Encoding Visual Modality for Robotic Manipulation Task of Peg-in-Hole-Insertion

Hao Li
Department of Mechanical Engineering
Stanford University
li2053@stanford.edu

Kaixin Chen
Department of Mechanical Engineering
Stanford University
kaixinc@stanford.edu

Abstract

This paper focuses on the application of vision encoders in the robot manipulation task of peg insertion. This is a joint project with CS229, while the CS229 project builds some control module to control our robot to perform peg insertion, the CS231N project takes the upstream task of building an vision module that encodes the vision input from a camera. We used a ResNet-18 [3] network pretrained on ImageNet as our baseline method and proposed a new algorithm. This new algorithm is based on ResNet-18 but with an additional multi-head self-attention layer. Our experiments showed that this additional multi-head self-attention provides better performance in our setup.

1. Introduction

In the past decade, deep learning in computer vision was in a phase of rapid development due to the leap forward in data sets, algorithms and computing power. This revolution has also influenced the field of robotics, which has been a major application of computer vision. Robots, empowered by better computer vision modules, are now more capable of understanding more and more complex surroundings and work spaces than ever before. At the same time, the development of deep learning in the field of controls has also enabled robots to plan and execute more advanced and more generalizable manipulation policies. The enhanced performance in sensing and understanding complex and even unseen surroundings and tasks enables robots to perform more diverse tasks in a wider range of settings with higher performance. Currently, these computer vision encoders has empowered robots to be able to perform complex tasks like grasping, placing, throwing and inserting.

This project is about the application of computer vision in robotics. Among the variety of robotics problems, using computer vision to help a robot perform peg insertion is the focus of this project as peg insertion requires high precision and could be applied to many applications, for example, a robot working in homes plugging in something in to a power supply. The application of peg insertion is not limited to inserting something, but successful peg insertion shows a robot is capable of aligning its end gripper precisely with some object, which empowers the robot to perform similar tasks like pressing a small button. As such, peg insertion is not only an active area of research, but also a useful and challenging task.

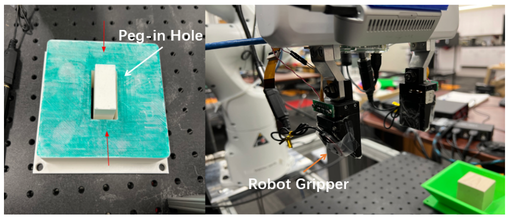

Figure 1. Franka Panda Robot Arm and Peg-in-Hole Task Setup. The red arrow on the hole base indicates the inserting direction of the peg. The robot gripper grasp the peg from the green bin and then move to some position above the hole base. Then, the robot gripper will follow the human demonstration policy to insert the peg into the hole.

Our entire algorithm of performing peg insertion with guide from vision consist of two modules, a vision module and a control module. This is a joint project with CS229. The CS231N project explores the process of designing a vision module to encode the visual input, and the CS229 project builds control modules. The combination of the two projects will make a robot able sense the position of the peg in its end gripper relative to the hole and learn a control policy using behavior cloning from human demonstration of manually controlling the robot to perform peg insertion using a key board to input action commands. Note that our demonstration and final result only performs insertion, but do not include the grasping and releasing steps. For the combination of the vision and control module, the inputs and output are the following. At train time, the available input is images taken by a third person view RGB camera over time. The outputs is predicted control action at each time step, and the loss is calculated comparing this predicted action to the action demonstration from an expert (person). At test time, the input are the same. For the vision module in specific, it takes the visual input stated earlier and outputs a single vector to that encodes useful information extracted from the input images. The imitation model (control module) build in CS229 will take this spacial representation as an input to create the final action output.

It is also necessary to state the difference and commonalities between the CS231N and CS229 project as they are not the same project. These two projects jointly performs peg insertion, and the CS231N project takes the up steam tasks of perception and the CS229 project takes the down stream task of control. Because these two projects performs the same task, they share the same data set. Also, since these two modules supposed to be trained end to end, to be able to evaluate the encoder built in the CS231N project, it is necessary to borrow a imitation network (control module) from the CS229 baseline as a tool for evaluation. However, this control module is treated as if it from an external library and the CS231N project takes no credit for it. Similarly, the CS229 project will also borrow one, and only one, encoder from this project (the baseline) as a tool of evaluating the imitation models built over there and not take any credit for it too. In addition to these differences, the most important perspective in the CS231N project is proposing a new vision encoder, and this new vision encoder is also completely owned by this project and is not used by the CS229 project.

2. Related work

The application of computer vision in robot manipulation tasks is commonly seen and have been applied to a variety of Tasks. For example, [6] [7] used CNN to guide a robot to grasp, [9] used a CNN encoder to help a robot perform imitation learning: grasping and placing, [12] used a visual encoder to grasp and throw object, and [4] used a CNN to encode the spacial features for peg insertion, similar to our work. Most of these related work all have some sort of vision module processing upstream vision input and have some sort of controlling module down stream, using the output of the vision module to control the robot.

In this paper, we categorize related works, computer vision techniques used for robot manipulation tasks, into two categories. In the first category, the vision module generates a fixed set of human interpretable representation of the robot's workspace. These features can be object segmentation of an input image or pose estimation of objects. These vision modules can be either a neural network or an algorithm with fixed policy. In the case where a neural network is used, the vision module and control module are trained separately , for example, the vision module is trained to predict object segmentation using vision input, and the control module is learning to predict the optimal action given object segmentation, [11] and [1] are examples of work with vision module that outputs object segmentation information. The meanings of these representations are usually hand crafted; for example, considering the task of grasping, the engineer can decide whether the final output of the visual module is a representation of the position of the object to grasp or a repeasentation of a 3D Reconstruction of the robot's work space. Due to the hard coded interpretation of the representations, the draw back of these methods are the weakness in expressivity and generalizability. These algorithms also sometimes relies on certain database, for example, the need of preknown 3D meshes in [8], creating greater difficulties in generalizing.

Recent related works that provides state of the art performance mostly belongs to the the second category, where the vision module,either pretrained on some other task or not, has been trained end to end with the control module eventually. Since it is trained with the control module end to end, it generates a representation that is not interpretable by humans. The intrinsic meaning of the representation depends on the data set that it is trained on. Our novel approach belongs to this category. Arguably, in this category, the most common algorithm used is performing transfer learning and fine tuning on ImageNet classifiers. [9], [10] and [4] all used ResNet-18 pretrained on ImageNet as the encoder. While most work used transfer learning, there are some exceptions, for example, [12] performed end to end training without pretraining instead.

These related works used similar hard wares, the sensory inputs are usually given by RGBD Camera [12] [2], with a few exceptions, for example using RGB camera in [9].

3. Data Set

The human (expert) demonstration data set is collected manually by the team from Stanford Vision and Learning lab. The data set includes about 80 peg insertion demonstrations of a robot controlled by a person, performing peg insertion. Within each of the demonstrations, there are approximately 100 time steps, which provides approximately 8,000 data points to work on. The train and validation split is approximately a seventy thirty split, giving approximately 5,600 training time steps and 2,400 evaluation time steps. Each of the data point includes all the necessary inputs needed for this project, namely the images taken by the cameras and the corresponding action made by the human demonstrating. Note that this is not a large data-set, especially when it comes to training some large vision encoder. As such, the method introduced in the following section focused on pretraining and fine tuning.

For human demonstration (and also the desired imitation), the action is in a three dimensional space and is discrete. More specifically, the robot arm's end gripper can only have translation motion, and is not free to rotate. At each of the time step, the end gripper can only move either negative one, zero or one unit step alone each on the x, y and z axis of some coordinate, making the action space discrete and only has twenty seven options. The size of a unit step is 1.5mm. Note that the position of the hole is the same during train, validation and test time. The vision inputs are 8 bit 480p colored images. Specifically, it has a resolution of 480 by 640. The images are not taken at a constant frame per second, instead, each image was taken after an action command was given by the expert and before an action was executed.

The first prepossessing performed is unzipping all the information saved in a hdf5 file into separate folders that a PyTorch data loader can read from. During train time, before the visual data was passed into the vision encoder, the images are resized to 180 by 120 then randomly cropped to 162 by 108. Afterward, color jitter was applied with brightness=.2, contrast=.2, saturation=.2, hue=.2. Data was transformed such a way so it act as data augmentation and alleviate the issue that the data was scarce. The transformation for test time is similar, but changing the random crop to center crop, and also removing the color jitter. The demonstration data have also been prepossessed, the initial input are a three by one vector with each of the element being in -1.5, 0, 1.5 (this is because the robot can only move either negative one, zero or one unit step along each axis and each unit step is 1.5mm) are mapped to their index 0, 1, 2 with -1.5 mapping to 0 as it is the the zeroth element in -1.5, 0, 1.5, 0 mapping to 1 and 1.5 mapping to 2. This mapping is performed as this peg insertion problem has been treated as a sequence of classification problems due to the discrete action space.

4. Methods

4.1. Vision Encoder Baseline

The baseline encoder was adapted from Res-Net-18. Since this is a transfer learning problem, Res-Net-18 Was modified to change it task form image classification to feature extraction. As such, the head of Res-Net was removed (fully connected and Soft-max non-linearity was removed), using only the convolution part of Res-Net. This baseline is chosen because it was used in [9], [10] and [4], and demonstrated strong performance. The baseline model has been trained for four times, this is both to make sure that our baseline method has been given enough chances to performs well and also to explore how to choose between pretraining, fine tuning and end to end training with random initialization. The four times of training have the following setup. Note that all setups are trained end to end.

Experiment 1.1 Used Res-Net-18 pretrained on ImageNet, performed pure transfer learning. Namely, the encoder was frozen through out the training process. Trained for 80 epochs.

Experiment 1.2 Used Res-Net-18 pretrained on ImageNet, performed fine tuning. Namely, the encoder was free through out the training process. Trained for 80 epochs.

Experiment 1.3 Used Res-Net-18 pretrained on ImageNet, performed a combination fine tuning and pertaining. Namely, the encoder was frozen for 15 epochs and unfroze afterward. Trained for 80 epochs.

Experiment 1.4. Used Res-Net-18 with random initialization. The encoder was free through out the training process, and trained for 120 epochs. This variant has been trained for a much larger number of epochs because the vision encoder was not pretrained. As such, its performance is expected to take more epoch to plateau.

These four training methods are chosen as either can potentially have the best performance. Option one has a significantly smaller Network which can avoid over-fitting. Option two and three are more expressive and will perform better if they do not over-fit too much. Lastly, since Res-Net was initially trained on classification tasks, thus there is a chance that its weights does not generalize well on the task of this project, in that case starting from scratch might be a better option.

4.2. Novel Vision Encoder

In addition to the baseline method, a novel model is proposed in this paper. It is an variant of the Res-Net-18 encoder presented above. The novel vision encoder was implemented as the following steps:

1. Obtain output from baseline Res-Net-18 encoder. This encoder should be pretrained on ImageNet.

2. Split and stack the output from Res-Net-18 encoder so that the output of the Res-Net-18 encoder has been chopped into four portions along the last dimension.

3. Perform self attention on the output of step two. Namely, perform attention with key, query and value all being the output of step two.

4. Split and stack the output of step three so that it has the same shape of the shape matches with the output of step one.

5. Add the output of step one with the out put of step four.

The rationale behind this newly proposed model is the following. Firstly, the use of self attention was inspired by the fact that what is important to the robot is the relative position between the peg and the hole, which made certain part of the image input significantly more important than the the remaining part of the image. Secondly, inspired by Res-Net, the output of the self attention layer is added on top

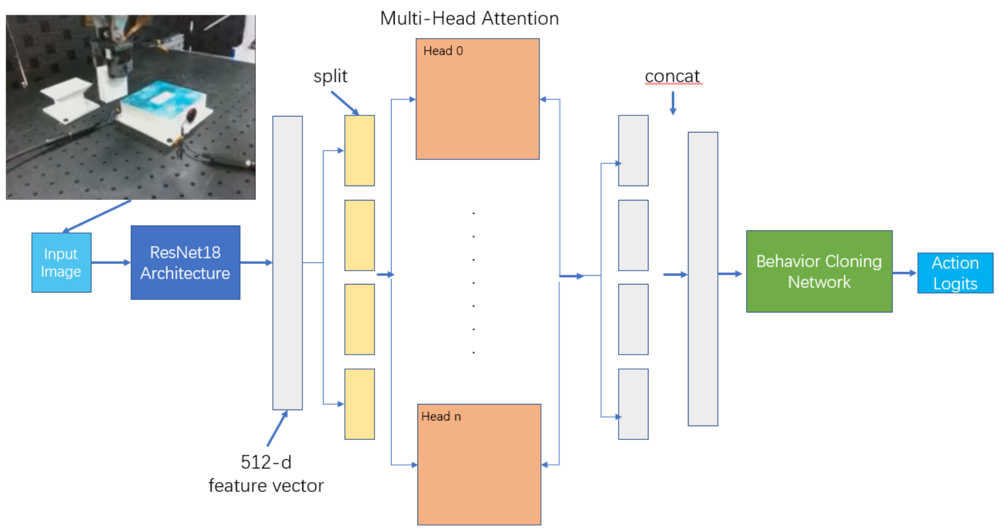

Figure 2. The Architecture of Our Proposed Model. To better address the features of the input images. We design the model as above. The model architecture starts with a stardard Resnet18 structure without the last classification layer. Therefore, the output will be a 512 dimension feature vector, including the features of the input images. Then, we implemented a multi-head attention structure. The output feature is splited input 4 different pieces, of which has 128 dimension. The split feature vectors will be input into the multi-head attention to get a more fine-grained feature of the input images. Finally, the features will be input into the Behavior Cloning Network to get the action logits.

of the output of the baseline Res-Net-18 encoder, providing better gradient flow.

The novel vision encoder is pretrained on ImageNet and then performed a combination fine tuning and pertaining. Namely, Res-Net was frozen for 15 epochs and unfroze afterward, and the self attention layer and control module was free for the entire training process. It was trained for 80 epochs. The training was also end to end. This is experiment is indexed as experiment 2.1.

4.3. Control Module

Though the control module is not what was developed in this class and we do not take any credit for it for CS231N, but due to the fact that the vision encoders are trained end to end with the control module, it is still necessary to provide some basic information about the control module. The control module is a fully connected network that performs behavior cloning of the human demonstration. The fully connected network has the following structure:

1. A batch norm layer.
2. A fully connected layer with input dimension of d and output dimension of 512, where d is the output dimension of the visual encoder. d is 512 for ResNet and 768 for ConvNeXt.
3. A batch norm layer.
4. ReLU nonlinearity.
5. A fully connected layer with input dimension of 512 and output dimension of 100.
6. ReLU nonlinearity.
7. A fully connected layer with input dimension of 100 and output dimension of 9.

Note that this is not the best model available for imitation learning, this model is chosen merely because it both provide reasonable performance and is fast to train. The human demonstrations are sub-optimal in a variety of perspectives, for example, human actions are often non-Markovian. As such, the ideal controller would be transformers or LSTM which was presented in the CS229 project, but because of the lack of computing power, these model takes approximately twelve hours to train. Since this project only use the control Module as a tool of evaluation, choosing a faster model serves similar purpose.

4.4. Optimizing

Since end to end training is used, a suitable loss function would be a loss function that evaluate the performance of the entirety of the vision module and the control module. The loss function used is a cross entropy loss, which its equation will be covered in the later section. As stated in the introduction, the action space of the robot is discrete, making cross entropy good loss metric. The ground truth (demonstration) action has a shape of three by one, each of the elements giving the index of the optimal action in each of the x,y and z coordinate, and when it was passed in to the loss function, the values in the the ground truth action has already mapped into 0,1,2. Here the label 0 represents moving one unit step in the negative direction, 1 represents not moving and vice versa. The output of out control module has the shape of nine by one. The nine by one vector is reshaped to three by three. Each of the row in the three by three matrix represents an channel (x, y or z) of the ground truth action, and the three elements in the row represents the scores of the label of 0, 1 and 2. With the predicted scores given by the control module, the ground truth action and the loss function, the loss and gradients are calculated. The optimizer used is Adam, as it provides an state of the art performance in a variety of aspects.

4.5. Implementation Details

The model was implemented using PyTorch Lightening with guidance from the PyTorch Lightening documentation and examples in the documentation. PyTorch Lighting was chosen because it saves much burden like organizing device, switching between grad and no grad and help us focus more on the model and hyper-parameters than irrelevant things. The models are trained using an Linux virtual machine with a dual-core CPU, 16GB RAM and a T4 GPU.

5. Experiments

5.1. Evaluation Metric

Since the vision module and the and control module are trained end to end, evaluating the vision module would need help from a control module. Using the control module stated in the early section (which is borrowed from CS229), the performance of the entirety is evaluated. Note that even though the controller is not optimal, but since each of the vision encoder is concatenated with the same controller, which make it comparing apple with apples. As such, the performance of each vision encoder in our setup can be compared.

There are two major evaluation metric used. The first is a mean reduced cross entropy loss mentioned earlier.

$$CrossEntropyLoss = \frac{\sum_{t=0}^{n-1} \sum_{a \in x,y,z} l_{t,a}}{3n} \tag{1}$$

With

$$l_{t,a} = -\sum_{d \in 0,1,2} 1\{A_{experts,t,a} = c\} log(\frac{exp(s_{t,a,d})}{\sum_{i \in 0,1,2} exp(s_{t,a,i})}) \tag{2}$$

Where n is the max number of time step, $A_{experts,t,a}$ the expert (ground truth) action at time step t along axis a and $s_{t,a,d}$ is the algorithm's output: the predicted score for direction (class) d at time t along axis a.

Secondly, as the action space is discrete, a good evaluation metric would be accuracy. Here, the accuracy of the action in each of the three (x,y and z) dimensions are calculated separately within each time step. For example, if the output action predicts (0,0,1) at certain time step and the ground truth is (0,1,1), then the accuracy at that time step is 2/3 instead of 0.

$$Accuracy = \frac{\sum_{t=0}^{n-1} \sum_{a \in x,y,z} 1\{A_{predicted,t,a} = A_{experts,t,a}\}}{3n} \tag{3}$$

Where predicted action $A_{predicted}$ is defined as:

$$A_{predicted,t,a} = \underset{d}{\text{argmax}} s_{t,a,d} \tag{4}$$

The validation steps take place at the end of the epoch. train loss is loaded into TensorBoard every batch and validation loss is loaded at the end of each epoch. Both train and validation accuracy are loaded into TensorBoard at the end of an epoch, recording the average accuracy over the epoch.

Among the four evaluation metric: training accuracy, validation accuracy, training loss and validation loss, the most important evaluation metric is the validation accuracy. This is because to be able to control a robot well is our end goal, which the other evaluation metrics does not characterize directly.

5.2. Results

Here another experiment is introduced:

Experiment 3.1 Used ConvNeXt-tiny [5] pretrained on ImageNet, performed a combination fine tuning and pertaining. Namely, the encoder was frozen for 15 epochs and unfroze afterward. Trained for 80 epochs.

This experiment is neither the baseline nor an novel approach, but only one of the experiment that was performed while developing a novel method. This additional experiment is shown here to show that the base line already gives high performance, and it is chosen to be displayed in the report over other experiments we perform is because it used ConvNeXt, which provides the state-of-the-art performance in the task of image classification.

The following presented results of the baseline method are trained using the following hyper-parameters. The batch size is chosen to be 8. While it might be using the GPU more efficiently with a larger batch size, it is observed that the CPU is the bottleneck. As such, it is observed that such a batch size provides fastest training. Also a small batch size give higher implicit regularization effect which is desired since the data set we have is not very large. L2 Regularization of 10^{-5} is used to ensure better generalizability as the vision encoder network is a large neural net, the regularization is small as it is only used as a tie breaker. Learning rate of 0.02 is used, and this is chosen purely by some experimentation. For the hyper-parameters used for Adam, betas are 0.9 and 0.999, these are simply the default value of the

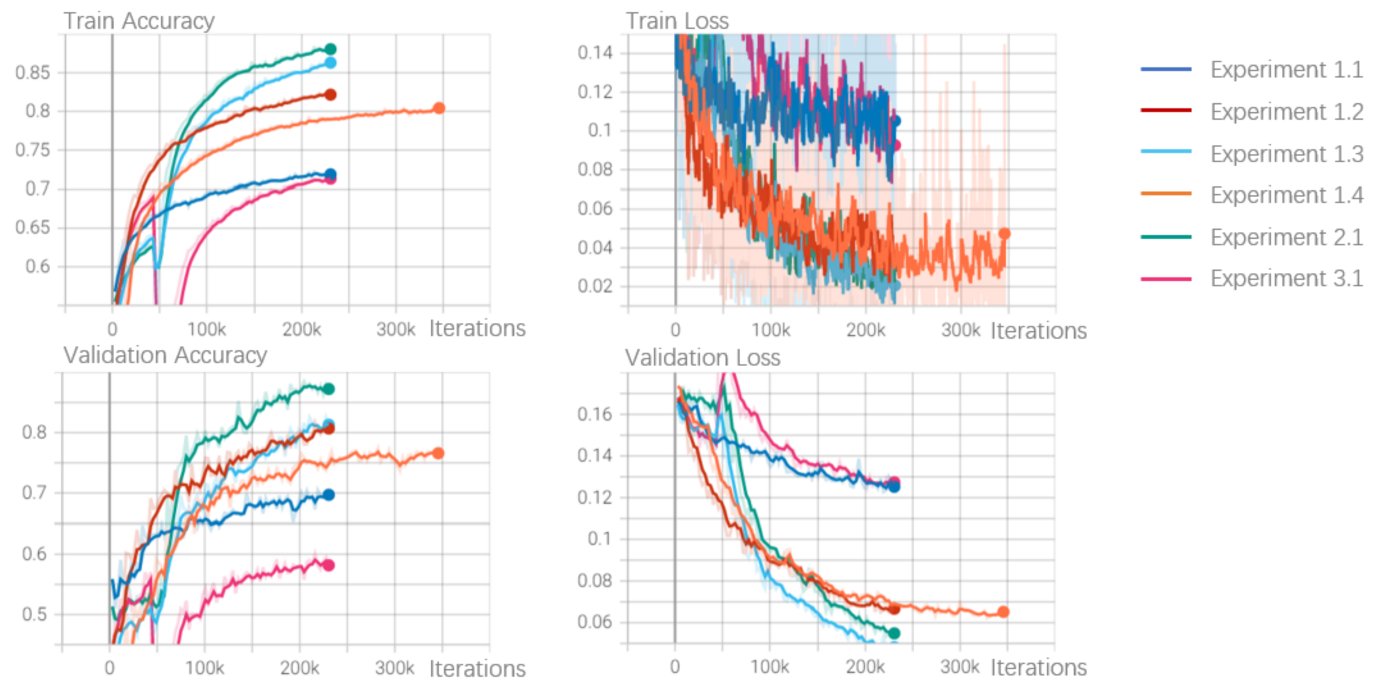

Figure 3. Experiment Results for Model Training. Every line of different color indicates an experiment we have done with different hyperparameters and models. As can be seen in the fig, our model achieves the highest accuracy among all.

ID	Model	Pretrained	Frozen for (epochs)	Max Epoch	Maximum Training Accuracy	Maximum Validation Accuracy	Difference
1.1	Res-Net-18	Yes	80	80	0.7222	0.7047	0.0175
1.2	Res-Net-18	Yes	0	80	0.8237	0.8173	0.0064
1.3	Res-Net-18	Yes	15	80	0.8648	0.8257	0.0391
1.4	Res-Net-18	No	0	120	0.8071	0.7744	0.0327
2.1	Res-Net-18 + self-attention	Yes	15	80	0.8848	0.8827	0.0021
3.1	ConvNeXt	Yes	15	80	0.72	0.6009	0.1143

Table 1. Results for each of the experiments

PyTorch Adam optimizer. Note that we did not share any experiment result with CS229, even that there is one experiment that have the same setup as another experiments in the CS229 project, we trained it twice, once for each of the CS229 and CS231N report.

Each of the baseline methods take about four hours to train for 80 epochs. Through out the base line method, and using the evaluation metric of validation accuracy, Experiment 1.3 (unfreezing at 15 epoch) performs the best. However, the experiment that performs second best, experiment 1.2 (fine tuning with no freezing) is only worse by a very Small margin. These two results are better than experiment 1.4, showing the fact that the weights in Res-Net transfers well in the task of robot manipulation. Experiment 1.1 performs the worst. Its performance is significantly lower than the variants of the base line those freed the weights in Res-Net suggested the lack of expressivity has hindered its performance. Comparing the max train accuracy and test accuracy between the four experiments, it can be seen that either of the four experiments of the baseline over fits. Also, because of the small gap between train accuracy and test accuracy, it is possible that making the model more expressive would result a better performance, which also enouraged us to add an additional self attention layer.

However, this is not the case in ConvNeXt. The gap between train accuracy and validation accuracy is significantly

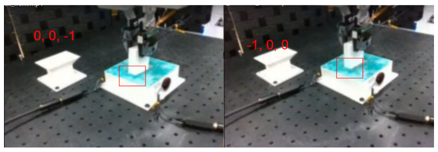

Figure 4. Real Robot Test for Our Best Model. On the left it is the wrong action that the robot is taking. On the right, it is correct action that the robot should take.

wider than use Res-Net. This suggests that the model is over fitting, which might be caused by that ConvNeXt too expressive for our dataset. Also, even though there has been over fitting, the train accuracy is not high. It is worse than all the experiments using the baseline model. This suggested that the weights or even the structure used in ConvNeXt does not transfer well to our setup.

Eventually, is the proposed method. The proposed methods take about four hours to train for 80 epochs. Experiment 2.1 beats the four experiments using the baseline in our setup in both validation accuracy and train accuracy by a large margin. Though this experiment has only shown that the proposed method is better in this set up (including the hard ware, controlling module and etc), but many related work shares the similar set up, for example, [4]. Also, the large margin also suggest that even if the setup changes, the proposed method is still highly likely to perform better than Res-Net itself. Also, it is worth noting that the proposed method also do not have a large gap between training and validation accuracy, which suggested that it has not over fit to the data.

5.3. Real Robot Experiment

We used our proposed model to test on the real robot. We performed 50 real-robot test with each trail starting at a random initial position. If the peg is inserted in the hole in the end, we will count it as a successful case. Otherwise, we will consider it as a failure case. Finally, our model achieves a success rate of 84%. In the failure cases, as shown in figure 4., the model predicts the wrong action at the edge of the hole base. This is because there is some vague representation in the vision. On the edge of the hole base, the correct action is moving back, but sometime, vision did not see it clearly, therefore, it predicted moving down and stuck on the edge.

Video clip of the demonstration of successful peg insertion serves as the qualitative result for our project, which is available at:

https://youtu.be/mbyZ9o9rxJU

6. Conclusion

The the application of deep learning and computer vision in the field of robotics and robot manipulation has been a hot topic of research. While many related work have presented successful examples of using ResNet-18 as the vision encoder for robot manipulation tasks, this paper presented a novel vision that beats ResNet-18 in the robot manipulation task of peg insertion by demonstrating higher validation accuracy. The new algorithm is based on ResNet-18, pretrained on ImageNet, with an additional self attention layer.

There are many possible future directions. First is that because it have been demonstrated that adding an additional self attention layer can enhance the performance of a CNN encoder, it is highly likely that transformers will work well in similar setup, as transformers include various multi-head self-attention layers. While exploring in this direction, a good baseline would be performing transfer learning using Swin transformers, which provides the state-of-the-art performance in image classification. The second possible direction is to also explore the performance of our variant and the base line method when the input is from multiple cameras or RGBD cameras. However, one limitation is that since the number of image inputs are doubled or even tripled at each time step, the training time will increase significantly.

References

[1] Matei Ciocarlie, Kaijen Hsiao, Edward Gil Jones, Sachin Chitta, Radu Bogdan Rusu, and Ioan A. Şucan. *Towards Reliable Grasping and Manipulation in Household Environments*, pages 241–252. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. 2

[2] Eppner Clemens, Höfer Sebastian, Jonschkowski Rico, Martín-Martín Roberto, Sieverling Arne, Wall Vincent, and Brock Oliver. Four aspects of building robotic systems: lessons from the amazon picking challenge 2015. *Auton Robot*, 42, 2018. 2

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 1

[4] Yifang Liu, Diego Romeres, Devesh K. Jha, and Daniel Nikovski. Understanding multi-modal perception using behavioral cloning for peg-in-a-hole insertion tasks, 2020. 2, 3, 7

[5] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022. 5

[6] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. 2017. 2

[7] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. Learning ambidextrous robot grasping policies. *Science Robotics*, 4(26):eaau4984, 2019. 2

[8] Jeffrey Mahler, Florian T Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James Kuffner, and Ken Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1957–1964. IEEE, 2016. 2

[9] Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. Learning to generalize across long-horizon tasks from human demonstrations, 2020. 2, 3

[10] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation, 2021. 2, 3

[11] D. Morrison, A. W. Tow, M. McTaggart, R. Smith, N. Kelly-Boxall, S. Wade-McCue, J. Erskine, R. Grinover, A. Gurman, T. Hunn, D. Lee, A. Milan, T. Pham, G. Rallos, A. Razjigaev, T. Rowntree, K. Vijay, Z. Zhuang, C. Lehnert, I. Reid, P. Corke, and J. Leitner. Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge, 2017. 2

[12] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics, 2019. 2