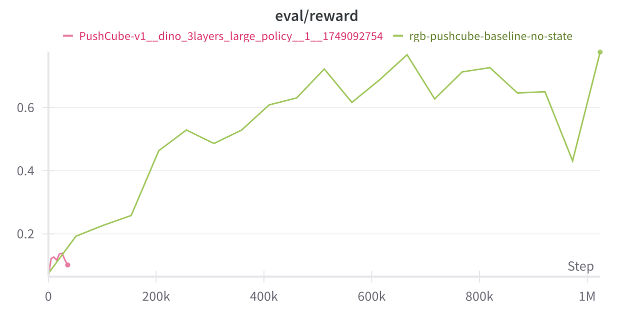

Figure 6. DINO

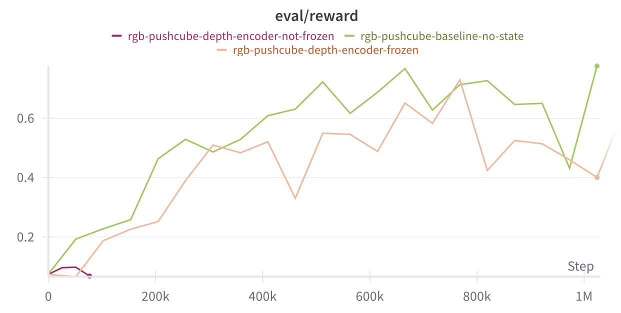

Figure 7. Depth Anything V2

5.4. Depth Anything V2

We then explored using the Depth Anything V2 model with pre-trained weights. We used the smallest pre-trained model size (*vits*) available on hugging face due to compute limitations. For simplicity and to explore the full potential of the model architecture we initially did not freeze the encoder weights and saw that the model failed to exceed 100k steps and earn rewards within 15 minutes of training time. We then froze the encoder later and saw performance that appeared within range of the baseline model (see Figure 7). We freeze the encoder weights to preserve what the model has learned previously as well as make for more efficient training. We watched videos from the evaluation roll-out and saw that policy learned to move the robot arm in circular paths that maintained an equal distance to the table while directionally aiming towards the cube to push, successfully solving the task. We trained 5 more policies and noticed similar circular motions emerging during the training process.

5.5. ResNet18 and ResNet50

We returned to explore models with more similar architectures to the baseline CNN, and started with ResNet18. We saw that a model with randomly initialized weights was not able to consistently improve despite matching the evaluation rewards of the baseline at one point near 400k steps (see Figure 10). We then saw that a model with pre-trained

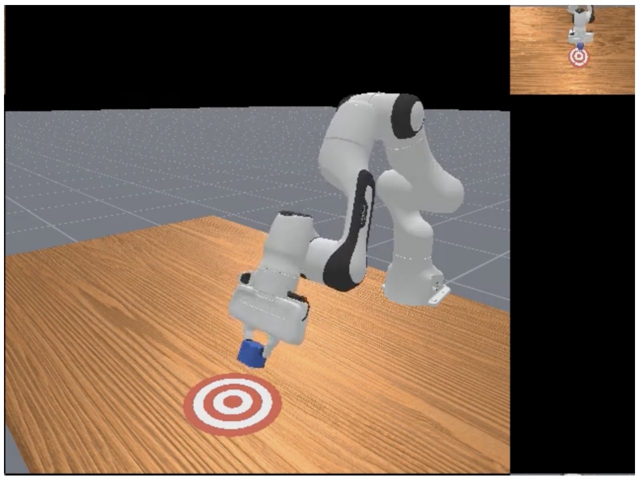

Figure 8. Emerging Behavior: Carrying

weights avoided earlier spikes but still failed to match our baseline within 15 minutes of compute (given time constraints). Therefore we froze all but the 3rd and fully connected layers within the blocks, saw encouraging performance on ResNet18 (see the red curve in Figure 9) and had sufficient GPU VRAM budget to expand to a pre-trained ResNet50 with all layers frozen except the same two layers.

The ResNet50 model was the first to more closely match the baseline model performance, though training wall-clock time for both partially-frozen ResNet models are a minimum of 10x slower than the baseline CNN. Both ResNet models learned a shoveling policy that more closely matched the baseline model motion, which we hypothesize is related to the convolution layers in both CNN and ResNet architectures.

Surprisingly the ResNet18 model developed an emerging behavior (see Figure 8): the policy picked up the cube to carry it to the destination when the cube could not be shoveled directly towards the target. We did not observe this behavior in any other model but we were able to observe the same behavior across multiple ResNet18 and ResNet50 models with pre-trained weights, suggesting a link between the weights learned from ImageNet input imagery and the learned robot policy.

5.6. Theia

Given the marketing hype of foundation models that have been pre-trained on related robotics tasks, we tried the new Theia robotics foundation model from the AI team at Boston Dynamics. Theia processes RGB camera images with a learned encoder and backbone layer to map inputs to a latent space. The latent space can then be decoded into outputs that reportedly accurately simulates what a variety of models including CLIP, Segment Anything, DepthAnythingV2, and ViT models from Google. The Theia paper suggests two approaches to training robotics tasks: training

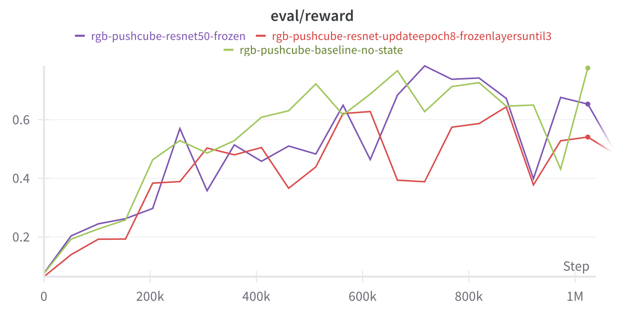

Figure 9. Resnet18 and Resnet50 - Both Partially Frozen

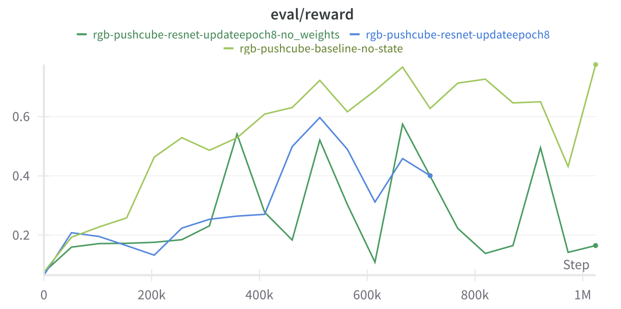

Figure 10. Resnet18 with Pre-trained and Random Weights

models directly using the learned latent representation, as well as decoding the latents into several simulated model outputs (e.g. CLIP and DepthAnything) then concatenating these outputs together.

We trained models with 30 variations of Theia model sizes (base, small, and tiny), frozen layers, output concatenations, and direct use of latents. We found all underperformed the baseline model, however we included the best model we were able to train from the Theia model (see Figure 11 for results from a Theia model trained with unfrozen weights on Theia small). This best run took 2.5 hours for 1M steps compared to the baseline model which took 7 minutes for 1M steps.

5.7. Discussion

Within all simulated environments the baseline CNNs appear to outperform more complex models. We suspect that this is due to the CNNs relatively simple structure and strong inductive biases (local receptive fields and translational invariance) that match visual patterns in PushCube, thus allowing task-relevant features directly from pixel input. Unlike ViTs or frozen DINO encoders, CNNs can be effectively trained end to end on limited robot data without requiring massive compute times or pretraining. For ResNets, we hypothesize that they perform almost equally as well if not better at times than CNNs because their design is inherently meant to perform as well as a simpler model due to the skip connections. For ViTs, even the lightweight ViT model was unable to converge, likely due to lack of pretrained weights. Given longer compute times and offline datasets, it may be able to converge, but with randomly initialized weights and lower compute, it was not able to do so.

An interesting discrepancy is the failure of DINO to converge while DepthAnythingv2, built using DINO as an encoder, achieves significant success. We suspect this is due to the additional decoder on DepthAnythingv2 which is trained to extract depth information from DINO features. Without this, we are expecting DINO or the policy network on top of it to be able to reconstruct informative depth and positional information from its features. Given that DINO is trained on ImageNet which contains millions of diverse images, it excels at identifying semantic similarity. However, given that the simulation environment's uniform, synthetic scene, its feature extraction is not as useful and thus the policy head is unable to extract reliable control signals. In contrast, DepthAnythingv2 pairs the DINO backbone with a pretrained depth encoder which converts features into coherent, per-pixel distance maps. Since these depth outputs directly encode the cube and table geometry, the policy has an immediately useful and stable representation, thus explaining its ability to converge when DINO features fail.

We hypothesize that Theia did worse than the CNN as, similar to a ViT, contained a minimum of 10 million learnable parameters and thus even with pretrained weights could not converge within a reasonable number of training steps. We observed that the inference time per step of Theia models on a set of 256 environments was a minimum of 2 minutes compared to less than 5 seconds with the baseline CNN, which would be challenging to achieve with our available GPUs. We believe additional hyper-parameter tuning may enable training on the Theia-tiny model and that a more complex task may benefit more from the latent-space representation that Theia generates. As none of the simulated outputs from Theia such as DepthAnythingV2 outperform the base CNN we are not convinced that Theia's latents are more valuable to learn than a simple CNN given our simulation environment and task.

We know that larger models can take longer to fit the training data and are more likely to overfit a given training set. We hypothesize that the out-performance is related to the relative consistency of the environments (particularly the similarity between the training environment and the evaluation environments). We also hypothesize that the visual sparsity of the simulated environments eliminates the value that larger models would bring to these policies when they are deployed in the real-world. Future work would be to test policies trained with each of these architectures in the real world to determine their their ability to bridge the sim-to-real gap.

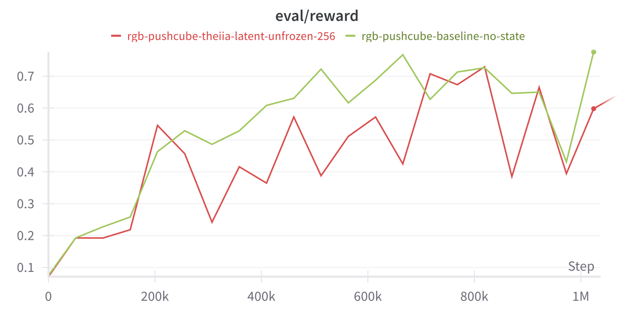

Figure 11. Theia

We hypothesize that pre-trained models may be more well-suited to appropriately responding to real-world camera visual inputs as their weights were trained on real world imagery instead of simulated imagery. Therefore it may be valuable to use a pre-trained ResNet model rather than a randomly initialized and purely simulation-trained CNN when performance is similar in simulation, until it is tested in the real-world.

6. Conclusion

In this paper, we investigated the use of CNNs as the vision backbone for simulated robotic picking tasks. Specifically, we compare a simple CNN to a ResNet50, DINOv1, Depth-Anythingv2, and Theia. We found that a simple CNN architecture with randomly initialized weights outperformed all other models in both the MuJoCo Playground environment as well as in Maniskill3. These results point to a few interesting conclusion. One is that for relatively simple scenes with limited visual occlusion, simple architectures are the way to go. We hypothesize that based on this, the more complicated and visually cluttered a scene gets, the better the complicated architectures will do compared to the simple CNN. This has important real world implications when choosing a vision backbone to deploy; it is desirable to not over-complicate the system while also maintaining functionality.

Future work could include validating the hypothesis that more complicated models will indeed outperfrom the simple CNN as visual occlusion increases by slowly populating the scene with more distractor objects and evaluating performance. Another direction would be to investigate specifically where the other models besides CNNs perform best in order to inform deployment.

7. Links

Maniskill Modified
MuJoCo Playground Modified

References

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. 2

[2] S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, and C. Finn. Robonet: Large-scale multi-robot learning. *CoRR*, abs/1910.11215, 2019. 3

[3] C.-Y. Hung, Q. Sun, P. Hong, A. Zadeh, C. Li, U. Tan, N. Majumder, S. Poria, et al. Nora: A small open-sourced generalist vision language action model for embodied tasks. *arXiv preprint arXiv:2504.19854*, 2025. 2

[4] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. 2, 3, 5

[5] L. G. Rosenzweig, B. Shacklett, W. Xia, and K. Fatahalian. High-throughput batch rendering for embodied ai. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–9, 2024. 2

[6] J. Shang, K. Schmeckpeper, B. B. May, M. V. Minniti, T. Kelestemur, D. Watkins, and L. Herlant. Theia: Distilling diverse vision foundation models for robot learning. *arXiv preprint arXiv:2407.20179*, 2024. 2, 4

[7] S. Tao, F. Xiang, A. Shukla, Y. Qin, X. Hinrichsen, X. Yuan, C. Bao, X. Lin, Y. Liu, T.-k. Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024. 1, 2

[8] Y. Wu, X. Li, J. Li, K. Yang, P. Zhu, and S. Zhang. Dino is also a semantic guider: Exploiting class-aware affinity for weakly supervised semantic segmentation. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 1389–1397, 2024. 4

[9] L. Yang, B. Kang, Z. Huang, X. Xu, J. Feng, and H. Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. In *CVPR*, 2024. 4

[10] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao. Depth anything v2. *Advances in Neural Information Processing Systems*, 37:21875–21911, 2024. 4

[11] K. Zakka, B. Tabanpour, Q. Liao, M. Haiderbhai, S. Holt, J. Y. Luo, A. Allshire, E. Frey, K. Sreenath, L. A. Kahrs, et al. Mujoco playground. *arXiv preprint arXiv:2502.08844*, 2025. 1, 2