Learning Robotic Tasks from Video Demonstrations

Jeremy Collins
jcollins90@gatech.edu

Alan Hesu
ahesu6@gatech.edu

Cody Houff
chouff3@gatech.edu

Dane Wang
ywang3943@gatech.edu

Abstract

Training an agent to solve a wide variety of complex tasks has remained an open problem for decades; long-horizon, reward-sparse environments are notoriously difficult to learn from scratch via reinforcement learning. One such domain is robotic manipulation, which has been a long-standing problem facing classical control theorists and roboticists alike. At the core of this problem is the curse of dimensionality – the space of possible trajectories is so large that a complete search through this space to learn a policy is intractable for even short-horizon tasks. Additionally, very precise sensing is often required for robustness but often leads to expensive solutions that are prone to hardware failure. We present an alternative approach that uses visual data alone to learn a control policy for a robotic arm by observing expert video demonstrations. We implement and test several models, accomplishing an 85% success for a pick-and-place task.

1. Introduction

Robot arms are being increasingly used in factories, distribution centers, and even hospitals to increase our quality of life. However, reliable robot arm planning and control remain relatively challenging both in industry and academia. Because of the large number of joints, the physical model is complex and high-dimensional, which makes the problem hard to solve from both classical control and deep learning approaches. Our goal for this project is to build an end-to-end deep learning model that can control the robot arm for a specific task.

Learning robotic manipulation tasks requires knowledge of the robot state and the surrounding environment. While typical reinforcement learning approaches have used privileged information of the joint space and object positions known directly from the environment simulation [5], such information may not always be known or available. To determine a robot arm's state space and sense its manipulation target, it is common to use a single observing camera mounted on the robot or pointed at the robot to capture the scene. The use of a camera lends itself naturally to predicting motion and control policies. In a collaborative robotic manipulation task, for example, each robot would like to observe the other and predict the trajectory in the short horizon in order to avoid collisions. This capability for action prediction is akin to generating a control policy by applying a predictive model autoregressively.

At the same time, reinforcement learning normally struggles to generate control models that can handle long horizon or sparse reward environments as shown in [2]. The application of reinforcement learning requires a great setup of the simulation environment and the corresponding reward function which could both be difficult to accomplish for a complex environment or long-horizon tasks.

In this project, we would like to propose an alternative approach to generate a robot arm control policy: behavioral cloning (BC). We build an end-to-end BC model for a pick-and-place task that is trained on video and action data from an expert demonstrator which can be either a human being or a pre-trained model. We showed that our BC model can learn a robot arm control strategy from the expert and then accomplish the same tasks autonomously.

Current state-of-the-art methods for this spatiotemporal reasoning often involve using convolutional neural networks (CNNs) to embed spatial information and long short-term memory (LSTMs) for prediction from a temporal sequence [13]. Applying advances in language modeling, we are using a transformer [18] in place of the LSTM for robot action prediction. We show that this approach offers accuracy that matches or exceeds the LSTM method in robot behavior cloning.

We also train models using the privileged robot and object state information as inputs instead of the video data from an external observing camera. The performance of these models is comparable to the models trained using the video data, showing that usage of the CNN to embed the spatial information from the environment state is sufficient to recover the necessary information to accomplish the task that an expert or agent with more privileged information would require.

By using video data from the camera directly, we reach an 85% success rate for a pick-and-place task by combing a CNN encoder, a transformer, and an action prediction network.

2. Related Works

2.1. Behavioral cloning

Behavioral cloning (BC) is a subset of imitation learning that involves training a model on observation-action pairs to replicate the actions and behaviors in the training dataset. While it typically relies on labeled observation-action data, recent works have demonstrated successful approaches that infer actions from solely observation data, replicating the performance of other state-of-the-art learning from demonstration (LfD) methods [11], [16].

This work has been leveraged to achieve novel performance in high dimensional environments such as Minecraft, where BC along with other fine-tuning methods were trained on a large volume of video data to achieve game playing performance comparable with human players [2]. Hard-exploration tasks such as Minecraft have been historically impossible to learn from scratch via reinforcement learning. Baker et al. utilize transformer network architectures for video prediction for generating actions as game inputs (ie. keyboard, mouse) from video data, using this method for both labeling large volumes of unlabeled video data and subsequently using the labeled data for BC.

2.2. Robot manipulation

Many current approaches that use cameras for robotic manipulation require the ability to estimate a robot's pose in 3D space. This may involve placing fiducial markers on different joints on the robot and undergoing a calibration procedure. A potentially simpler approach involves using a learned model to implicitly transform the 2D image to 3D space and perform pose estimation. Sithamparanathan et al. demonstrate a method using a CNN to generate a 3D point cloud from a 2D image, and the point cloud is then mapped to the most likely robot pose [15]. Lee at al. also utilize a CNN-based approach, though their output involves placing keypoints in the 2D image and then using perspective-n-point (PnP) and known forward kinematics of the robot to determine the 3D pose [9]. Widmaier et al. use a random forest approach trained on images containing depth information. They also propose a DISP distance metric, which can be more suitable for measuring error in pose estimation for multi-DOF robots compared to simple euclidean distance [20].

While CNNs are capable of spatial reasoning for instantaneous robot state estimation, predicting the next robot action also requires temporal reasoning using information from a sequence of past robot states. Although 3D convolution is a potential solution to this, it is often prohibitively expensive. LSTMs have long been used for temporal modeling, including motion prediction applications [1]. They have also been extended to perform the spatiotemporal reasoning necessary for the application of robot motion and action prediction. Rodrigues et al. use a CNN for pose estimation by estimating keypoints. These keypoints are then used to construct a sequence to predict future keypoints using an LSTM [13]. By replacing the traditional fully connected architecture in an LSTM with a CNN, Shi et al. allow the LSTM itself to perform spatial reasoning, enabling a more end-to-end model architecture [14].

2.3. Transformers

Transformer-based methods use an architecture that relies on an attention mechanism which considers relationships between elements of an input sequence to generate low-dimensional embeddings, from which the desired output may be decoded. The use of multiheaded attention enables a much larger degree of parallelization of the data processing as opposed to the sequential processing traditionally used in recurrent neural networks and has been demonstrated to show state-of-the-art performance on a number of language processing and computer vision tasks [19].

However, the use of transformers in robot motion prediction is less thoroughly investigated. Levine et al. demonstrate a convolutional neural network (CNN) model trained on real-world video data for grasping and servoing of a robotic arm in specific manipulation tasks [10]. In addition to a convolutional long short-term memory (LSTM) approach, Finn et al. use a spatial transformation prediction (STP) method for generating affine transformations for motion prediction from videos. They demonstrate trained results on both robotic and human motions [4]. Multiple datasets have been created across a wide variety of robotic manipulation tasks. Using their crowd-sourced RoboTurk dataset, Mandlekar et al. investigate BC but report poor results given the diversity of the manipulation tasks [12].

3. Methods

3.1. Task description

The environment setup from the expert demonstration data [5] consists of a single Panda robotic arm on a flat surface, as shown in Fig 1. There is a single red cube, which the robot attempts to pick up and move to the goal location, which is represented by a collision-less yellow cube. The robot's action space is represented by a 1x4 vector consisting of the *x, y,* and *z* velocity commands for the end effector and a single value representing the gripper position. Because the end-effector is not granted any rotational de-

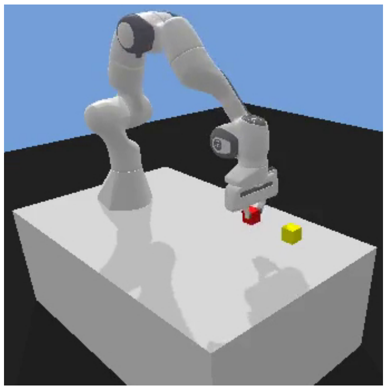


Figure 1. Panda-Gym simulation Environment

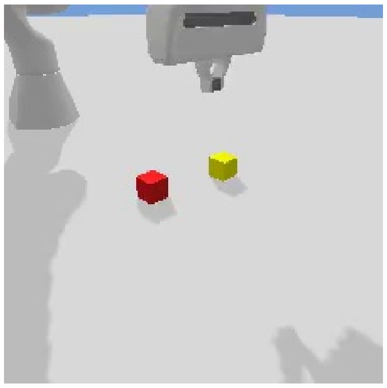


Figure 2. Example input frame

grees of freedom, this vector is sufficient to capture all possible movements. The observation space, generated from the simulation environment and containing all information needed to represent the robot and task state, is a 1x25 vector. This contains the gripper speed and position and the position of its finger, the position of the goal, and the position, orientation, and linear and rotational velocity of the object. A camera observing the environment also captures images of the environment, including the robot, goal, and object. An example of such a frame is seen in Fig 2.

3.2. Machine learning models

The models we implemented can be divided into four sections as shown by Fig 3. The horizontal axis demarcates whether the instantaneous information from a single timestep is used as an input or if a sequence of multiple timesteps is used. The vertical axis divides the input representation between the privileged environment state information, which is represented as a 1x25 vector, or the frame recorded by the camera, which is resized to a 96x96x3 RGB image. The upper left corner of the matrix in Fig 3 represents the least complex model, while the lower right corner of the matrix represents the models with the highest complexity. This is due to the input format and the ways in which the input information is processed before being passed into the action predictor network. In the right half, a CNN network is used to process image frames, and in the bottom half, an LSTM or transformer is used as a sequence model to process the sequential input.

3.3. Basic Behavior Cloning

The first model is the basic BC model, which is shown in 3a. This architecture functions as the foundation of the other models presented in the figure and is based on the BC method described in [3] and implemented in [6]. This combines a variational autoencoder (VAE) with a discriminator mapping the latent space to action predictions to build a data-efficient learning architecture. Specifically, our implementation of the BC action predictor is shown in 4. The input features are fed into a multilayer perceptron (MLP), which encodes the input's latent space. From this latent space, an action probability distribution is learned using fully connected layers to predict the mean and variance of the distribution, which is then sampled to generate the predicted action. While Chen at al. show high performance in the domain of video game playing, we show that this generative modeling can also be applied to robot arm manipulation.

The loss function of our BC model is shown in Equation 1 and is composed of three main terms. The first term, $-logP(x|\mathcal{N}(\mu,\sigma))$ is the negative log probability of the true action given the predicted action distribution. This represents how well the action distribution predicts the true action. The second term, $\alpha H(x)$, is the entropy of the predicted action distribution, which minimizes the degree of uncertainty in the distribution. The final term, $\lambda||w||_2^2$, is an L_2 weight regularization term that helps prevent the model from overfitting on the training data. The weight terms α and λ modify how much the entropy and L_2 regularization term contribute to the overall loss function respectively.

This loss function enables the probabilistic predictive modeling described above, which is necessary to accurately predict the control inputs for the robotic manipulation task. A more deterministic loss function and modeling approach such as mean squared error (MSE) loss was observed to be insufficient. To minimize the euclidean distance between the predicted and actual actions, the model ends up predicting the overall mean of the training data.

$$loss = -logP(x|\mathcal{N}(\mu,\sigma)) - \alpha H(x) + \lambda||w||_2^2 \qquad (1)$$

3.4. LSTM + Behavior Cloning

Instead of using single timesteps as the input, adding some temporal understanding of the environment state to

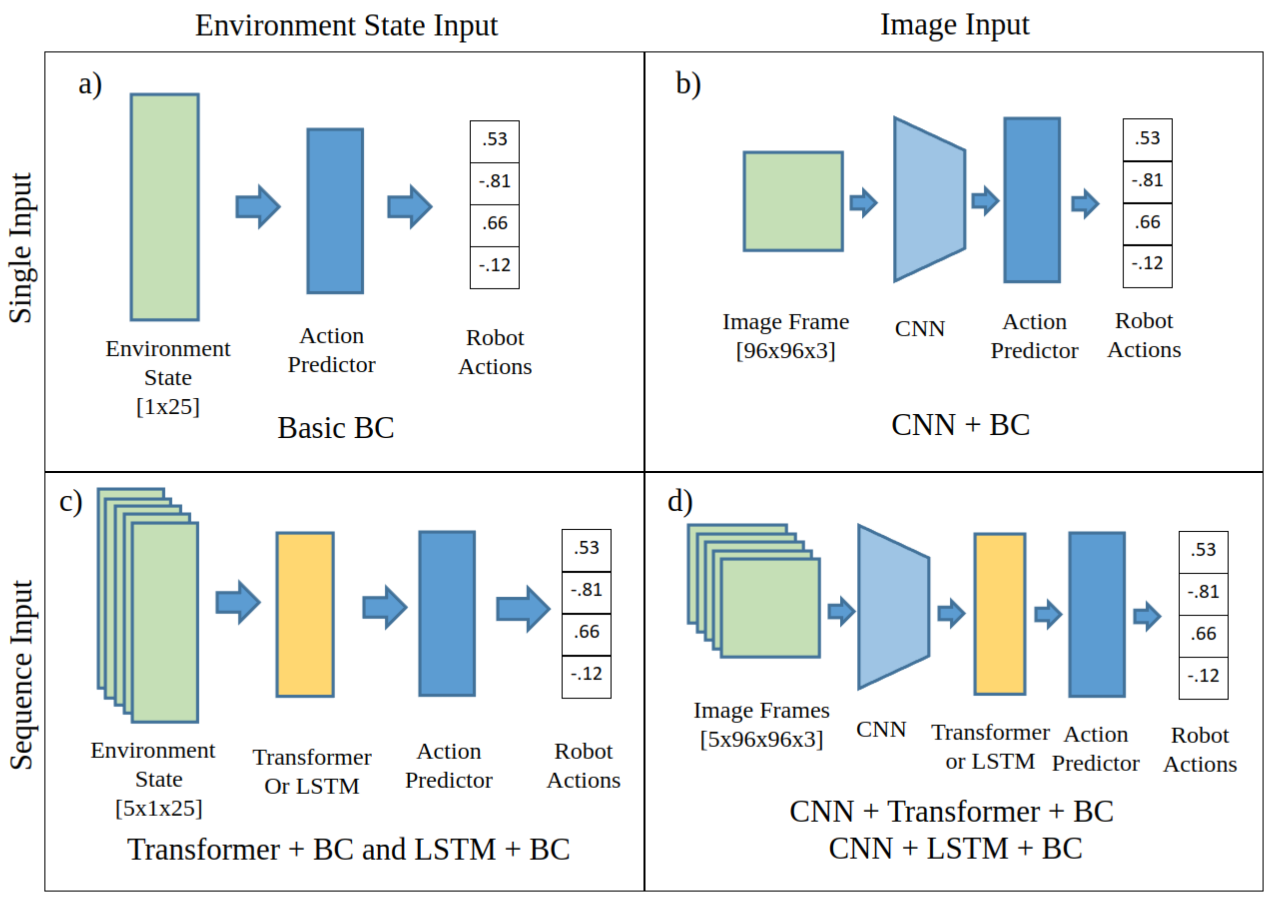


Figure 3. Type of Models

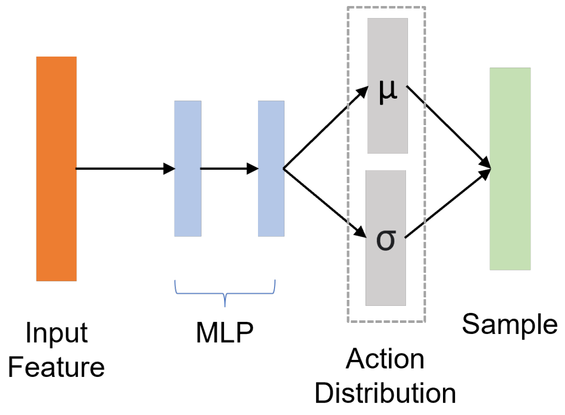


Figure 4. Behavior Cloning Structure

the model may improve its task reasoning by better contextualizing its own actions and movements as well as any motion in the object. To accomplish this, a long-short term memory (LSTM) architecture was used as a sequence model [7]. As shown in Fig 3c, a sequence of five consecutive 1x25 environment state vectors from $t-4$ to t is used as an input to the LSTM. The LSTM itself consists of two layers with a hidden dimension size of 32. The final hidden state in the LSTM is then projected back to a 1x25 vector using a final fully connected layer, which now represents the model's understanding of the environment along with prior temporal information. This vector is then passed into the behavior cloning architecture described in 3.3

3.5. Transformer + Behavior Cloning

An alternative to the LSTM as a sequence model is the transformer. A causally masked encoder-decoder transformer is used based on the time series forecasting architecture described in [21]. As with the LSTM, the transformer input consists of a sequence of five environment state vectors. The chosen transformer model uses a hidden dimension of 32, 1 encoder layer, 1 decoder layer, and two attention heads. Similar to the LSTM, the last token from the output of the decoder is projected back to a 1x25 vector and passed into the behavior cloning architecture in 3.3.

3.6. CNN + Behavior Cloning

As mentioned before, instead of using the 1x25 observation space vector, we also tried to take the 96x96x3 raw image of the simulation environment as the input. This algorithm is shown in Fig 3b, where a CNN is added to project the image into a visual feature vector that can be used by the behavior cloning model. The CNN network we used is called the MAGICAL CNN from [17]. The CNN takes the environment as input and returns a 1x128 learned visual feature vector which is then passed into the behavior cloning architecture described in 3.3.

3.7. CNN + LSTM + Behavior Cloning

This algorithm is a combination of the 3.6 and 3.4, and it is shown as Fig 3d. It takes a sequence of five consecutive 96x96x3 simulation environment pictures as the input and then passes into the CNN and LSTM networks described above. The output is then passed to the behavior cloning architecture described in 3.3 like all other models.

3.8. CNN + Transformer + Behavior Cloning

Just like 3.7, this model also takes a sequence of five consecutive 96x96x3 simulation environment images as the input. The only difference is that we are using the transformer from 3.5 instead of LSTM as the sequential model. This model is also represented in Fig 3d.

4. Data Collection

We collect a labeled dataset including videos of robot arm operations with corresponding control commands. For our simulation environment, we choose panda-gym. A sample simulation environment is presented as Figure 1.

Panda-gym is an open-source goal-conditioned environment for Panda robotic arm based on OpenAI Gym. [5]. It includes pre-trained models that can control a robot arm to complete five different simulated tasks: reach, push, slide, pick and place, and stack. There is also a virtual camera, which allows for the recording of RGB videos of the robot arm operation.

For this project, we chose to generate a dataset of demonstrations on the pick and place task, each of which lasts for 50 simulation steps. The Panda-gym environment randomly initializes the position of the goal and object. If the robot arm can pick up and target an object and move it to the goal position, the task is considered a success. The pretrained model within the Panda-gym has a 98.8% success rate. The speed at which the robot can pick up and move the object and keep it in the desired goal position is also measured. This metric arises from the reward function in the reinforcement learning environment used to train the expert agent, which receives a reward of -1 for each step the object is not in the goal and 0 for each step it is in the goal, generating a total reward per demonstration in the range [-50, 0]. We convert this to a positive score value by subtracting the absolute value of the accrued reward from the maximum of 50, yielding a score in the range [0, 50], where a higher score indicates more effective completion of the task. The expert model obtains an average score of 41.96.

We use this pretrained expert model to generate the dataset, which consists of 3000 demonstrations, with each demonstration including the 1x25 observation space, 1x4 action command, and recorded image frame for each timestep. Each demonstration is 50 steps, so altogether, 150000 data points were collected and used. The set of 3000 demonstrations were then split into a training and testing dataset, with the training dataset containing 80% of the data and the testing dataset containing the other 20%. Because the environment used in the generation of the dataset also randomly initializes the starting positions of the goal and object, the dataset is representative of the possible states the robot may encounter during evaluation of the model.

To load data for the single input models (Fig 3a, b), a single observation vector or frame is used as the input, and the corresponding action vector is used as the ground truth prediction. To generate the sequential inputs for the sequence models (Fig 3c, d), sequences of five contiguous observations or image frames are collected from each demonstration. Because evaluating the model requires starting the simulation at time $t = 0$ without any information from prior timesteps, sequences at the start of each demonstration that don't contain enough frames from the actual sample are padded with zeros so that the model is still trained on data representative of the initial state of the task.

5. Experiments and Results

5.1. Training

All of the models were trained on the 2400 demonstration samples found in the training dataset using the Adam optimizer [8] with a learning rate of 1e-4 and batch size of 32. In the loss function (1), the entropy weight α was set to 1e-3, and the L2 weight λ was set to 1e-6. Each model was trained until the training and testing loss were observed to converge. A plot comparing the convergence rates of the testing loss for each model is shown in Fig 5.

5.2. Evaluation criteria

To evaluate the performance of a model, we select two criteria. The first criterion is the success rate. We run 1000 tests in simulation after the training process is finished and then record the number of times that the model is able to move the object to the goal. The higher this success rate is, the more this model learned from the expert.

Input	Model	Score	Success Rate (%)	# Parameters	Batch/s
State	Expert	41.96	98.9		
State	Basic BC	32.18	80.6	2189	121.49
State	Transformer+BC	**36.68**	**89.7**	21894	59.52
State	LSTM+BC	34.28	82.5	19014	75.69
Image	CNN+BC	31.81	82.5	755405	58.02
Video	CNN+Transformer+BC	32.17	84.1	785101	33.82
Video	CNN+LSTM+BC	33.67	86.1	788813	31.53

Table 1. Model performance metrics

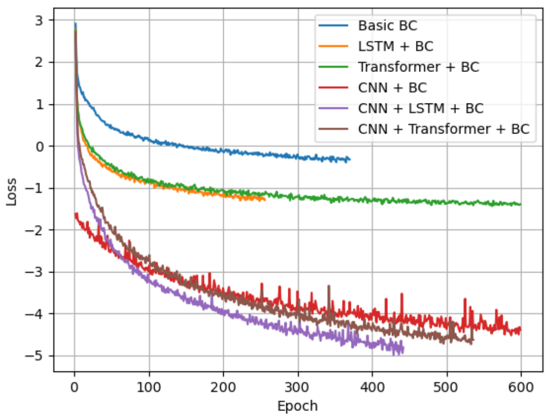


Figure 5. Test loss per epoch

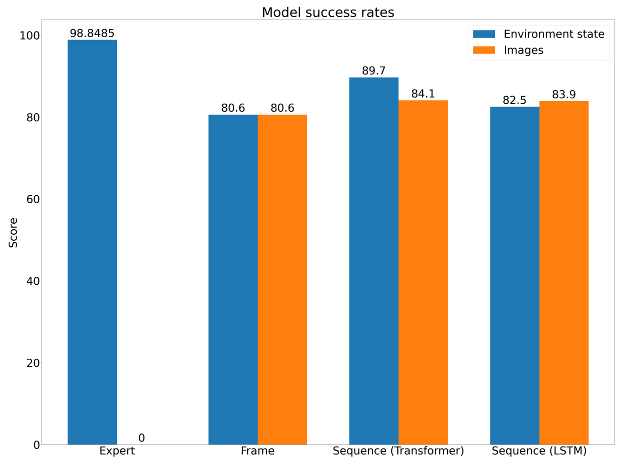


Figure 6. Comparing privileged vs video only results

The second criterion is the score, which is explained in section 4. The value of the score measures the speed of the model to complete the mission. The faster a model solves the problem, the higher score it will receive.

We also measure two metrics concerning the size and computation requirements to train each model. For each model, the number of trainable parameters as well as the training speed measured as batches per second are recorded. These two values indicate the complexity and expressivity of each model, where a higher number of parameters may allow the model a higher degree of expressivity, while a higher batch/s indicates the model may train more quickly.

5.3. Results and analysis

A table describing the training results and the metrics for each model is shown in Table 1. The expert demonstrator itself is measured as well, achieving both a high score and success rate, showing that it both quickly and consistently accomplishes the task. From the test results shown above, all 6 models we trained are able to complete the pick-and-place task with a relatively high success rate which means our models do learn from the expert about how to operate the robot arm via Behaviour Cloning.

We observe that models that take visual data as input achieve success rates comparable with that of models with access to privileged, low-dimensional information about the environment and task state as shown in Figure 6. This result demonstrates that the CNN and sequence models successfully extracted task-relevant information from images and videos only, and shows that this information is nearly as salient as a task-specific, hand-crafted representation of the environment state.

By comparing the performance of the sequential models, we find that modeling temporal information can add slight improvements to the model performance, as each sequential model outperforms the corresponding single input model. Furthermore, the transformer models perform as well or better than the LSTM models. These results confirm the increased performance in sequence modeling displayed by transformers in recent work [21], [18].

As shown by the test loss plot in Fig 5, the models all begin to converge without overfitting on the training data. This is because all of the data was generated with randomized environment initialization, so the 2400 training demonstrations sufficiently capture the possible configurations in the testing dataset as well. However, there is still a gap in performance between the trained models and the expert demonstrator. Some models may be trained for even longer, allowing the loss to continue to decrease, though larger models such as those including a CNN would require far more computation to do so. The autoregressive nature of the BC method also contributes to a lower performance and potential lack of generalizability. If the model sees an input that it has not been trained on and is out of distribution, it may make a poor prediction. This poor prediction then leads to an even worse state in the next timestep, with the error propagating because the model has no understanding of how to recover from these failing trajectories.

5.4. Model size

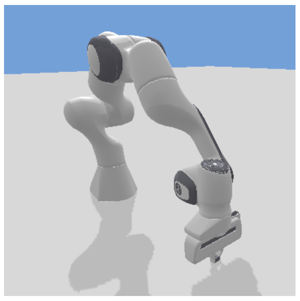
The size and training speed of each model are also shown in Table 1. The basic BC model is the simplest and fastest training model, while adding sequence modeling increases the model size by approximately an order of magnitude. Adding the CNN for processing images further increases the size by another order of magnitude, as the CNN's role in processing spatial information in an image dominates the contribution to the model size. The increase in complexity is also reflected by the batch/s values. Notably, the sequence models' addition to the CNN in the two video models show a significant decrease in batch/s, despite the proportionally minimal increase in model size. This is potentially due to the corresponding increase in size of the input itself as well as the sequential calculations in a model such as an LSTM.

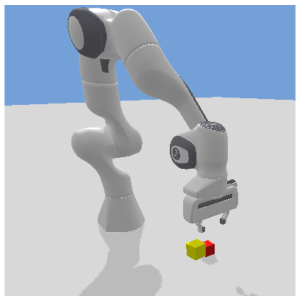
5.5. Failure modes

From qualitatively observing the resulting evaluations of the models in simulation, it is not clear why some environment states cause the model to fail. Some of these failing trajectories will closely miss the object or drop it, while others involve the robot arm moving to an entirely different position. Some failure modes for the image input, however, can be more clearly explained by the model's poor depth perception and potential for objects to be occluded. Examples are shown in Fig 7. In Fig 7a, the robot arm is fully occluding both the goal and object from the camera, so the model does not know where to move. In Fig 7b, the goal slightly occludes the object, so the model is poor at recognizing the location of the object and fails to move the robot arm to pick it up. Finally, in Fig 7c, the object is occluding the goal. This most clearly illustrates the model's poor depth perception, as the robot is holding the object at what the model perceives is the goal location, even though the actual location is farther back along the camera depth direction.

5.6. Dataset size

We conducted additional experiments using different sizes of demonstration datasets, with the models retrained using 240 demonstrations, 2400 demonstrations (Table 1), and 24000 demonstrations. The results of these tests are shown in Fig 8. While there is a clear trend of increasing dataset size improving model performance, the overall impact is relatively small. Among the models, the largest range in performance is a success rate of 14%, while the smallest range is only 0.9%. This shows that the models are able to learn the given task even with a smaller number of demonstrations, suggesting that the behavioral cloning approach can be successful when gathering expert demonstrations is difficult and the size of the dataset may be limited.


(a) Fully occluded object and goal


(b) Goal occludes object

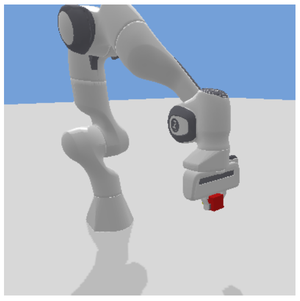

(c) Object occludes goal

Figure 7. Different failure modes for models using image inputs

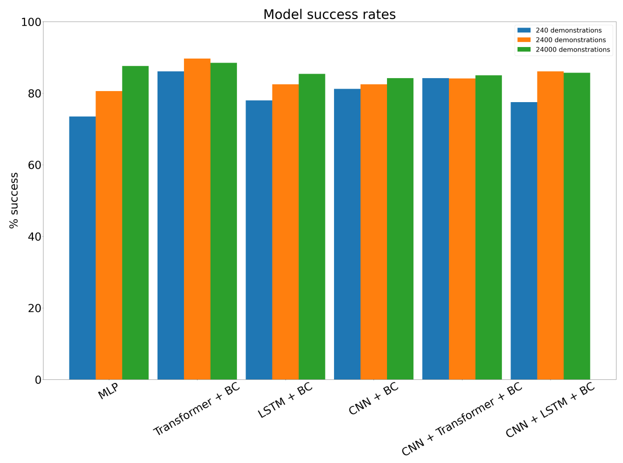


Figure 8. Model performance vs dataset size

5.7. Transformer size

Our original transformer model was relatively small compared to the original model proposed in [18]. To investigate whether transformer size would affect performance, we tested a version of the CNN + Transformer + BC model with larger input layers, more encoder and decoder layers, and more attention heads. For the same training time, the model performed slightly worse, obtaining a lower score of 30.01 compared to 32.17 and a lower success rate of 83.1% compared to 84.1%. This performance may be due to the larger size and slower training speed of the model, as it had 16% more trainable parameters. In general, however, these results suggest a more expressive model is not necessary to achieve higher performance on the task.

6. Conclusion

We successfully built an end-to-end model that learns to control a robot arm by observing video demonstrations of a robot arm solving a pick and place task and achieves an 85% success rate.

We also analyze the performance of the model when it takes the environment state or raw images as the input. We prove that the CNN is able to extract enough information about the environment for our model to learn from the expert.

In addition, we prove that our BC model could be trained with a reasonable amount of demonstrations, taking only 240 demonstrations to reach an 80% success rate. The model achieves 95% relative performance with only one-tenth of the maximum data used which means we can easily apply this model to different tasks without needing an excessive amount of video demonstrations.

There are many potential future works that can be extended from this project. For instance, we could fine-tune the model to see if it is could achieve better performance or improve the robustness of our model. One idea is to use images from different perspectives to investigate whether the model could generalize to these different angles. Lastly, we would love to explore the possibility of applying our model to other domains such as reward sparse environments where RL may not work or more complex robotic tasks. Our model is general, so we would expect it to work in a variety of environments. Lastly, we believe as access to data accelerates behavior cloning will play an increasing role in future automation technologies that will help make our lives better.

References

[1] Florent Altché and Arnaud de La Fortelle. An LSTM network for highway trajectory prediction. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 353–359, Oct. 2017. ISSN: 2153-0017. 2

[2] Bowen Baker, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos, 2022. 1, 2

[3] Brian Chen, Siddhant Tandon, David Gorsich, Alex Gorodetsky, and Shravan Veerapaneni. Behavioral cloning in atari games using a combined variational autoencoder and predictor model. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 2077–2084, 2021. 3

[4] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction, 2016. 2

[5] Quentin Gallouédec, Nicolas Cazin, Emmanuel Dellandréa, and Liming Chen. panda-gym: Open-source goal-conditioned environments for robotic learning, 2021. 1, 2, 5

[6] Adam Gleave, Mohammad Taufeeque, Juan Rocamonde, Erik Jenner, Steven H. Wang, Sam Toyer, Maximilian Ernestus, Nora Belrose, Scott Emmons, and Stuart Russell. imitation: Clean imitation learning implementations. arXiv:2211.11972v1 [cs.LG], 2022. 3

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 4

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. 5

[9] Timothy E. Lee, Jonathan Tremblay, Thang To, Jia Cheng, Terry Mosier, Oliver Kroemer, Dieter Fox, and Stan Birchfield. Camera-to-Robot Pose Estimation from a Single Image. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9426–9432, May 2020. ISSN: 2577-087X. 2

[10] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018. 2

[11] YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation, 2017. 2

[12] Ajay Mandlekar, Jonathan Booher, Max Spero, Albert Tung, Anchit Gupta, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. Scaling robot supervision to hundreds of hours with roboturk: Robotic manipulation dataset through human reasoning and dexterity. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1048–1055, 2019. 2

[13] Iago Richard Rodrigues, Marrone Dantas, Assis Oliveira Filho, Gibson Barbosa, Daniel Bezerra, Ricardo Souza, Maria Valéria Marquezini, Patricia Takako Endo, Judith Kelner, and Djamel H. Sadok. A framework for robotic arm pose estimation and movement prediction based on deep and extreme learning models, May 2022. arXiv:2205.13994 [cs]. 1, 2

[14] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Convolutional LSTM

Network: A Machine Learning Approach for Precipitation Nowcasting, Sept. 2015. arXiv:1506.04214 [cs]. 2

[15] Kiruthikan Sithamparanathan, Sarangan Rajendran, Pirakash Thavapirakasam, A.M. Harsha, and S. Abeykoon. Pose Estimation of a Robot Arm from a Single Camera. In *2021 3rd International Conference on Electrical Engineering (EECon)*, pages 131–136, Sept. 2021. 2

[16] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation, 2018. 2

[17] Sam Toyer, Rohin Shah, Andrew Critch, and Stuart Russell. The magical benchmark for robust imitation. *Advances in Neural Information Processing Systems*, 33:18284–18295, 2020. 5

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, Dec. 2017. arXiv:1706.03762 [cs]. 1, 6, 8

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. 2

[20] Felix Widmaier, Daniel Kappler, Stefan Schaal, and Jeannette Bohg. Robot arm pose estimation by pixel-wise regression of joint angles. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 616–623, May 2016. 2

[21] Neo Wu, Bradley Green, Xue Ben, and Shawn O'Banion. Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case, Jan. 2020. arXiv:2001.08317 [cs, stat]. 4, 6

A. Team Contributions Table

Name	Contribution
Alan Hesu	Model development, model training and tuning, data exploration
Cody Houff	Model development, model training and tuning, data collection
Dane Wang	Data and results analysis, report, poster
Jeremy Collins	Training pipeline development, sequence model development