Scalable Representations for Vision and Robotics

Tete Xiao

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-189
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-189.html

June 9, 2023

Scalable Representations for Vision and Robotics

By

Tete Xiao

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Trevor Darrell, Chair
Professor Jitendra Malik
Associate Professor Anca Dragan
Doctor Ross Girshick

Spring 2023

Scalable Representations for Vision and Robotics

Abstract

Scalable Representations for Vision and Robotics

by

Tete Xiao

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Trevor Darrell, Chair

Artificial intelligence systems have shown remarkable advancements in recent years. However, the challenge of scalability and generalization to real-world problems remains a significant issue. In this thesis, we explore the three key components of building scalable artificial intelligence systems for computer vision, including model optimizability, learning objectives, and large-scale datasets, and apply these outcomes for robotics.

Our work begins with an examination of the optimizability of vision transformers, proposing a new set of optimizability metrics and an alternative design for their patchify stem. Next, we introduce a contrastive self-supervised learning objective that reduces inductive biases in self-supervised learning, resulting in superior performance across various datasets. We then showcase the effectiveness of self-supervised visual pre-training from real-world images for learning motor control tasks from pixels, outperforming supervised baselines and matching oracle state performance.

Expanding on this, we explore self-supervised visual pre-training on images from diverse, in-the-wild videos for real-world robotic tasks, demonstrating the effectiveness of pre-trained representations across a range of tasks and embodiments. In addition, we present a sim-to-real learning-based approach for real-world humanoid locomotion using a causal Transformer, marking the first fully learning-based method for real-world full-sized humanoid locomotion. Finally, we conclude the thesis and discuss potential future directions for further research in the field.

To my parents

Contents

List of Figures **v**

List of Tables **xii**

Acknowledgments **xv**

1 Introduction **1**

2 Optimizability of Scalable Models: A Study of Vision Transformers **3**
2.1 Introduction . . . 3
2.2 Related Work . . . 5
2.3 Vision Transformer Architectures . . . 7
2.4 Measuring Optimizability . . . 9
2.5 Stability Experiments . . . 10
2.5.1 Training Length Stability . . . 10
2.5.2 Optimizer Stability . . . 12
2.5.3 Learning Rate and Weight Decay Stability . . . 13
2.5.4 Experimental Details . . . 13
2.6 Peak Performance . . . 14
2.7 Additional Study and Experimental Details . . . 16
2.8 Conclusion . . . 24

3 Reducing Inductive Bias in Contrastive Self-Supervised Learning **25**
3.1 Introduction . . . 25
3.2 Background: Contrastive Learning Framework . . . 26
3.3 LooC: Leave-one-out Contrastive Learning . . . 28
3.4 Experiments . . . 30
3.5 Related Work . . . 36
3.6 Conclusions . . . 37

4 Masked Visual Pre-training for Motor Control 38
4.1 Introduction . . . 38
4.2 Masked Visual Pre-training for Motor Control . . . 40
4.2.1 Masked Visual Pre-training . . . 40
4.2.2 Learning Motor Control from Pixels . . . 41
4.3 Benchmark Suite . . . 42
4.4 Experimental Setup . . . 44
4.5 Experimental Results . . . 45
4.5.1 Sample Complexity . . . 45
4.5.2 Pre-training Framework Comparison . . . 46
4.5.3 Comparison to In-domain Training Framework . . . 46
4.5.4 Ablations . . . 47
4.5.5 Representation Analysis . . . 48
4.6 Related Work . . . 49

5 Real-World Robot Learning with Masked Visual Pre-training 52
5.1 Introduction . . . 52
5.2 Related Work . . . 54
5.3 Framework . . . 54
5.3.1 Masked Visual Pre-training . . . 54
5.3.2 Real-World Robot Learning . . . 56
5.4 Experimental Setup . . . 56
5.5 Experimental Results . . . 58
5.5.1 Basic Motor Control . . . 59
5.5.2 Visually Diverse Scenes and Objects . . . 59
5.5.3 Scaling Model and Data Size . . . 61
5.5.4 Comparison to Concurrent Work . . . 62
5.5.5 Ablation Studies . . . 62
5.5.6 Case Study: Multi-finger Hand . . . 64
5.5.7 Data Collection . . . 65
5.5.8 Evaluation Protocol . . . 66
5.6 Conclusion . . . 67

6 Learning Humanoid Locomotion with Transformers 68
6.1 Introduction . . . 68
6.2 Related Work . . . 70
6.3 Method . . . 71
6.3.1 Model Architecture . . . 72
6.3.2 State-policy Supervision . . . 73

6.3.3 Joint Optimization with Reinforcement Learning 73
6.4 Experimental Setup 74
6.4.1 Digit Humanoid Robot 74
6.4.2 Simulation Environment 74
6.4.3 Reward 75
6.4.4 Reinforcement Learning Algorithm 79
6.4.5 Neural Network Model 79
6.5 Experiments 80
6.5.1 Simulation Experiments 80
6.5.2 Real-World Experiments 81
6.5.3 Dirty Laundry 85
6.6 Conclusion 86

7 Summary and Future Directions **87**

Bibliography **88**

List of Figures

2.1 **Early convolutions help transformers see better**: We hypothesize that the substandard optimizability of ViT models compared to CNNs primarily arises from the *early* visual processing performed by its *patchify stem*, which is implemented by a non-overlapping stride-p $p \times p$ convolution, with $p = 16$ by default. We *minimally* replace the patchify stem in ViT with a standard *convolutional stem* of only ∼5 convolutions that has approximately the same complexity as a *single* transformer block. We reduce the number of transformer blocks by one (*i.e.*, $L - 1$ *vs.* L) to maintain parity in flops, parameters, and runtime. We refer to the resulting model as ViT$_C$ and the original ViT as ViT$_P$. The vast majority of computation performed by these two models is identical, yet surprisingly we observe that ViT$_C$ (i) converges faster, (ii) enables, for the first time, the use of either AdamW or SGD without a significant accuracy drop, (iii) shows greater stability to learning rate and weight decay choice, and (iv) yields improvements in ImageNet top-1 error allowing ViT$_C$ to outperform state-of-the-art CNNs, whereas ViT$_P$ does not. 4

2.2 **Training length stability**: We train 9 models for 50 to 400 epochs on ImageNet-1k and plot the Δtop-1 error to the 400 epoch result for each. ViT$_C$ demonstrates faster convergence than ViT$_P$ across the model complexity spectrum, and helps close the gap to CNNs (represented by RegNetY). 11

2.3 **Optimizer stability**: We train each model for 50 to 400 epochs with AdamW (upward triangle .9513.6) and SGD (downward triangle .9513.6). For the baseline ViT$_P$, SGD yields significantly worse results than AdamW. In contrast, ViT$_C$ and RegNetY models exhibit a much smaller gap between SGD and AdamW across all settings. Note that for long schedules, ViT$_P$ often fails to converge with SGD (*i.e.*, loss goes to NaN), in such cases we copy the best results from a shorter schedule of the same model (and show the results via a dashed line). 11

2.4 **Hyperparameter stability for AdamW (*lr* and *wd*)**: For each model, we train 64 instances of the model for 50 epochs each with a random *lr* and *wd* (in a fixed width interval around the optimal value for each model). *Top*: Scatterplots of the *lr*, *wd*, and *lr* $\cdot$*wd* for three 4GF models. Vertical bars indicate optimal *lr*, *wd*, and *lr* $\cdot$*wd* values for each model. *Bottom*: For each model, we generate an EDF of the errors by plotting the cumulative distribution of the Δtop-1 errors (Δ to the optimal error for each model). A steeper EDF indicates better stability to *lr* and *wd* variation. ViT$_C$ significantly improves the stability over the baseline ViT$_P$ across the model complexity spectrum, and matches or even outperforms the stability of the CNN model (RegNetY). 12

2.5 **Hyperparameter stability for SGD (*lr* and *wd*)**: We repeat the setup from Figure 2.4 using SGD instead of AdamW. The stability improvement of ViT$_C$ over the baseline ViT$_P$ is even larger than with AdamW. *E.g.*, ~60% of ViT$_C$-18GF models are within 4% Δtop-1 error of the best result, while less than 20% of ViT$_P$-18GF models are (in fact most ViT$_P$-18GF runs don't converge). 13

2.6 **Peak performance (epoch training time *vs.* ImageNet-1k val top-1 error)**: Results of a fair, controlled comparison of ViT$_P$, ViT$_C$, and CNNs. Each curve corresponds to a model complexity sweep resulting in a training speed spectrum (minutes per ImageNet-1k epoch). *Left:* State-of-the-art CNNs. Equipped with a modern training recipe, ResNets are highly competitive in the faster regime, while RegNetY and Z perform similarly, and better than EfficientNets. *Middle:* Selected CNNs compared to ViTs. With access to only ImageNet-1k training data, RegNetY *and* ResNet outperform ViT$_P$ across the board. ViT$_C$ is more competitive with CNNs. *Right:* Pretraining on ImageNet-21k improves the ViT models more than the CNNs, making ViT$_P$ competitive. Here, the proposed ViT$_C$ outperforms all other models across the full training speed spectrum. . . 15

2.7 **Stem normalization and non-linearity**: We apply BN and ReLU after the patchify stem and train ViT$_P$-4GF (*left plot*), or replace BN with layer norm (LN) in the convolutional stem of ViT$_C$-4GF (*middle plot*). EDFs are computed by sampling *lr* and *wd* values and training for 50 epochs. The table (*right*) shows 100 epoch results using best *lr* and *wd* values found at 50 epochs. The minor gap in error in the EDFs and at 100 epochs indicates that these choices are fairly insignificant. 17

2.8 **Deeper models**: We increase the depth of ViT$_P$-4GF from 12 to 48 blocks, termed as ViT$_P$-16GF (48 blocks), and create a counterpart with a convolutional stem, ViT$_C$-16GF (47 blocks); all models are trained for 50 epochs. *Left*: The convolutional stem significantly improves error and stability despite accounting for only ~2% total flops. *Middle*, *Right*: The deeper 16GF ViTs clearly outperform the shallower 4GF models and achieve similar (slightly worse) error to the shallower and wider 18GF models. The deeper ViT$_P$ also has better lr/wd stability than the shallower ViT$_P$ models. 18

2.9 **Complexity measures *vs.* runtime**: We plot the GPU runtime of models versus three commonly used complexity measures: *parameters*, *flops*, and *activations*. For all models, including ViT, *runtime is most correlated with activations*, not flops, as was previously shown for CNNs. 19

2.10 **Impact of training recipes on convergence**: We train ViT models using the DeiT recipe *vs.* our simplified counterpart. *Left and middle*: Δtop-1 error of 4GF and 18GF models at 50, 100 and 200 epoch schedules, and asymptotic performance at 400 epochs. *Right*: Absolute top-1 error of 18GF models. Removing augmentations and using model EMA accelerates convergence for both ViT$_P$ and ViT$_C$ models while slightly improving upon our reproduction of DeiT's top-1 error. 23

3.1 Self-supervised contrastive learning relies on data augmentations as depicted in (a) to learn visual representations. However, current methods introduce inductive bias by encouraging neural networks to be less sensitive to information w.r.t. augmentation, which may help or may hurt. As illustrated in (b), rotation invariant embeddings can help on certain flower categories, but may hurt animal recognition performance; conversely color invariance generally seems to help coarse grained animal classification, but can hurt many flower categories and bird categories. Our method, shown in the following figure, overcomes this limitation. 27

3.2 **Framework of the Leave-one-out Contrastive Learning approach**, illustrated with two types of augmentations, i.e., random rotation and color jittering. We generate multiple views with leave-one-out strategy, then project their representations into separate embedding spaces with contrastive objective, where each embedding space is either invariant to all augmentations, or invariant to all but one augmentation. The learnt representation can be the general embedding space $\mathcal{V}$ (blue region), or the concatenation of embedding sub-spaces $\mathcal{Z}$ (grey region). Our results show that either of our proposed representations are able to outperform baseline contrastive embeddings and do not suffer from decreased performance when adding augmentations to which the task is not invariant (i.e., the red X's in Figure 1). 29
3.3 **Top nearest-neighbor retrieval results** of LooC vs. corresponding invariant MoCo baseline with color (left) and rotation (right) augmentations on IN-100 and iNat-1k. The results show that our model can better preserve information dependent on color and rotation despite being trained with those augmentations. 34
3.4 **Histograms of correct predictions (activations×weights of classifier) by each augmentation-dependent head** from IN-100 and iNat-1k. The classifier on IN-100 heavily relies on texture-dependent information, whereas it is much more balanced on iNat-1k. This is consistent with the improvement gains observed when learning with multiple augmentations. 36

4.1 We explore learning visual representations from large scale collections of images "in the wild", e.g., from YouTube or Egocentric videos, and using them to learn to perform a range of different motor control tasks from pixels. Please see the supplementary materials for videos. 39
4.2 **Masked visual pre-training for motor control:** *Left:* We first *pre-train* visual representations using *self-supervision* through masked image modeling from *real-world* images. *Right:* We then *freeze* the image encoder and train task-specific controllers on top with reinforcement learning (RL). The *same* visual representations are used for all motor control tasks. . . 40
4.3 **Sample complexity:** We plot the success rate as a function of environment steps on the 8 PixMC tasks. Each task uses either the Franka arm with a parallel gripper or the Kuka arm with a multi-finger hand. The MVP approach outperforms the supervised baseline on all tasks and closely matches the oracle state model (considered the upper bound of RL) on 6 tasks at convergence. The result shows that self-supervised pre-training improves representation quality for motor control tasks. 43

4.4 **Pre-training framework:** MVP vs. CLIP and MoCo visual pre-training frameworks. 45
4.5 **In-domain training comparisons:** MVP vs. state-of-the-art methods on two benchmarks. 45
4.6 **Ablation studies:** pre-training data, encoder sizes, sensors, and action learning framework. 47
4.7 **Disentangles shape and color:** The robots are trained to pick up a blue box of 4.5cm side length. At *test time*, we add a distractor in terms of color (blue vs. green), shape (cube vs. sphere), or size (4.5cm vs. 6cm), shown at the top. MVP maintains high success rates for color and shape, whereas the scale ambiguity is likely due to single first-person camera setup. 49
4.8 **Handles objects of various shapes:** We import three additional objects (i.e., can, mug, and banana) from the YCB dataset and re-train the controller to pick up the individual object category. The Kuka robot with the Allegro hand can pick up all objects with at least 50% success rate. These results highlight the generality of our visual representations. 49
4.9 **Attention maps visualization.** We observe that MVP models (c) capture a notion of objects which is not represented by the supervised recognition models (b). Note that our approach is self-supervised and the representations emerges automatically. Interestingly, the results are surprisingly consistent across the different attention heads (3 shown here). 50

5.1 **Real-world robot learning with masked visual pre-training.** We learn visual representations from a massive collection of Internet and egocentric data. We pre-train representations with masked image modeling, freeze the encoder, and learn control policies for robotic tasks on top. 53
5.2 **One encoder for all robots and tasks.** We train control policies per task, on top of the frozen encoder. The same vision encoder is used for all downstream robotic tasks and embodiments. 55
5.3 **Real-world robotic tasks.** We perform extensive real robot evaluations using a 7 DoF robot arm with a parallel jaw gripper. Our tasks include basic motor control skills (reaching a red block, pushing a wooden cube, and picking a yellow cube), variations in scenes (closing a fridge), objects (picking fruits), and scenes and objects (picking a detergent bottle from a cluttered sink). 56

5.4 **Comparison to vision encoders.** We compare our approach to visual encoders trained with CLIP, supervised learning on the ImageNet, and from scratch on the task at hand. In all cases, we observe that our approach consistently outperforms the baselines by a considerable margin. 58
5.5 **Sample complexity.** We show the performance of our approach as the number of demonstrations varies from 20 to 80. CLIP performance at 80 demonstrations is shown with a dashed lined for reference. Our approach is comparable to CLIP using only half the number of demonstrations. . . 58
5.6 **Variations in scenes and objects.** We compare our approach to CLIP on tasks with variations in scenes (closing the fridge), objects (picking fruits), and scenes and objects (picking an object from a cluttered sink). The models are ViT-Base. Our approach considerably outperforms CLIP and the gap is larger than in simpler settings (see Figure 5.4). This may suggest that our representations capture more precise spatial structure that is helpful for robotic tasks in more realistic contexts. 60
5.7 **Scaling model and data.** We study the scaling properties of our approach. We observe that scaling the model size alone from ViT-S to ViT-B while keeping the dataset fixed (HoI image collection; see text for details) does not improve the performance and even hurts (left). However, when we scale both the model and data (our massive Ego image collection; see text for details) we see clear benefits from a larger model. The trend continues when going further from the 86M ViT-B to the 307M ViT-L model (middle & right). Moreover, the gains are larger for harder tasks (right). 61
5.8 **Ablation studies.** We conduct ablation studies on the camera setups; input modality; commonly used image augmentations; from-scratch training architecture; CLIP pre-training architecture, and end-to-end finetuning on downstream tasks. See Section 5.5.5 text for more details. 63
5.9 **Multi-finger hand.** We show that our framework readily generalizes to a different robot morphology. We experiment with finger reaching, using seen and unseen objects, and cube flipping. 64
5.10 **Data collection.** We show our setup for collecting demonstrations in the real world with human operation. *Left:* We collect xArm demonstrations using an HTC Vive VR controller. *Right:* We collect Allegro hand demonstrations using an Meta Quest 2 device. See text for more details. 65

6.1 **Humanoid locomotion.** We present a learning-based approach for humanoid locomotion and evaluate it on a full-sized real-world Digit robot. Our policies are trained entirely in simulation and successfully transferred to real hardware zero-shot. Our robot can adapt to external disturbances such as carrying a backpack or a handbag; being pushed by a stick, pulled by cables, or having a yoga ball thrown at it. Moreover, it can walk over terrains with different friction, texture, and geometry. 69
6.2 **Humanoid Transformer.** Our neural network controller is a causal Transformer model trained by autoregressive prediction of future actions from the history of observations and actions. We hypothesize that the observation-action history contains useful information about the world that a powerful Transformer model can leverage to adjust its actions in-context. 71
6.3 **External disturbance.** We include carrying constant loads and withstanding external forces. See also Figure 6.1. 81
6.4 **Rough terrain.** The controller undergoes tests on eight different types of challenging terrain in the laboratory, with three being depicted in the figure (slippery surfaces, cables, and rubber). The robot is instructed to walk forward at a constant velocity of 0.15 m/s. See also Figure 6.1. . . . 83
6.5 **Adaptation, steps.** We test the robot's ability to climb steps despite the controller not being exposed to such terrains. We observe that the controller initially makes a mistake, but quickly adapts by lifting the leg faster and higher on the second attempt, which suggests adaptability of the controller. 84
6.6 **Adaptation, motor malfunction.** We simulate a sudden malfunction of the left knee motor by decreasing its PD gains by 50%. The figure shows the changes in the position, velocity, and torque of both the left and right knee motors, with the vertical dashed line indicating the moment of the simulated malfunction. Despite the critical role of knee motors in the robot's balance, our approach is able to dynamically adjust and stabilize, demonstrating its ability to adapt in-context. 85
6.7 **Emergent arm swing.** We find that our Transformer-based controller leads to emergent human-like arm swing behaviors in coordination with leg movements. 86
6.8 **Arm swing analysis.** The learned humanoid locomotion in our experiments exhibits human-like arm swing behaviors in coordination with leg movements, i.e., a contralateral relationship between the arms and the legs. 86

List of Tables

2.1 **Model definitions**: *Left*: Our ViT$_P$ models at various complexities, which use the original *patchify* stem and closely resemble the original ViT models. To facilitate comparisons with CNNs, we modify the original ViT-Tiny, -Small, -Base, -Large models to obtain models at 1GF, 4GF, 18GF, and 36GF, respectively. The modifications are indicated in blue and include reducing the MLP multiplier from 4× to 3× for the 1GF and 4GF models, and reducing the number of transformer blocks from 24 to 14 for the 36GF model. *Right*: Our ViT$_C$ models at various complexities that use the *convolutional* stem. The only additional modification relative to the corresponding ViT$_P$ models is the removal of 1 transformer block to compensate for the increased flops of the convolutional stem. We show complexity measures for all models (flops, parameters, activations, and epoch training time on ImageNet-1k); the corresponding ViT$_P$ and ViT$_C$ models match closely on all metrics. 8

2.2 **Peak performance (grouped by model family)**: Model complexity and validation top-1 error at 100, 200, and 400 epoch schedules on ImageNet-1k, and the top-1 error after pretraining on ImageNet-21k (IN 21k) and fine-tuning on ImageNet-1k. This table serves as reference for the results shown in Figure 2.6. Blue numbers: best model trainable under 20 minutes per ImageNet-1k epoch. Batch sizes and training times are reported normalized to 8 32GB Volta GPUs (see Appendix). Additional results on the ImageNet-V2 test set are presented in the Appendix. . . . 16

2.3 **Stem designs**: We compare ViT's standard patchify stem (P) and our convolutional stem (C) to four alternatives ($S1$ - $S4$) that each include a *patchify layer*, *i.e.*, a convolution with kernel size (> 1) equal to stride (highlighted in blue). Results use 50 epoch training, 4GF model size, and optimal *lr* and *wd* values for all models. We observe that increasing the pixel size of the patchify layer ($S1$ - $S4$) systematically degrades both top-1 error and optimizer stability (Δ) relative to C. 17

2.4 **Learning rate and weight decay used in §2.5**: *Left*: Per-model *lr* and *wd* values used for the experiments in §2.5.1 and §2.5.2, optimized for ImageNet-1k at 50 epochs. *Right*: Per-model *lr* and *wd* ranges used for the experiments in §2.5.3. Note that for our final experiments in §2.6, we constrained the *lr* and *wd* values further, using a single setting for all CNN models, and just two settings for all ViT models. We recommend using this simplified set of values in §2.6 when comparing models for fair and easily reproducible comparisons. All *lr* values are normalized w.r.t. a minibatch size of 2048. 21

2.5 **Ablation of data augmentation and regularization**: We use the *lr* and *wd* from Table 2.4 (left), except for ViT$_P$-18GF models with RandAugment which benefit from stronger *wd* (we increase *wd* to 0.5). Original DeiT ablation results are copied for reference in gray (*last column*); these use a *lr*/*wd* of 1e−3/0.05 (*lr* normalized to minibatch size 2048), which leads to some training failures (we note our *wd* is 5-10× higher). Our default training setup (*first row* in each set) uses AutoAugment, mixup, CutMix, label smoothing, and model EMA. Compared to the DeiT setup (*second row* in each set), we do not use erasing, stochastic depth, or repeating. Although our setup is equally effective, it is simpler and also converges much faster (see Figure 2.10). 22

3.1 **Classification accuracy on 4-class rotation and IN-100** under linear evaluation protocol. Adding rotation augmentation into baseline MoCo significantly reduces its capacity to classify rotation angles while downgrades its performance on IN-100. In contrast, our method better leverages the information gain of the new augmentation. 30

3.2 **Evaluation on multiple downstream tasks**. Our method demonstrates superior generalizability and transferability with increasing number of augmentations. 31

3.3 **Evaluation on datasets of real-world corruptions.** Rotation augmentation is beneficial for ON-13, and texture augmentation if beneficial for IN-C-100. 33

3.4 **Comparisons of concatenating features from different embedding spaces in LooC++** jointly trained on color, rotation and texture augmentations. Downstream tasks show nonidentical preferences for augmentation-dependent or invariant representations. 33

3.5 **Comparisons of LooC vs. MoCo** trained with all augmentations. 35

4.1 **Existing benchmarks:** Compared to existing benchmarks, ours features a unique combination of hand-designed tasks, dense rewards, and complex robots (e.g., multi-finger hands). Crucially, it leverages a fast simulator and provides distributed training for scaling learning-based motor control from pixel observations. 41

5.1 **Comparison to concurrent work.** All of our vision models, trained with image-only self-supervision, considerably outperform the strongest available R3M model trained on paired video-language labels from Ego4D. The gains are larger for larger models. Evaluated on the PickFruit task. . 62

6.1 **Domain randomization.** We show the types of randomization and their corresponding ranges. Additive randomization adds a value drawn from a specified range to the original value. Scaling randomization multiplies the original value by a value drawn from a specified range. The range is specified as a lower and upper bound for a uniform distribution or as a mean and std for a Gaussian distribution. 76

6.2 **Input commands**. We independently generate commands for forward/backward walking, sideway walking, and turning. The input is set to zero if the sampled value falls below the specified threshold. The commands are re-sampled periodically at fixed intervals. 77

6.3 **Observation and state spaces.** The state policy's actor and the critics of both policies utilize states as input. 78

6.4 **Hyperparameters of PPO.** 79

6.5 **Evaluation results in simulation.** We compare our approach to an MLP baseline, a CNN baseline, an LSTM baseline, and the company controller developed by Agility robotics. We observe that our approach outperforms the neural network baselines considerably and shows respectable performance compared to the state-of-the-art company controller. 80

Acknowledgments

My time at Berkeley has been marked by the lovely Berkeley Hills and the beautiful Cal campus with enriching academic environment. As I reach the culmination of my PhD journey, I am filled with a deep sense of gratitude for the individuals who have supported, guided, and accompanied me throughout this adventure.

I would like to express my deepest appreciation to my PhD advisor, Trevor Darrell. Trevor has been an incredible mentor throughout this journey. He has granted me the freedom to pursue ideas, while his encouragement has motivated me to push the boundaries of knowledge. Trevor's mentorship has been instrumental in my growth over the past four years, and I am truly grateful for the opportunity to learn from him.

I would also like to extend my gratitude to my committee members, Anca Dragan, Jitendra Malik, and Ross Girshick, for their thoughtful feedback and valuable insights, which have greatly contributed to the improvement of my thesis work.

I am grateful for the collaboration I shared with my closest collaborator, Ilija Radosavovic, whose expertise and friendship have enriched many works in this thesis. I have been fortunate to share this journey with my (other) lab mates, Tim Brooks, Bill Peebles, Yu Sun, Haozhi Qi, and many others. Their camaraderie and support have been invaluable, and I have learned much from each one of them.

My heartfelt thanks go to the experts I have had the privilege of working alongside, including Ross Girshick, Piotr Dollár, Jitendra Malik, Alyosha Efros, Pieter Abbeel, Koushil Sreenath, and Kurt Keutzer, as well as my collaborators, Ilija Radosavovic, Xiaolong Wang, Colorado Reed, Bike Zhang, Alex Kirillov, Stephen James, Eric Minton, Hanzi Mao, Nikhila Ravi, and Lerrel Pinto. Their expertise have significantly shaped my academic development. I am eternally grateful to my past mentors, especially the late Jian Sun, whose teachings continue to guide me.

A special mention goes to the wonderful staff at the Berkeley AI Research Lab. Their dedication and hard work have supported my research and activities a lot.

I cherish the support and connection I have with Yingcheng Liu, Huiwen Chang, Hexiang Hu, and Haoyue Shi, as well as my college friends Borui Jiang and Ruixuan Luo. Our shared experiences and memories are among the most priceless treasures

of my life. I am also fortunate to have my closest friends Junjie Yu and Ziqi Lu, who have been by my side through life's many ups and downs. Their unwavering support has been a pillar of strength throughout this journey.

Finally, I would like to express my sincere gratitude to my parents, for their unconditional love and support throughout my life, and for giving me the courage to pursue my goals. Their influence has shaped me into the person I am today, and I cannot thank them enough for their selflessness and dedication.

To all those who have contributed to this journey, both named and unnamed, I am deeply grateful. Your influence has shaped my academic trajectory and will remain an integral part of my story and life.

Chapter 1

Introduction

In 2012, the landscape of artificial intelligence (AI) experienced a groundbreaking moment with the advent of AlexNet [1]. Utilizing the power of high-performing GPU accelerators, AlexNet significantly reduced the top-5 error in the ImageNet Large Scale Visual Recognition Challenge [2] to 15.3%, a remarkable improvement of over 10% compared to the previous year's best result of 26.1%. This achievement fueled rapid advancements in AI, and within a mere three years, researchers designed increasingly sophisticated neural networks that leveraged enhanced computational power to achieve even lower error rates of 11.2%, 6.7%, and 3.6%, surpassing human-level performance (5%) [3].

One key aspect of these developments is the concept of task-specific representation learning. Models are trained on fixed datasets and excel at tasks within the scope of these datasets. However, they often struggle to generalize beyond the confines of the training data [4, 5], which poses challenges for their applicability to real-world problems, particularly in unstructured environments like those encountered by robots in everyday situations. Thus, the task-specific representation learning paradigm faces *scalability* limitations.

A major milestone in the AI domain came with the introduction of GPT-3 [6], a revolutionary generative pre-trained transformer model that significantly expanded the frontier of natural language processing (NLP). GPT-3 achieved exceptional performance across a wide range of NLP tasks through three critical factors: (1) high-capacity transformer models [7], (2) self-supervised objectives that do not overly rely on human labels [8, 9], and (3) the use of diverse, real-world data from the Internet [8,9]. These factors emphasize the importance of integrating the right model, learning objective, and data to build scalable systems [10].

In light of these advances, this thesis aims to extend the scalable paradigm to the fields of computer vision and robotics, ultimately pursuing the development

of general AI systems. We explore the three key components of creating scalable systems, including model optimizability, learning objectives, and large-scale datasets, and applying these outcomes for robotics.

In Chapter 2, we examine the optimizability of vision transformers (ViT) [11], a novel family of models derived from the high-capacity transformer models used in NLP. We propose a new set of optimizability metrics to analyze these high-capacity models and suggest an alternative design for their patchify stem, which significantly enhances optimization stability and peak performance.

Chapter 3 introduces a contrastive self-supervised learning objective that does not presume knowledge of specific, task-dependent invariances, reducing inductive biases in self-supervised learning. By constructing separate embedding spaces for varying and invariant factors, our model captures information across each augmentation and demonstrates superior performance across various datasets.

In Chapter 4, we showcase the effectiveness of self-supervised visual pre-training from real-world images for learning motor control tasks from pixels. We first train the visual representations using masked modeling of natural images. Subsequently, we freeze the visual encoder and train neural network controllers on top with reinforcement learning. Without performing any task-specific fine-tuning of the encoder, the same visual representations are used for all motor control tasks. To the best of our knowledge, this is the first self-supervised model to leverage real-world images at scale for motor control. Our results consistently outperform supervised baselines, sometimes even matching the oracle state performance.

Chapter 5 extends the work of the previous chapter, exploring self-supervised visual pre-training on images from diverse, in-the-wild videos for real-world robotic tasks. We demonstrate the effectiveness of pre-trained representations across a range of real-world robotic tasks and embodiments. Our encoder consistently outperforms other approaches, including CLIP [12], supervised ImageNet pre-training, and training from scratch. We also highlight the benefits of scaling visual pre-training for robot learning.

In Chapter 6, we present a sim-to-real learning-based approach for real-world humanoid locomotion using a causal Transformer trained by autoregressive prediction of future actions from the history of observations and actions. Our controller is trained on an ensemble of randomized environments in simulation and successfully deployed to the real world zero-shot, marking the first fully learning-based method for real-world full-sized humanoid locomotion.

Lastly, in Chapter 7, we conclude this thesis and discuss potential future directions for further research in the field.

Chapter 2

Optimizability of Scalable Models: A Study of Vision Transformers

2.1 Introduction

Vision transformer (ViT) models [11] offer an alternative design paradigm to convolutional neural networks (CNNs) [13]. ViTs replace the inductive bias towards local processing inherent in convolutions with global processing performed by multi-headed self-attention [7]. The hope is that this design has the potential to improve performance on vision tasks, akin to the trends observed in natural language processing [9]. While investigating this conjecture, researchers face another unexpected difference between ViTs and CNNs: ViT models exhibit substandard *optimizability*. ViTs are sensitive to the choice of optimizer [14] (AdamW [15] *vs*. SGD), to the selection of dataset specific learning hyperparameters [11, 14], to training schedule length, to network depth [16], *etc*. These issues render former training recipes and intuitions ineffective and impede research.

Convolutional neural networks, in contrast, are exceptionally easy and robust to optimize. Simple training recipes based on SGD, basic data augmentation, and standard hyperparameter values have been widely used for years [17]. Why does this difference exist between ViT and CNN models? In this paper we hypothesize that the issues lies primarily in the *early* visual processing performed by ViT. ViT "patchifies" the input image into $p \times p$ non-overlapping patches to form the transformer encoder's input set. This *patchify stem* is implemented as a stride-p $p \times p$ convolution, with $p = 16$ as a default value. This large-kernel plus large-stride convolution runs counter to the typical design choices used in CNNs, where best-practices have converged to a small stack of stride-two 3×3 kernels as the network's stem (*e.g.*, [18–20]).

To test this hypothesis, we *minimally* change the early visual processing of ViT

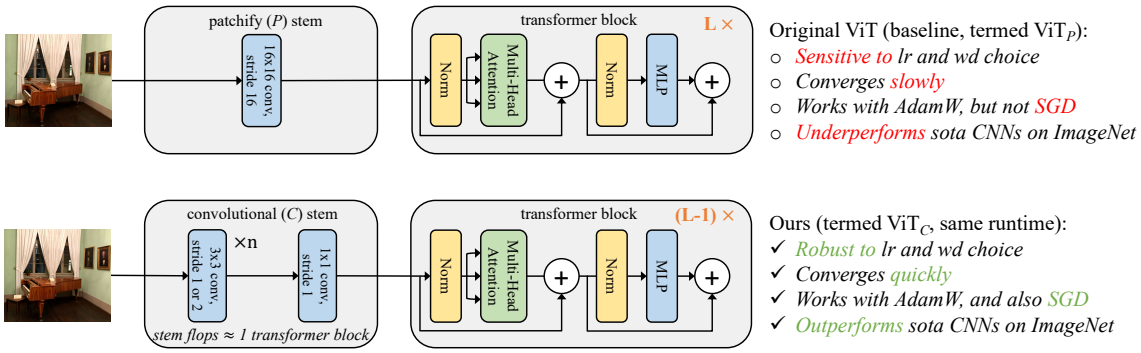


Figure 2.1: **Early convolutions help transformers see better**: We hypothesize that the substandard optimizability of ViT models compared to CNNs primarily arises from the *early* visual processing performed by its *patchify stem*, which is implemented by a non-overlapping stride-p $p \times p$ convolution, with $p = 16$ by default. We *minimally* replace the patchify stem in ViT with a standard *convolutional stem* of only ~5 convolutions that has approximately the same complexity as a *single* transformer block. We reduce the number of transformer blocks by one (*i.e.*, $L - 1$ *vs.* L) to maintain parity in flops, parameters, and runtime. We refer to the resulting model as ViT$_C$ and the original ViT as ViT$_P$. The vast majority of computation performed by these two models is identical, yet surprisingly we observe that ViT$_C$ (i) converges faster, (ii) enables, for the first time, the use of either AdamW or SGD without a significant accuracy drop, (iii) shows greater stability to learning rate and weight decay choice, and (iv) yields improvements in ImageNet top-1 error allowing ViT$_C$ to outperform state-of-the-art CNNs, whereas ViT$_P$ does not.

by replacing its patchify stem with a standard *convolutional stem* consisting of only ~5 convolutions, see Figure 2.1. To compensate for the small addition in flops, we remove one transformer block to maintain parity in flops and runtime. We observe that even though the vast majority of the computation in the two ViT designs is identical, this small change in early visual processing results in markedly different training behavior in terms of the sensitivity to optimization settings as well as the final model accuracy.

In extensive experiments we show that replacing the ViT patchify stem with a more standard convolutional stem (i) allows ViT to converge faster (§2.5.1), (ii) enables, for the first time, the use of either AdamW or SGD without a significant drop in accuracy (§2.5.2), (iii) brings ViT's stability w.r.t. learning rate and weight decay closer to that of modern CNNs (§2.5.3), and (iv) yields improvements in ImageNet [2] top-1 error of ~1-2 percentage points (§2.6). We consistently observe these improvements across a wide spectrum of model complexities (from 1G flops to 36G flops) and dataset scales (ImageNet-1k to ImageNet-21k).

These results show that injecting some convolutional inductive bias into ViTs can

be beneficial under commonly studied settings. We did *not* observe evidence that the hard locality constraint in early layers hampers the representational capacity of the network, as might be feared [21]. In fact we observed the opposite, as ImageNet results improve even with larger-scale models and larger-scale data when using a convolution stem. Moreover, under carefully controlled comparisons, we find that ViTs are only able to surpass state-of-the-art CNNs when equipped with a convolutional stem (§2.6).

We conjecture that restricting convolutions in ViT to *early* visual processing may be a crucial design choice that strikes a balance between (hard) inductive biases and the representation learning ability of transformer blocks. Evidence comes by comparison to the "hybrid ViT" presented in [11], which uses 40 convolutional layers (most of a ResNet-50) and shows no improvement over the default ViT. This perspective resonates with the findings of [21], who observe that early transformer blocks prefer to learn more local attention patterns than later blocks. Finally we note that exploring the design of hybrid CNN/ViT models is *not* a goal of this work; rather we demonstrate that simply using a minimal convolutional stem with ViT is sufficient to dramatically change its optimization behavior.

In summary, the findings presented in this paper lead us to recommend using a standard, lightweight convolutional stem for ViT models in the analyzed dataset scale and model complexity spectrum as a more robust and higher performing architectural choice compared to the original ViT model design.

2.2 Related Work

Convolutional neural networks (CNNs). The breakthrough performance of the AlexNet [1] CNN [13, 22] on ImageNet classification [2] transformed the field of recognition, leading to the development of higher performing architectures, *e.g.*, [17, 18, 23, 24], and scalable training methods [25, 26]. These architectures are now core components in object detection (*e.g.*, [27]), instance segmentation (*e.g.*, [28]), and semantic segmentation (*e.g.*, [29]). CNNs are typically trained with stochastic gradient descent (SGD) and are widely considered to be easy to optimize.

Self-attention in vision models. Transformers [7] are revolutionizing natural language processing by enabling scalable training. Transformers use multi-headed self-attention, which performs global information processing and is strictly more general than convolution [30]. Wang *et al.* [31] show that (single-headed) self-attention is a form of non-local means [32] and that integrating it into a ResNet [17] improves several tasks. Ramachandran *et al.* [33] explore this direction further with stand-alone self-attention networks for vision. They report difficulties in designing an attention-based

network stem and present a bespoke solution that avoids convolutions. In contrast, we demonstrate the benefits of a convolutional stem. Zhao *et al.* [34] explore a broader set of self-attention operations with hard-coded locality constraints, more similar to standard CNNs.

Vision transformer (ViT). Dosovitskiy *et al.* [11] apply a transformer encoder to image classification with minimal vision-specific modifications. As the counterpart of input token embeddings, they partition the input image into, *e.g.*, 16×16 pixel, non-overlapping patches and linearly project them to the encoder's input dimension. They report lackluster results when training on ImageNet-1k, but demonstrate state-of-the-art transfer learning when using large-scale pretraining data. ViTs are sensitive to many details of the training recipe, *e.g.*, they benefit greatly from AdamW [15] compared to SGD and require careful learning rate and weight decay selection. ViTs are generally considered to be difficult to optimize compared to CNNs (*e.g.*, see [11, 14, 16]). Further evidence of challenges comes from Chen *et al.* [35] who report ViT optimization instability in self-supervised learning (unlike with CNNs), and find that freezing the patchify stem at its random initialization improves stability.

ViT improvements. ViTs are gaining rapid interest in part because they may offer a novel direction away from CNNs. Touvron *et al.* [14] show that with more regularization and stronger data augmentation ViT models achieve competitive accuracy on ImageNet-1k alone (*cf.* [11]). Subsequently, works concurrent with our own explore numerous other ViT improvements. Dominant themes include multi-scale networks [36–40], increasing depth [16], and locality priors [21, 36, 41–43]. In [21], d'Ascoli *et al.* modify multi-head self-attention with a convolutional bias at initialization and show that this prior improves sample efficiency and ImageNet accuracy. Resonating with our work, [36, 41–43] present models with convolutional stems, but do not analyze optimizability (our focus).

Discussion. Unlike the concurrent work on locality priors in ViT, our focus is studying *optimizability* under *minimal* ViT modifications in order to derive crisp conclusions. Our perspective brings several novel observations: by adding only ~5 convolutions to the stem, ViT can be optimized well with either AdamW or SGD (*cf.* all prior works use AdamW to avoid large drops in accuracy [14]), it becomes less sensitive to the specific choice of learning rate and weight decay, and training converges faster. We also observe a consistent improvement in ImageNet top-1 accuracy across a wide spectrum of model complexities (1G flops to 36G flops) and dataset scales (ImageNet-1k to ImageNet-21k). These results suggest that a (hard) convolutional bias early in the network does not compromise representational capacity, as conjectured in [21], and is beneficial within the scope of this study.

2.3 Vision Transformer Architectures

Next, we review vision transformers [11] and describe the convolutional stems used in our work.

The vision transformer (ViT). ViT first partitions an input image into *non-overlapping* $p \times p$ patches and linearly projects each patch to a d-dimensional feature vector using a learned weight matrix. A patch size of $p = 16$ and an image size of 224×224 are typical. The resulting patch embeddings (plus positional embeddings and a learned classification token embedding) are processed by a standard transformer encoder [7, 44] followed by a classification head. Using common network nomenclature, we refer to the portion of ViT before the transformer blocks as the network's *stem*. ViT's stem is a specific case of convolution (stride-p, $p \times p$ kernel), but we will refer to it as the *patchify stem* and reserve the terminology of *convolutional stem* for stems with a more conventional CNN design with multiple layers of *overlapping* convolutions (*i.e.*, with stride smaller than the kernel size).

ViT$_P$ models. Prior work proposes ViT models of various sizes, such as ViT-Tiny, ViT-Small, ViT-Base, *etc.* [11, 14]. To facilitate comparisons with CNNs, which are typically standardized to 1 gigaflop (GF), 2GF, 4GF, 8GF, *etc.*, we modify the original ViT models to obtain models at about these complexities. Details are given in Table 2.1 (left). For easier comparison with CNNs of similar flops, and to avoid subjective size names, we refer the models by their flops, *e.g.*, ViT$_P$-4GF in place of ViT-Small. We use the P subscript to indicate that these models use the original *patchify* stem.

Convolutional stem design. We adopt a typical minimalist convolutional stem design by stacking 3×3 convolutions [18], followed by a single 1×1 convolution at the end to match the d-dimensional input of the transformer encoder. These stems quickly downsample a 224×224 input image using overlapping strided convolutions to 14×14, matching the number of inputs created by the standard patchify stem. We follow a simple design pattern: all 3×3 convolutions either have stride 2 and double the number of output channels or stride 1 and keep the number of output channels constant. We enforce that the stem accounts for approximately the computation of one transformer block of the corresponding model so that we can easily control for flops by removing one transformer block when using the convolutional stem instead of the patchify stem. Our stem design was chosen to be purposefully simple and we emphasize that it was not designed to maximize model accuracy.

ViT$_C$ models. To form a ViT model with a convolutional stem, we simply replace the patchify stem with its counterpart convolutional stem and *remove one transformer block* to compensate for the convolutional stem's extra flops (see Figure 2.1). We

model	ref model	hidden size	MLP mult	num heads	num blocks	flops (B)	params (M)	acts (M)	time (min)
ViT_P-1GF	~ViT-T	192	3	3	12	1.1	4.8	5.5	2.6
ViT_P-4GF	~ViT-S	384	3	6	12	3.9	18.5	11.1	3.8
ViT_P-18GF	=ViT-B	768	4	12	12	17.5	86.7	24.0	11.5
ViT_P-36GF	$\frac{3}{5}$ViT-L	1024	4	16	14	35.9	178.4	37.3	18.8

model	hidden size	MLP mult	num heads	num blocks	flops (B)	params (M)	acts (M)	time (min)
ViT_C-1GF	192	3	3	11	1.1	4.6	5.7	2.7
ViT_C-4GF	384	3	6	11	4.0	17.8	11.3	3.9
ViT_C-18GF	768	4	12	11	17.7	81.6	24.1	11.4
ViT_C-36GF	1024	4	16	13	35.0	167.8	36.7	18.6

Table 2.1: **Model definitions**: *Left*: Our ViT_P models at various complexities, which use the original *patchify* stem and closely resemble the original ViT models. To facilitate comparisons with CNNs, we modify the original ViT-Tiny, -Small, -Base, -Large models to obtain models at 1GF, 4GF, 18GF, and 36GF, respectively. The modifications are indicated in blue and include reducing the MLP multiplier from 4× to 3× for the 1GF and 4GF models, and reducing the number of transformer blocks from 24 to 14 for the 36GF model. *Right*: Our ViT_C models at various complexities that use the *convolutional* stem. The only additional modification relative to the corresponding ViT_P models is the removal of 1 transformer block to compensate for the increased flops of the convolutional stem. We show complexity measures for all models (flops, parameters, activations, and epoch training time on ImageNet-1k); the corresponding ViT_P and ViT_C models match closely on all metrics.

refer to the modified ViT with a convolutional stem as ViT_C. Configurations for ViT_C at various complexities are given in Table 2.1 (right); corresponding ViT_P and ViT_C models match closely on all complexity metrics including flops and runtime.

Convolutional stem details. Our convolutional stem designs use four, four, and six 3×3 convolutions for the 1GF, 4GF, and 18GF models, respectively. The output channels are [24, 48, 96, 192], [48, 96, 192, 384], and [64, 128, 128, 256, 256, 512], respectively. All 3×3 convolutions are followed by batch norm (BN) [25] and then ReLU [45], while the final 1×1 convolution is not, to be consistent with the original patchify stem. Eventually, matching stem flops to transformer block flops results in an unreasonably large stem, thus ViT_C-36GF uses the same stem as ViT_C-18GF.

Convolutions in ViT. Dosovitskiy *et al.* [11] also introduced a "hybrid ViT" architecture that blends a modified ResNet [17] (BiT-ResNet [46]) with a transformer encoder. In their hybrid model, the patchify stem is replaced by a partial BiT-ResNet-50 that terminates at the output of the conv4 stage or the output of an extended conv3 stage. These image embeddings replace the standard patchify stem embeddings. This partial BiT-ResNet-50 stem is *deep*, with 40 convolutional layers. In this chapter, we explore *lightweight* convolutional stems that consist of only 5 to 7 convolutions in total, instead of the 40 used by the hybrid ViT. Moreover, we emphasize that the goal of our work is *not* to explore the hybrid ViT design space, but rather to study the optimizability effects of simply replacing the patchify stem with a *minimal* convolutional stem that follows standard CNN design practices.

2.4 Measuring Optimizability

It has been noted in the literature that ViT models are challenging to optimize, *e.g.*, they may achieve only modest performance when trained on a mid-size dataset (ImageNet-1k) [11], are sensitive to data augmentation [14] and optimizer choice [14], and may perform poorly when made deeper [16]. We empirically observed the general presence of such difficulties through the course of our experiments and informally refer to such optimization characteristics collectively as *optimizability*.

Models with poor optimizability can yield very different results when hyperparameters are varied, which can lead to seemingly bizarre observations, *e.g.*, removing *erasing* data augmentation [47] causes a catastrophic drop in ImageNet accuracy in [14]. Quantitative metrics to measure optimizability are needed to allow for more robust comparisons. In this section, we establish the foundations of such comparisons; we extensively test various models using these optimizability measures in §2.5.

Training length stability. Prior works train ViT models for lengthy schedules, *e.g.*, 300 to 400 epochs on ImageNet is typical (at the extreme, [36] trains models for 1000 epochs), since results at a formerly common 100-epoch schedule are substantially worse (2-4% lower top-1 accuracy, see §2.5.1). In the context of ImageNet, we define top-1 accuracy at 400 epochs as an approximate asymptotic result, *i.e.*, training for longer will not meaningfully improve top-1 accuracy, and we compare it to the accuracy of models trained for only 50, 100, or 200 epochs. We define *training length stability* as the gap to asymptotic accuracy. Intuitively, it's a measure of convergence speed. Models that converge faster offer obvious practical benefits, especially when training many model variants.

Optimizer stability. Prior works use AdamW [15] to optimize ViT models from random initialization. Results of SGD are not typically presented and we are only aware of Touvron *et al.* [14]'s report of a dramatic ~7% drop in ImageNet top-1 accuracy. In contrast, widely used CNNs, such as ResNets, can be optimized equally well with either SGD or AdamW (see §2.5.2) and SGD (always with momentum) is typically used in practice. SGD has the practical benefit of having fewer hyperparameters (*e.g.*, tuning AdamW's β_2 can be important [48]) and requiring 50% less optimizer state memory, which can ease scaling. We define *optimizer stability* as the accuracy gap between AdamW and SGD. Like training length stability, we use optimizer stability as a proxy for the ease of optimization of a model.

Hyperparameter (*lr*, *wd*) stability. Learning rate (lr) and weight decay (wd) are among the most important hyperparameters governing optimization with SGD and AdamW. New models and datasets often require a search for their optimal values as the choice can dramatically affect results. It is desirable to have a model and optimizer

that yield good results for a wide range of learning rate and weight decay values. We will explore this *hyperparameter stability* by comparing the error distribution functions (EDFs) [20] of models trained with various choices of *lr* and *wd*. In this setting, to create an EDF for a model we randomly sample values of *lr* and *wd* and train the model accordingly. Distributional estimates, like those provided by EDFs, give a more complete view of the characteristics of models that point estimates cannot reveal [20, 49]. We will review EDFs in §2.5.3.

Peak performance. The maximum possible performance of each model is the most commonly used metric in previous literature and it is often provided without carefully controlling training details such as data augmentations, regularization methods, number of epochs, and *lr*, *wd* tuning. To make more robust comparisons, we define *peak performance* as the result of a model at 400 epochs using its best-performing optimizer and *parsimoniously* tuned *lr* and *wd* values (details in §2.6), *while fixing justifiably good values for all other variables that have a known impact on training*. Peak performance results for ViTs and CNNs under these carefully controlled training settings are presented in §2.6.

2.5 Stability Experiments

In this section we test the *stability* of ViT models with the original patchify (P) stem *vs*. the convolutional (C) stem defined in §2.3. For reference, we also train RegNetY [49, 50], a state-of-the-art CNN that is easy to optimize and serves as a reference point for good stability.

We conduct experiments using ImageNet-1k [2]'s standard training and validation sets, and report top-1 error. Following [50], for all results, we carefully control training settings and we use a minimal set of data augmentations that still yields strong results, for details see §2.5.4. In this section, unless noted, for each model we use the optimal *lr* and *wd* found under a 50 epoch schedule (see Appendix).

2.5.1 Training Length Stability

We first explore how rapidly networks converge to their asymptotic error on ImageNet-1k, *i.e.*, the highest possible accuracy achievable by training for many epochs. We approximate asymptotic error as a model's error using a 400 epoch schedule based on observing diminishing returns from 200 to 400. We consider a grid of 24 experiments for ViT: $\{P, C\}$ stems $\times$ $\{1, 4, 18\}$ GF model sizes $\times$ $\{50, 100, 200, 400\}$ epochs. For reference we also train RegNetY at $\{1, 4, 16\}$ GF. We use the best optimizer choice for each model (AdamW for ViT models and SGD for

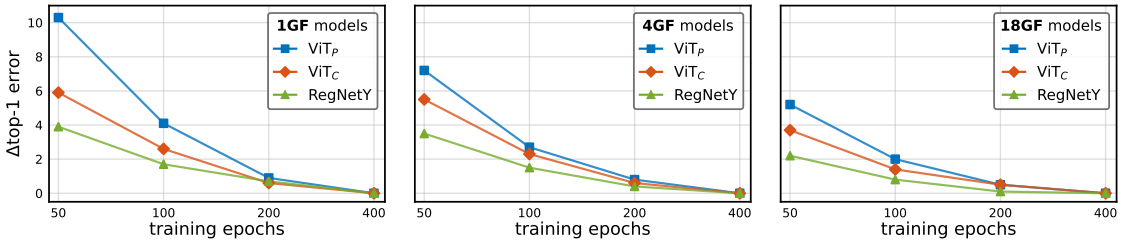


Figure 2.2: **Training length stability**: We train 9 models for 50 to 400 epochs on ImageNet-1k and plot the Δtop-1 error to the 400 epoch result for each. ViT$_C$ demonstrates faster convergence than ViT$_P$ across the model complexity spectrum, and helps close the gap to CNNs (represented by RegNetY).

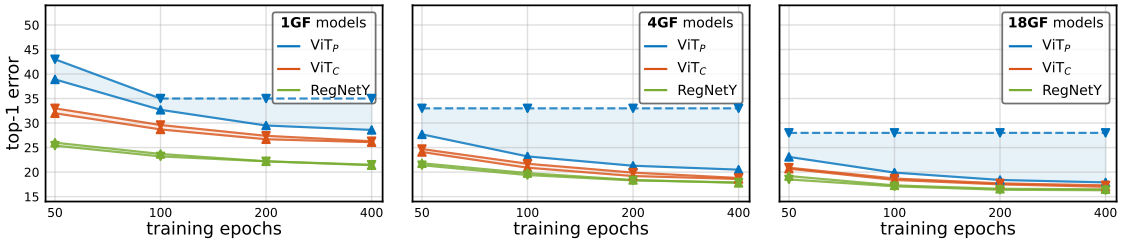


Figure 2.3: **Optimizer stability**: We train each model for 50 to 400 epochs with AdamW (upward triangle ▲) and SGD (downward triangle ▼). For the baseline ViT$_P$, SGD yields significantly worse results than AdamW. In contrast, ViT$_C$ and RegNetY models exhibit a much smaller gap between SGD and AdamW across all settings. Note that for long schedules, ViT$_P$ often fails to converge with SGD (*i.e.*, loss goes to NaN), in such cases we copy the best results from a shorter schedule of the same model (and show the results via a dashed line).

RegNetY models).

Results. Figure 2.2 shows the absolute error *deltas* (Δtop-1) between 50, 100, and 200 epoch schedules and asymptotic performance (at 400 epochs). ViT$_C$ demonstrates faster convergence than ViT$_P$ across the model complexity spectrum, and closes much of the gap to the rate of CNN convergence. The improvement is most significant in the shortest training schedule (50 epoch), *e.g.*, ViT$_P$-1GF has a 10% error delta, while ViT$_C$-1GF reduces this to about 6%. This opens the door to applications that execute a large number of short-scheduled experiments.

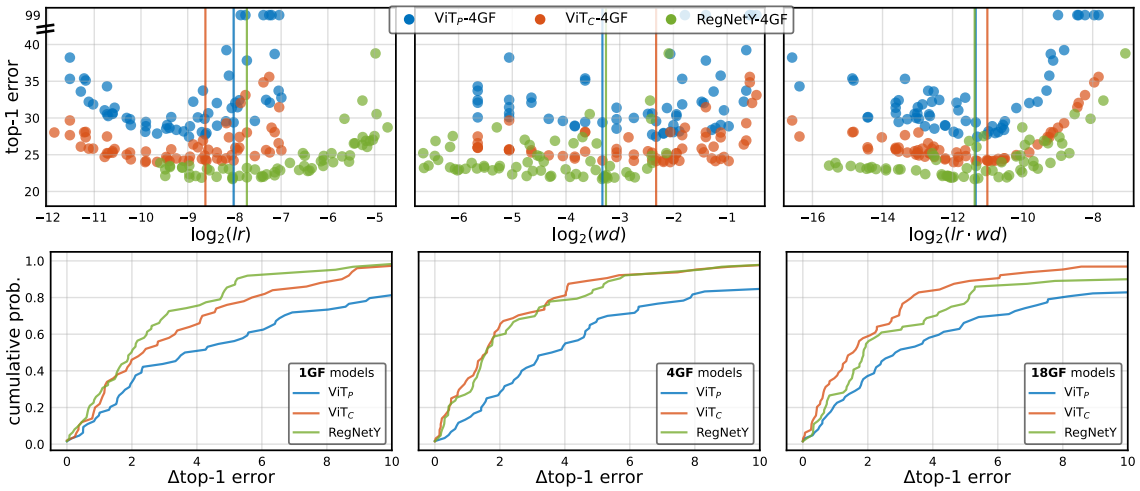


Figure 2.4: **Hyperparameter stability for AdamW (*lr* and *wd*)**: For each model, we train 64 instances of the model for 50 epochs each with a random *lr* and *wd* (in a fixed width interval around the optimal value for each model). *Top*: Scatterplots of the *lr*, *wd*, and *lr* ·*wd* for three 4GF models. Vertical bars indicate optimal *lr*, *wd*, and *lr* ·*wd* values for each model. *Bottom*: For each model, we generate an EDF of the errors by plotting the cumulative distribution of the Δtop-1 errors (Δ to the optimal error for each model). A steeper EDF indicates better stability to *lr* and *wd* variation. ViT$_C$ significantly improves the stability over the baseline ViT$_P$ across the model complexity spectrum, and matches or even outperforms the stability of the CNN model (RegNetY).

2.5.2 Optimizer Stability

We next explore how well AdamW and SGD optimize ViT models with the two stem types. We consider the following grid of 48 ViT experiments: $\{P, C\}$ stems $\times$ $\{1, 4, 18\}$ GF sizes $\times$ $\{50, 100, 200, 400\}$ epochs $\times$ $\{$AdamW, SGD$\}$ optimizers. As a reference, we also train 24 RegNetY baselines, one for each complexity regime, epoch length, and optimizer.

Results. Figure 2.3 shows the results. As a baseline, RegNetY models show virtually no gap when trained using either SGD or AdamW (the difference ~0.1-0.2% is within noise). On the other hand, *ViT$_P$ models suffer a dramatic drop when trained with SGD* across all settings (of up to 10% for larger models and longer training schedules). With a convolutional stem, ViT$_C$ models exhibit much smaller error gaps between SGD and AdamW across all training schedules and model complexities, including in larger models and longer schedules, where the gap is reduced to less than 0.2%. In other words, both RegNetY and ViT$_C$ can be easily trained via either SGD or AdamW, but ViT$_P$ cannot.

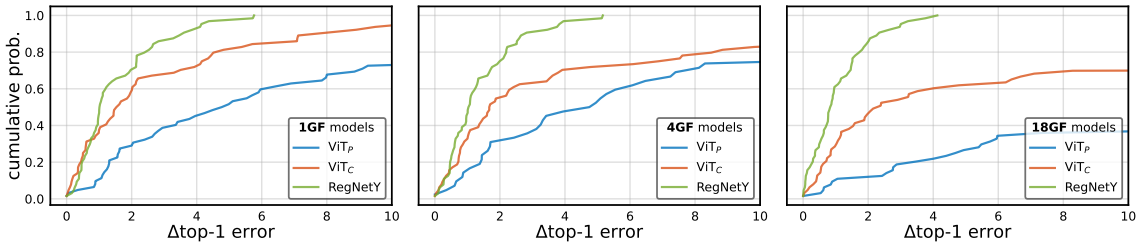


Figure 2.5: **Hyperparameter stability for SGD (*lr* and *wd*)**: We repeat the setup from Figure 2.4 using SGD instead of AdamW. The stability improvement of ViT$_C$ over the baseline ViT$_P$ is even larger than with AdamW. *E.g.*, ~60% of ViT$_C$-18GF models are within 4% Δtop-1 error of the best result, while less than 20% of ViT$_P$-18GF models are (in fact most ViT$_P$-18GF runs don't converge).

2.5.3 Learning Rate and Weight Decay Stability

Next, we characterize how sensitive different model families are to changes in learning rate (*lr*) and weight decay (*wd*) under both AdamW and SGD optimizers. To quantify this, we make use of error distribution functions (EDFs) [20]. An EDF is computed by sorting a set of results from low-to-high error and plotting the cumulative proportion of results as error increases, see [20] for details. In particular, we generate EDFs of a model as a function of *lr* and *wd*. The intuition is that if a model is robust to these hyperparameter choices, the EDF will be steep (all models will perform similarly), while if the model is sensitive, the EDF will be shallow (performance will be spread out).

We test 6 ViT models ($\{P, C\} \times \{1, 4, 18\}$ GF) and 3 RegNetY models ($\{1, 4, 16\}$ GF). For each model and each optimizer, we compute an EDF by randomly sampling 64 (*lr*, *wd*) pairs with learning rate and weight decay sampled in a fixed width interval around their optimal values for that model and optimizer (see the Appendix for sampling details). Rather than plotting absolute error in the EDF, we plot Δtop-1 error between the best result (obtained with the optimal *lr* and *wd*) and the observed result. Due to the large number of models, we train each for 50 epochs.

Results. Figure 2.4 shows scatterplots and EDFs for models trained by AdamW. Figure 2.5 shows SGD results. In all cases we see that ViT$_C$ significantly improves the *lr* and *wd* stability over ViT$_P$ for both optimizers. This indicates that the *lr* and *wd* are easier to optimize for ViT$_C$ than for ViT$_P$.

2.5.4 Experimental Details

In all experiments we train with a single half-period cosine learning rate decay schedule with a 5-epoch linear learning rate warm-up [26]. We use a minibatch

size of 2048. Crucially, weight decay is *not* applied to the gain factors found in normalization layers nor to bias parameters anywhere in the model; we found that decaying these parameters can dramatically reduce top-1 accuracy for small models and short schedules. For inference, we use an exponential moving average (EMA) of the model weights (*e.g.*, [51]). The *lr* and *wd* used in this section are reported in the Appendix. Other hyperparameters use defaults: SGD momentum is 0.9 and AdamW's $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

Regularization and data augmentation. We use a simplified training recipe compared to recent work such as DeiT [14], which we found to be equally effective across a wide spectrum of model complexities and dataset scales. We use AutoAugment [52], mixup [53] ($\alpha = 0.8$), CutMix [54] ($\alpha = 1.0$), and label smoothing [55] ($\epsilon = 0.1$). We prefer this setup because it is similar to common settings for CNNs (*e.g.*, [50]) except for stronger mixup and the addition of CutMix (ViTs benefit from both, while CNNs are not harmed). We compare this recipe to the one used for DeiT models in the Appendix, and observe that *our setup provides substantially faster training convergence* likely because we remove repeating augmentation [56, 57], which is known to slow training [56].

2.6 Peak Performance

A model's peak performance is the most commonly used metric in network design. It represents what is possible with the best-known-so-far settings and naturally evolves over time. Making fair comparisons between different models is desirable but fraught with difficulty. Simply citing results from prior work may be negatively biased against that work as it was unable to incorporate newer, yet applicable improvements. Here, we strive to provide a *fairer comparison* between state-of-the-art CNNs, ViT$_P$, and ViT$_C$. We identify a set of factors and then strike a pragmatic balance between which subset to optimize for each model *vs.* which subset share a constant value across all models.

In our comparison, all models share the same epochs (400), use of model weight EMA, and set of regularization and augmentation methods (as specified in §2.5.4). All CNNs are trained with SGD with *lr* of 2.54 and *wd* of 2.4e−5; we found this single choice worked well across all models, as similarly observed in [50]. For all ViT models we found AdamW with a *lr*/*wd* of 1.0e−3/0.24 was effective, except for the 36GF models. For these larger models we tested a few settings and found a *lr*/*wd* of 6.0e−4/0.28 to be more effective for both ViT$_P$-36GF and ViT$_C$-36GF models. For training and inference, ViTs use 224×224 resolution (we do *not* fine-tune at higher resolutions), while the CNNs use (often larger) optimized resolutions

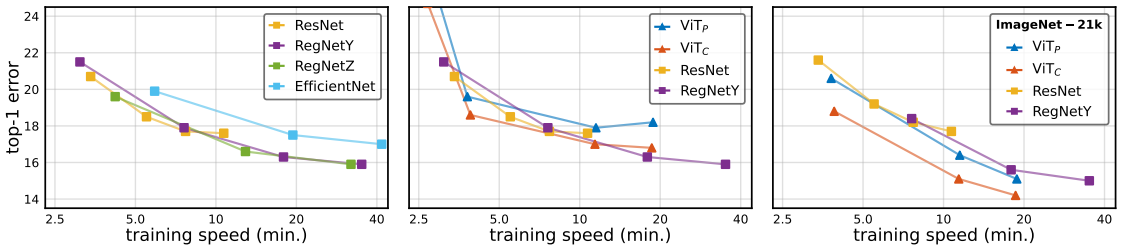


Figure 2.6: **Peak performance (epoch training time *vs.* ImageNet-1k val top-1 error)**: Results of a fair, controlled comparison of ViT$_P$, ViT$_C$, and CNNs. Each curve corresponds to a model complexity sweep resulting in a training speed spectrum (minutes per ImageNet-1k epoch). *Left:* State-of-the-art CNNs. Equipped with a modern training recipe, ResNets are highly competitive in the faster regime, while RegNetY and Z perform similarly, and better than EfficientNets. *Middle:* Selected CNNs compared to ViTs. With access to only ImageNet-1k training data, RegNetY *and* ResNet outperform ViT$_P$ across the board. ViT$_C$ is more competitive with CNNs. *Right:* Pretraining on ImageNet-21k improves the ViT models more than the CNNs, making ViT$_P$ competitive. Here, the proposed ViT$_C$ outperforms all other models across the full training speed spectrum.

specified in [19, 50]. Given this protocol, we compare ViT$_P$, ViT$_C$, and CNNs across a spectrum of model complexities (1GF to 36GF) and dataset scales (directly training on ImageNet-1k *vs.* pretraining on ImageNet-21k and then fine-tuning on ImageNet-1k).

Results. Figure 2.6 shows a progression of results. Each plot shows ImageNet-1k val top-1 error *vs.* ImageNet-1k epoch training time.[1] The left plot compares several state-of-the-art CNNs. RegNetY and RegNetZ [50] achieve similar results across the training speed spectrum and outperform EfficientNets [19]. Surprisingly, ResNets [17] are highly competitive at fast runtimes, showing that under a fairer comparison these years-old models perform substantially better than often reported (*cf*. [19]).

The middle plot compares two representative CNNs (ResNet and RegNetY) to ViTs, still using only ImageNet-1k training. The baseline ViT$_P$ underperforms RegNetY across the entire model complexity spectrum. To our surprise, *ViT$_P$ also underperforms ResNets* in this regime. ViT$_C$ is more competitive and outperforms CNNs in the middle-complexity range.

The right plot compares the same models but with ImageNet-21k pretraining (details in Appendix). In this setting ViT models demonstrates a greater capacity to benefit from the larger-scale data: now ViT$_C$ strictly outperforms both ViT$_P$

[1]We time models in PyTorch on 8 32GB Volta GPUs. We note that batch inference time is highly correlated with training time, but we report epoch time as it is easy to interpret and does not depend on the use case.

model	flops (B)	params (M)	acts (M)	time (min)	batch size	epochs 100	epochs 200	epochs 400	IN 21k
ResNet-50	4.1	25.6	11.3	3.4	2048	22.5	21.2	20.7	21.6
ResNet-101	7.8	44.5	16.4	5.5	2048	20.3	19.1	18.5	19.2
ResNet-152	11.5	60.2	22.8	7.7	2048	19.5	18.4	17.7	18.2
ResNet-200	15.0	64.7	32.3	10.7	1024	19.5	18.3	17.6	17.7
RegNetY-1GF	1.0	9.6	6.2	3.1	2048	23.2	22.2	21.5	-
RegNetY-4GF	4.1	22.4	14.5	7.6	2048	19.4	18.3	17.9	18.4
RegNetY-16GF	15.5	72.3	30.7	17.9	1024	17.1	16.4	16.3	15.6
RegNetY-32GF	31.1	128.6	46.2	35.1	512	16.2	15.9	15.9	15.0
RegNetZ-1GF	1.0	11.0	8.8	4.2	2048	20.8	20.2	19.6	-
RegNetZ-4GF	4.0	28.1	24.3	12.9	1024	17.4	16.9	16.6	-
RegNetZ-16GF	16.0	95.3	51.3	32.0	512	16.0	15.9	15.9	-
RegNetZ-32GF	32.0	175.1	79.6	55.3	256	16.3	16.2	16.1	-

model	flops (B)	params (M)	acts (M)	time (min)	batch size	epochs 100	epochs 200	epochs 400	IN 21k
EffNet-B2	1.0	9.1	13.8	5.9	2048	21.4	20.5	19.9	-
EffNet-B4	4.4	19.3	49.5	19.4	512	18.5	17.8	17.5	-
EffNet-B5	10.3	30.4	98.9	41.7	256	17.3	17.0	17.0	-
ViT$_P$-1GF	1.1	4.8	5.5	2.6	2048	33.2	29.7	27.7	-
ViT$_P$-4GF	3.9	18.5	11.1	3.8	2048	23.3	20.8	19.6	20.6
ViT$_P$-18GF	17.5	86.6	24.0	11.5	1024	19.9	18.4	17.9	16.4
ViT$_P$-36GF	35.9	178.4	37.3	18.8	512	19.9	18.8	18.2	15.1
ViT$_C$-1GF	1.1	4.6	5.7	2.7	2048	28.6	26.1	24.7	-
ViT$_C$-4GF	4.0	17.8	11.3	3.9	2048	20.9	19.2	18.6	18.8
ViT$_C$-18GF	17.7	81.6	24.1	11.4	1024	18.4	17.5	17.0	15.1
ViT$_C$-36GF	35.0	167.8	36.7	18.6	512	18.3	17.6	16.8	14.2

Table 2.2: **Peak performance (grouped by model family)**: Model complexity and validation top-1 error at 100, 200, and 400 epoch schedules on ImageNet-1k, and the top-1 error after pretraining on ImageNet-21k (IN 21k) and fine-tuning on ImageNet-1k. This table serves as reference for the results shown in Figure 2.6. Blue numbers: best model trainable under 20 minutes per ImageNet-1k epoch. Batch sizes and training times are reported normalized to 8 32GB Volta GPUs (see Appendix). Additional results on the ImageNet-V2 test set are presented in the Appendix.

and RegNetY. Interestingly, *the original ViT$_P$ does not outperform a state-of-the-art CNN* even when trained on this much larger dataset. Numerical results are presented in Table 2.2 for reference to exact values. This table also highlights that flop counts are not significantly correlated with runtime, but that activations are (see Appendix for more details), as also observed by [50]. *E.g.*, EfficientNets are slow relative to their flops while ViTs are fast.

These results verify that ViT$_C$'s convolutional stem improves not only optimization stability, as seen in the previous section, but also peak performance. Moreover, this benefit can be seen across the model complexity and dataset scale spectrum. Perhaps surprisingly, given the recent excitement over ViT, we find that ViT$_P$ struggles to compete with state-of-the-art CNNs. We only observe improvements over CNNs when using *both* large-scale pretraining data *and* the convolutional stem.

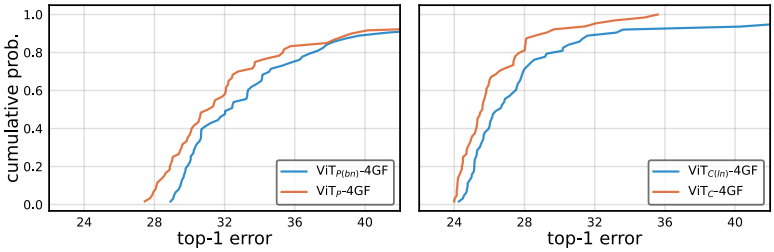
2.7 Additional Study and Experimental Details

Stem Design Ablation Experiments

ViT's patchify stem differs from the proposed convolutional stem in the type of convolution used and the use of normalization and a non-linear activation function. We investigate these factors next.

stem	kernel size	stride	padding	channels	flops (M)	params (M)	acts (M)	top-1 error AdamW	top-1 error SGD	Δ
P	[16]	[16]	[0]	[384]	58	0.3	0.8	27.7	33.0	5.3
C	[3, 3, 3, 3, 1]	[2, 2, 2, 2, 1]	[1, 1, 1, 1, 0]	[48, 96, 192, 384, 384]	435	1.0	1.2	**24.0**	**24.7**	**0.7**
$S1$	[3, 3, 3, 2, 1]	[2, 2, 2, 2, 1]	[1, 1, 1, 0, 0]	[42, 104, 208, 416, 384]	422	0.8	1.3	24.3	25.1	0.8
$S2$	[3, 3, 3, 4, 1]	[2, 2, 1, 4, 1]	[1, 1, 1, 0, 0]	[32, 64, 128, 256, 384]	422	0.7	1.1	24.3	25.3	1.0
$S3$	[3, 3, 3, 8, 1]	[2, 1, 1, 8, 1]	[1, 1, 1, 0, 0]	[17, 34, 68, 136, 384]	458	0.7	1.6	25.1	26.2	1.1
$S4$	[3, 3, 3, 16, 1]	[1, 1, 1, 16, 1]	[1, 1, 1, 0, 0]	[8, 16, 32, 64, 384]	407	0.6	2.9	26.2	27.9	1.3

Table 2.3: **Stem designs**: We compare ViT's standard patchify stem (P) and our convolutional stem (C) to four alternatives ($S1$ - $S4$) that each include a *patchify layer*, *i.e.*, a convolution with kernel size (> 1) equal to stride (highlighted in blue). Results use 50 epoch training, 4GF model size, and optimal *lr* and *wd* values for all models. We observe that increasing the pixel size of the patchify layer ($S1$ - $S4$) systematically degrades both top-1 error and optimizer stability (Δ) relative to C.



model	top-1 err.
ViT$_P$-4GF	23.2
ViT$_{P(bn)}$-4GF	23.3
ViT$_C$-4GF	20.9
ViT$_{C(ln)}$-4GF	21.1

Figure 2.7: **Stem normalization and non-linearity**: We apply BN and ReLU after the patchify stem and train ViT$_P$-4GF (*left plot*), or replace BN with layer norm (LN) in the convolutional stem of ViT$_C$-4GF (*middle plot*). EDFs are computed by sampling *lr* and *wd* values and training for 50 epochs. The table (*right*) shows 100 epoch results using best *lr* and *wd* values found at 50 epochs. The minor gap in error in the EDFs and at 100 epochs indicates that these choices are fairly insignificant.

Stem design. The focus of this paper is studying the large, positive impact of changing ViT's default patchify stem to a simple, standard convolutional stem constructed from stacked stride-two 3×3 convolutions. Exploring the stem design space, and more broadly "hybrid ViT" models [11], to maximize peak performance is an explicit *anti-goal* because we want to study the impact under minimal modifications. However, we can gain additional insight by considering alternative stem designs that fall between the patchify stem (P) the standard convolutional stem (C). Four alternative designs ($S1$ - $S4$) are presented in Table 2.3. The stems are designed so that overall model flops remain comparable. Stem $S1$ modifies C to include a small 2×2 patchify layer, which slightly worsens results. Stems $S2$ - $S4$ systematically increase the pixel size p of the patchify layer from $p = 2$ up to 16, matching the size used in stem P. *Increasing p reliably degrades both error and optimizer stability.* Although we

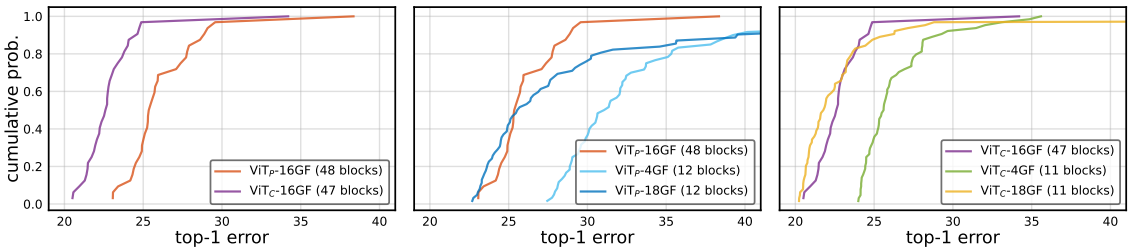


Figure 2.8: **Deeper models**: We increase the depth of ViT$_P$-4GF from 12 to 48 blocks, termed as ViT$_P$-16GF (48 blocks), and create a counterpart with a convolutional stem, ViT$_C$-16GF (47 blocks); all models are trained for 50 epochs. *Left*: The convolutional stem significantly improves error and stability despite accounting for only ~2% total flops. *Middle*, *Right*: The deeper 16GF ViTs clearly outperform the shallower 4GF models and achieve similar (slightly worse) error to the shallower and wider 18GF models. The deeper ViT$_P$ also has better *lr*/*wd* stability than the shallower ViT$_P$ models.

selected the C design *a priori* based on existing best-practices for CNNs, we see *ex post facto* that it outperforms four alternative designs that each include one patchify layer.

Stem normalization and non-linearity. We investigate normalization and non-linearity from two directions: (1) adding BN and ReLU to the default patchify stem of ViT, and (2) changing the normalization in the proposed convolutional stem. In the first case, we simply apply BN and ReLU after the patchify stem and train ViT$_P$-4GF (termed ViT$_{P(bn)}$-4GF) for 50 and 100 epochs. For the second case, we run four experiments with ViT$_C$-4GF: {50, 100} epochs × {BN, layer norm (LN)}. As before, we tune *lr* and *wd* for each experiment using the 50-epoch schedule and reuse those values for the 100-epoch schedule. We use AdamW for all experiments. Figure 2.7 shows the results. From the EDFs, which use a 50 epoch schedule, we see that the addition of BN and ReLU to the patchify stem slightly worsens the best top-1 error but does not affect *lr* and *wd* stability (*left*). Replacing BN with LN in the convolutional stem marginally degrades both best top-1 error and stability (*middle*). The table (*right*) shows 100 epoch results using optimal *lr* and *wd* values chosen from the 50 epoch runs. At 100 epochs the error gap is small indicating that these factors are likely insignificant.

Deeper Model Ablation Experiments

Touvron *et al.* [16] found that deeper ViT models are more unstable, *e.g.*, increasing the number of transformer blocks from 12 to 36 may cause a ~10 point drop in top-1 accuracy given a fixed choice of *lr* and *wd*. They demonstrate that

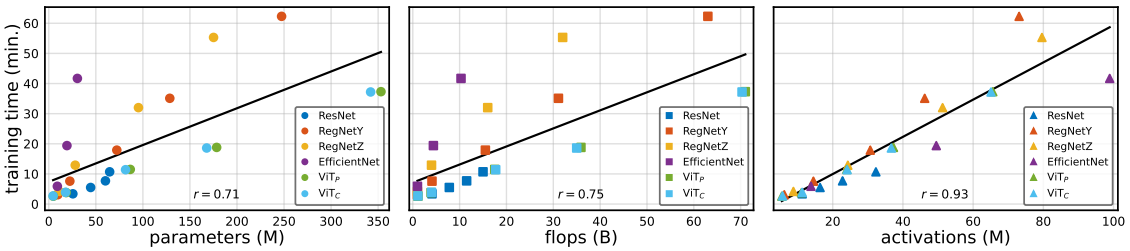


Figure 2.9: **Complexity measures *vs.* runtime**: We plot the GPU runtime of models versus three commonly used complexity measures: *parameters*, *flops*, and *activations*. For all models, including ViT, *runtime is most correlated with activations*, not flops, as was previously shown for CNNs.

stochastic depth and/or their proposed LayerScale can remedy this training failure. Here, we explore deeper models by looking at EDFs created by sampling *lr* and *wd*. We increase the depth of a ViT$_P$-4GF model from 12 blocks to 48 blocks, termed ViT$_P$-16GF (48 blocks). We then remove one block and use the convolutional stem from ViT$_C$-4GF, yielding a counterpart ViT$_C$-16GF (47 blocks) model. Figure 2.8 shows the EDFs of the two models and shallower models for comparison, following the setup in §2.5.3. Despite the convolutional stem accounting for only 1/48 (~2%) total flops, it shows solid improvement over its patchify counterpart. We find that a variety of *lr* and *wd* choices allow deeper ViT models to be trained without a large drop in top-1 performance and without additional modifications. In fact, the deeper ViT$_P$-16GF (48 blocks) has better *lr* and *wd* stability than ViT$_P$-4GF and ViT$_P$-18GF over the sampling range (Figure 2.8, *middle*).

Larger Model ImageNet-21k Experiments

In Table 2.2 we reported the peak performance of ViT models on ImageNet-21k up to 36GF. To study larger models, we construct a 72GF ViT$_P$ by using 22 blocks, 1152 hidden size, 18 heads, and 4 MLP multiplier. For ViT$_C$-72GF, we use the same C-stem design used for ViT$_C$-18GF and ViT$_C$-36GF, but *without* removing one transformer block since the flops increase from the C-stem is marginal in this complexity regime.

Our preliminary explorations into 72GF ViT models directly adopted hyper-parameters used for 36GF ViT models. Under this setting, we observed that the convolutional stem still improves top-1 error, however, we also found that a new form of instability arises, which causes training error to randomly spike. Sometimes training may recover within the same epoch, and subsequently the final accuracy is not impacted; or, it may take several epochs to recover from the error spike, and in

this case we observe suboptimal final accuracy. The first type of error spike is more common for ViT$_P$-72GF, while the latter type of error spike is more common for ViT$_C$-72GF.

To mitigate this instability, we adopt two measures: (i) For both models, we lower *wd* from 0.28 to 0.15 as we found that it significantly reduces the chance of error spikes. (ii) For ViT$_C$-72GF, we initialize its stem from the ImageNet-21k pre-trained ViT$_C$-36GF and keep it frozen throughout training. These modifications make training ViT-72GF models on ImageNet-21k feasible. When fine-tuned on ImageNet-1k, ViT$_P$-72GF reaches 14.2% top-1 error and ViT$_C$-72GF reaches 13.6% top-1 error, showing that ViT$_C$ still outperforms its ViT$_P$ counterpart. Increasing fine-tuning resolution from 224 to 384 boosts the performance of ViT$_C$-72GF to 12.6% top-1 error, while significantly increasing the fine-tuning model complexity from 72GF to 224GF.

Model Complexity and Runtime

In previous sections, we reported error *vs*. training time. Other commonly used complexity measures include *parameters*, *flops*, and *activations*. Indeed, it is most typical to report accuracy as a function of model flops or parameters. However, flops may fail to reflect the bottleneck on modern memory-bandwidth limited accelerators (*e.g.*, GPUs, TPUs). Likewise, parameters are an even more unreliable predictor of model runtime. Instead, activations have recently been shown to be a better proxy of runtime on GPUs (see [49, 50]). We next explore if similar results hold for ViT models.

For CNNs, previous studies [49, 50] defined *activations* as the *total size of all output tensors of the convolutional layers*, while disregarding normalization and non-linear layers (which are typically paired with convolutions and would only change the activation count by a constant factor). In this spirit, for transformers, we define *activations as the size of output tensors of all matrix multiplications*, and likewise disregard element-wise layers and normalizations. For models that use both types of operations, we simply measure the output size of all convolutional and vision transformer layers.

Figure 2.9 shows the runtime as a function of these model complexity measures. The Pearson correlation coefficient (r) confirms that activations have a much stronger linear correlation with actual runtime ($r = 0.93$) than flops ($r = 0.75$) or parameters ($r = 0.71$), confirming that the findings of [50] for CNNs also apply to ViTs. While flops are somewhat predictive of runtime, models with a large ratio of activations to flops, such as EfficientNet, have much higher runtime than expected based on flops. Finally, we note that ViT$_P$ and ViT$_C$ are nearly identical on all measures.

model	AdamW		SGD	
	lr	*wd*	*lr*	*wd*
RegNetY-∗	3.8e-3	0.1	2.54	2.4e-5
ViT$_P$-1GF	2.0e-3	0.20	1.9	1.3e-5
ViT$_P$-4GF	2.0e-3	0.20	1.9	1.3e-5
ViT$_P$-18GF	1.0e-3	0.24	1.1	1.2e-5
ViT$_C$-1GF	2.5e-3	0.19	1.9	1.3e-5
ViT$_C$-4GF	1.0e-3	0.24	1.3	2.2e-5
ViT$_C$-18GF	1.0e-3	0.24	1.1	2.7e-5

model	AdamW	
	lr	*wd*
ViT-∗	(2.5e−4, 8.0e−3)	(0.02, 0.8)
RegNetY-∗	(1.25e−3, 4.0e−2)	(0.0075, 0.24)

model	SGD	
	lr	*wd*
ViT-∗	(0.1, 3.2)	(4.0e−6, 1.2e−4)
RegNetY-∗	(0.25, 8.0)	(3.0e−6, 8.0e−5)

Table 2.4: **Learning rate and weight decay used in §2.5**: *Left*: Per-model *lr* and *wd* values used for the experiments in §2.5.1 and §2.5.2, optimized for ImageNet-1k at 50 epochs. *Right*: Per-model *lr* and *wd* ranges used for the experiments in §2.5.3. Note that for our final experiments in §2.6, we constrained the *lr* and *wd* values further, using a single setting for all CNN models, and just two settings for all ViT models. We recommend using this simplified set of values in §2.6 when comparing models for fair and easily reproducible comparisons. All *lr* values are normalized w.r.t. a minibatch size of 2048.

Timing. Throughout the paper we report *normalized* training time, as if the model were trained on a single 8 V100 GPU server, by multiplying the actual training time by the number of GPUs used and dividing by 8. (Due to different memory requirements of different models, we may be required to scale up the number of GPUs to accommodate the target minibatch size.) We use the number of minutes taken to process one ImageNet-1k epoch as a standard unit of measure. We prefer training time over inference time because inference time depends heavily on the use case (*e.g.*, a streaming, latency-oriented setting requires a batch size of 1 *vs.* a throughput-oriented setting that allows for batch size $\gg 1$) and the hardware platform (*e.g.*, smartphone, accelerator, server CPU).

Additional Experimental Details

Stability experiments. For the experiments in §2.5.1 and §2.5.2, we allow each CNN and ViT model to select a different *lr* and *wd*. We find that all CNNs select nearly identical values, so we normalize them to a single choice as done in [50]. ViT models prefer somewhat more varied choices. Table 2.4 (*left*) lists the selected values. For the experiments in §2.5.3, we use *lr* and *wd* intervals shown in Table 2.4 (*right*). These ranges are constructed by (i) obtaining initial good *lr* and *wd* choices for each model family; and then (ii) multiplying them by 1/8 and 4.0 for left and right interval endpoints (we use an asymmetric interval because models are trainable with smaller but not larger values). Finally we note that if we were to redo the experiments, the setting used in §2.5.1/§2.5.2 could be simplified.

model	Augment	Mixup	CutMix	Label Smooth	Model EMA	Erasing	Stoch Depth	Repeating	100 epochs	400 epochs	300 epochs [14]
ViT$_P$-4GF	Auto	✓	✓	✓	✓				23.2	20.5	-
	Rand	✓	✓	✓		✓	✓	✓	25.4	20.7	-
	Rand	✓	✓	✓		✓		✓	24.9	20.5	-
	Rand	✓	✓	✓		✓			23.6	20.4	-
	Rand	✓	✓	✓					23.5	20.3	-
	Auto	✓	✓	✓					23.0	20.3	-
ViT$_P$-18GF	Auto	✓	✓	✓	✓				19.9	17.9	-
	Rand	✓	✓	✓		✓	✓	✓	22.5	18.6	18.2
	Rand	✓	✓	✓		✓		✓	25.1	19.2	96.6
	Rand	✓	✓	✓		✓			21.2	19.9	-
	Rand	✓	✓	✓					20.9	19.7	-
	Auto	✓	✓	✓					20.4	20.0	-
	Rand	✓	✓	✓		✓	✓		-	-	22.6
	Rand	✓	✓	✓			✓	✓	-	-	95.7
	Rand	✓	✓	✓	✓	✓	✓	✓	-	-	18.1

Table 2.5: **Ablation of data augmentation and regularization**: We use the *lr* and *wd* from Table 2.4 (left), except for ViT$_P$-18GF models with RandAugment which benefit from stronger *wd* (we increase *wd* to 0.5). Original DeiT ablation results are copied for reference in gray (*last column*); these use a *lr*/*wd* of 1e−3/0.05 (*lr* normalized to minibatch size 2048), which leads to some training failures (we note our *wd* is 5-10× higher). Our default training setup (*first row* in each set) uses AutoAugment, mixup, CutMix, label smoothing, and model EMA. Compared to the DeiT setup (*second row* in each set), we do not use erasing, stochastic depth, or repeating. Although our setup is equally effective, it is simpler and also converges much faster (see Figure 2.10).

Peak performance on ImageNet-1k. We note that in later experiments we found tuning *lr* and *wd* per model is *not* necessary to obtain competitive results. Therefore, for our final experiments in §2.6, we constrained the *lr* and *wd* values further, using a single setting for all CNN models, and just two settings for all ViT models, as discussed in §2.6. We recommend using this simplified set of values when comparing models for fair and easily reproducible comparisons. Finally, for these experiments, when training is memory constrained (*i.e.*, for EfficientNet-{B4,B5}, RegNetZ-{4,16,32}GF), we reduce the minibatch size from 2048 and linearly scale the *lr* according to [26].

Peak performance on ImageNet-21k. For ImageNet-21k, a dataset of 14M images and ~21k classes, we pretrain models for 90 (ImageNet-21k) epochs, following [11]. We do *not* search for the optimal settings for ImageNet-21k and instead use the identical training recipe (up to minibatch size) used for ImageNet-1k. To reduce training time, we distribute training over more GPUs and use a larger minibatch size of 4096 with the *lr* scaled accordingly. For simplicity and reproducibility, we use a

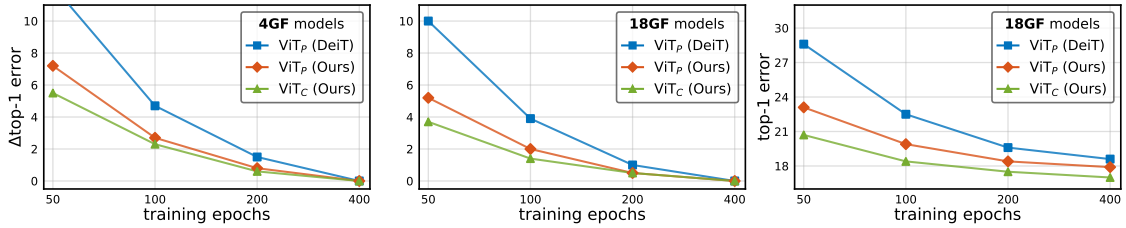


Figure 2.10: **Impact of training recipes on convergence**: We train ViT models using the DeiT recipe *vs*. our simplified counterpart. *Left and middle*: Δtop-1 error of 4GF and 18GF models at 50, 100 and 200 epoch schedules, and asymptotic performance at 400 epochs. *Right*: Absolute top-1 error of 18GF models. Removing augmentations and using model EMA accelerates convergence for both ViT$_P$ and ViT$_C$ models while slightly improving upon our reproduction of DeiT's top-1 error.

single label per image, unlike some prior work (*e.g.*, [58, 59]) that uses WordNet [60] to expand single labels to multiple labels. After pretraining, we fine-tune for 20 epochs on ImageNet-1k and use a small-scale grid search of *lr* while keeping *wd* at 0, similar to [11, 58].

Regularization and Data Augmentation

At this study's outset, we developed a simplified training setup for ViT models. Our goals were to design a training setup that is as simple as possible, resembles the setup used for state-of-the-art CNNs [50], and maintains competitive accuracy with DeiT [14]. Here, we document this exploration by considering the baseline ViT$_P$-4GF and ViT$_P$-18GF models. Beyond simplification, we also observe that our training setup yields faster convergence than the DeiT setup, as discussed below.

Table 2.5 compares our setup to that of DeiT [14]. Under their *lr*/*wd* choice, [14] report failed training when removing *erasing* and *stochastic depth*, as well as significant drop of accuracy when removing *repeating*. We find that they can be safely disabled as long as a higher *wd* is used (our *wd* is 5-10× higher). We observe that we can remove model EMA for ViT$_P$-4GF, but that it is essential for the larger ViT$_P$-18GF model, especially at 400 epochs. Without model EMA, ViT$_P$-18GF can still be trained effectively, but this requires additional augmentation and regularization (as in DeiT).

Figure 2.10 shows that our training setup accelerates convergence for both ViT$_P$ and ViT$_C$ models, as can be seen by comparing the error *deltas* (Δtop-1) between the DeiT baseline and ours (*left* and *middle* plots). Our training setup also yields slightly better top-1 error than our reproduction of DeiT (*right* plot). We conjecture that

faster convergence is due to removing repeating augmentation [56, 57], which was shown in [56] to slow convergence. Under some conditions repeating augmentation may improve accuracy; we did not observe such improvements in our experiments.

2.8 Conclusion

In this chapter we demonstrated that the optimization challenges of ViT models are linked to the large-stride, large-kernel convolution in ViT's patchify stem. The seemingly trivial change of replacing this patchify stem with a simple convolutional stem leads to a remarkable change in optimization behavior. With the convolutional stem, ViT (termed ViT_C) converges faster than the original ViT (termed ViT_P) (§2.5.1), trains well with either AdamW or SGD (§2.5.2), improves learning rate and weight decay stability (§2.5.3), and improves ImageNet top-1 error by ~1-2% (§2.6). These results are consistent across a wide spectrum of model complexities (1GF to 36GF) and dataset scales (ImageNet-1k to ImageNet-21k). Our results indicate that injecting a small dose of convolutional inductive bias into the early stages of ViTs can be hugely beneficial. Looking forward, we are interested in the theoretical foundation of why such a minimal architectural modification can have such large (positive) impact on optimizability. We are also interested in studying larger models. Our preliminary explorations into 72GF models reveal that the convolutional stem still improves top-1 error, however we also find that a *new* form of instability arises that causes training error to randomly spike, especially for ViT_C.

Chapter 3

Reducing Inductive Bias in Contrastive Self-Supervised Learning

3.1 Introduction

Self-supervised learning, which uses raw image data and/or available pretext tasks as its own supervision, has become increasingly popular as the inability of supervised models to generalize beyond their training data has become apparent. Different pretext tasks have been proposed with different transformations, such as spatial patch prediction [61, 62], colorization [63–65], rotation [66]. Whereas pretext tasks aim to recover the transformations between different "views" of the same data, more recent contrastive learning methods [67–70] instead try to learn to be *invariant* to these transformations, while remaining discriminative with respect to other data points. Here, the transformations are generated using classic data augmentation techniques which correspond to common pretext tasks, e.g., randomizing color, texture, orientation and cropping.

Yet, the inductive bias introduced through such augmentations is a double-edged sword, as each augmentation encourages invariance to a transformation which can be beneficial in some cases and harmful in others: e.g., adding rotation may help with view-independent aerial image recognition, but significantly downgrade the capacity of a network to solve tasks such as detecting which way is up in a photograph for a display application. Current self-supervised contrastive learning methods assume implicit knowledge of downstream task invariances. In this chapter, we propose to learn visual representations which capture individual factors of variation in a contrastive learning framework without presuming prior knowledge of downstream invariances.

Instead of mapping an image into a single embedding space which is invariant to

all the hand-crafted augmentations, our model learns to construct separate embedding sub-spaces, each of which is sensitive to a specific augmentation while invariant to other augmentations. We achieve this by optimizing multiple augmentation-sensitive contrastive objectives using a multi-head architecture with a shared backbone. Our model aims to preserve information with regard to each augmentation in a unified representation, as well as learn invariances to them. The general representation trained with these augmentations can then be applied to different downstream tasks, where each task is free to selectively utilize different factors of variation in our representation. We consider transfer of either the shared backbone representation, or the concatenation of all the task-specific heads; both outperform all baselines; the former uses same embedding dimensions as typical baselines, while the latter provides greatest overall performance in our experiments. In this paper, we experiment with three types of augmentations: rotation, color jittering, and texture randomization, as visualized in Figure 3.1. We evaluate our approach across a variety of diverse tasks including large-scale classification [2], fine-grained classification [71, 72], few-shot classification [73], and classification on corrupted data [5, 74]. Our representation shows consistent performance gains with increasing number of augmentations. Our method does not require hand-selection of data augmentation strategies, and achieves better performance against state-of-the-art MoCo baseline [69, 75], and demonstrates superior transferability, generalizability and robustness across tasks and categories. Specifically, we obtain around 10% improvement over MoCo in classification when applied on the iNaturalist [72] dataset.

3.2 Background: Contrastive Learning Framework

Contrastive learning learns a representation by maximizing similarity and dissimilarity over data samples which are organized into similar and dissimilar pairs, respectively. It can be formulated as a dictionary look-up problem [69], where a given reference image $\mathcal{I}$ is augmented into two views, query and key, and the query token q should match its designated key k^+ over a set of sampled negative keys $\{k^-\}$ from other images. In general, the framework can be summarized as the following components: (i) A data augmentation module $\mathcal{T}$ constituting n atomic augmentation operators, such as random cropping, color jittering, and random flipping. We denote a pre-defined atomic augmentation as random variable X_i. Each time the atomic augmentation is executed by sampling a specific augmentation parameter from the random variable, i.e., $x_i \sim X_i$. One sampled data augmentation module transforms image $\mathcal{I}$ into a random view $\widetilde{\mathcal{I}}$, denoted as $\widetilde{\mathcal{I}} = \mathcal{T}[x_1, x_2, \ldots, x_n](\mathcal{I})$. Positive pair (q, k^+) is generated by applying two randomly sampled data augmentation on the

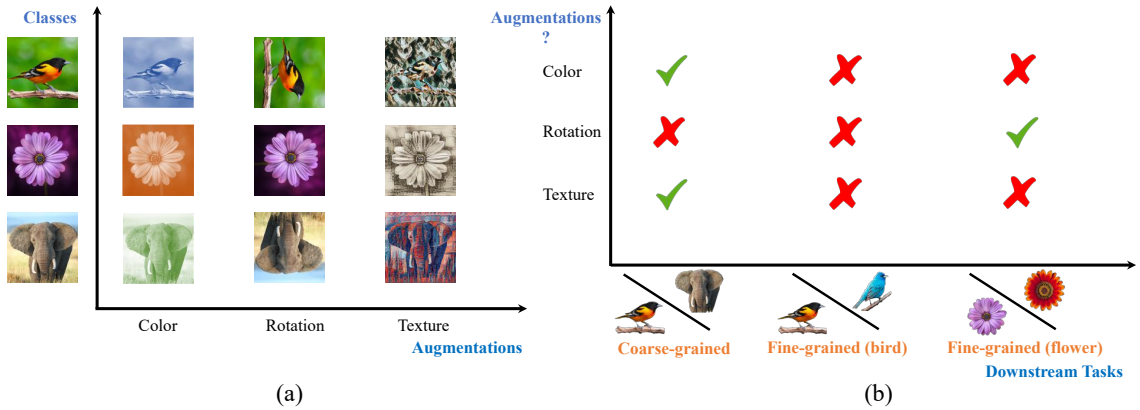


Figure 3.1: Self-supervised contrastive learning relies on data augmentations as depicted in (a) to learn visual representations. However, current methods introduce inductive bias by encouraging neural networks to be less sensitive to information w.r.t. augmentation, which may help or may hurt. As illustrated in (b), rotation invariant embeddings can help on certain flower categories, but may hurt animal recognition performance; conversely color invariance generally seems to help coarse grained animal classification, but can hurt many flower categories and bird categories. Our method, shown in the following figure, overcomes this limitation.

same reference image. (ii) An encoder network f which extracts the feature $\boldsymbol{v}$ of an image $\mathcal{I}$ by mapping it into a d-dimensional space $\mathbb{R}^d$. (iii) A projection head h which further maps extracted representations into a hyper-spherical (normalized) embedding space. This space is subsequently used for a specific pretext task, i.e., contrastive loss objective for a batch of positive/negative pairs. A common choice is InfoNCE [76]:

$$\mathcal{L}_q = -\log \frac{\exp\left(q \cdot k^+/\tau\right)}{\exp\left(q \cdot k^+/\tau\right) + \sum_{k^-} \exp\left(q \cdot k^-/\tau\right)}, \tag{3.1}$$

where τ is a temperature hyper-parameter scaling the distribution of distances.

As a key towards learning a good feature representation [70], a strong augmentation policy prevents the network from exploiting naïve cues to match the given instances. However, inductive bias is introduced through the selection of augmentations, along with their hyper-parameters defining the strength of each augmentation, manifested in Equation 3.1 that *any* views by the stochastic augmentation module $\mathcal{T}$ of the same instance are mapped onto the same point in the embedding space. The property negatively affects the learnt representations: 1) Generalizability and transferability are harmed if they are applied to the tasks where the discarded information is essential, e.g., color plays an important role in fine-grained classification

of birds; 2) Adding an extra augmentation is complicated as the new operator may be helpful to certain classes while harmful to others, e.g., a rotated flower could be very similar to the original one, whereas it does not hold for a rotated car; 3) The hyper-parameters which control the strength of augmentations need to be carefully tuned for each augmentation to strike a delicate balance between leaving a short-cut open and completely invalidate one source of information.

3.3 LooC: Leave-one-out Contrastive Learning

We propose Leave-one-out Contrastive Learning (LooC), a framework for multi-augmentation contrastive learning. Our framework can selectively prevent information loss incurred by an augmentation. Rather than projecting every view into a single embedding space which is invariant to all augmentations, in our LooC method the representations of input images are projected into several embedding spaces, each of which is *not* invariant to a certain augmentation while remaining invariant to others, as illustrated in Figure 3.2. In this way, each embedding sub-space is specialized to a single augmentation, and the shared layers will contain both augmentation-varying and invariant information. We learn a shared representation jointly with the several embedding spaces; we transfer either the shared representation alone, or the concatenation of all spaces, to downstream tasks.

View Generation. Given a reference image and n atomic augmentations, we first augment the reference image with two sets of independently sampled augmentation parameters into the query view $\mathcal{I}_q$ and the first key view $\mathcal{I}_{k_0}$, i.e., $\mathcal{I}_{\{q,k_0\}} = \mathcal{T}[x_1^{\{q,k_0\}}, x_2^{\{q,k_0\}}, \ldots, x_n^{\{q,k_0\}}](\mathcal{I})$. Additionally, we generate n views from the reference image as extra key views, denoted as $\mathcal{I}_{k_i}$, $\forall i \in \{1, \ldots, n\}$. For the i^{th} additional key view, the parameter of i^{th} atomic augmentation is copied from it of the query view, i.e., $x_i^{k_i} \equiv x_i^q$, $\forall i \in \{1, \ldots, n\}$; whereas the parameter of other atomic augmentations are still independently sampled, i.e., $x_j^{k_i} \sim X_j$, $\forall j \neq i$. For instance, assume that we have a set of two atomic augmentations {`random_rotation`, `color_jitter`}, $\mathcal{I}_q$ and $\mathcal{I}_{k_1}$ are always augmented by the same rotation angle but different color jittering; $\mathcal{I}_q$ and $\mathcal{I}_{k_2}$ are always augmented by the same color jittering but different rotation angle; $\mathcal{I}_q$ and $\mathcal{I}_{k_0}$ are augmented independently, as illustrated in the left part of Figure 3.2.

Contrastive Embedding Space. The augmented views are encoded by a neural network encoder $f(\cdot)$ into feature vectors $\boldsymbol{v}^q, \boldsymbol{v}^{k_0}, \cdots, \boldsymbol{v}^{k_n}$ in a joint embedding space $\mathcal{V} \in \mathbb{R}^d$. Subsequently, they are projected into n+1 normalized embedding spaces $\mathcal{Z}_0, \mathcal{Z}_1, \cdots, \mathcal{Z}_n \in \mathbb{R}^{d'}$ by projection heads $h : \mathcal{V} \mapsto \mathcal{Z}$, among which $\mathcal{Z}_0$ is invariant to all types of augmentations, whereas $\mathcal{Z}_i$ ($\forall i \in \{1, 2, \cdots, n\}$) is dependent on the

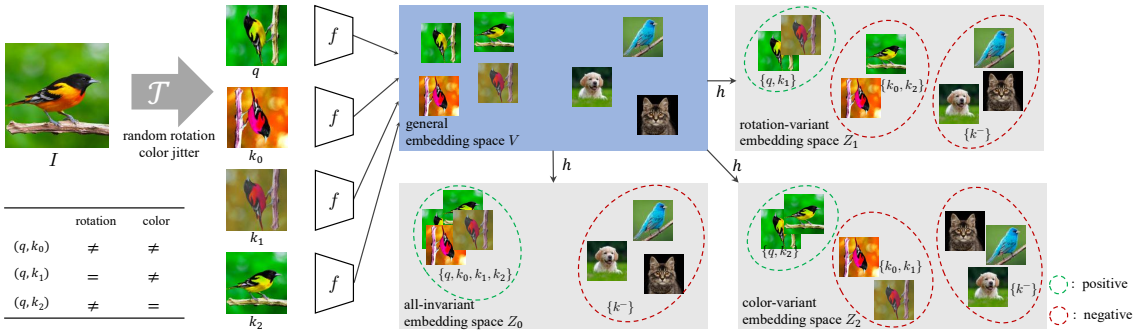


Figure 3.2: **Framework of the Leave-one-out Contrastive Learning approach**, illustrated with two types of augmentations, i.e., random rotation and color jittering. We generate multiple views with leave-one-out strategy, then project their representations into separate embedding spaces with contrastive objective, where each embedding space is either invariant to all augmentations, or invariant to all but one augmentation. The learnt representation can be the general embedding space $\mathcal{V}$ (blue region), or the concatenation of embedding sub-spaces $\mathcal{Z}$ (grey region). Our results show that either of our proposed representations are able to outperform baseline contrastive embeddings and do not suffer from decreased performance when adding augmentations to which the task is not invariant (i.e., the red X's in Figure 1).

i^{th} type of augmentation but invariant to other types of augmentations. In other words, in $\mathcal{Z}_0$ all features $\boldsymbol{v}$ should be mapped to a single point, whereas in $\mathcal{Z}_i$ $(\forall i \in \{1, 2, \cdots, n\})$ only $\boldsymbol{v}^q$ and $\boldsymbol{v}^{k_i}$ should be mapped to a single point while $\boldsymbol{v}^{k_j}$ $\forall j \neq i$ should be mapped to $n-1$ separate points, as only $\mathcal{I}_q$ and $\mathcal{I}_{k_i}$ share the same i^{th} augmentation.

We perform contrastive learning in all normalized embedding spaces based on Equation 3.1, as shown in the right part of Figure 3.2. For each query $\boldsymbol{z}^q$, denote $\boldsymbol{z}^{k^+}$ as the keys from the *same* instance, and $\boldsymbol{z}^{k^-}$ as the keys from *other* instances. Since all views should be mapped to the single point in $\mathcal{Z}_0$, the positive pair for the query $\boldsymbol{z}_0^q$ is $\boldsymbol{z}_0^{k_0^+}$, and the negative pairs are embeddings of other instances in this embedding space $\{\boldsymbol{z}_0^{k_0^-}\}$; for embedding spaces $\mathcal{Z}_1, \cdots, \mathcal{Z}_n$, the positive pair for the query $\boldsymbol{z}_i^q$ is $\boldsymbol{z}_i^{k_i^+}$, while the negative pairs are embeddings of other instances in this embedding space $\{\boldsymbol{z}_i^{k_i^-}\}$, *and* $\{\boldsymbol{z}_i^{k_j^+} \mid \forall j \in \{0, 1, \cdots, n\} \text{ and } j \neq i\}$, which are the embeddings of the *same* instance with different i^{th} augmentation. The network then learns to be sensitive to one type of augmentation while insensitive to other types of augmentations in one embedding space. Denote $E_{i,j}^{\{+,-\}} = \exp\left(\boldsymbol{z}_i^q \cdot \boldsymbol{z}_i^{k_j^{\{+,-\}}} / \tau\right)$. The

model	Rotation Acc.	IN-100 top-1	IN-100 top-5
Supervised	72.3	83.7	95.7
MoCo	61.1	81.0	95.2
MoCo + Rotation	43.3	79.4	94.1
MoCo + Rotation (same for q and k)	45.5	78.1	94.3
LooC + Rotation [ours]	65.2	80.2	95.5

Table 3.1: **Classification accuracy on 4-class rotation and IN-100** under linear evaluation protocol. Adding rotation augmentation into baseline MoCo significantly reduces its capacity to classify rotation angles while downgrades its performance on IN-100. In contrast, our method better leverages the information gain of the new augmentation.

overall training objective for q is:

$$\mathcal{L}_q = -\frac{1}{n+1}\left(\log\frac{E^+_{0,0}}{E^+_{0,0}+\sum_{k^-}E^-_{0,0}} + \sum_{i=1}^{n}\log\frac{E^+_{i,i}}{\sum_{j=0}^{n}E^+_{i,j}+\sum_{k^-}E^-_{i,i}}\right), \tag{3.2}$$

The network must preserve information w.r.t. all augmentations in the general embedding space $\mathcal{V}$ in order to optimize the combined learning objectives of all normalized embedding spaces.

Learnt representations. The representation for downstream tasks can be from the general embedding space $\mathcal{V}$ (Figure 3.2, blue region), or the concatenation of all embedding sub-spaces (Figure 3.2, grey region). LooC method returns $\mathcal{V}$; we term the implementation using the concatenation of all embedding sub-spaces as LooC++.

3.4 Experiments

Methods. We adopt Momentum Contrastive Learning (MoCo) [69] as the backbone of our framework for its efficacy and efficiency, and incorporate the improved version from [75]. We use three types of augmentations as pretext tasks for static image data, namely color jittering (including random gray scale), random rotation (90°, 180°, or 270°), and texture randomization [4, 77]. We apply random-resized cropping, horizontal flipping and Gaussian blur as augmentations without designated embedding spaces. Note that random rotation and texture randomization are not utilized in state-of-the-art contrastive learning based methods [69, 70, 75] and for good reason, as we will empirically show that naïvely taking these augmentations negatively affects the performance on some specific benchmarks. For LooC++, we include `Conv5` block into the projection head h, and use the concatenated features at the last layer of `Conv5`, instead of the last layer of h, from each head. Note than for both LooC and

model	Augmentation		iNat-1k		CUB-200		Flowers-102		IN-100	
	Color	Rotation	top-1	top-5	top-1	top-5	5-shot	10-shot	top-1	top-5
MoCo	✓		36.2	62.0	36.7	64.7	67.9 (± 0.5)	77.3 (± 0.1)	81.0	95.2
LooC	✓		41.2	67.0	40.1	69.7	68.2 (± 0.6)	77.6 (± 0.1)	81.1	95.3
		✓	40.0	65.4	38.8	67.0	70.1 (± 0.4)	79.3 (± 0.1)	80.2	95.5
	✓	✓	44.0	69.3	39.6	69.2	70.9 (± 0.3)	80.8 (± 0.2)	79.2	94.7
LooC++	✓	✓	46.1	71.5	39.3	69.3	68.1 (± 0.4)	78.8 (± 0.2)	81.2	95.2

Table 3.2: **Evaluation on multiple downstream tasks**. Our method demonstrates superior generalizability and transferability with increasing number of augmentations.

LooC++ the augmented additional keys are only fed into the key encoding network, which is not back-propagated, thus it does not much increase computation or GPU memory consumption.

Datasets and evaluation metrics. We train our model on the 100-category ImageNet (IN-100) dataset, a subset of the ImageNet [2] dataset, for fast ablation studies of the proposed framework. We split the subset following [68]. The subset contains ∼125k images, sufficiently large to conduct experiments of statistical significance. After training, we adopt *linear* classification protocol by training a *supervised* linear classifier on *frozen* features of feature space $\mathcal{V}$ for LooC, or concatenated feature spaces $\mathcal{Z}$ for LooC++. This allows us to directly verify the quality of features from a variation of models, yielding more interpretable results. We test the models on various downstream datasets: 1) IN-100 validation set; 2) The iNaturalist 2019 (iNat-1k) dataset [72], a large-scale classification dataset containing 1,010 species. Top-1 and top-5 accuracy on this dataset are reported; 3) The Caltech-UCSD Birds 2011 (CUB-200) dataset [71], a fine-grained classification dataset of 200 bird species. Top-1 and top-5 classification accuracy are reported. 4) VGG Flowers (Flowers-102) dataset [73], a consistent of 102 flower categories. We use the dataset for few-shot classification and report 5-shot and 10-shot classification accuracy over 10 trials within 95% confidence interval. Unlike many few-shot classification methods which conduct evaluation on a subset of categories, we use all 102 categories in our study; 5) ObjectNet dataset [5], a test set collected to intentionally show objects from new viewpoints on new backgrounds with different rotations of real-world images. We only use the 13 categories which overlap with IN-100, termed as ON-13; 6) ImageNet-C dataset [74], a benchmark for model robustness of image corruptions. We use the 100 categories as IN-100, termed as IN-C-100. Note that ON and IN-C are test sets, so we do not train a supervised linear classifier exclusively while directly benchmark the linear classifier trained on IN-100 instead.

Implementation details. We closely follow [75] for most training hyper-parameters.

We use a ResNet-50 [17] as our feature extractor. We use a two-layer MLP head with a 2048-d hidden layer and ReLU for each individual embedding space. We train the network for 500 epochs, and decrease the learning rate at 300 and 400 epochs. We use separate queues [69] for individual embedding space and set the queue size to 16,384. The batch size during training is set to 256.

Study on augmentation inductive biases. We start by designing an experiment which allows us to directly measure how much an augmentation affects a downstream task which is sensitive to the augmentation. For example, consider two tasks which can be defined on IN-100: Task A is 4-category classification of rotation degrees for an input image; Task B is 100-category classification of ImageNet objects. We train a supervised *linear* classifier for task A with randomly rotated IN-100 images, and another classifier for task B with *unrotated* images. In Table 3.1 we compare the accuracy of the original MoCo (w/o rotation augmentation), MoCo w/ rotation augmentation, and our model w/ rotation augmentation. A priori, with no data labels to perform augmentation selection, we have no way to know if rotation should be utilized or not. Adding rotation into the set of augmentations for MoCo downgrades object classification accuracy on IN-100, and significantly reduces the capacity of the baseline model to distinguish the rotation of an input image. We further implement a variation enforcing the random rotating angle of query and key always being the same. Although it marginally increases rotation accuracy, IN-100 object classification accuracy further drops, which is inline with our hypothesis that the inductive bias of discarding certain type of information introduced by adopting an augmentation into contrastive learning objective is significant and cannot be trivially resolved by tuning the distribution of input images. On the other hand, our method with rotation augmentation not only sustains accuracy on IN-100, but also leverages the information gain of the new augmentation. We can include all augmentations with our LooC multi-self-supervised method and obtain improved performance across all condition without any downstream labels or a prior knowledged invariance.

Fine-grained recognition results. A prominent application of unsupervised learning is to learn features which are transferable and generalizable to a variety of downstream tasks. To fairly evaluate this, we compare our method with original MoCo on a diverse set of downstream tasks. Table 3.2 lists the results on iNat-1k, CUB-200 and Flowers-102. Although demonstrating marginally superior performance on IN-100, the original MoCo trails our LooC counterpart on all other datasets by a noticeable margin. Specifically, applying LooC on random color jiterring boosts the performance of the baseline which adopts the same augmentation. The comparison shows that our method can better preserve color information. Rotation augmentation also boosts the performance on iNat-1k and Flowers-102, while yields smaller improvements on CUB-

model	Aug.		ON-13		IN-C-100 (top-1)						IN-100	
	Rot.	Tex.	top-1	top-5	Noise	Blur	Weather	Digital	All	$d \geq 3$	top-1	top-5
Supervised			30.9	54.8	28.4	47.1	44.9	58.5	47.2	36.5	83.7	95.7
MoCo			29.2	54.2	37.9	38.5	47.7	60.1	48.2	37.2	81.0	95.2
LooC	✓		34.2	59.6	31.3	33.1	42.4	54.9	42.7	31.8	80.2	95.5
		✓	30.1	54.1	42.4	39.6	54.0	61.9	51.3	41.9	81.0	94.7
	✓	✓	33.3	59.2	37.0	35.2	50.2	56.9	46.5	37.2	79.4	94.3
LooC++	✓	✓	32.6	57.3	38.3	37.6	52.0	60.0	48.8	38.9	82.1	95.1

Table 3.3: **Evaluation on datasets of real-world corruptions.** Rotation augmentation is beneficial for ON-13, and texture augmentation if beneficial for IN-C-100.

Model	Variance Head			IN-100		iNat-1k		Flowers-102		IN-C-100
	Col.	Rot.	Tex.	top-1	top-5	top-1	top-5	5-shot	10-shot	all-top-1
LooC++				78.5	94.3	38.5	64.7	68.6 (± 0.6)	77.6 (± 0.1)	48.0
	✓			79.7	94.4	42.9	68.7	69.1 (± 0.7)	79.5 (± 0.2)	47.1
		✓		81.5	94.9	41.4	67.4	70.5 (± 0.6)	80.0 (± 0.2)	52.6
			✓	80.3	94.9	43.0	68.6	70.4 (± 0.5)	80.5 (± 0.2)	44.1
	✓	✓	✓	82.2	95.3	45.9	71.4	71.0 (± 0.7)	81.9 (± 0.3)	48.0

Table 3.4: **Comparisons of concatenating features from different embedding spaces in LooC++** jointly trained on color, rotation and texture augmentations. Downstream tasks show nonidentical preferences for augmentation-dependent or invariant representations.

200, which supports the intuition that some categories benefit from rotation-invariant representations while some do not. The performance is further boosted by using LooC with both augmentations, demonstrating the effectiveness in simultaneously learning the information w.r.t. multiple augmentations.

Interestingly, LooC++ brings back the slight performance drop on IN-100, and yields more gains on iNat-1k, which indicates the benefits of explicit feature fusion without hand-crafting what should or should not be contrastive in training.

Robustness learning results. Table 3.3 compares our method with MoCo and supervised model on ON-13 and IN-C-100, two testing sets for real-world data generalization under a variety of noise conditions. The linear classifier is trained on standard IN-100, without access to the testing distribution. The fully supervised network is most sensitive to perturbations, albeit it has highest accuracy on the source dataset IN-100. We also see that rotation augmentation is beneficial for ON-13, but significantly downgrades the robustness to data corruptions in IN-C-100. Conversely, texture randomization increases the robustness on IN-C-100 across all corruption types, particularly significant on "Blur" and "Weather", and on the severity level above or equal to 3, as the representations must be insensitive to local

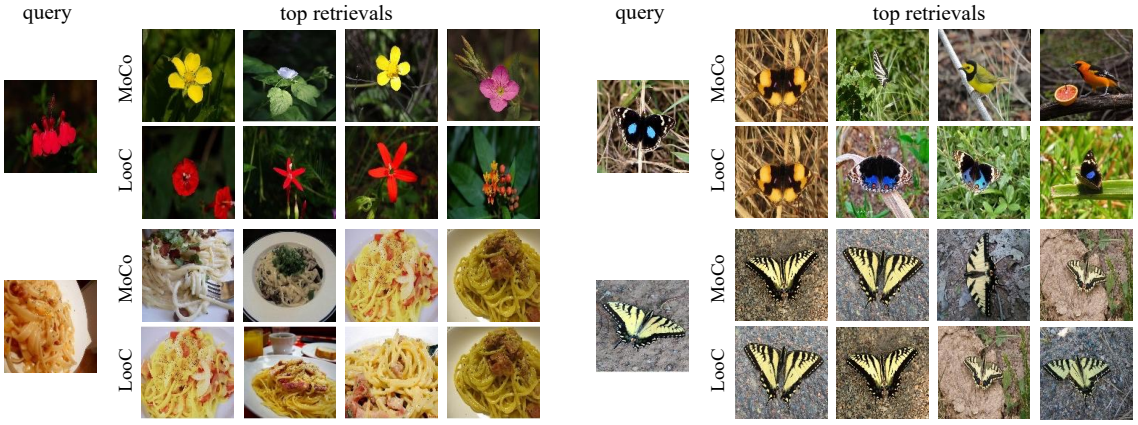


Figure 3.3: **Top nearest-neighbor retrieval results** of LooC vs. corresponding invariant MoCo baseline with color (left) and rotation (right) augmentations on IN-100 and iNat-1k. The results show that our model can better preserve information dependent on color and rotation despite being trained with those augmentations.

noise to learn texture-invariant features, but its improvement on ON-13 is marginal. Combining rotation and texture augmentation yields improvements on both datasets, and LooC++ further improves its performance on IN-C-100.

Qualitative results. In Figure 3.3 we show nearest-neighbor retrieval results using features learnt with LooC vs. corresponding MoCo baseline. The top retrieval results demonstrate that our model can better preserve information which is not invariant to the transformations presented in the augmentations used in contrastive learning.

Ablation: MoCo w/ all augmentations vs. LooC. We compare our method and MoCo trained with all augmentations. We also add multiple `Conv5` heads to MoCo, termed as MoCo++, for a fair comparison with LooC++. The results are listed in Table 3.5. Using multiple heads boosts the performance of baseline MoCo, nevertheless, our method achieves better or comparable results compared with its baseline counterparts.

Note that the results in Table 2 to 5 should be interpreted in the broader context of Table 3.1. Table 1 illustrates the catastrophic consequences of not separating the varying and invariant factors of an augmentation (in this case, rotation). It can be imagined that if we add "rotation classification" as one downstream task in Table 4, MoCo++ will perform as poorly as in Table 1. The key of our work is to avoid what has happened in Table 1 and simultaneously boosts performance.

Ablation: Augmentation-dependent embedding spaces vs. tasks. We train a LooC++ with all types of augmentations, and subsequently train multiple linear classifiers with concatenated features from different embedding spaces: all-invariant,

Model	IN-100		iNat-1k		Flowers-102		IN-C-100
	top-1	top-5	top-1	top-5	5-shot	10-shot	all-top-1
MoCo	77.9	93.7	39.5	65.1	72.1 (± 0.4)	81.1 (± 0.2)	47.4
LooC	78.5	94.0	41.7	67.5	72.1 (± 0.7)	81.4 (± 0.2)	45.4
MoCo++	80.8	94.6	43.4	68.5	70.0 (± 0.8)	80.5 (± 0.3)	48.3
LooC++	82.2	95.3	45.9	71.4	71.0 (± 0.7)	81.9 (± 0.3)	48.0

Table 3.5: **Comparisons of LooC vs. MoCo** trained with all augmentations.

color, rotation and texture. Any additional variance features boost the performance on IN-100, iNat-1k and Flowers-102. Adding texture-dependent features decreases the performance on IN-C-100: Textures are (overly) strong cues for ImageNet classification [4], thus the linear classifier is prone to use texture-dependent features, loosing the gains of texture invariance. Adding rotation-dependent features increases the performance on IN-C-100: Rotated objects of most classes in IN-100 are rare, thus the linear classifier is prone to use rotation-dependent features, so that drops on IN-C-100 triggered by rotation-invariant augmentation are re-gained. Using all types of features yields best performance on IN-100, iNat-1k and Flowers-102; the performance on IN-C-100 with all augmentations remains comparable to MoCo, which does not suffer from loss of robustness introduced by rotation invariance.

In Figure 3.4 we show the histogram of correct predictions (activations×weights of classifier) by each augmentation-dependent head of a few instances from IN-100 and iNat-1k. The classifier prefers texture-dependent information over other kinds on an overwhelmingly majority of samples from IN-100, even for classes where shape is supposed to be the dominant factor, such as "pickup" and "mixing bowl" ((a), top row). This is consistent with the findings from [4] that ImageNet-trained CNNs are strongly biased towards texture-like representations. Interestingly, when human or animal faces dominant an image ((a), bottom-left), LooC++ sharply prefers rotation-dependent features, which also holds for face recognition of humans. In contrast, on iNat-1k LooC++ prefers a more diverse set of features, such as color-dependent feature for a dragonfly species, rotation and texture-dependent features for birds, as well as rotation-invariant features for flowers. Averaged over the datasets, the distribution of classifier preferences is more balanced on iNat-1k than IN-100, as the entropy of the distribution on iNat-1k is close to 2 bits, whereas it is close to 1 bit on IN-100, as it is dominated by only two elements. It corroborates the large improvements on iNat-1k gained from multi-dependent features.

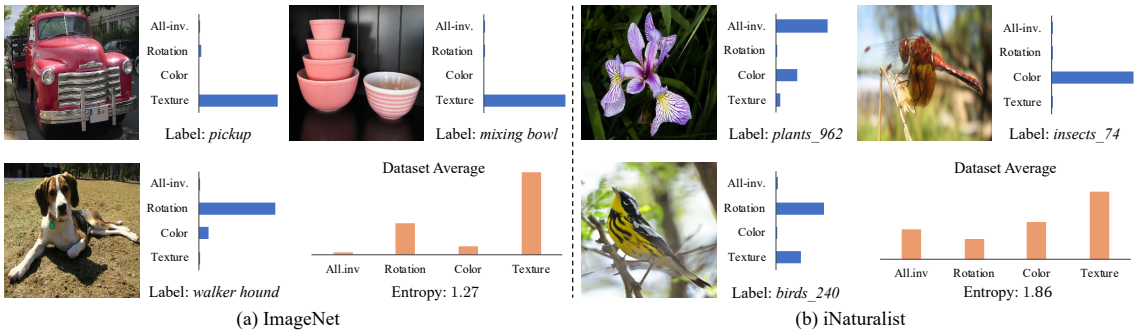


Figure 3.4: **Histograms of correct predictions (activations×weights of classifier) by each augmentation-dependent head** from IN-100 and iNat-1k. The classifier on IN-100 heavily relies on texture-dependent information, whereas it is much more balanced on iNat-1k. This is consistent with the improvement gains observed when learning with multiple augmentations.

3.5 Related Work

Pretext Tasks. In computer vision, feature design and engineering used to be a central topic before the wide application of deep learning. Researchers have proposed to utilize cue combination for image retrieval and recognition tasks [78–82]. For example, the local brightness, color, and texture features are combined together to represent an image and a simple linear model can be trained to detect boundaries [78]. Interestingly, the recent development of unsupervised representation learning in deep learning is also progressed by designing different self-supervised pretext tasks [61–63, 66, 83–85]. For example, relative patch prediction [61] and rotation prediction [66] are designed to discover the underlined structure of the objects; image colorization task [63] is used to learn representations capturing color information. The inductive bias introduced by each pretext task can often be associated with a corresponding hand-crafted descriptor.

Multi-Task Self-Supervised Learning. Multi-task learning has been widely applied in image recognition [28, 86, 87]. However, jointly optimizing multiple tasks are not always beneficial. As shown in [86], training with two tasks can yield better performance than seven tasks together, as some tasks might be conflicted with each other. This phenomenon becomes more obvious in multi-task self-supervised learning [88–92] as the optimization goal for each task can be very different depending on the pretext task. To solve this problem, different weights for different tasks are learned to optimize for the downstream tasks [91]. However, searching the weights typically requires labels, and is time-consuming and does not generalize to different tasks. To train general representations, researchers have proposed to utilize sparse

regularization to factorize the network representations to encode different information from different tasks [88, 93]. In this paper, we also proposed to learn representation which can factorize and unify information from different augmentations. Instead of using sparse regularization, we define different contrastive learning objective in a multi-head architecture.

Contrastive Learning. Instead of designing different pretext tasks, recent work on contrastive learning [67–70, 76, 94] trained networks to be invariant to various corresponding augmentations. Researchers [70] elaborated different augmentations and pointed out which augmentations are helpful or harmful for ImageNet classification. It is also investigated in [68] that different augmentations can be beneficial to different downstream tasks. Instead of enumerating all the possible selections of augmentations, we proposed a unified framework which captures different factors of variation introduced by different augmentations.

3.6 Conclusions

Current contrastive learning approaches rely on specific augmentation-derived transformation invariances to learn a visual representation, and may yield suboptimal performance on downstream tasks if the wrong transformation invariances are presumed. We propose a new model which learns both transformation dependent and invariant representations by constructing multiple embeddings, each of which is *not* contrastive to a single type of transformation. Our framework outperforms baseline contrastive method on coarse-grained, fine-grained, few-shot downstream classification tasks, and demonstrates better robustness of real-world data.

Chapter 4

Masked Visual Pre-training for Motor Control

4.1 Introduction

The last decade of machine learning has been powered by learning representations with large neural networks and augmenting them with a relatively small amount of domain knowledge where appropriate. This paradigm has led to substantial progress across a range of domains. Examples include visual recognition [28, 95], natural language [6,8,9,96], and audio [97]. And the trend continues. Motor control, however, remains a notable exception. In this chapter, we show that self-supervised visual pre-training on real-world images is effective for learning motor control tasks from pixels. These self-supervised representations consistently outperform supervised representations.

Consider tasks shown in Figure 4.1 (right). The required movement types vary from simple reaching to object interactions. We also see variations in robots, scene configurations, and objects. Control inputs are high-dimensional and difficult to search (e.g., 23 DoF robot with a multi-finger hand). We explore learning complex tasks such as these from high-dimensional pixel observations. To tackle this setting, we use the neural network architecture shown in Figure 4.2b. Our network encodes the input image using a high-capacity visual encoder [11] and combines it with proprioceptive information to obtain an embedding. A light-weight neural network controller takes in the embedding and predicts actions. The whole system can be trained end-to-end with reinforcement learning (RL).

Indeed, this design is akin to architectures typically used in approaches that learn control policies end-to-end with RL, e.g., [98]. While conceptually appealing, the latter has two main challenges in practice. First, training is computationally

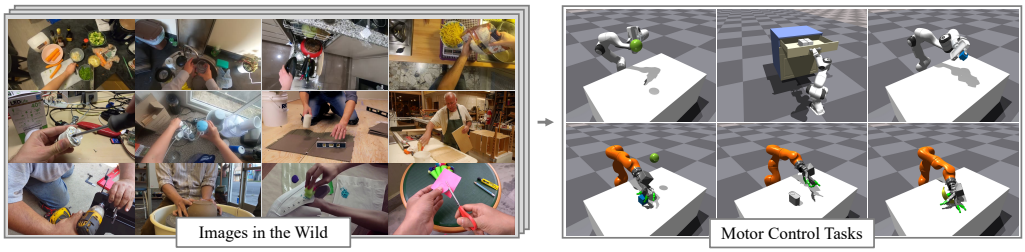
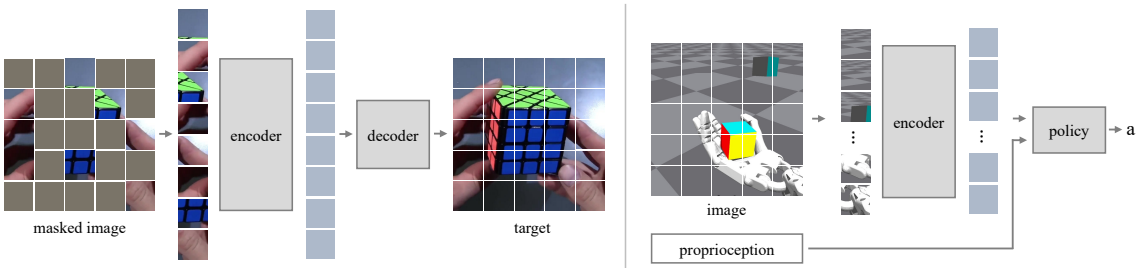


Figure 4.1: We explore learning visual representations from large scale collections of images "in the wild", e.g., from YouTube or Egocentric videos, and using them to learn to perform a range of different motor control tasks from pixels. Please see the supplementary materials for videos.

expensive and has poor sample complexity (especially with high-dimensional inputs and actions). Second, the learned solutions typically overfit to the setting at hand and thus do not generalize to new scenes and objects. One way to offset the high sample complexity of end-to-end RL is to employ auxiliary objectives [76,99–101]. For example, [101] show excellent performance in vision-based RL by using contrastive learning with data augmentations. However, such representations are still trained using only environment-specific experience.

The key aspect of our approach is in how we train the visual representations. We do *not* train the visual encoder while learning specific motor control tasks. Instead, we *pre-train* the visual encoder by *self-supervision* from *real-world images* (Figure 4.1 & 4.2). We build on the masked autoencoder (MAE) [102] work, and learn visual representation by masked image modeling. Thanks to the Internet and ubiquitous portable cameras, we now have access to large amounts of unlabeled data for self-supervision. MAE does not require human labels or make strong assumptions about data distributions, e.g., centered objects or pre-defined augmentation invariances [103], making it an excellent framework for learning general visual representations from in-the-wild images. Given the visual encoder, we train neural network controllers on top. We keep the visual representations frozen and do not perform *any* task-specific fine-tuning of the encoder; all motor control tasks use the *same* visual representations. We call our approach MVP (for **M**asked **V**isual **P**re-training for Motor Control).

We compare our self-supervised approach to baselines that follow the same architecture (Figure 4.2b) but use different visual representations. As an upper bound, we consider oracle hand-engineered states for solving a task (e.g., 3D poses and direction-to-goal vectors). We also compare our method to visual encoders trained by supervised pre-training on ImageNet [2] and CLIP [12], and state-of-the-art in domain training techniques [101, 104]. We summarize our main results as follows:



(a) masked visual pretraining (b) learning motor control

Figure 4.2: **Masked visual pre-training for motor control:** *Left:* We first *pre-train* visual representations using *self-supervision* through masked image modeling from *real-world* images. *Right:* We then *freeze* the image encoder and train task-specific controllers on top with reinforcement learning (RL). The *same* visual representations are used for all motor control tasks.

- 1) We show that a *single* visual encoder pre-trained on real-world images can solve various motor control tasks *without* fine-tuning per-task, state estimation, or demonstrations.

- 2) Our *self-supervised* approach consistently outperforms *supervised* pre-training (up to 80% absolute success rate), state-of-the-art in-domain training (up to 70% absolute success rate), and even matches the oracle performance in some cases.

- 3) We find that pre-training on *images in the wild*, e.g., from YouTube [105] or Egocentric [106, 107] videos, works better for manipulation tasks than ImageNet [2] images.

- 4) We show that our visual representations *generalize* in various ways. For example, our visual encoder disentangles shape and color and is able to handle a range of different object geometries and configurations.

4.2 Masked Visual Pre-training for Motor Control

4.2.1 Masked Visual Pre-training

We aim to leverage large amounts of visual data for learning a *general* visual representation for motor control tasks. To this end, we need a scalable learning framework that can work on natural images without human labels. Self-supervised learning, and specifically the masked image modeling through the masked autoencoder (MAE) [102] framework, naturally fits our goal. The most appealing property of MAE is its simplicity and minimal reliance on dataset-specific augmentation engineering; for example, it works well even with minimal data augmentations (center crop and

	Robosuite	MetaWorld	Ours
Simulator	MuJoCo	MuJoCo	IsaacGym
Fast			✓
#Arms	8	1	2
#Hands			✓
#Tasks	9	50	8
Rewards	✓	✓	✓

Table 4.1: **Existing benchmarks:** Compared to existing benchmarks, ours features a unique combination of hand-designed tasks, dense rewards, and complex robots (e.g., multi-finger hands). Crucially, it leverages a fast simulator and provides distributed training for scaling learning-based motor control from pixel observations.

color). MAEs mask-out random patches of the input image and reconstruct the missing pixels with a Vision Transformer (ViT) [11]. During training, only unmasked patches are fed into the MAE; this strategy makes training more efficient. It is critical to train MAE with a high masking ratio (e.g., masking 75% of all patches) and use a heavy encoder with a light decoder.

We learn visual representations from data collected from Internet and/or portable camera devices (e.g., phones, glasses, etc.). We use images from the egocentric Epic Kitchens dataset [106, 107], the YouTube 100 Days of Hands dataset [105], and the crowd-sourced Something-Something dataset [108]. Combined, these sources yield a collection of ~700K images that we refer to as the Human-Object Interaction dataset (HOI). Note that we do *not* exploit any human labels or temporal information even if it is possible. We also apply our approach using the ImageNet dataset [2] for controlled comparisons with supervised baselines.

4.2.2 Learning Motor Control from Pixels

Given the pre-trained visual encoder, we now turn to learning motor control from pixels. We freeze the visual representations and use them for all downstream motor control tasks; we do not perform any task-specific fine-tuning of the image encoder. This design has two main benefits. First, it prevents the encoder from overfitting to the setting at hand and thus preserves general visual representations for learning new tasks. Second, it leads to considerable memory and run time savings since there is no need to back-propagate through the encoder. Freezing visual encoder makes using large vision models in the RL loop fast and feasible.

In Figure 4.2b, we show our architecture for learning motor control from pixels. We first extract a fixed-sized vector of image features using our pre-trained visual

encoder. Notice that all of the image patches are passed through the encoder, unlike in the masked pre-training stage. We additionally compile proprioceptive robot information in the form of joint positions and velocities into a second vector. This proprioceptive information is readily available on real robot hardware. We concatenate these two vectors to obtain the input embedding for the neural network controller.

We then train task-specific motor control policies on top of this embedding with model-free reinforcement learning. Our policy is a small multi-layer perceptron (MLP) network. In addition, we train a critic that has the same architecture as the policy using the same representations. The policy and the critic do not share weights. We use the proximal policy optimization (PPO) algorithm [109]. PPO is a state-of-the-art policy gradient method that has shown excellent performance on complex motor control tasks and successful transfer to real hardware [110, 111]. We will show in Section 4.5.3 and 4.5.4 that MVP also works with off-policy RL algorithms [112] and imitation learning.

4.3 Benchmark Suite

We construct a new benchmark suite of tasks for studying motor control from pixels, described next.

Motivation. While there exist a number of excellent benchmarks for motor control, e.g., DMC [113], RLBench [114], Robosuite [115], MetaWorld [116], they all fall short on one or more of our requirements. In particular, there is no benchmark suite for learning motor control algorithms that has high-resolution images, realistic robots, fast physics simulation, efficient training, and appropriate reward functions and metrics. To this end, we introduce a new benchmark suite for *Pixel Motor Control*, which we call PixMC. We show the key aspects of PixMC to several existing benchmarks in Table 4.1.

Simulator. We leverage the recent NVIDIA IsaacGym simulator [117] to build our benchmark. The core design idea of IsaacGym is to perform simulation on a GPU in a shared context. IsaacGym allows for very fast training times. For example, we are able to train our oracle state-based models in ~12 minutes and our image models in ~5 hours (~8 million environment steps) on a single NVIDIA 2080 Ti GPU. This training speed is considerably faster than it would be in other simulators commonly used for motor control such as MuJoCo [118]. Rudin et al. [119] have shown that sim-to-real transfer is feasible based on policies trained in IsaacGym.

Robots. PixMC includes two robot arms, a parallel jaw gripper, and a multi-finger hand combined as follows: **(1)** *Franka:* A Franka Emika robot commonly used for

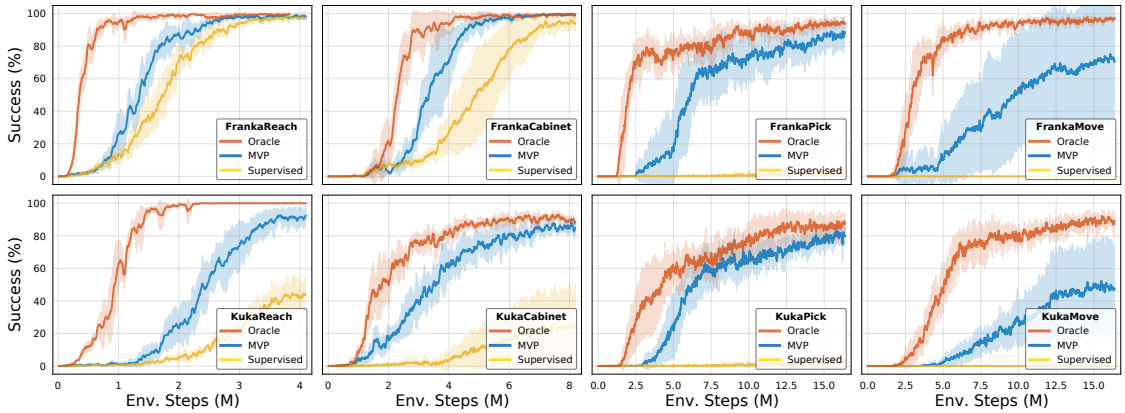


Figure 4.3: **Sample complexity:** We plot the success rate as a function of environment steps on the 8 PixMC tasks. Each task uses either the Franka arm with a parallel gripper or the Kuka arm with a multi-finger hand. The MVP approach outperforms the supervised baseline on all tasks and closely matches the oracle state model (considered the upper bound of RL) on 6 tasks at convergence. The result shows that self-supervised pre-training improves representation quality for motor control tasks.

research. It has a 7-DoF arm with a 2-DoF gripper. **(2)** *Kuka with Allegro:* A Kuka LBR iiwa arm with 7 DoFs and a 4-finger Allegro hand with 16 DoFs (4 DoFs per finger), for 23 DoFs in total. For brevity, we refer to it as "Kuka."

Observations and actions. Our benchmark supports rendering high-resolution pixel observations for each robot. For both the Franka and Kuka setups and each of the tasks, we use wrist-mounted cameras by default. The benchmark provides proprioceptive information for the robots, as well as hand-engineered states typically including 3D poses or relevant objects, goals, and their relations. All of our default settings use position control in joint angle space with a control frequency of 60Hz.

Tasks, rewards, and metrics. PixMC tasks include several movement types from basic reaching to interacting with objects. The objects in the environment vary in positions, scale, color, and shape. Figure 4.1 show a few example scenes. We hand-design task-specific dense reward functions for training RL policies. We define reward-independent success metrics that typically quantify the distance from the agent or an object to a specified goal location over sufficient time steps.

Distributed training. The scarcity of GPU memory is a major bottleneck. For our typical setup with 224×224 images, we can fit at most 256 environments on a single 2080 Ti GPU. We implement PPO with distributed training to support large batch sizes. Similar to data parallel training, we create a model replica per-GPU, collect rollouts on each GPU, and synchronize gradients across GPUs.

4.4 Experimental Setup

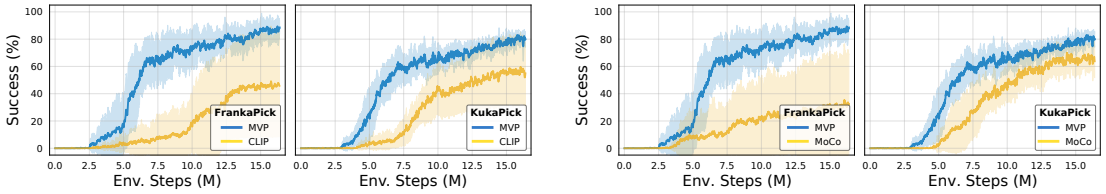
Data for pre-training. We consider two kinds of pre-training data: ImageNet [2] and a joint Human-Object Interaction (HOI) dataset. We construct the HOI data by combining Epic-Kitchens [106, 107], Something-Something [26], and 100 Days of Hands (100-DOH) [105]. To build HOI, we sample frames from Epic-Kitchens and Something-Something at 1fps and 0.3fps, respectively. This yields 700k images including 100k from 100-DOH.

Encoder. The image encoder follows standard ViT architecture [11]. We use the ViT-Small model [14] with a 16×16 patch size, 384 hidden size, 6 attention heads, an MLP multiplier of 4, and 12 blocks. The model runs at 4.6 gigaflops for input images of 224×224, approximately 1.2× as many as the ResNet-50 [17] model.

We pre-train supervised and self-supervised variants of the ViT model. For the supervised model, we use the recipe in [120] and train on the ImageNet dataset for 400 epochs. We use the MAE framework for the self-supervised counterpart [102]. We use an auxiliary dummy classification token in the MAE for downstream finetuning and transfer [102]. We use a crop ratio of [0.2, 1.0] for ImageNet and [0.1, 0.75] for HOI, due to the larger width-over-height aspect ratio of HOI images. We train the MAE models for 1600 epochs on 16 GPUs for both HOI and ImageNet datasets.

Controller. The controller is a simple MLP with each hidden layer followed by a SeLU [121] activation. We use a four-layer MLP with hidden size [256, 128, 64] for all tasks, following [117]. The (dummy) classification token of the ViT encoder yields the image features and a linear layer projects the features to 128 dimensions. The controller takes in the linearly-projected (128-d) proprioceptive state of the robot along with the projected image features. The controller outputs delta joint angles.

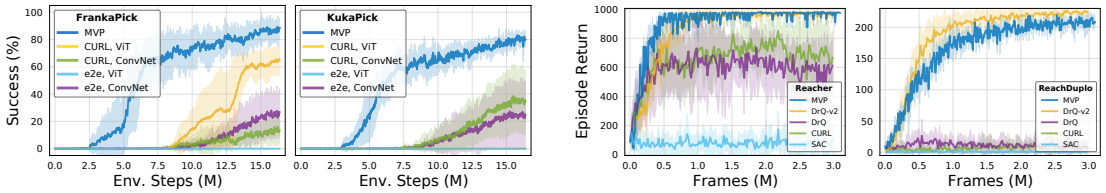
Training with RL. We freeze the visual encoder throughout the entire training horizon. We train for 500 iterations for reach, 1000 iterations for cabinet, and 2000 iterations for pick and move. In each iteration, we collect samples from 256 environments which have 32 steps each. We train for 10 epochs on these collected samples per iteration. We compose 4 minibatches per epoch, i.e., 4 gradient updates, leading to a minibatch size of 2048 per gradient update. We choose this configuration because it maximizes the memory on a single NVIDIA 2080 Ti GPU. In all experiments we train with a cosine learning rate decay schedule [26]. To reduce randomness in the RL experiments [122], for each task and model we search for the best learning rate in {0.0005, 0.001, 0.0015} with 5 seeds per learning rate (15 runs for each task and model). We always report the performance yielded by the best learning rate aggregated over seeds unless otherwise specified. Other hyperparams use defaults: Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, gradient norm of 1, noise std of 1.0.



(a) **CLIP comparison:** MAE outperforms CLIP pre-training that uses large-scale language supervision.

(b) **MoCo-v3 comparison:** MAE outperforms MoCo-v3 contrastive self-supervised pre-training.

Figure 4.4: **Pre-training framework:** MVP vs. CLIP and MoCo visual pre-training frameworks.



(a) **PixMC tasks:** MVP outperforms end-to-end training from scratch as well as CURL with both a small ConvNet and a large ViT.

(b) **DMC tasks:** Despite a larger domain gap, MVP shows strong performance using the same visual representation for different downstream tasks.

Figure 4.5: **In-domain training comparisons:** MVP vs. state-of-the-art methods on two benchmarks.

4.5 Experimental Results

4.5.1 Sample Complexity

Figure 4.3 shows success rates over training on 8 challenging tasks from PixMC. We consider the oracle state model (i.e., position, orientation, and velocity of the object, goal and robot in world-coordinate system, which is difficult to estimate in real-world settings) as the upper bound of RL. MVP significantly outperforms the supervised baseline on all tasks. At convergence, MVP closely matches the oracle state model on 6 tasks. The supervised baseline is flat at zero success rate on the pick and move tasks with both robots; MVP rivals the oracle on the pick task and achieves high success rate on the move task. These results show that self-supervised pre-training markedly improves representation quality for motor control tasks.

4.5.2 Pre-training Framework Comparison

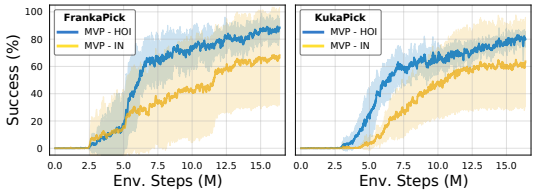
CLIP. We experiment with substituting a pre-trained CLIP visual encoder [12] in place of our MAE encoders. The CLIP pre-training framework uses 400M labeled text-image pairs and has shown excellent performance across a wide range of visual tasks. We opt to use the ViT-Base CLIP encoder as it is closest in size to our ViT-Small encoders. In Figure 4.4a we show the results. We observe a promising signal that self-supervised representations can outperform strong CLIP encoders. By inversing embedding back to images using a generative model, recent work [123] shows that from CLIP image representation to image mapping is generally one to many. This finding suggests that the inferior performance of CLIP in motor control tasks may be partly due to encoding ambiguity and failing to capture object geometries for manipulation.

MoCo-v3. We compare visual encoders trained with the Momentum Contrastive (MoCo) self-supervised learning framework instead of MAE used in MVP. We opt to use the latest MoCo-v3 [35] designed for ViT models. We show results in Figure 4.4b. We observe that MAE pre-training yields superior results than MoCo on the tasks. Furthermore, we note it may be harder to adapt contrastive self-supervised learning frameworks like MoCo-v3 to in-the-wild non-curated images.

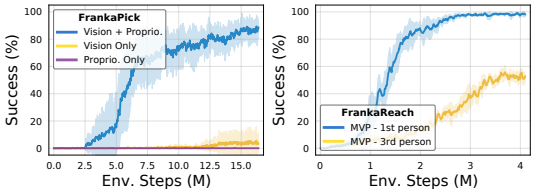
4.5.3 Comparison to In-domain Training Framework

PixMC tasks. In-domain training opts for learning visual encoders and control policies end-to-end with RL. We first compare MVP to two common setups that train the vision encoder with environment data on PixMC tasks: 1) end-to-end training from scratch, and 2) CURL [101] which uses an auxiliary training objective. We experiment with both a small ConvNet [104] and a large ViT, and note that the former is the most popular choice in previous works and the latter is identical to MVP's visual encoder. CURL is trained with PPO and MoCo-v3 [35] data augmentation recipe. Figure 4.5a shows that MVP significantly outperforms the in-domain training frameworks with both small and large networks. Note that MVP that uses a frozen backbone is much less computationally expensive compared to end-to-end training a large network (1 vs. 8 GPUs). This result highlights the superior sample complexity of MVP over in-domain training frameworks.

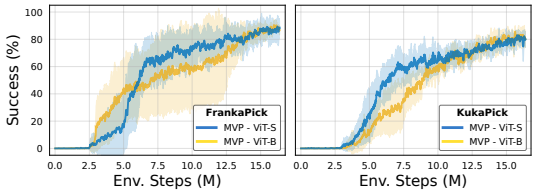
DMC tasks. We further compare to state-of-the-art in-domain training methods [100, 101, 104, 124] on the DeepMind Control Suite (DMC) [113]. We follow [104] and train MVP using DDPG [112]. As proprioception is unavailable and velocity estimation is essential for DMC tasks, we encode three frames with the visual encoder on HOI



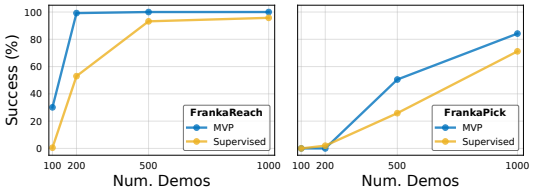
(a) **Pre-training data:** We compare using HOI and ImageNet images as the source of pre-training data. The representations learned from HOI data perform better on motor control tasks.



(c) **Sensor modalities:** We ablate proprioception input (left) and first-person vs. third-person camera setup (right). First-person camera observation combined with proprioception is superior.



(b) **Larger encoders:** We use a pre-trained ViT-Base model. We do not observe clear gains from preliminary model scaling and believe that scaling data and model size is an exciting area for future work.



(d) **Imitation learning:** We collect demos and train behavior cloning policies on top of frozen representations. MVP improves demo efficiency and success rates by up to 2x. See texts for details.

Figure 4.6: **Ablation studies:** pre-training data, encoder sizes, sensors, and action learning framework.

data and combine the features. Figure 4.5b shows that, despite pre-training and downstream domain gap, MVP significantly outperforms SAC, DrQ, and CURL baselines and is comparable to the state-of-the-art DrQ-v2. This result shows that MVP's potential in solving a diverse range of tasks *using the same backbone representation*. It also confirms that MVP works with off-policy RL algorithms.

4.5.4 Ablations

Pre-training data. We train MVP on HOI and ImageNet data, respectively. Figure 4.6a shows the results on the two pick tasks. MVP trained on HOI data outperforms the counterpart trained on ImageNet data on motor control tasks. Whereas ImageNet is dominated by images of animals and objects, HOI contains many images demonstrating object manipulation from a first-person camera view. We hypothesize that this difference is why HOI is empirically the superior choice for

motor control tasks. See the Appendix for full results.

Larger encoders. We pre-train a ViT-Base encoder (18 gigaflops) and conduct a preliminary transfer study on the Franka/Kuka pick tasks. Figure 4.6b shows the results. We observe that the larger encoder does not improve performance. A larger encoder potentially requires more data and/or a different training recipe. Overall, scaling data and models in the context of self-supervised representations for motor control remains an exciting area for future work.

Sensor modalities. We ablate removing visual observation *or* proprioception input (Figure 4.6c, left), and first-person vs. third-person camera setup (Figure 4.6c, right). The result shows that proprioception is essential with vision and first-person camera significantly improves sample efficiency.

Imitation learning. We experiment with imitation learning in replacement of RL for learning control policies. We collect demos using a pre-trained expert policy. We divide demos into subsets of various sizes and train behavior cloning on top of frozen representations. Figure 4.6d shows success rates of MVP and the supervised baseline at different numbers of demos. MVP improves demo efficiency by ~1.7x to 2x (horizontal comparison, $\geq$90% success rate in reach; $\geq$ 50% success rate in pick), and success rates by a relative 2x (vertical comparison, 200 demos in reach; 500 demos in pick). This set of imitation learning results is identical to it of RL in terms of ranking order, and thus reaffirms the effectiveness of MVP.

4.5.5 Representation Analysis

Shape and color disentanglement. In the pick task the robots are trained to pick up a blue box of 4.5cm side length. In this experiment we add a distractor object that differs from the training object in terms of color, shape, or scale, *at test time* without additional training. Specifically, we have 1) a *green* box of 4.5cm side length (color distractor); 2) a blue *sphere* of 4.5cm diameter (shape distractor); or 3) a blue box of *6cm side length* (scale distractor). Figure 4.7 shows the results of our MVP model. Color and shape distractors only marginally decrease the success rate, suggesting that MVP is able to disentangle the color and shape of objects. The model, however, is less sensitive to scale variation as the 50% success rate suggests that the distractor or the original box is picked up by chance. We believe it is due to scale ambiguity from single first-person camera setup.

Diverse objects. To understand how diverse are the object representations captures by our model, we import various objects from the YCB dataset [125]—box, can, mug, and banana—for the pick task and re-train the model for each individual object. Figure 4.8 visualizes the experiment setup and shows the results. The Kuka robot

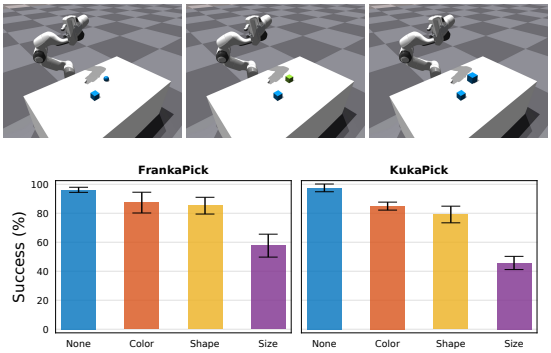


Figure 4.7: **Disentangles shape and color:** The robots are trained to pick up a blue box of 4.5cm side length. At *test time*, we add a distractor in terms of color (blue vs. green), shape (cube vs. sphere), or size (4.5cm vs. 6cm), shown at the top. MVP maintains high success rates for color and shape, whereas the scale ambiguity is likely due to single first-person camera setup.

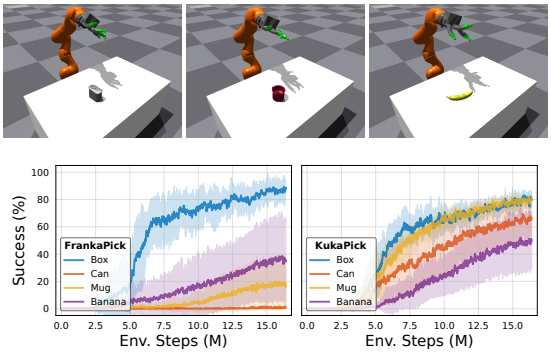


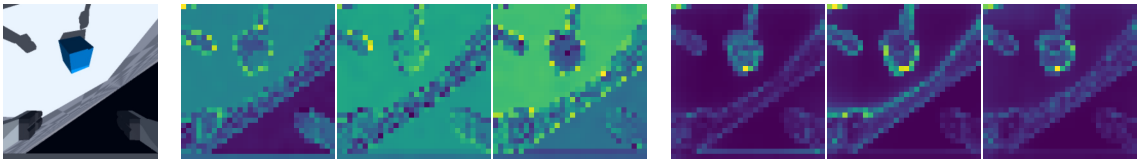
Figure 4.8: **Handles objects of various shapes:** We import three additional objects (i.e., can, mug, and banana) from the YCB dataset and re-train the controller to pick up the individual object category. The Kuka robot with the Allegro hand can pick up all objects with at least 50% success rate. These results highlight the generality of our visual representations.

with the Allegro hand can pick up all of the objects with at least a 50% success rate. This shows MVP representations can capture objects of different geometries.

Feature visualization. We visualize attention maps for different models in Figure 4.9. We observe that, in contrast to the supervised recognition model (b), our self-supervised model (c) has a notion of objects. Although it may not be necessary for single object classification, as local texture may largely suffice, objectness is essential for motor control tasks that require precise spatial structure. The representation of objects emerges automatically as a byproduct of masked prediction.

4.6 Related Work

Self-supervised pre-training in computer vision. Self-supervised learning has been gaining momentum in computer vision. The approaches often rely on pretext tasks for pre-training [61–63, 66, 83, 84]. More recently, contrastive learning methods, e.g., [67, 69, 70, 76, 126–128], have been popular. These techniques try to learn to be invariant to a set of hand-crafted augmentations. Xiao et al. [103] have shown that the augmentations introduce inductive bias and may harm downstream transfer. Masked image autoencoding [48, 102, 129] pursues a different direction by learning



(a) Image (b) Supervised (c) MVP

Figure 4.9: **Attention maps visualization.** We observe that MVP models (c) capture a notion of objects which is not represented by the supervised recognition models (b). Note that our approach is self-supervised and the representations emerges automatically. Interestingly, the results are surprisingly consistent across the different attention heads (3 shown here).

to recover masked pixels. Specifically, the Masked Autoencoders (MAE) [102] have shown excellent performance on recognition tasks. We adopt the MAE as our visual pre-training strategy for learning motor control.

RL with self-supervision. One way to overcome the high sample complexity of RL is to employ auxiliary objectives, where in addition to learning the task, a model learns to predict some property of the environment (e.g., depth) that potentially help representations [99, 130–132]. Beyond hand-designed environment properties, researchers have explored using more general self-supervised objectives [76, 100, 101]. For example, Srinivas et al. [101] show excellent performance in vision-based RL tasks. Representations can also be pre-trained on the data from the environment [101, 133]. Overall, we share the goal of learning good visual representations with self-supervision. In contrast, we learn representations from large collections of natural images rather than environment-specific experience.

Self-supervision in robotics. Self-supervised learning has also been used in various robotic settings. Pinto and Gupta [134] and Agrawal et al. [135] learn representations through interaction. Sermanet et al. [136] learn representations from multiview video using contrastive learning and use them for imitation learning. Florence et al. [137] learn dense image descriptors with self-supervision. Zhan et al. [138] learn robotic manipulation with RAD [139] and CURL [101]. Pari et al. [140] show the effectiveness of visual representations with non-parametric nearest neighbor controllers. All of these approaches learn representations in a specific robotic setting of interest (e.g., videos in the lab), rather than general visual representations from image collections like ours.

Supervised pre-training. Sax et al. [141] and Chen et al. [142] show that representations learned from performing a set of mid-level vision tasks using label supervision benefits downstream navigation and manipulation tasks, respectively. Zhou et al. [143] show the effectiveness on visual representations in driving settings.

Lin et al. [144] transfer image models trained on supervised vision tasks, e.g., edge detection and semantic segmentation, to affordance prediction models for object manipulation. Shah and Kumar [145] use ImageNet representations for dexterous manipulation. In contrast to all of these, our approach is self-supervised and does not rely on labeled datasets.

Chapter 5

Real-World Robot Learning with Masked Visual Pre-training

5.1 Introduction

Learning representations with large neural networks is the powerhouse of modern deep learning. This has enabled impressive results in computer vision [1, 95], natural language processing [6, 8, 9], and audio generation [97, 146]. How can we transfer the success stories of representation learning to robotics? We can approach this from two ends: shared representations on the perception side or shared representations on the action side. Our focus in this paper is on shared *visual* representations.

Of course, the devil is in the details. Recent developments in the field of visual learning have made this more feasible: (1) the use of diverse, real-world data from the Internet and egocentric videos, (2) self-supervised objectives that do not overly rely on data augmentations or other forms of strong human-designed priors, (3) scalable and high-capacity transformer models [7, 11], and (4) training of control policies on top of frozen visual representations. In our recent work [147], we have shown that this recipe for self-supervised visual pre-training is effective for motor control in simulation.

In this paper, we show that this framework is effective for real-world robotic tasks as well (Figure 5.1). We build on our prior work, but make significant advances in terms of data scale and diversity (7× larger), model size (15× bigger), and real-world experiments (extensive real robot evaluations).

In particular, we train *self-supervised* visual representations on real-world images and videos from the Internet [2, 26, 105] and egocentric video datasets [106, 148]. We leverage the masked autoencoders [102] that learn visual representations by masked prediction. The hope is that, by learning to predict the missing content in

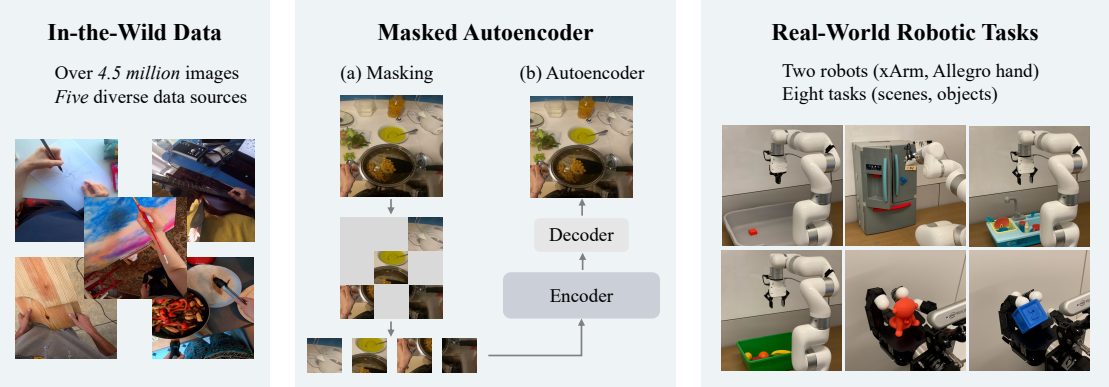


Figure 5.1: **Real-world robot learning with masked visual pre-training.** We learn visual representations from a massive collection of Internet and egocentric data. We pre-train representations with masked image modeling, freeze the encoder, and learn control policies for robotic tasks on top.

real-world images, the model will learn useful properties of the visual world that will enable it to learn to perform real-world robotic tasks. Given the pre-trained vision encoder, we *freeze* the encoder and learn control policies on top. The *same* visual representations are used for all downstream robotic tasks and embodiments. We focus on efficient real-world learning through behavior cloning with a handful of human-provided demonstrations per task (20 - 80).

We evaluate our approach in an extensive real-world study and report results from 981 real-world experiments. We consider basic motor control tasks (reach, push, pick), as well as tasks with variations in scenes and objects (Figure 5.1, right). We find that our approach achieves considerably higher performance than CLIP (up to 75%), supervised pre-training (up to 81%), and training from scratch (up to 81%). Furthermore, we observe that our representations lead to large improvements in sample complexity, reaching the strongest baseline performance with half the number of demonstrations.

In addition, we demonstrate the benefits of scaling visual pre-training for robotics by training a 307M parameter vision encoder [11] on a massive collection of 4.5M images from ImageNet [2], Epic Kitchens [107], Something Something [26], 100 Days of Hands [105], and Ego4D [148] datasets. Importantly, we observe that it is not sufficient to scale the model alone and that larger models require bigger datasets. To the best of our knowledge, ours is the largest vision model deployed for robotics, and demonstrates clearly the benefits of visual pre-training scale for robot learning.

5.2 Related Work

End-to-end control is concerned with learning to predict robot actions (*e.g.*, joint velocities, end-effector poses, etc) directly from observations [98, 149, 150], without the need to perform explicit 3D pose estimation [110], grasp planning [151], and motion planning [152]. However, these end-to-end approaches tend to be too sample inefficient for real-world training. Some works have tried to find a balance between these explicitly pipelined approaches and end-to-end approaches [153–155].

Supervised pre-training for robotics learns one or more pretext tasks through strong supervision and then transfers the representations to downstream robotic tasks. Lin et al. [144] shows that representations learned from semantic tasks such as detection and segmentation correlate with affordance maps for object manipulation. [156] use language-supervised CLIP model [12] for learning language-conditioned imitation policy. In concurrent work, [157] explore pre-training visual representations using time contrastive learning and language descriptions from human annotators. These methods all require expert labels or cross-domain supervision.

Self-supervised learning in robotics has been explored as a means of improving sample efficiency. Examples include: learning a dynamic model from interaction with environments [135]; learning visual representation from interaction with environments [158]; learning vision-based policies on self-collected trajectories [134, 159]; learning visual autoencoders on trajectories [160]; learning spatiotemporal representations through videos [136, 161]; learning visual correspondence [162]; utilizing non-parametric nearest-neighbor retrieval [140]; and conducting visual self-supervised learning on pre-collected demonstrations [138]. These methods require in-domain data collection, and thus may be difficult to extend beyond the training environment and task. In contrast, our approach uses a large-scale and diverse collection of real-world images and videos, making it more generalizable.

5.3 Framework

5.3.1 Masked Visual Pre-training

Data collection. We first compile a large-scale dataset for learning visual representations. We primarily use Ego4D [148], a massive scale, egocentric dataset from nine countries recorded via portable devices, covering over 3,670 hours of daily-life activities. We combine the Ego4D data with the ImageNet [2], as well as the Hand-object Interaction (HoI) data used in [147], which comprises of the egocentric Epic Kitchens [107] dataset, the YouTube 100 Days of Hands dataset [105], and

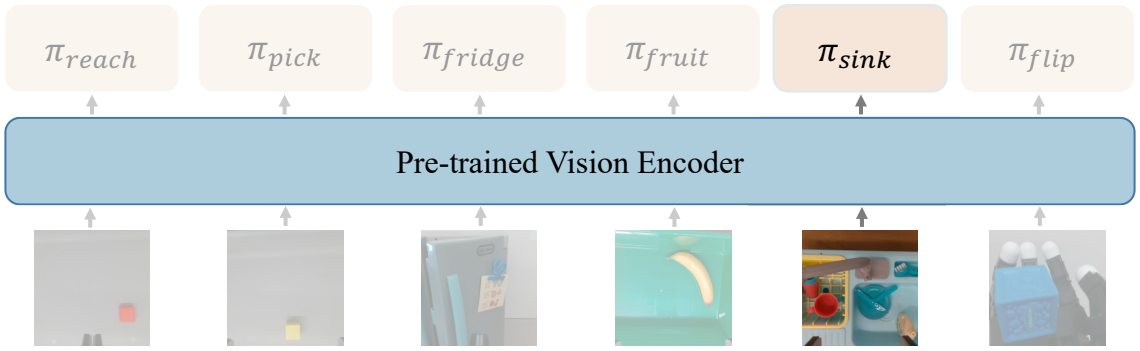


Figure 5.2: **One encoder for all robots and tasks.** We train control policies per task, on top of the frozen encoder. The same vision encoder is used for all downstream robotic tasks and embodiments.

the crowd-sourced Something-Something dataset [26]. Our training data totals 4.5 million images, 6.5x of the HoI data. We find that a sufficiently large and diverse pre-training dataset to perform the mask image modeling self-supervisory task is critical to scale up the vision backbone for real robot tasks.

Self-supervised objective. At the core of our self-supervised representation learning approach is masked image modeling via the masked autoencoders (MAE) [102]. MAE masks out random patches in an image and reconstructs the missing content with a vision transformer (ViT) [11]. A high masking ratio, *e.g.*, 75%, and asymmetrical heavy-encoder light-decoder design, are important for learning good visual representations efficiently. Simple and free from dataset or task-specific augmentations [103], MAE is the state-of-the-art self-supervised framework in computer vision [163–166], and has been demonstrated to work well for motor control tasks in simulation as well [147].

Architecture. We use the ViT models as our vision encoders. While the MAE-trained ViT models yield improving performance in vision tasks as model sizes grow [11, 102, 167], previous work [147] does not show improvement from switching a ViT-Small model to the ViT-Base counterpart of 4x as many parameters. In this chapter, we scale the model up to the ViT-Large and deploy it on the real robot. The model contains 307M parameters and runs at ~64 gigaflops at input size 224×224, approximately 15x as many as the commonly adopted ResNet-50 [17], the largest vision model deployed for robotics. As we will show in the experiments, scaling model sizes while training on sufficiently large data leads to consistent performance improvement on downstream robotic tasks.

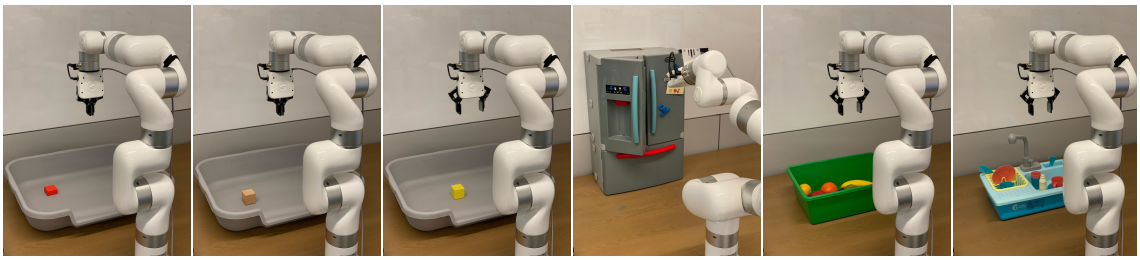


Figure 5.3: **Real-world robotic tasks.** We perform extensive real robot evaluations using a 7 DoF robot arm with a parallel jaw gripper. Our tasks include basic motor control skills (reaching a red block, pushing a wooden cube, and picking a yellow cube), variations in scenes (closing a fridge), objects (picking fruits), and scenes and objects (picking a detergent bottle from a cluttered sink).

5.3.2 Real-World Robot Learning

We learn to perform real-robot tasks through behavior cloning (BC). We collect demonstrations containing trajectories of RGB images from a wrist-mounted camera and the robot's joint state at each time step. For most of the tasks, we use the motion-tracked HTC Vive VR system to control the end-effector. For some tasks that are difficult to demonstrate via the motion controller, *e.g.*, closing fridge door, we use kinesthetic teaching. Given the recorded demonstrations, we train a control policy that takes in the input image features and proprioceptive states (joint positions) at time step t and predicts the action at time step $t + 1$. We perform joint position control; we do not use any end-effector information (*e.g.*, the 6-DoF pose). We build on our MVP pipeline [147] and freeze the image encoder throughout the policy learning, which prevents large pre-trained encoders from overfitting to a specific setting or task, and greatly reduces GPU memory footprint and training time.

5.4 Experimental Setup

In this section we provide implementation details for our approach and describe our evaluation setup.

Data. We extract frames from Ego4D, Epic Kitchens, and Something-Something at 0.2 fps, 1fps and 0.3fps, respectively. We then combine the Ego4d with ImageNet and the YouTube 100 Days of Hands dataset. This process yields 2.6M frames from Ego4D, 1.2M images from ImageNet, and 700k HoI images from the rest, a total of 4.5M images. We term the combined dataset as "Ego" for abbreviation. Note that [147] only uses the 700k HoI images, excluding the Ego4D and ImageNet.

Encoders. We use the standard Vision Transformer (ViT) architecture as the image encoder. We use three models of various sizes: ViT-Small [14], ViT-Base, and ViT-Large models, of 22M, 86M, and 307M parameters, respectively. The ViT-Small model is approximately the same size as the ResNet-50 model, while the ViT-Large model has ~15x as many parameters as the ResNet-50 model.

Pre-training. We pre-train the models via the MAE framework [102]. The training recipe closely follows [102], with dataset specific settings from [147]. We use the auxiliary dummy classification token for transferring to downstream robotics tasks. We train the MAE models for 400 epochs for the combined Ego dataset; 1600 epochs for the HOI dataset; and 1600 epochs for ImageNet dataset. We use the pre-training recipe in [120] for the study that involves ImageNet supervised models.

Controllers. The controller takes in both image features and the robot's proprioceptive state. We use joint positions as the proprioceptive state *without* explicitly appending the end-effector pose to the state. We do not use velocity in the state as our low-cost arm does not support true velocity sensing. The controller outputs *delta joint angles*. The controller's design closely follows [117], *i.e.*, a four-layer MLP with a SeLU [121] activation following each hidden layer. The hidden size is [256, 128, 64] for most tasks and [512, 256, 128] for the `PickSink` task. We linearly project the image features and the proprioceptive states to a joint embedding space as the controller's input.

Robot and robotic tasks. We use the low-cost UFACTORY xArm7 robot (a 7-DoF arm) and a 1-DoF parallel jaw gripper. We use the arm's maximum control frequency of 5 Hz for both collecting demonstrations and control. We use a first-person wrist-mounted RealSense camera for all tasks. We do *not* use depth information from the camera. We consider basic motor control tasks, *i.e.*, `ReachBlock`, `PushCube`, and `PickCube`, and more challenging in-context tasks, *i.e.*, `CloseFridge`, `PickFruit`, and `PickSink`. The tasks are shown in Figure 5.3 (more details in the next section).

Demonstrations. We collect 80 demonstrations per task. We use the motion-tracked HTC Vive VR system for most tasks, except for `CloseFridge` we use kinematics teaching. We use trajectory replay on the robot for trajectory pruning. We do *not* use the key-frame information for the learning.

Evaluation. We systematically sweep across 16 variations of the environment, *e.g.*, shifting the target object. For consistency and reliability of the study, we use the *same* 16 variations for all models in an individual task, and evaluate models *sequentially* at each variation, in order for similar lighting conditions, robot conditions, precise object initial locations, etc (see also Appendix B).

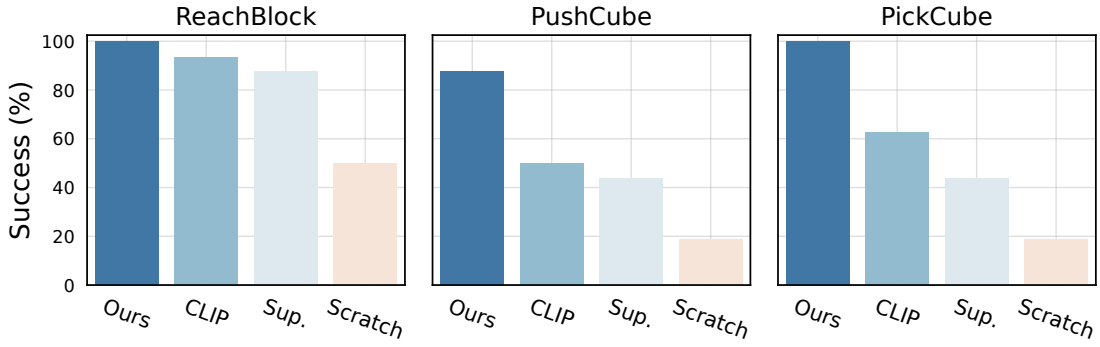


Figure 5.4: **Comparison to vision encoders.** We compare our approach to visual encoders trained with CLIP, supervised learning on the ImageNet, and from scratch on the task at hand. In all cases, we observe that our approach consistently outperforms the baselines by a considerable margin.

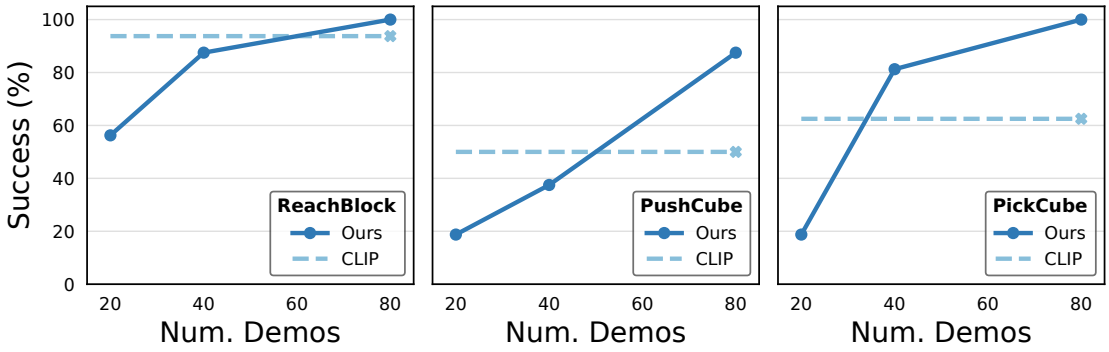


Figure 5.5: **Sample complexity.** We show the performance of our approach as the number of demonstrations varies from 20 to 80. CLIP performance at 80 demonstrations is shown with a dashed lined for reference. Our approach is comparable to CLIP using only half the number of demonstrations.

5.5 Experimental Results

We perform extensive evaluations across a range of visual backbones, real-world robotic tasks, objects, and environments (Figure 5.3). In total, we report results from 981 real-world experiments.

5.5.1 Basic Motor Control

We evaluate three basic motor control tasks in visually simple contexts: reaching a red block, pushing a wooden cube, and picking a yellow cube (see Figure 5.3 for task visualization). These tasks serve as stepping stones for more complex tasks, and the visual representations that can potentially be fundamental for robotics should learn these tasks efficiently. In all cases, we randomize the initial object and robot positions (see Appendix B for details about the learning and task setup).

Comparison to various vision encoders. In Figure 5.4 we compare our approach to a set of state-of-the-art vision backbones: CLIP [8] trained on 400M text-image pairs, supervised model trained on the ImageNet, and a model trained from scratch with in-domain demonstration data. For fair comparisons, we use the ViT-Base [11] vision encoder for all methods. We empirically observe that the CLIP encoder performs the best among the baselines, and the ranking order is consistent across the benchmark tasks. Our approach consistently outperforms the baselines by a considerable margin.

Sample complexity. In Figure 5.5 we study the performance of our approach as the number of demonstrations varies from 20 to 80. For reference, we show the performance of the most competitive baseline, CLIP, trained with 80 demonstrations (dashed horizontal line). In aggregate, we observe that our approach reaches CLIP performance while using 50% fewer demonstrations. This result is a promising signal for using our models for learning more complex robotic tasks, as discussed next.

5.5.2 Visually Diverse Scenes and Objects

We have previously shown that our approach can learn basic motor control tasks, such as reaching, pushing, and picking, at a higher success rate and better sample complexity than the baseline vision backbones. To focus on the motor control aspect, we used a visually simple environment and basic objects. However, one of the main potential benefits of our approach is that learning visual representations from diverse, real-world images may enable to solve robotic tasks that involve the interaction with everyday objects in more visually complex environments. In this subsection, we evaluate our visual representations on robotic tasks with variations in scenes and objects in more realistic setups.

Scene context. We first evaluate our approach in a more realistic scene by considering the task of closing the door of a toy fridge that is left open (termed `CloseFridge`). We randomize the location of the fridge, which side of the door to close, and initial angle of the door. As shown in Figure 5.6, bottom-left, the initial configuration of the fridge and the robot vary considerably, which is quite common in everyday settings. Figure 5.6, top-left, shows that our approach outperforms all baselines.

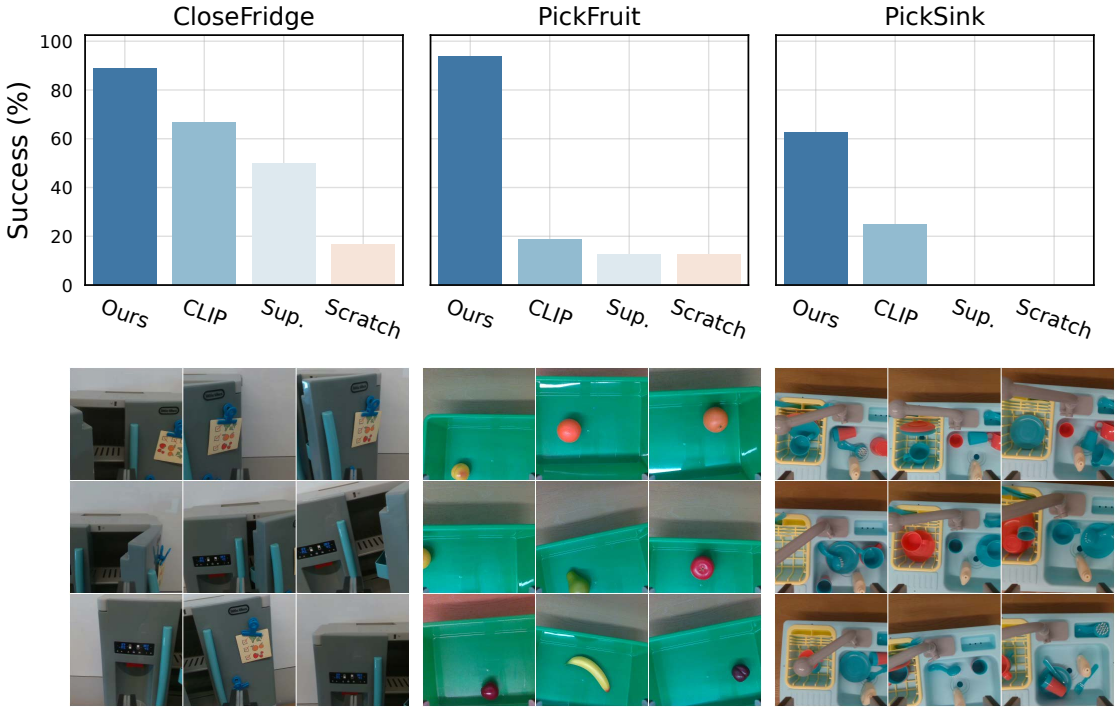


Figure 5.6: **Variations in scenes and objects.** We compare our approach to CLIP on tasks with variations in scenes (closing the fridge), objects (picking fruits), and scenes and objects (picking an object from a cluttered sink). The models are ViT-Base. Our approach considerably outperforms CLIP and the gap is larger than in simpler settings (see Figure 5.4). This may suggest that our representations capture more precise spatial structure that is helpful for robotic tasks in more realistic contexts.

Different objects. Next, we evaluate on a task with a variety of objects. The goal is to pick up eight different fruits that vary in color, shape, and size (termed `PickFruit`). In each trial, a fruit is selected at random, and both the fruit and the robot's positions are randomized. The number of training demonstrations remain unchanged, *i.e.*, we provide 10 demonstrations for each fruit. In Figure 5.6, middle, we show the evaluation results (top) and starting configuration samples (bottom). We see that baselines struggles in this setting while our approach achieves nearly perfect score.

Objects in context. Finally, we evaluate our approach on a task that features interacting with objects in everyday contexts. We task the robot with picking a detergent bottle from a cluttered sink (termed `PickSink`). The task is challenging as the visual configuration of the scene, such as the toy plates, mug cups, and

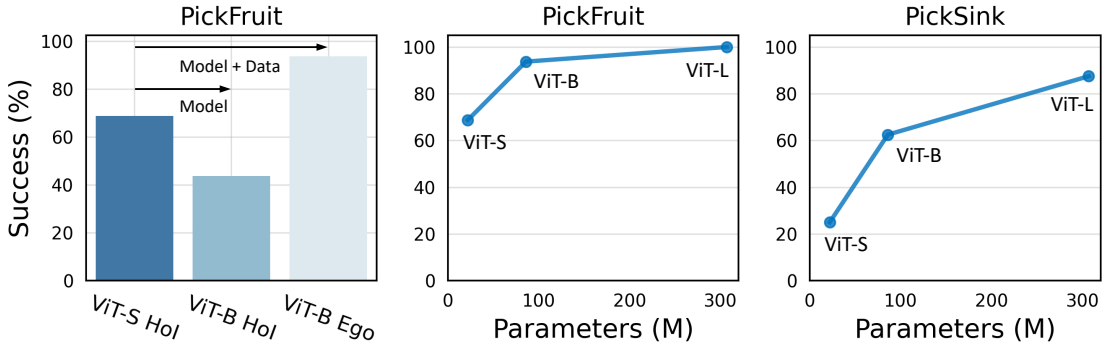


Figure 5.7: **Scaling model and data.** We study the scaling properties of our approach. We observe that scaling the model size alone from ViT-S to ViT-B while keeping the dataset fixed (HoI image collection; see text for details) does not improve the performance and even hurts (left). However, when we scale both the model and data (our massive Ego image collection; see text for details) we see clear benefits from a larger model. The trend continues when going further from the 86M ViT-B to the 307M ViT-L model (middle & right). Moreover, the gains are larger for harder tasks (right).

silverware, can vary in unlimited ways, as shown in Figure 5.6 right We observe that our approach considerably outperforms baseline approaches using the same ViT-B encoder.

5.5.3 Scaling Model and Data Size

Importantly, our visual pre-training approach uses a self-supervised objective [102] that makes few assumptions about the data distribution, and does not rely on human-designed pretext tasks such as data augmentations. Therefore, the framework is well-suited for pre-training from a massive collection of unlabeled and in-the-wild visual data. Here we study scaling model and data size.

We first consider increasing the model capacity. In Figure 5.7, left, we see that increasing the model size (~4.5x) from ViT-S to ViT-B, while keeping the data size fixed (HOI image collection [147]), does not increase performance and even hurts. This is consistent with the in-simulation results reported in [147]. However, if we also scale the data size from HOI to our massive Ego data collection, ViT-B yields better results. These results suggests that we must scale *both* the model and the data.

In Figure 5.7, middle & right, we show the performance as a function of model size. Additionally increasing the model size from the 86M parameter ViT-B to the 307M parameter ViT-L leads to further improvements. The gain is larger for the

	supervision	params (M)	success (%)
R3M	video-text	23	31.3
CLIP	image-text	86	18.8
Ours	image-only	22	68.8
		86	93.8
		307	100.0

Table 5.1: **Comparison to concurrent work.** All of our vision models, trained with image-only self-supervision, considerably outperform the strongest available R3M model trained on paired video-language labels from Ego4D. The gains are larger for larger models. Evaluated on the PickFruit task.

visually more challenging task (`PickSink`). To the best of our knowledge, our work is the largest vision model deployed to real robot tasks, which clearly demonstrates the benefits of scaling visual pre-training for real-world robot learning.

5.5.4 Comparison to Concurrent Work

We compare our approach to a concurrent work [157], submitted to the same conference (CoRL 2022). Similarly, it pre-trains visual representations on in-the-wild video data from Ego4D [148]. However, it relies on paired language-video annotations that are available as part of Ego4D. In contrast, our approach is fully self-supervised and makes minimal assumptions about the data distribution, enabling us to leverage massive collections of uncurated data (*e.g.*, from the Internet). In Table 5.1, we compare our models to the strongest available R3M ResNet-50 model. We observe that our medium-sized ViT-B model outperforms R3M ResNet-50 by a large margin (93.8% *vs*. 31.3%). We also see that our smallest ViT-S model from [147] outperforms it as well (68.8% *vs*. 31.3%).

5.5.5 Ablation Studies

We conduct ablation studies on camera setups, input modality, training settings, model architectures for ours and baselines, as well as downstream finetuning. Results in Figure 5.8 and discussed next.

Cameras. We compare using the first-person wrist camera with the third-person table camera (Figure 5.8, top-left). The wrist-mounted camera yields significantly better results than the third-person camera. During the evaluation trials we observe that the third-person camera model struggles with fine-grained localization, and the farther the object is from the camera, the worse the predicted trajectory is. This

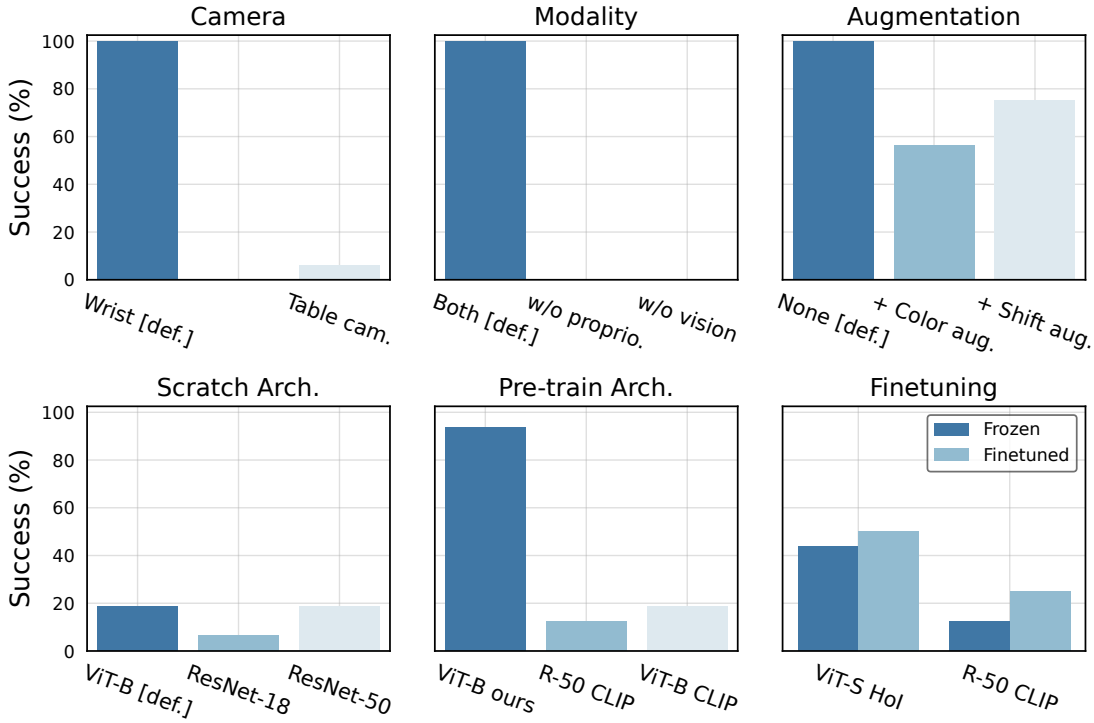


Figure 5.8: **Ablation studies.** We conduct ablation studies on the camera setups; input modality; commonly used image augmentations; from-scratch training architecture; CLIP pre-training architecture, and end-to-end finetuning on downstream tasks. See Section 5.5.5 text for more details.

suggests that the model may be confused by the scale (size) ambiguity in the setting because we do not use depth, and the third-person observation does not change as the robot acts.

Modality. We experiment with removing proprioception states or images from the model's input (Figure 5.8, top-center); both yield zero success rate, as neither is fully-observed input for the task.

Augmentations. We experiment with adding common data augmentations for training the task policy (Figure 5.8, top-right). We do not observe any benefits from these augmentations likely since our vision encoder is frozen and overfits significantly less from training on limited amount of data.

Training-from-scratch architectures. We compare various model architectures of smaller sizes for from-scratch training (Figure 5.8, bottom-right). ResNet-18 and ResNet-50 models, which are of 11% and 22% of the referenced ViT-B model (FLOP-

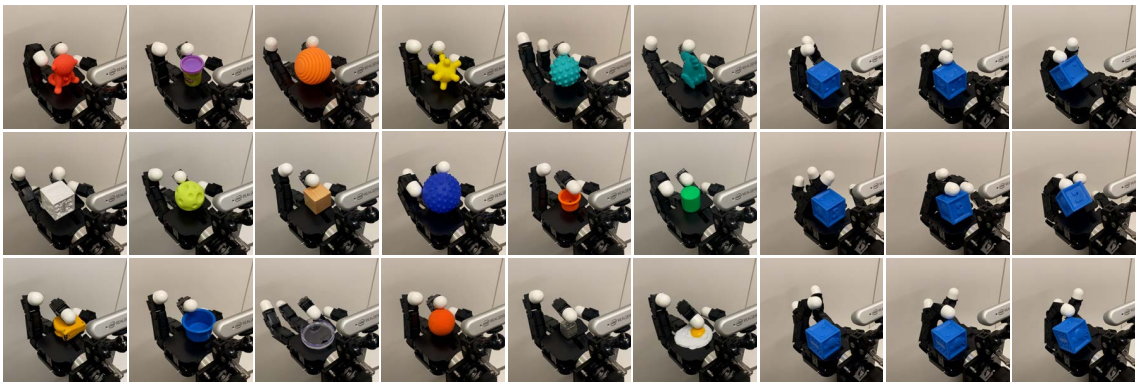


Figure 5.9: **Multi-finger hand.** We show that our framework readily generalizes to a different robot morphology. We experiment with finger reaching, using seen and unseen objects, and cube flipping.

wise), respectively, do not yield better result.

CLIP pre-training architectures. We compare the publicly-released CLIP ResNet-50 and ViT-B models. The CLIP ViT-B model yields better performance for the robotics tasks. This result aligns with the ranking order of the CLIP models' computer vision task performance as reported in [12].

Finetuning. We finetune the ViT-S HoI model and the CLIP ResNet-50 model as they are relatively smaller models that achieve modest performance on downstream tasks. Finetuning these models marginally improves the performance, suggesting that finetuning alone on limited number of demonstrations does not close the gap with superior visual representations from pre-training.

5.5.6 Case Study: Multi-finger Hand

Our approach makes no assumptions about the downstream robotic tasks or embodiments. In particular, our policies take pixel images as input and predict joint position angles as actions (rather than, *e.g.*, end-effector pose). In this subsection, we test the generality of our approach by applying it for downstream tasks with a multi-finger Allegro hand (see Appendix A for more details on the setup).

Visual reaching. We design a reaching task in which the hand learns to reach the top of an object with the tip of the index finger. The object's position is randomized across the palm. We provide 10 demonstrations for each of the 8 different objects. At test time, we evaluate the trained policy on the 8 seen objects as well as 45 unseen objects (see Figure 5.9). The success rate is ~50% across both.

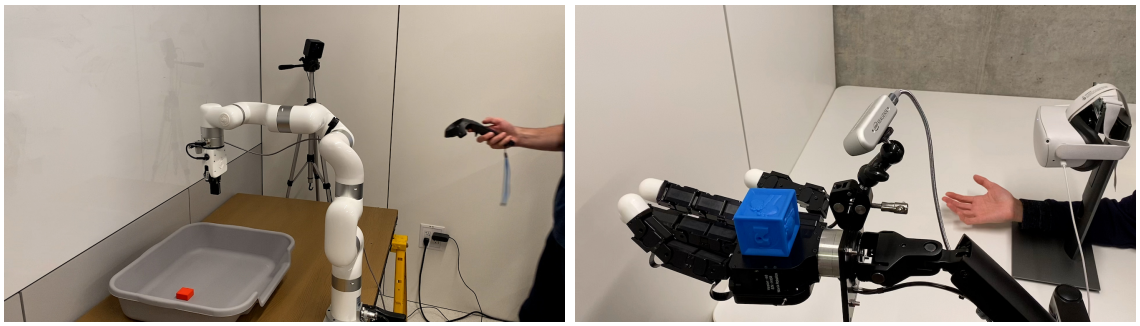


Figure 5.10: **Data collection.** We show our setup for collecting demonstrations in the real world with human operation. *Left:* We collect xArm demonstrations using an HTC Vive VR controller. *Right:* We collect Allegro hand demonstrations using an Meta Quest 2 device. See text for more details.

Visual flipping. Next, we consider a cube flipping task in which the goal is to flip a rubber cube that is placed in the palm. The position of the cube is randomized across the palm. Thus, the policy must rely on visual cues to accomplish the task. In aggregate, we observe a success rate of 50% across 30 trials. See Figure 5.9 for example key frames and also check out the videos on the project page.

Understanding vision through action. Training policies on top of frozen visual representations enables us to perform studies to understand what the pre-trained visual representations utilize for downstream tasks. Here we first train a policy to reach a yellow cube, but test with objects of different shape and color. First, we find that when given the same shape of different color (wooden cube) or same color and different shape (yellow ball), the policy reaches for the object. Next, when given both the yellow cube and a distractor it reaches for the yellow cube. Finally, when given an object of different shape and color (blue cube) the hand stays still.

5.5.7 Data Collection

We describe our setup for collecting demonstrations for the xArm and the Allegro hand (Figure 5.10).

xArm demonstrations. To collect demonstrations for the xArm, we use an HTC Vive VR controller (Figure 5.10, left). The setup includes two external sensors that track the position of the VR controller and estimate its 6-DoF pose. Given the 6-DoF pose of the controller, we use it to control the end-effector (EE) pose of the robot. Specifically, we map the EE pose to the joint angles via inverse kinematics (IK) following [168]. We control the gripper open/close state via a button press on the controller. Using this pipeline, the user controls the system in real-time while

having direct view of the robot. While the user is performing the task, we save the camera feed and robot state information as demonstrations. On average, it takes about 1 hr to collect 40 demonstrations using this setup.

Allegro hand demonstrations. We would like to have a similar setup for collecting demonstrations for the Allegro hand. However, it is hard to control a multi-finger hand using a joystick or a VR controller. We thus require a different user interface. To this end, we develop a new approach based on the Meta Quest 2 device (Figure 5.10, right). In particular, we use the hand tracking provided by the Quest 2 to obtain the 3D keypoints of the human hand. We then map the human keypoints to the Allegro hand joint angles using IK following [169]. As before, our system runs in real-time and records demonstration data while the human is controlling the robot to complete the desired task.

We note that our demonstration recording approach relies exclusively on consumer grade VR software and hardware. Namely, VR controller tracking in the case of the HTC Vive and hand tracking in the case of the Meta Quest 2. This has three desirable properties for research use. First, it is likely more accurate and reliable than research prototypes. Second, it is likely to improve with future software updates. Finally, it is easy to acquire and set up by other researchers around the world.

5.5.8 Evaluation Protocol

Our goal is to conduct fair evaluation of different pre-training methods and control for any confounding factors that might impact the performance. In the early stages of this project, we observed that the performance of the same model varies non-trivially based on the robot up time, *i.e.*, after running for many hours our low-cost robot arm starts to execute actions less precisely. Similarly, the difference in lighting conditions may impact the performance of different methods. Therefore, we employ a systematic evaluation protocol. At data collection and training time, we randomize the initial object and robot positions. At test time, we fix the robot position to the center of the randomization space, and slide the object along K pre-defined and carefully-measured locations (we use $k = 16$ in most experiments). For each location of the target object, we benchmark a set of models *sequentially*, so that each model manipulates the object at the exact same location and at roughly the same time. We then move the object to the next location and repeat the evaluation step. Overall, this procedure ensures the models are benchmarked with the similar robot and lighting conditions.

5.6 Conclusion

We explore learning visual representations from a massive collection of real-world data and using them for downstream robotic tasks. We pre-train representations with masked modeling, freeze the encoder, and learn control policies on top. We perform extensive evaluations in the real world and show that, across various robotic tasks, our approach leads to higher success rate and better sample complexity than CLIP, supervised ImageNet pre-training, and training from scratch. We further demonstrate the benefits of scaling the model and data size for real world robot learning.

Chapter 6

Learning Humanoid Locomotion with Transformers

6.1 Introduction

We study the problem of humanoid locomotion. This setting poses both hardware and control challenges. Our focus in this work is on learning-based control. Still, many challenges remain: unsafe exploration, inaccurate simulation, hybrid nonlinear control, and high-dimensional under-actuated system.

There is a rich body of literature on control theory and optimization that has shown excellent results [170, 171]. Perhaps the most well-known are the videos of the Boston Dynamics Atlas robot doing back flips, jumping over obstacles, dancing, and doing all of that while making it look easy [172].

While these approaches have made great progress, learning-based methods have become of increasing interest due to their ability to learn from diverse simulations or real environments. For example, learning-based approaches have proven very effective in dexterous manipulation [110, 111], quadrupedal locomotion [173–175], and bipedal locomotion [176–178]. There have been approaches that combine learning components with model-based controllers for humanoids as well [179, 180]. Moreover, it is possible that the existing proprietary controllers (e.g., from Agility Robotics) use at least some learning components. Yet, there has not been published work on purely learning-based humanoid locomotion to date.

In this chapter, we propose a learning-based approach for humanoid locomotion. We present a Transformer-based controller that predicts future actions autoregressively from the history of past observations and actions (Figure 6.2). Our model is trained with large-scale reinforcement learning (RL) on an ensemble of randomized environments in simulation and deployed to the real world in a zero-shot fashion.

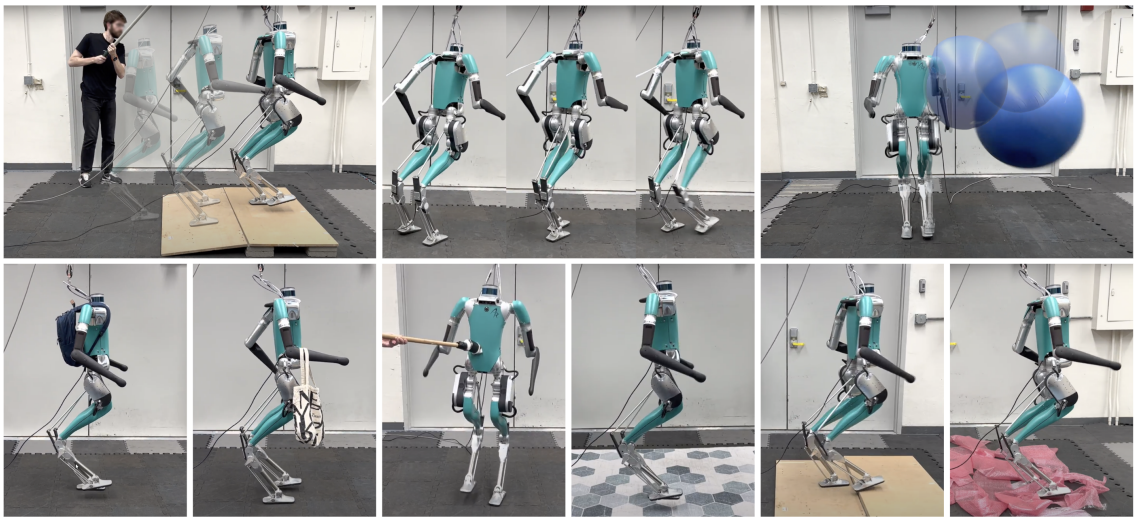


Figure 6.1: **Humanoid locomotion.** We present a learning-based approach for humanoid locomotion and evaluate it on a full-sized real-world Digit robot. Our policies are trained entirely in simulation and successfully transferred to real hardware zero-shot. Our robot can adapt to external disturbances such as carrying a backpack or a handbag; being pushed by a stick, pulled by cables, or having a yoga ball thrown at it. Moreover, it can walk over terrains with different friction, texture, and geometry.

We hypothesize that the history of observations and actions implicitly encodes the information about the world that a powerful Transformer model can use to adapt its behavior dynamically at test time. For example, the model can use the history of desired *vs*. actual states to figure out how to adjust its actions to better achieve future states. This can be seen as a form of in-context learning—changing model behavior without updating the model parameters—often found in large Transformer models like GPT-3 [6].

We evaluate our approach on a full-sized Digit humanoid robot. We find that our approach outperforms standard neural network model choices like Multi-Layer Perceptron (MLP), convolutional neural network (CNN), and Long Short-Term Memory (LSTM) in a high-fidelity simulator. We further show successful zero-shot transfer to hardware. Our model can handle external disturbances like carrying payloads and walking over different terrains (Figure 6.1).

6.2 Related Work

Humanoid Locomotion Humanoid locomotion has been a longstanding challenge in robotics. Roboticists designed the first full-sized real-world humanoid robot [181] in 1970s. Since then, researchers have developed a variety of humanoid robots to push the limits of robot locomotion research [182–185]. While classic control methods can achieve stable and robust locomotion [170, 186–188], optimization-based strategies show the advantage of simultaneously authoring dynamic behaviors and obeying constraints [171,172,189,190]. Meanwhile, since the 1990s, researchers have developed and applied RL-based approaches to robot locomotion [191], which have demonstrated remarkably robust and dynamic locomotion behaviors. For example, these approaches have enabled quadrupeds to traverse challenging terrain [173–175, 192], while also empowering bipeds with robust and diverse locomotion gaits [163, 176–178, 193]. In this chapter, we employ a RL-based strategy for humanoid locomotion, i.e., on a Digit robot. Recent works on Digit humanoid locomotion that involve RL [179, 180] are integrated with a model-based control design. In contrast, we approach humanoid locomotion from an end-to-end learning paradigm.

Domain Randomization One way to approach the sim-to-real problem is to perform RL with domain randomization [194–197]. The idea is to train an agent to be robust in a range of simulated environments, with the expectation that this will enable it to perform well in the real world, which can be seen as just another type of environment.

Adaptation Instead of learning to be robust to different variations of the environment, we can learn to adapt. One way to achieve this is through memory-based models like LSTMs which have been effective for dexterous manipulation [111] and bipedal locomotion [178]. An alternative approach is to learn an estimator of the latent environment properties to facilitate adaptation [174, 175], which has enabled quadrupeds to traverse diverse terrains in the wild. We hypothesize that our Transformer-based controller can learn to adapt in-context from the history of observations and actions.

Transformers in Robotics Over the last several years, we have witnessed impressive advances in natural language processing [6, 8, 9] and computer vision [11, 198] powered by large Transformer models. Recently, Transformers have been used in robotics as well. Example use cases include visual pre-training [199], object-centric representations [200], depth fusion [201], and language-conditioned behavior cloning [202, 203]. Likewise, we share the goal of using Transformer models for robotics. In contrast, our Transformer-based controllers are trained with online reinforcement learning without offline datasets, and our focus is on real-world humanoid locomotion.

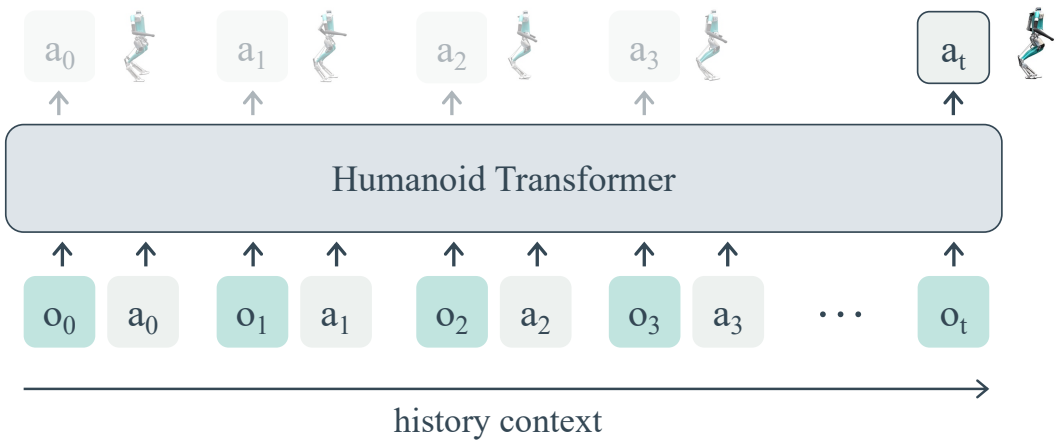


Figure 6.2: **Humanoid Transformer.** Our neural network controller is a causal Transformer model trained by autoregressive prediction of future actions from the history of observations and actions. We hypothesize that the observation-action history contains useful information about the world that a powerful Transformer model can leverage to adjust its actions in-context.

6.3 Method

We formulate the control problem as a Markov Decision Process (MDP), which provides a mathematical framework for modeling discrete-time decision-making processes with partially stochastic outcomes. The MDP comprises the following elements: a state space S, an action space A, a transition function $P(s_{t+1}|s_t, a_t)$, which determines the probability of transitioning from state s_t to s_{t+1} after taking action a_t at time step t, and a scalar reward function $R(s_{t+1}|s_t, a_t)$, which assigns a scalar value to each state-action-state transition, serving as feedback to the agent on the quality of its actions. Our approach to solving the MDP problem is through Reinforcement Learning (RL), which aims to find an optimal policy that maximizes the expected cumulative reward over a finite or infinite horizon.

In practice, estimating true underlying state of an environment is impossible for real-world applications. In the presence of a noisy observation space, the MDP framework needs to be modified to reflect the uncertainty in the observations. This can be done by introducing an observation space O and an observation function $Z(o_t|s_t)$, which determines the probability of observing state s_t as o_t. The MDP now becomes a Partially Observable Markov Decision Process (POMDP), where the agent must make decisions based on its noisy observations rather than the true state of the environment. The composition of the action, observation and state spaces is described in Section 6.4.4.

We illustrate our framework in Figure 6.2 and provide a comprehensive description of the method below.

6.3.1 Model Architecture

Our aim is to find a policy π_o for real-world deployment in the POMDP problem. Our policy takes as input a history trajectory of observation-action pairs over a context window of length l, represented as $o_t, a_{t-1}, o_{t-1}, a_{t-2}, ..., o_{t-l+1}, a_{t-l}$, and outputs the next action a_t. To achieve this, we utilize Transformers [7] for sequential trajectory modeling and action prediction.

Transformers are a type of neural network architecture that has been widely used in sequential modeling tasks, such as natural language processing [6, 8, 9], audio processing [204], and increasingly in computer vision [11, 198] as well. The key feature of Transformers is the use of a self-attention mechanism, which allows the model to weigh the importance of each input element in computing the output. The self-attention mechanism is implemented through a self-attention function, which takes as input a set of queries Q, keys K, and values V and outputs a weighted sum, computed as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{6.1}$$

where d_k is the dimensionality of the key. The self-attention mechanism enables the Transformer to capture long-range dependencies between input elements.

We represent each observation-action pair in the locomotion trajectory as a token. Transformers are able to extract the structural information of these tokens through a repeated process of assigning weights to each token (softmax on Q and K) in time, and mapping the tokens (V) into features spaces, effectively highlighting relevant observations and actions and thus enabling the inference of important information such as gait and contact states. We employ Multi-Layer Perceptrons (MLPs) to embed each observation-action pair into a feature space. To capture the positional information of each token in the sequence, we add sinusoidal positional encodings to the features. We leverage the temporal dependencies among the observations and actions by restricting the self-attention mechanism to only attend to preceding tokens, resulting in a Causal Transformer [8].

Transformers have proven to be effective in the realm of in-context learning, where a model's behavior can be dynamically adjusted based on the information present in its context window. Unlike gradient-based methods that require fine-tuning on task-specific data samples and explicit state estimation methods that rely on

pre-defined state spaces, Transformers can learn in-context, providing them with the flexibility to handle diverse inputs.

6.3.2 State-policy Supervision

In Reinforcement Learning (RL), an agent must continuously gather experience through trial-and-error and update its policy in order to optimize the decision-making process. However, this process can be challenging, particularly in complex and high-dimensional environments, where obtaining a useful reward signal may require a significant number of interactions and simulation steps. Through our investigation, we found that directly optimizing a policy using RL in observation space is slow and resource-intensive, due to limited sample efficiency, which impairs our iteration cycles.

To overcome these limitations, we adopt a two-step approach. First, we assume that the environment is fully observable and train a state policy $\pi_s(a_t|s_t)$ using simulation. This training is fast and resource-efficient, and we tune the reward functions, such as gait-parameters, until an optimal state policy is obtained in simulation. Next, we distill the learned state policy to an observation policy through Kullback-Leibler (KL) divergence. This approach shares some similarities with the teacher-student paradigm utilized in [174, 175], with the most notable difference being that we do not impose a bottleneck representation or explicitly design a state space.

6.3.3 Joint Optimization with Reinforcement Learning

The discrepancy between the state space and the observation space can result in suboptimal decision-making if relying solely on state-policy supervision, as policies based on these separate spaces may have different reward manifolds with respect to the state and observation representations. To overcome this issue, we utilize a joint optimization approach combining RL loss with state-policy supervision. The objective function is defined as:

$$L(\pi_o) = L_{RL}(\pi_o) + \lambda D_{KL}(\pi_o \parallel \pi_s) \tag{6.2}$$

where λ is a weighting factor representing the state-policy supervision, $L_{RL}(\pi_o)$ is the RL loss, and $D_{KL}(\pi_o \parallel \pi_s)$ is the KL divergence between the observation policy π_o and the state policy π_s. The weighting factor λ is gradually annealed to zero over the course of the training process, typically reaching zero at the mid-point of the training horizon. It is important to note that our approach does not require any

pre-computed trajectories or offline datasets, as both the state-policy supervision and RL-supervision are optimized through on-policy learning.

6.4 Experimental Setup

6.4.1 Digit Humanoid Robot

Digit is a general-purpose humanoid robot developed by Agility Robotics, standing at approximately 1.6 meters tall with a total weight of 45 kilograms. The robot's floating-base model is equipped with 30 degrees of freedom, including four actuated joints in each arm and eight joints in each leg, of which six are actuated. The passive joints, the shin and tarsus, are designed to be connected through the use of leaf springs and a four-bar linkage mechanism, while the toe joint is actuated by means of rods attached at the tarsus joint. Digit robot has been used as a humanoid platform for mechanical design [205], locomotion control [180, 206, 207], state estimation [208], planning [209–211], etc.

6.4.2 Simulation Environment

In our simulation environment, we use the Isaac Gym simulator [117,119] to model the rigid-body and contact dynamics of the Digit robot. Given the closed kinematic chains and non-fully actuated nature of the knee-shin-tarsus and tarsus-toe joints of the robot, Isaac Gym is unable to effectively model these dynamics. To address this limitation, we introduce a novel "virtual spring" model with high stiffness to represent the rods. We apply forces calculated from the spring's deviation from its nominal length to the rigid bodies. Additionally, we employ an alternating simulation sub-step method to quickly correct the length of the virtual springs to their nominal values. We found that these efforts collectively make sim-to-real transfer feasible. To the best of our knowledge, this represents the first successful attempt to perform sim-to-real transfer from Isaac Gym to an under-actuated full-sized humanoid robot.

We randomize various elements in the simulation, including dynamics properties of the robot, control parameters, and environment physics, as well as adding noise and delay to the observations. Table 6.1 summarizes the domain randomization items and the corresponding ranges and distributions.

For the robot's walking environment, we randomize the terrain types, which include smooth planes, rough planes, and smooth slopes. The robot executes a variety of walking commands such as walking forward, sideward, turning, or a combination thereof, which are randomly resampled at a fixed interval. We set the commands

below a small cut-off threshold to zero. Table 6.2 lists the ranges of the commands used in our training process.

6.4.3 Reward

Our reward is a combination of the following terms:

- Linear velocity tracking reward (r_{lv}): This reward tracks the targets of forward and sideway walking velocity.

$$r_{lv} \coloneqq \exp(-||v_{xy} - v^*_{xy}||^2_2/\sigma_{xy}), \tag{6.3}$$

  where v_{xy} and v^*_{xy} represent the realized and commanded base linear velocity, respectively. We set σ_{xy} to 0.2.

- Angular velocity tracking reward (r_{av}): This reward tracks the target of turning velocity,

$$r_{av} \coloneqq \exp(-(\omega_z - \omega^*_z)^2/\sigma_\omega), \tag{6.4}$$

  where ω_z and ω^*_z represent the realized and commanded base angular velocity along the z-axis, respectively. We set σ_ω to 0.2.

- Base motion reward (r_{bm}): This reward penalizes the vertical and roll-pitch motions of the robot's base.

$$r_{bm} \coloneqq v^2_z + 0.5 * ||\omega_{xy}||^2_2, \tag{6.5}$$

  where v_z is the base linear velocity on the z-axis and ω_{xy} is the base's roll-pitch velocity.

- Base orientation reward (r_{bo}): This reward penalizes the base's orientation of the robot.

$$r_{bo} \coloneqq ||g_{xy}||^2_2, \tag{6.6}$$

  where g_{xy} is the x-and-y component of the projected gravity vector.

- Base height reward (r_{bh}): This reward penalizes overly low base height.

$$r_b \coloneqq \begin{cases} (h_l - h)^2 & h < h_l \\ 0 & h \geq h_l, \end{cases} \tag{6.7}$$

  where h_l and h represent the lower limit and actual base height, respectively. In this study, h_l was set to 1.0 meter.

Parameter	Unit	Range	Operator	Distribution
Joint Position	rad	[0.0, 0.175]	additive	Gaussian
Joint Velocity	rad/s	[0.0, 0.15]	additive	Gaussian
Base Lin. Vel.	m/s	[0.0, 0.15]	additive	Gaussian
Base Ang. Vel.	rad/s	[0.0, 0.15]	additive	Gaussian
Gravity Projection	-	[0.0, 0.075]	additive	Gaussian
Observation Delay	B(p)×dt	[0.0, 0.2]	-	uniform
Action Delay	B(p)×dt	[0.0, 0.2]	-	uniform
Motor Offset	rad	[0.0, 0.035]	additive	uniform
Motor Strength	%	[0.85, 1.15]	scaling	uniform
Motor Damping	Nms/rad	[0.3, 4.0]	scaling	loguniform
Mass	kg	[0.5, 1.5]	scaling	uniform
Kp Factor	%	[0.9, 1.1]	scaling	uniform
Kd Factor	%	[0.9, 1.1]	scaling	uniform
Gravity	m/s^2	[0.0, 0.67]	additive	uniform
Friction	-	[0.3, 2.0]	scaling	uniform
Restitution	-	[0.0, 0.4]	additive	uniform

Table 6.1: **Domain randomization.** We show the types of randomization and their corresponding ranges. Additive randomization adds a value drawn from a specified range to the original value. Scaling randomization multiplies the original value by a value drawn from a specified range. The range is specified as a lower and upper bound for a uniform distribution or as a mean and std for a Gaussian distribution.

- In-the-air reward (r_{air}): penalizes the robot for lifting both feet off the ground.

$$r_{air} \coloneqq \mathbb{1}[\sum_{i \in \text{foot}} \mathbb{1}[||F(i)||_2 > 0] = 0], \tag{6.8}$$

  where F represents the contact forces on the robot's foot.

- Foot contact force reward (r_f): This reward penalizes overly high contact force on the robot's feet.

$$r_f \coloneqq \sum_{i \in \text{foot}} \begin{cases} ||F(i) - f_{max}||_2 & ||F(i)||_2 > f_{max} \\ 0 & ||F(i)||_2 \leq f_{max}, \end{cases} \tag{6.9}$$

  where f_{max} represents the maximum foot contact force threshold and is set as a hyperparameter.

Parameter	Unit	Range	Cut-off	Change Interval
Forward Speed	m/s	[-0.3, 1.0]	0.10	10 sec.
Sideway Speed	m/s	[-0.3, 0.3]	0.10	10 sec.
Turn Speed	rad/s	[-1.0, 1.0]	0.26	10 sec.

Table 6.2: **Input commands**. We independently generate commands for forward/backward walking, sideway walking, and turning. The input is set to zero if the sampled value falls below the specified threshold. The commands are re-sampled periodically at fixed intervals.

- Footswing trajectory tracking (r_{fs}): This reward penalizes the deviation of the footswing trajectory from heuristic trajectories.

$$\begin{aligned} r_{fs} \coloneqq & \sum_{i \in \text{foot}} ||\text{foot\_traj}_{i,xy} - \text{foot\_traj}^*_{i,xy}||_2^2 \\ & + 5.0 * \sum_{i \in \text{foot}} (\text{foot\_traj}_{i,h} - \text{foot\_traj}^*_{i,h})^2, \end{aligned} \tag{6.10}$$

  where foot_traj_{xy} and foot_traj_h are x-y and z-component of heuristic foot trajectories. The x-y component uses Raibert heuristics [170] and the z-component uses von Mises distributions ($\kappa = 0.04$).

- Joint torques reward (r_τ): This reward penalizes output torques to prevent hardware damage and reduce energy consumption.

$$r_\tau \coloneqq ||\tau||_2^2, \tag{6.11}$$

  where τ is the joint torques.

- Joint acceleration reward (r_{acc}): This reward penalizes joint accelerations to reduce shakiness.

$$r_{acc} \coloneqq ||\ddot{q}||_2^2, \tag{6.12}$$

  there $\ddot{q}$ is the joint acceleration.

- Action rate reward (r_a): This reward penalizes excessive changes in consecutive actions.

$$r_a \coloneqq ||a_t - a_{t-1}||_2^2, \tag{6.13}$$

  where a_t represents the predicted actions at time step t.

Input	Dimensionality	π_o (Actor)	π_s and Critic
Base Linear Velocity	3	✓	✓
Base Angular Velocity	3	✓	✓
Joint Positions	26	✓	✓
Joint Velocities	26	✓	✓
Projected Gravity	3	✓	✓
Clock Input	2	✓	✓
Commands	3	✓	✓
Gait Heuristics	6		✓
Height Map	121		✓
Diff. Noisy Actions	36		✓
Diff. Noisy Obs.	61		✓
Robot, Env. Params.	147		✓
Kp, Kd	40		✓
Gravity	3		✓

Table 6.3: **Observation and state spaces.** The state policy's actor and the critics of both policies utilize states as input.

- Joint target smoothing (r_s): This reward penalizes sudden changes in predicted joint targets.

$$r_s \coloneqq ||q'_t - q'_{t-1}||_2^2 + ||q'_t - 2q'_{t-1} + q'_{t-2}||_2^2, \tag{6.14}$$

  where q'_t represents the joint target at time step t.

- Selected joint position penalty (r_{jp}): This reward penalizes deviations from a "neutral" joint position for selected joints.

$$r_{jp} \coloneqq \sum_{j \in M} \alpha_j (q(j) - q_0(j))^2, \tag{6.15}$$

  where $q(j)$ and $q_0(j)$ represent the current and neutral joint positions of joint j, respectively. The joints penalized include shoulder's roll and yaw with a weight of $\alpha = 2.0$, elbow with a weight of $\alpha = 1.0$, hip's roll and yaw with a weight of $\alpha = 0.5$, and shoulder and hip's pitch with a weight of $\alpha = 0.1$.

- Termination reward (r_k): penalizes self-collision or base collision that immediately terminates an episode.

$$r_k \coloneqq \mathbb{1}[\text{self\_collision}||\text{base\_collision}] \tag{6.16}$$

Parameter	Value
Number of GPUs	4 A100s
Number of Environments	8192 (π_s) / 4096 (π_o)
Learning Epochs	5
Steps per Environment	24
Minibatch Size	49152 (π_s) / 24576 (π_o)
Episode Length	20 seconds
Discount Factor (γ)	0.99
Generalised Advantage Estimation (λ)	0.95
Entropy Regularization Coefficient	0.001
PPO Clipping Parameter	0.2
Optimizer	AdamW
Learning Rate (Actor)	5e-4
Learning Rate (Critic)	5e-4
Learning Rate Schedule (Actor)	cosine
Learning Rate Schedule (Critic)	constant
Weight Decay	0.01
Training Iterations	6000
Normalize Input	True
Normalize Value	True

Table 6.4: **Hyperparameters of PPO.**

6.4.4 Reinforcement Learning Algorithm

We use the proximal policy optimization (PPO) algorithm [109] for training RL policies. We use the actor-critic method and do not share weights. We always feed states into the critic [212]. Table 6.4 shows the hyperparameters used in our experiments. Table 6.3 lists the composition of the state and observation spaces. We predict PD targets and include P and D gains of the leg motors in the action space. We do not train the policy to control the four toe motors, and instead we set the motors as their default positions using fixed PD gains.

6.4.5 Neural Network Model

The Transformer model used in this study has four blocks, each of which has an embedding dimension of 192 and employs a multi-head attention mechanism with 4 heads. The MLP ratio of the transformer is set to 2.0. The hidden size of the MLP

	Success Rate (%)	Max Cmd. Vel. (m/s)
MLP	50	1.0
CNN	50	0.8
LSTM	60	0.3
Ours	65	1.0
Company	70	1.0

Table 6.5: **Evaluation results in simulation.** We compare our approach to an MLP baseline, a CNN baseline, an LSTM baseline, and the company controller developed by Agility robotics. We observe that our approach outperforms the neural network baselines considerably and shows respectable performance compared to the state-of-the-art company controller.

for projecting input observations is [512, 512]. The action prediction component of the model uses an MLP with hidden sizes of [256, 128]. Overall, the model contains 1.4M parameters. We use a context window of 16. The state model component is composed of an MLP with hidden sizes of [512, 512, 256, 128].

6.5 Experiments

In this section, we evaluate our learning-based approach in a high-fidelity simulator and in the real world.

6.5.1 Simulation Experiments

We begin by evaluating our approach in high fidelity AR-Sim simulator developed by Agility robotics. This enables us to evaluate unsafe controllers and control for factors of variations. Unlike the Isaac Gym simulator that was used for training, AR-Sim accurately simulates the dynamics and physical properties of the Digit robot.

We compare our proposed method with three neural network-based baselines: 1) a Multi-Layer Perceptron (MLP), 2) a Convolutional Neural Network (CNN), and 3) a Long Short-Term Memory (LSTM). In addition, we compare it to 4) a proprietary controller developed by Agility Robotics (Company). We select the MLP, CNN, and LSTM neural network baselines as they are commonly used for learning-based robot locomotion. On the other hand, we use the proprietary Company controller as a baseline to demonstrate the state-of-the-art performance of model-based control for the Digit robot. To ensure a fair comparison, we disabled the perception module of the Company controller in AR-Sim during the evaluation, as both the neural

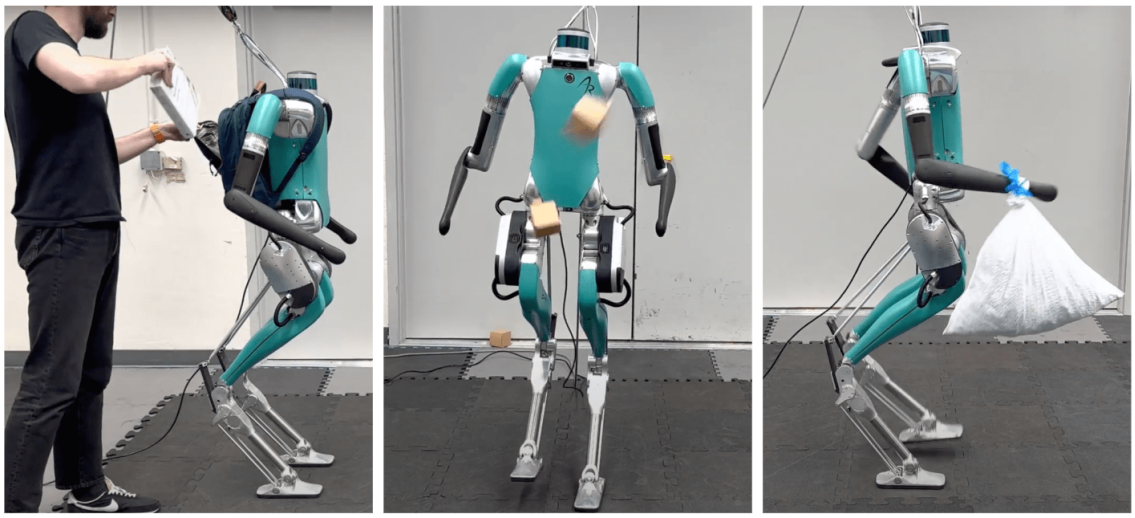


Figure 6.3: **External disturbance.** We include carrying constant loads and withstanding external forces. See also Figure 6.1.

network-based baselines and our proposed method operate in a blind manner.

In Table 6.5, we show the results on two tasks. First, we consider a walking experiment in a challenging environment containing steps of increasing height (ranging from 4 cm to 16 cm) that were not present during the training phase. We then command the robot to walk forward at 0.3 m/s and measure the success rate across five trials for each of the methods. We observe that our approach outperforms the neural-network based baselines and shows respectable performance compared to the state-of-the-art company proprietary controller. Next, we compare different methods in terms of the maximal command speed they are able to track. In particular, we command different desired velocities and record the highest velocity that a method is able to track without falling. Note that the actual achieved velocities may still differ from the commanded velocities. We observe that the MLP, ours, and the Agility controller are able to walk with 1.0 m/s commands while the CNN and the LSTM require lower commands.

6.5.2 Real-World Experiments

Next, we evaluate our Transformer-based controller on real hardware. We find that other neural network baselines were not stable enough to transfer to real hardware.

External Disturbance

Robustness and adaptation to external disturbances are critical requirements for real-world deployment of a humanoid robot. To evaluate these qualities, we conducted a series of experiments in the laboratory environment to test the performance of our proposed controller. In Figure 6.3, we present example photographs from these experiments . These experiments can be broadly categorized into two groups: 1) carrying loads, and 2) subjected to external forces.

In the first group of experiments, we evaluate the robot's ability to carry loads of varying mass, shape, and center of mass while walking forward. We conduct five experiments, each with the robot carrying a different type of load: an empty backpack, a loaded backpack, a cloth handbag, a loaded trash bag, and a paper bag. Our results demonstrate that the robot is able to successfully complete its walking route while carrying each of these loads. Notably, our Transformer-based controller is able to adapt to the presence of a loaded trash bag attached to its arm, despite the reliance of our policy on arm swing movements for balancing (as described in Section 6.5.2). This suggests that our controller is able to adapt its behavior according to the context and overcome the external disturbance presented in the experiment.

In the second group of experiments, we test the robot's ability to handle sudden external forces while standing still or walking. These experiments include being pushed with a wooden stick or being hit with a thrown yoga ball while standing, and being pulled from the back with cables while walking forward. Our results show that the robot's behavior is able to stabilize in each of these scenarios. Given that these disturbances are sudden and require fast changes in behavior, especially in the case of a humanoid, to prevent the robot from falling, this suggests that our transformer-based controller is able to infer the necessary adaptations from the context.

Rough Terrain

In addition to handling external disturbances, a humanoid robot must also be able to navigate various terrains. To assess the capabilities of our transformer-based controller in this regard, we conducted a series of experiments on different terrains in the laboratory (Figure 6.4). Each experiment involved commanding the robot to walk forward at a constant velocity of 0.15 m/s.

Next, we covered the floor with four different types of items: rubbers, cloths, cables, and bubble wraps, which altered the roughness of the terrain and could potentially lead to challenging entanglement and slipping situations, as the robot does not utilize exteroceptive sensing. Despite these difficulties, our controller was

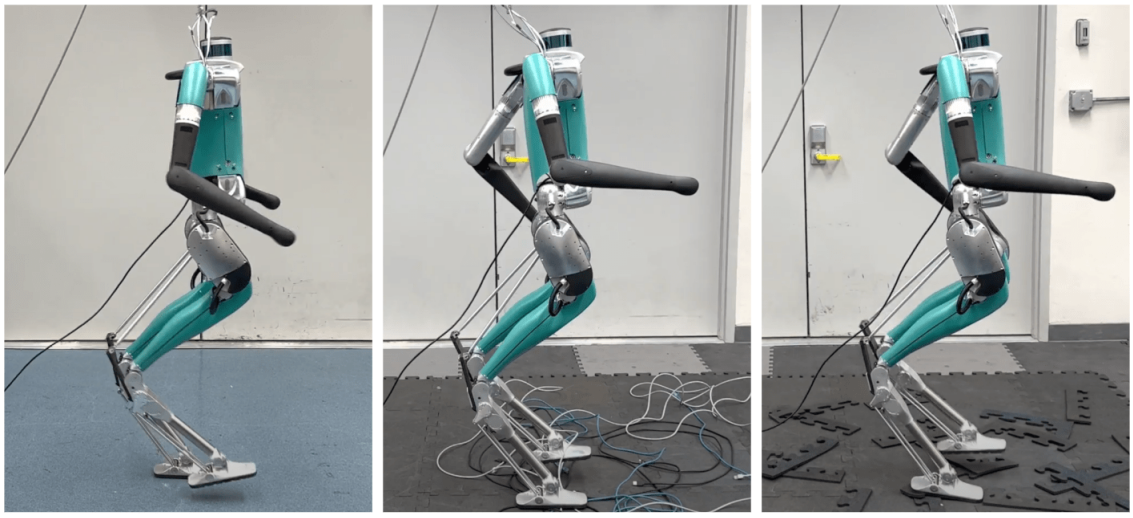


Figure 6.4: **Rough terrain.** The controller undergoes tests on eight different types of challenging terrain in the laboratory, with three being depicted in the figure (slippery surfaces, cables, and rubber). The robot is instructed to walk forward at a constant velocity of 0.15 m/s. See also Figure 6.1.

able to traverse all three terrain types.

Finally, we evaluated the controller's performance on two different slope difficulties. The simulated training slopes are up to 10% (percentage), and our testing slopes are up to 8.7%. Our results demonstrate that the robot was able to successfully cross both slopes, with higher confidence at higher velocity (0.2 m/s) on steeper slopes.

Adaptation

In this section, we further investigate the in-context adaptation ability of our Transformer-based controller for humanoid locomotion. To do this, we evaluate the performance of the controller on a novel terrain type that was not encountered during training. In particular, we test the robot's ability to climb small steps.

It is important to note that while our controller was trained on rough terrains and slopes in simulation, it was not exposed to steps or any terrains that involve discrete changes in height. We command it to walk forward at a velocity of 0.15 m/s and place the steps in front of it. An example episode is depicted in Figure 6.5. We observe that our controller initially makes a mistake in ascending the step, but quickly adapts and lifts the leg faster and higher on the second attempt. This suggests that the controller is able to dynamically adapt its behavior relying solely on the history of past observations and actions.

Next, we assess the adaptability of our approach by simulating a sudden motor

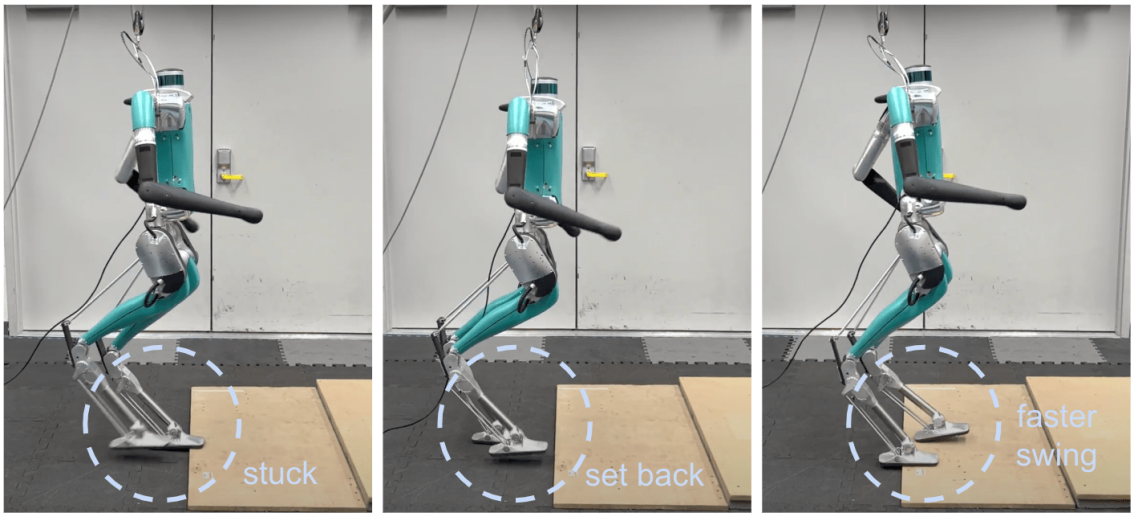


Figure 6.5: **Adaptation, steps.** We test the robot's ability to climb steps despite the controller not being exposed to such terrains. We observe that the controller initially makes a mistake, but quickly adapts by lifting the leg faster and higher on the second attempt, which suggests adaptability of the controller.

malfunction scenario while the robot is stepping in place. To achieve this, we reduce the PD gains of the selected left knee motor by half. The positions, velocities, and torques of both the left and right knee motors are plotted in Figure 6.6. The dashed line indicates the moment of the simulated malfunction. As seen from the figure, the left knee's position and velocity patterns undergo a significant change immediately after the malfunction, accompanied by a slight increase in its torque output. Meanwhile, the right knee's patterns adjust accordingly. The robot stabilizes after a few cycles. Despite the crucial role of the knee motors in maintaining the robot's balance, our approach demonstrates its ability to adapt in-context and prevent a catastrophic failure.

Emergent Arm Use

We observe emergent arm swing behaviors during walking, as shown in Figure 6.7. Note that we do not have explicit reference trajectories to guide these motions. Instead, this behavior emerges as a result of reinforcement learning in the training environments. Moreover, the swinging arm is coordinated with the legs, which is similar to the walking pattern of humans. Specifically, when the left leg is lifting up, the right arm swings forward. As pointed in [213, 214], arm swing helps maintain the stability of human locomotion and minimize energy consumption, which supports

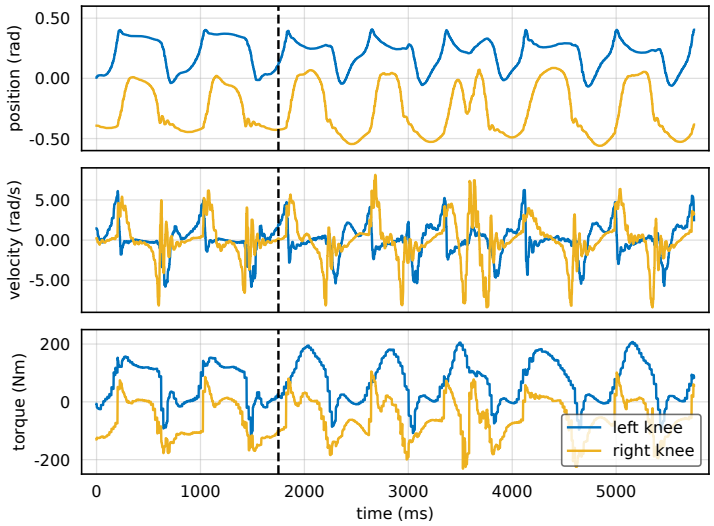


Figure 6.6: **Adaptation, motor malfunction.** We simulate a sudden malfunction of the left knee motor by decreasing its PD gains by 50%. The figure shows the changes in the position, velocity, and torque of both the left and right knee motors, with the vertical dashed line indicating the moment of the simulated malfunction. Despite the critical role of knee motors in the robot's balance, our approach is able to dynamically adjust and stabilize, demonstrating its ability to adapt in-context.

our learned locomotion behavior from a biomechanics perspective.

6.5.3 Dirty Laundry

In our evaluations on real hardware, our approach shows promising results in terms of adaptability and robustness to different terrains and external disturbances. However, it still has some limitations that need to be addressed in future work.

One limitation is that the policy is not perfectly symmetrical, as the motors on two sides do not produce identical trajectories. This results in a slight asymmetry in the robot's movement, with the controller being better at lateral movements to the left compared to the right. Additionally, our policy has errors in tracking the commanded velocity. We also notice that here is a minor difference in trajectories between two consecutive time cycles. Despite these limitations, our approach has shown potential in adapting to different terrains and handling external disturbances in real-world scenarios.

Another limitation is that under excessive external disturbances, for example, a very strong pull of a cable in our experiments, can cause the robot to fall. Please see the supplementary materials for video examples of failure cases.

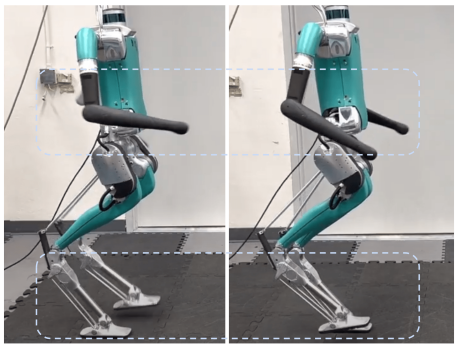


Figure 6.7: **Emergent arm swing.** We find that our Transformer-based controller leads to emergent human-like arm swing behaviors in coordination with leg movements.

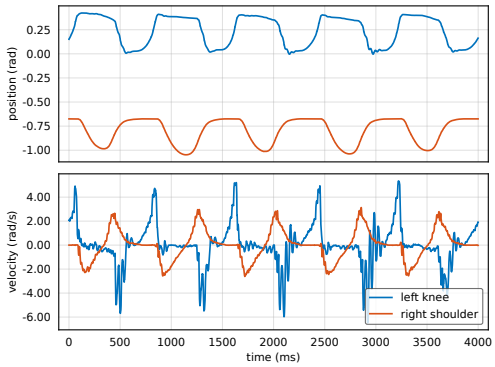


Figure 6.8: **Arm swing analysis.** The learned humanoid locomotion in our experiments exhibits human-like arm swing behaviors in coordination with leg movements, i.e., a contralateral relationship between the arms and the legs.

Finally, the Isaac Gym simulator does not support accurate simulation of under-actuated systems. In this study, we employed approximation methods to represent the four-bar linkage structure. We believe that our framework will benefit from improving the simulator in the future.

6.6 Conclusion

We present a sim-to-real learning-based approach for real-world humanoid locomotion. Our controller is a causal Transformer trained by autoregressive prediction of future actions from the history of observations and actions. Our approach is trained with reinforcement learning in simulation and deployed zero-shot. We evaluate our model in high-fidelity simulation and successfully deploy it to a real robot as well.

Chapter 7

Summary and Future Directions

This thesis has made strides towards learning scalable representations for computer vision and robotics, with the ultimate goal of developing general AI systems. Nevertheless, there is still much work to be done.

In Chapters 2 and 3, we investigated the optimizability of vision transformers and an inductive-bias-free self-supervised learning objective. However, vision models have not yet been scaled to the same level as large language models (LLMs), which have demonstrated emergent behaviors. As research progresses, there is a need to explore how to effectively scale these models to further improve their performance.

In Chapters 4 and 5, we showcased the power of learning scalable visual representations for motor control and robotics through masked visual pre-training. This pre-training does not include semantics, i.e., languages, which are essential for human-robot interaction as humans rely on language instructions to communicate with robots. Our tasks involved only a single object, and do not require the robot to learn feedback control, or to learn multi-step planning. Overcoming these limitations is essential as we move toward developing, benchmarking, and widely deploying pre-trained models for real-world robotic applications.

Moreover, integrating vision into our humanoid locomotion models (Chapter 6) can significantly benefit the overall system. By incorporating visual inputs, the humanoid can better perceive and understand its environment, allowing for more accurate locomotion and decision-making. This integration is essential for deploying robots in real-world scenarios, where they need to navigate complex and dynamic environments.

In summary, by continuing to explore and refine the principles and techniques presented in this work, we can move ever closer to the development of truly general AI systems that can effectively tackle a wide array of complex, real-world challenges.

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *NeurIPS*, 2012. 1, 5, 52

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *CVPR*, 2009. 1, 4, 5, 10, 26, 31, 39, 40, 41, 44, 52, 53, 54

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *ICCV*, 2015. 1

[4] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, "Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness," in *ICLR*, 2019. 1, 30, 35

[5] A. Barbu, D. Mayo, J. Alverio, W. Luo, C. Wang, D. Gutfreund, J. Tenenbaum, and B. Katz, "Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models," in *NeurIPS*, 2019. 1, 26, 31

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *NeurIPS*, 2020. 1, 38, 52, 69, 70, 72

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017. 1, 3, 5, 7, 52, 72

[8] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018. 1, 38, 52, 59, 70, 72

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *NAACL*, 2019. 1, 3, 38, 52, 70, 72

[10] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv:2001.08361*, 2020. 1

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2021. 2, 3, 5, 6, 7, 8, 9, 17, 22, 23, 38, 41, 44, 52, 53, 55, 59, 70, 72

[12] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.*, "Learning transferable visual models from natural language supervision," in *ICML*, 2021. 2, 39, 46, 54, 64

[13] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, 1989. 3, 5

[14] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *ICML*, 2021. 3, 6, 7, 9, 14, 22, 23, 44, 57

[15] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *ICLR*, 2019. 3, 6, 9

[16] H. Touvron, M. Cord, A. Sablayrolles, G. Synnaeve, and H. Jégou, "Going deeper with image transformers," *arXiv:2103.17239*, 2021. 3, 6, 9, 18

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016. 3, 5, 8, 15, 32, 44, 55

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015. 3, 5, 7

[19] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," *ICML*, 2019. 3, 15

[20] I. Radosavovic, J. Johnson, S. Xie, W.-Y. Lo, and P. Dollár, "On network design spaces for visual recognition," in *ICCV*, 2019. 3, 10, 13

[21] S. d'Ascoli, H. Touvron, M. Leavitt, A. Morcos, G. Biroli, and L. Sagun, "ConViT: Improving vision transformers with soft convolutional inductive biases," in *ICML*, 2021. 5, 6

[22] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980. 5

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015. 5

[24] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *CVPR*, 2017. 5

[25] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015. 5, 8

[26] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: Training ImageNet in 1 hour," *arXiv:1706.02677*, 2017. 5, 13, 22, 44, 52, 53, 55

[27] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *NeurIPS*, 2015. 5

[28] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *ICCV*, 2017. 5, 36, 38

[29] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, 2015. 5

[30] J.-B. Cordonnier, A. Loukas, and M. Jaggi, "On the relationship between self-attention and convolutional layers," *ICLR*, 2020. 5

[31] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *CVPR*, 2018. 5

[32] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *CVPR*, 2005. 5

[33] P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens, "Stand-alone self-attention in vision models," *NeurIPS*, 2019. 5

[34] H. Zhao, J. Jia, and V. Koltun, "Exploring self-attention for image recognition," in *CVPR*, 2020. 6

[35] X. Chen, S. Xie, and K. He, "An empirical study of training self-supervised vision transformers," in *ICCV*, 2021. 6, 46

[36] B. Graham, A. El-Nouby, H. Touvron, P. Stock, A. Joulin, H. Jégou, and M. Douze, "LeViT: a vision transformer in ConvNet's clothing for faster inference," in *ICCV*, 2021. 6, 9

[37] H. Fan, B. Xiong, K. Mangalam, Y. Li, Z. Yan, J. Malik, and C. Feichtenhofer, "Multiscale vision transformers," in *ICCV*, 2021. 6

[38] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z. Jiang, F. E. Tay, J. Feng, and S. Yan, "Tokens-to-token ViT: Training vision transformers from scratch on ImageNet," in *ICCV*, 2021. 6

[39] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions," in *ICCV*, 2021. 6

[40] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *ICCV*, 2021. 6

[41] Z. Chen, L. Xie, J. Niu, X. Liu, L. Wei, and Q. Tian, "Visformer: The vision-friendly transformer," in *ICCV*, 2021. 6

[42] H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang, "CvT: Introducing convolutions to vision transformers," in *ICCV*, 2021. 6

[43] K. Yuan, S. Guo, Z. Liu, A. Zhou, F. Yu, and W. Wu, "Incorporating convolution designs into visual transformers," in *ICCV*, 2021. 6

[44] Q. Wang, B. Li, T. Xiao, J. Zhu, C. Li, D. F. Wong, and L. S. Chao, "Learning deep transformer models for machine translation," in *ACL*, 2019. 7

[45] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010. 8

[46] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Big Transfer (BiT): General visual representation learning," in *ECCV*, 2020. 8

[47] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *AAAI*, 2020. 9

[48] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, and I. Sutskever, "Generative pretraining from pixels," in *ICML*, 2020. 9, 49

[49] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *CVPR*, 2020. 10, 20

[50] P. Dollár, M. Singh, and R. Girshick, "Fast and accurate model scaling," in *CVPR*, 2021. 10, 14, 15, 16, 20, 21, 23

[51] X. Dai, A. Wan, P. Zhang, B. Wu, Z. He, Z. Wei, K. Chen, Y. Tian, M. Yu, P. Vajda, *et al.*, "FBNetV3: Joint architecture-recipe search using neural acquisition function," in *CVPR*, 2021. 14

[52] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "AutoAugment: Learning augmentation policies from data," in *CVPR*, 2019. 14

[53] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," in *ICLR*, 2018. 14

[54] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "CutMix: Regularization strategy to train strong classifiers with localizable features," in *CVPR*, 2019. 14

[55] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016. 14

[56] M. Berman, H. Jégou, A. Vedaldi, I. Kokkinos, and M. Douze, "MultiGrain: a unified image embedding for classes and instances," *arXiv:1902.05509*, 2019. 14, 24

[57] E. Hoffer, T. Ben-Nun, I. Hubara, N. Giladi, T. Hoefler, and D. Soudry, "Augment your batch: better training with larger batches," *arXiv:1901.09335*, 2019. 14, 24

[58] M. Tan and Q. V. Le, "Efficientnetv2: Smaller models and faster training," in *ICML*, 2021. 23

[59] T. Ridnik, E. Ben-Baruch, A. Noy, and L. Zelnik-Manor, "Imagenet-21k pretraining for the masses," in *NeurIPS*, 2021. 23

[60] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, 1995. 23

[61] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *ICCV*, 2015. 25, 36, 49

[62] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *ECCV*, 2016. 25, 36, 49

[63] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *ECCV*, 2016. 25, 36, 49

[64] G. Larsson, M. Maire, and G. Shakhnarovich, "Learning representations for automatic colorization," in *ECCV*, 2016. 25

[65] R. Zhang, P. Isola, and A. A. Efros, "Split-brain autoencoders: Unsupervised learning by cross-channel prediction," in *CVPR*, 2017. 25

[66] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," in *ICLR*, 2018. 25, 36, 49

[67] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *CVPR*, 2018. 25, 37, 49

[68] Y. Tian, D. Krishnan, and P. Isola, "Contrastive multiview coding," *arXiv:1906.05849*, 2019. 25, 31, 37

[69] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *CVPR*, 2020. 25, 26, 30, 32, 37, 49

[70] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *ICML*, 2020. 25, 27, 30, 37, 49

[71] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-ucsd birds-200-2011 dataset," 2011. 26, 31

[72] G. Van Horn, O. Mac Aodha, Y. Song, Y. Cui, C. Sun, A. Shepard, H. Adam, P. Perona, and S. Belongie, "The inaturalist species classification and detection dataset," in *CVPR*, 2018. 26, 31

[73] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *ICVGIP*, 2008. 26, 31

[74] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," *ICLR*, 2019. 26, 31

[75] X. Chen, H. Fan, R. Girshick, and K. He, "Improved baselines with momentum contrastive learning," *arXiv:2003.04297*, 2020. 26, 30, 31

[76] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv:1807.03748*, 2018. 27, 37, 39, 49, 50

[77] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *CVPR*, 2016. 30

[78] D. R. Martin, C. C. Fowlkes, and J. Malik, "Learning to detect natural image boundaries using local brightness, color, and texture cues," *TPAMI*, 2004. 36

[79] A. Frome, Y. Singer, and J. Malik, "Image retrieval and classification using local distance functions," in *NeurIPS*, 2007. 36

[80] A. Frome, Y. Singer, F. Sha, and J. Malik, "Learning globally-consistent local distance functions for shape-based image retrieval and classification," in *ICCV*, 2007. 36

[81] T. Malisiewicz and A. A. Efros, "Recognition by association via learning per-exemplar distances," in *CVPR*, 2008. 36

[82] A. Rabinovich, T. Lange, J. Buhmann, and S. Belongie, "Model order selection and cue combination for image segmentation," in *CVPR*, 2006. 36

[83] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," in *ICCV*, 2015. 36, 49

[84] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *CVPR*, 2016. 36, 49

[85] A. Owens, J. Wu, J. H. McDermott, W. T. Freeman, and A. Torralba, "Ambient sound provides supervision for visual learning," in *ECCV*, 2016. 36

[86] I. Kokkinos, "Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory," in *CVPR*, 2017. 36

[87] M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, and R. Urtasun, "Multinet: Real-time joint semantic reasoning for autonomous driving," in *Intelligent Vehicles Symposium*, 2018. 36

[88] C. Doersch and A. Zisserman, "Multi-task self-supervised visual learning," in *ICCV*, 2017. 36, 37

[89] X. Wang, K. He, and A. Gupta, "Transitive invariance for self-supervised visual representation learning," in *ICCV*, 2017. 36

[90] L. Pinto and A. Gupta, "Learning to push by grasping: Using multiple tasks for effective learning," in *ICRA*, 2017. 36

[91] A. Piergiovanni, A. Angelova, and M. S. Ryoo, "Evolving losses for unsupervised video representation learning," in *CVPR*, 2020. 36

[92] H. Alwassel, D. Mahajan, L. Torresani, B. Ghanem, and D. Tran, "Self-supervised learning by cross-modal audio-video clustering," in *NeurIPS*, 2020. 36

[93] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *CVPR*, 2016. 37

[94] I. Misra and L. van der Maaten, "Self-supervised learning of pretext-invariant representations," in *CVPR*, 2020. 37

[95] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, 2014. 38, 52

[96] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," 2019. 38

[97] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio.," *SSW*, 2016. 38, 52

[98] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *JMLR*, 2016. 38, 54

[99] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," in *ICLR*, 2017. 39, 50

[100] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, "Improving sample efficiency in model-free reinforcement learning from images," *arXiv:1910.01741*, 2019. 39, 46, 50

[101] A. Srinivas, M. Laskin, and P. Abbeel, "Curl: Contrastive unsupervised representations for reinforcement learning," in *ICML*, 2020. 39, 46, 50

[102] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *CVPR*, 2022. 39, 40, 44, 49, 50, 52, 55, 57, 61

[103] T. Xiao, X. Wang, A. A. Efros, and T. Darrell, "What should not be contrastive in contrastive learning," in *ICLR*, 2021. 39, 49, 55

[104] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto, "Mastering visual continuous control: Improved data-augmented reinforcement learning," in *ICLR*, 2022. 39, 46

[105] D. Shan, J. Geng, M. Shu, and D. F. Fouhey, "Understanding human hands in contact at internet scale," in *CVPR*, 2020. 40, 41, 44, 52, 53, 54

[106] D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, *et al.*, "Scaling egocentric vision: The epic-kitchens dataset," in *ECCV*, 2018. 40, 41, 44, 52

[107] D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, and M. Wray, "The epic-kitchens dataset: Collection, challenges and baselines," *TPAMI*, 2021. 40, 41, 44, 53, 54

[108] R. Goyal, S. Ebrahimi Kahou, V. Michalski, J. Materzynska, S. Westphal, H. Kim, V. Haenel, I. Fruend, P. Yianilos, M. Mueller-Freitag, *et al.*, "The" something something" video database for learning and evaluating visual common sense," in *ICCV*, 2017. 41

[109] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017. 42, 79

[110] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *IJRR*, 2020. 42, 54, 68

[111] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving rubik's cube with a robot hand," *arXiv:1910.07113*, 2019. 42, 68, 70

[112] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv:1509.02971*, 2015. 42, 46

[113] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, *et al.*, "Deepmind control suite," *arXiv:1801.00690*, 2018. 42, 46

[114] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, "Rlbench: The robot learning benchmark & learning environment," *RA-L*, 2020. 42

[115] Y. Zhu, J. Wong, A. Mandlekar, and R. Martín-Martín, "robosuite: A modular simulation framework and benchmark for robot learning," *arXiv:2009.12293*, 2020. 42

[116] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *CoRL*, 2020. 42

[117] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," in *NeurIPS*, 2021. 42, 44, 57, 74

[118] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IROS*, 2012. 42

[119] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *CoRL*, 2021. 42, 74

[120] T. Xiao, P. Dollar, M. Singh, E. Mintun, T. Darrell, and R. Girshick, "Early convolutions help transformers see better," in *NeurIPS*, 2021. 44, 57

[121] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *NeurIPS*, 2017. 44, 57

[122] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare, "Deep reinforcement learning at the edge of the statistical precipice," *NeurIPS*, 2021. 44

[123] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *arXiv:2204.06125*, 2022. 46

[124] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018. 46

[125] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The ycb object and model set: Towards common benchmarks for manipulation research," in *ICAR*, 2015. 48

[126] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *CVPR*, 2006. 49

[127] O. Henaff, "Data-efficient image recognition with contrastive predictive coding," in *ICML*, 2020. 49

[128] A. Jabri, A. Owens, and A. Efros, "Space-time correspondence as a contrastive random walk," *NeurIPS*, 2020. 49

[129] H. Bao, L. Dong, and F. Wei, "Beit: Bert pre-training of image transformers," in *ICLR*, 2022. 49

[130] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, *et al.*, "Learning to navigate in complex environments," in *ICLR*, 2017. 50

[131] E. Shelhamer, P. Mahmoudieh, M. Argus, and T. Darrell, "Loss is its own reward: Self-supervision for reinforcement learning," in *ICLR*, 2017. 50

[132] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," in *AAAI*, 2017. 50

[133] D. Ha and J. Schmidhuber, "World models," *arXiv:1803.10122*, 2018. 50

[134] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *ICRA*, 2016. 50, 54

[135] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," *NeurIPS*, 2016. 50, 54

[136] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain, "Time-contrastive networks: Self-supervised learning from video," in *ICRA*, 2018. 50, 54

[137] P. R. Florence, L. Manuelli, and R. Tedrake, "Dense object nets: Learning dense visual object descriptors by and for robotic manipulation," in *CoRL*, 2018. 50

[138] A. Zhan, P. Zhao, L. Pinto, P. Abbeel, and M. Laskin, "A framework for efficient robotic manipulation," *arXiv:2012.07975*, 2020. 50, 54

[139] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," *NeurIPS*, 2020. 50

[140] J. Pari, N. Muhammad, S. P. Arunachalam, L. Pinto, *et al.*, "The surprising effectiveness of representation learning for visual imitation," in *RSS*, 2022. 50, 54

[141] A. Sax, B. Emi, A. R. Zamir, L. Guibas, S. Savarese, and J. Malik, "Mid-level visual representations improve generalization and sample efficiency for learning visuomotor policies," *arXiv:1812.11971*, 2018. 50

[142] B. Chen, A. Sax, G. Lewis, I. Armeni, S. Savarese, A. Zamir, J. Malik, and L. Pinto, "Robust policies via mid-level visual representations: An experimental study in manipulation and navigation," in *CoRL*, 2020. 50

[143] B. Zhou, P. Krähenbühl, and V. Koltun, "Does computer vision matter for action?," *Science Robotics*, 2019. 50

[144] L. Yen-Chen, A. Zeng, S. Song, P. Isola, and T.-Y. Lin, "Learning to see before learning to act: Visual pre-training for manipulation," in *ICRA*, 2020. 51, 54

[145] R. Shah and V. Kumar, "Rrl: Resnet as representation for reinforcement learning," *arXiv:2107.03380*, 2021. 51

[146] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," *arXiv:2005.00341*, 2020. 52

[147] T. Xiao, I. Radosavovic, T. Darrell, and J. Malik, "Masked visual pre-training for motor control," *arXiv:2203.06173*, 2022. 52, 54, 55, 56, 57, 61, 62

[148] K. Grauman, A. Westbury, E. Byrne, Z. Chavis, A. Furnari, R. Girdhar, J. Hamburger, H. Jiang, M. Liu, X. Liu, *et al.*, "Ego4d: Around the world in 3,000 hours of egocentric video," in *CVPR*, 2022. 52, 53, 54, 62

[149] S. James, A. J. Davison, and E. Johns, "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task," *CoRL*, 2017. 54

[150] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *CoRL*, 2018. 54

[151] A. T. Miller, S. Knoop, H. I. Christensen, and P. K. Allen, "Automatic grasp planning using shape primitives," in *ICRA*, 2003. 54

[152] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *ICRA*, 2000. 54

[153] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *IROS*, 2018. 54

[154] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani, *et al.*, "Transporter networks: Rearranging the visual world for robotic manipulation," *CoRL*, 2020. 54

[155] S. James and A. J. Davison, "Q-attention: Enabling Efficient Learning for Vision-based Robotic Manipulation," *RA-L*, 2022. 54

[156] M. Shridhar, L. Manuelli, and D. Fox, "Cliport: What and where pathways for robotic manipulation," in *CoRL*, 2021. 54

[157] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta, "R3m: A universal visual representation for robot manipulation," *CoRL*, 2022. 54, 62

[158] L. Pinto, D. Gandhi, Y. Han, Y.-L. Park, and A. Gupta, "The curious robot: Learning visual representations via physical interactions," in *ECCV*, 2016. 54

[159] 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," *IJRR*, 2018. 54

[160] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *ICRA*, 2016. 54

[161] P. Sermanet, K. Xu, and S. Levine, "Unsupervised perceptual rewards for imitation learning," in *RSS*, 2017. 54

[162] P. Florence, L. Manuelli, and R. Tedrake, "Self-supervised correspondence in visuomotor policy learning," *RA-L*, 2019. 54

[163] Y. Li, S. Xie, X. Chen, P. Dollar, K. He, and R. Girshick, "Benchmarking detection transfer learning with vision transformers," *arXiv:2111.11429*, 2021. 55, 70

[164] Z. Tong, Y. Song, J. Wang, and L. Wang, "Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training," in *NeurIPS*, 2022. 55

[165] C. Feichtenhofer, H. Fan, Y. Li, and K. He, "Masked autoencoders as spatiotemporal learners," in *NeurIPS*, 2022. 55

[166] R. Bachmann, D. Mizrahi, A. Atanov, and A. Zamir, "Multimae: Multi-modal multi-task masked autoencoders," in *ECCV*, 2022. 55

[167] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, "Scaling vision transformers," in *CVPR*, 2022. 55

[168] S. James, K. Wada, T. Laidlow, and A. J. Davison, "Coarse-to-Fine Q-attention: Efficient Learning for Visual Robotic Manipulation via Discretisation," *CVPR*, 2022. 65

[169] S. P. Arunachalam, S. Silwal, B. Evans, and L. Pinto, "Dexterous imitation made easy: A learning-based framework for efficient dexterous manipulation," in *ICRA*, 2023. 66

[170] M. H. Raibert, *Legged robots that balance*. MIT press, 1986. 68, 70, 77

[171] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous robots*, 2016. 68, 70

[172] S. Kuindersma, "Recent progress on atlas, the world's most dynamic humanoid robot," 2020. 68, 70

[173] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, 2019. 68, 70

[174] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science robotics*, 2020. 68, 70, 73

[175] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," *RSS*, 2021. 68, 70, 73

[176] Z. Xie, G. Berseth, P. Clary, J. Hurst, and M. van de Panne, "Feedback control for cassie with deep reinforcement learning," in *IROS*, 2018. 68, 70

[177] J. Siekmann, Y. Godse, A. Fern, and J. Hurst, "Sim-to-real learning of all common bipedal gaits via periodic reward composition," in *ICRA*, 2021. 68, 70

[178] J. Siekmann, K. Green, J. Warila, A. Fern, and J. Hurst, "Blind bipedal stair traversal via sim-to-real reinforcement learning," *RSS*, 2021. 68, 70

[179] G. A. Castillo, B. Weng, W. Zhang, and A. Hereid, "Reinforcement learning-based cascade motion policy design for robust 3d bipedal locomotion," *IEEE Access*, 2022. 68, 70

[180] L. Krishna, G. A. Castillo, U. A. Mishra, A. Hereid, and S. Kolathaya, "Linear policies are sufficient to realize robust bipedal walking on challenging terrains," *RA-L*, 2022. 68, 70, 74

[181] I. Kato, "Development of wabot 1," *Biomechanism*, 1973. 70

[182] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, "The development of honda humanoid robot," in *ICRA*, 1998. 70

[183] G. Nelson, A. Saunders, N. Neville, B. Swilling, J. Bondaryk, D. Billings, C. Lee, R. Playter, and M. Raibert, "Petman: A humanoid robot for testing chemical protective clothing," *Journal of the Robotics Society of Japan*, 2012. 70

[184] O. Stasse, T. Flayols, R. Budhiraja, K. Giraud-Esclasse, J. Carpentier, J. Mirabel, A. Del Prete, P. Souères, N. Mansard, F. Lamiraux, *et al.*, "Talos: A new humanoid research platform targeted for industrial applications," in *Humanoids*, 2017. 70

[185] M. Chignoli, D. Kim, E. Stanger-Jones, and S. Kim, "The mit humanoid robot: Design, motion planning, and control for acrobatic behaviors," in *Humanoids*, 2021. 70

[186] S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi, and H. Hirukawa, "The 3d linear inverted pendulum mode: A simple modeling for a biped walking pattern generation," in *IROS*, 2001. 70

[187] E. R. Westervelt, J. W. Grizzle, and D. E. Koditschek, "Hybrid zero dynamics of planar biped walkers," *IEEE transactions on automatic control*, 2003. 70

[188] S. Collins, A. Ruina, R. Tedrake, and M. Wisse, "Efficient bipedal robots based on passive-dynamic walkers," *Science*, 2005. 70

[189] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *IROS*, 2012. 70

[190] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," in

IEEE/RSJ international conference on intelligent robots and systems (IROS), 2018. 70

[191] H. Benbrahim and J. A. Franklin, "Biped dynamic walking using reinforcement learning," *Robotics and Autonomous Systems*, 1997. 70

[192] G. B. Margolis and P. Agrawal, "Walk these ways: Tuning robot control for generalization with multiplicity of behavior," *CoRL*, 2022. 70

[193] A. Kumar, Z. Li, J. Zeng, D. Pathak, K. Sreenath, and J. Malik, "Adapting rapid motor adaptation for bipedal robots," in *IROS*, 2022. 70

[194] R. Antonova, S. Cruciani, C. Smith, and D. Kragic, "Reinforcement learning for pivoting task," *arXiv:1703.00472*, 2017. 70

[195] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," in *RSS*, 2017. 70

[196] 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 *IROS*, 2017. 70

[197] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *ICRA*, 2018. 70

[198] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *ECCV*, 2020. 70, 72

[199] I. Radosavovic, T. Xiao, S. James, P. Abbeel, J. Malik, and T. Darrell, "Real-world robot learning with masked visual pre-training," *CoRL*, 2022. 70

[200] W. Yuan, C. Paxton, K. Desingh, and D. Fox, "Sornet: Spatial object-centric representations for sequential manipulation," in *CoRL*, 2022. 70

[201] R. Yang, M. Zhang, N. Hansen, H. Xu, and X. Wang, "Learning vision-guided quadrupedal locomotion end-to-end with cross-modal transformers," *arXiv:2107.03996*, 2021. 70

[202] M. Shridhar, L. Manuelli, and D. Fox, "Perceiver-actor: A multi-task transformer for robotic manipulation," *arXiv:2209.05451*, 2022. 70

[203] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, *et al.*, "Rt-1: Robotics transformer for real-world control at scale," *arXiv:2212.06817*, 2022. 70

[204] L. Dong, S. Xu, and B. Xu, "Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition," in *ICASSP*, 2018. 72

[205] A. K. Han, A. Hajj-Ahmad, and M. R. Cutkosky, "Bimanual handling of deformable objects with hybrid adhesion," *RA-L*, 2022. 74

[206] G. A. Castillo, B. Weng, W. Zhang, and A. Hereid, "Robust feedback motion policy design using reinforcement learning on a 3d digit bipedal robot," in *IROS*, 2021. 74

[207] Y. Gao, Y. Gong, V. Paredes, A. Hereid, and Y. Gu, "Time-varying alip model and robust foot-placement control for underactuated bipedal robot walking on a swaying rigid surface," *arXiv:2210.13371*, 2022. 74

[208] Y. Gao, C. Yuan, and Y. Gu, "Invariant filtering for legged humanoid locomotion on a dynamic rigid surface," *IEEE/ASME Transactions on Mechatronics*, 2022. 74

[209] A. Adu-Bredu, N. Devraj, and O. C. Jenkins, "Optimal constrained task planning as mixed integer programming," in *IROS*, 2022. 74

[210] K. S. Narkhede, A. M. Kulkarni, D. A. Thanki, and I. Poulakakis, "A sequential mpc approach to reactive planning for bipedal robots using safe corridors in highly cluttered environments," *RA-L*, 2022. 74

[211] A. Shamsah, J. Warnke, Z. Gu, and Y. Zhao, "Integrated task and motion planning for safe legged navigation in partially observable environments," *arXiv:2110.12097*, 2021. 74

[212] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric actor critic for image-based robot learning," in *RSS*, 2017. 79

[213] J. Park, "Synthesis of natural arm swing motion in human bipedal walking," *Journal of biomechanics*, 2008. 84

[214] S. H. Collins, P. G. Adamczyk, and A. D. Kuo, "Dynamic arm swinging in human walking," *Proceedings of the Royal Society B: Biological Sciences*, 2009. 84