Deep-Dynamic Movement Primitives learning with Convolutional Neural Networks

handed in
MASTER'S THESIS

Bachelor of Eng. Yuecheng Mao

born on the 04.09.1991
living in:
Schroefelhofstrasse 24
81375 Munich
Tel.: 017647386527

Human-centered Assistive Robotics
Technical University of Munich

Univ.-Prof. Dr.-Ing. Dongheui Lee

Supervisor:	M.Sc. Affan Pervez
Start:	01.01.2017
Intermediate Report:	19.05.2017
Delivery:	21.07.2017

Abstract

KUKA Light Weight Robot is a programmable machine with a single arm, which has seven degree of freedom in joint space and three in cartesian space. The robot can generate trajectories with its end effector to finish complex tasks. Executing different actions at each time step allows the robot to move from start point to goal point through a certain trajectory as human expected. Dynamic Movement Primitive encodes motions by using nonlinear forcing terms for planning and control complex trajectories. Extended Dynamic Movement Primitive injects task parameter dependence into the nonlinear forcing terms for intelligent movement of robot.
This thesis aims to model task parameterized forcing terms with high level representation of task parameter by using deep neural networks for doing sweeping task on KUKA robot. In this work, the task parameter is extracted from RGB images from camera. Two different approaches i.e., CNN and a combination of CNN and RNN are developed. The neural network is used as function approximator, which maps image and clock signal to the nonlinear forcing term directly. Additionally, the CNN model is generalized for multiple real objects by using data augmentation. After training the neural network, the robot is able to do sweeping task from start position to dustpan with tracking real object in real time.

Contents

1 Introduction **5**
1.1 Problem Statement 6
1.2 Related Work 7

2 Technical Approach **9**
2.1 Overview of Design 9
2.2 Relevant Methods 10
2.2.1 Discrete Convolution 10
2.2.2 Activation Function 11
2.2.3 Soft Attention 12
2.2.4 Long Short Term Memory 13
2.3 Implementation 13
2.3.1 Initial Researched Approach 13
2.3.2 Switch Approach from CNN + RNN to CNN 18
2.3.3 Main Approach: Convolutional Neural Network 19

3 Evaluation **29**
3.1 Experimental Setup 29
3.2 Evaluation with CONVRNN 30
3.2.1 Evaluation on Dataset 30
3.2.2 Evaluation on Robot 37
3.3 Evaluation with CNN 38
3.3.1 Error of End-to-End Training 38
3.3.2 Evaluation on Dataset 39
3.3.3 Evaluation on Robot 43

4 Discussion **47**

5 Conclusion **51**

A DVD **53**

List of Figures **55**

Bibliography **59**

Chapter 1

Introduction

Trajectory or motion planning is a common task in robotics. Basically, robot has to react and move based on its environment or more specifically sensor inputs. In dynamic environments, such as the real world, the state of objects are time variant and space variant. This increases difficulty for robot to do the correct action. The state space and action space of robot in the real world are essentially indefinably large. Nowadays hard coding every possible situation for robot controller becomes less efficient in industry. Fortunately, machine learning is field of study that "gives computers the ability to learn without being explicitly programmed" [Mun14]. However, it is hard to apply only a single machine learning algorithm for doing complex tasks, i.e. it can not do End-to-End learning for long term dependency between inputs and expected outputs. In the most case, it combines many algorithms to finish a complex task, e.g. combining vision part, feature part and controller part of robot. It still needs much effort to implement.

Recently, deep learning is increasingly becoming popular and important for learning representations of data. It has huge success in many fields, such as computer vision and natural language processing, e.g., Convolutional Neural Network (CNN) is mostly used in for visual tasks, e.g., image classification like VGGnet [SZ14] and Recurrent Neural Network (RNN) prefers to be used for handling sequential data [SVL14]. Theoretically, deep learning is able to approximate any kind of continues functions and can be used as Universal Function Approximator [GBC16]. This property makes lots of machine learning enthusiasts digging into deep learning.

The aim of this work is to learn a deep neural network that gives KUKA light weights robot ability to move correctly in on-line manner for sweeping task. Figure 1.1 shows the overview of sweeping task. $\mathcal{H}$ denotes the home position of robot and $\mathcal{G}$ is the goal position where robot moves to. The shape of trajectory depends on the trash position $\mathcal{T}$. This is to say, there are infinite different shapes of trajectory which can not be predefined. The trash position could also be changed during execution of a motion, i.e., the robot has to track the trash in on-line manner. In this work, the research work involves Recurrent Neural Networks and Convolutional Neural Networks. The trash is firstly a simple object glued with a special marker. Both

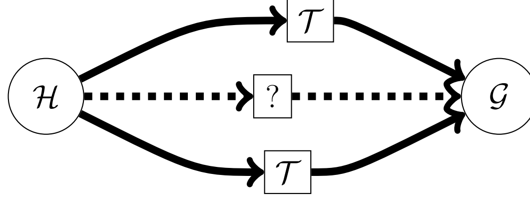


Figure 1.1: Sweeping task: robot starts from home position $\mathcal{H}$ and pushes trash $\mathcal{T}$ into goal position (dustpan) $\mathcal{G}$, where $\mathcal{H}$ and $\mathcal{G}$ are fixed, on the contrary, the trash position $\mathcal{T}$ is unknown and could be varying in real time.

approaches are successfully evaluated on real robot for marker object. As a extended task, the CNN is generalized for multiple real objects by using data augmentation.

1.1 Problem Statement

Dynamic Movement Primitive (DMP) provides a way for encoding motion data [INS03] and gives robot ability to generate different trajectories. DMP model consists of two dynamical systems with parameterized connection, where one system drives the other one by using a clock signal. The KUKA light weight robot has three Degree of Freedom (DoF) in Cartesian space, which are encoded with three DMPs, i.e., two for planar movement and one for rotation of robot's end effector.

The transform equation of the dynamical systems [PL17] is given by

$$\tau \dot{v} = K(g - x) - Dv - K(g - x_0)s + sK\mathcal{F}(s) \tag{1.1}$$

where s is the clock signal, $\mathcal{F}(s)$ is the forcing term and other components are not relevant to this work.

The three DMPs in Cartesian space correspond three forcing terms $\mathcal{F}(s)$ in (1.1). Hence, the robot is able to generate different trajectories by feeding the nonlinear forcing terms into (1.1). In case of sweeping task (see Fig. 1.1), the forcing term is not only dependent on the clock signal but also the task parameter $\mathcal{T}$, i.e., $\mathcal{F}(s, \mathcal{T})$. If all forcing terms are zeros, then the robot moves from home position to goal position over a straight line (see dashed line in Fig. 1.1).In a planar view, the trash position $\mathcal{T}$ is a two dimensional variable which has x and y direction. Hence, the forcing term is defined as $\mathcal{F}(s, \mathcal{T}_x, \mathcal{T}_y)$ [PL17]. Since the task parameter has to be replaced with high level representation i.e. images. The goal of this work is to find a mathematical mapping between inputs (image and clock signal) and outputs (forcing terms), as shown in Fig. 1.2. Since there is no simulation tool available in the lab, evaluation is done on real robot directly with a safety button.

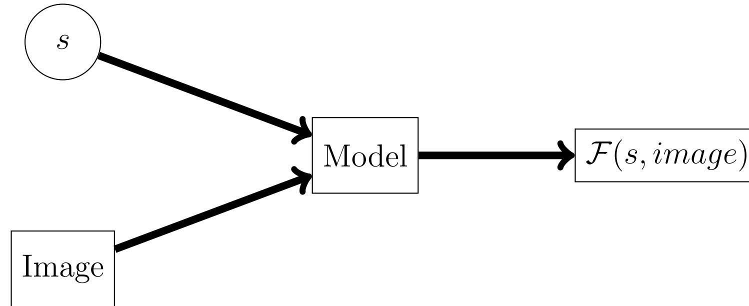


Figure 1.2: Research purpose: build a model using deep neural network, which end-to-end maps inputs to expected outputs.

1.2 Related Work

The essential goal of this work is to learn the forcing terms $\mathcal{F}$ in (1.1). Pervez et al. [PL17] have implemented an approach for learning the forcing terms. Programming by Demonstration (PbD) is used to demonstrate in [PL17] how the robot should move. The forcing terms of each demonstration is modeled by using Gaussian Mixture Model (GMM). Pervez et al. also compared different regression approach for predicting the forcing terms and evaluated on real robot. Their work is the major background of this thesis. The dataset of this work is collected by executing their program on robot, i.e., the predicted forcing terms of [PL17] are referred as the ground truth outputs.
The basic idea of this research comes from [LFDA16]. It is the most similar situation to this work. An end-to-end Convolutional Neural Network with a special softmax layer is introduced in [LFDA16]. The special softmax layer transforms object position in image into real world position. It helps a lot to this work. In their approach, inputs are mapped into motor torques and it needs reinforcement learning to train current movement at each time step for future reward. In our case, DMP model is already implemented and ensures that robot moves from home position to goal position. Hence, at each time step the forcing term can be handled independently in this work. Different hard tasks are evaluated in [LFDA16] and sweeping is main task of this work. Instead of hard covering background with green stuff within the images in [LFDA16], the images from our data set are pretty noisy with different objects in the background. This makes training process harder.
The rest of this report is structured as follows: Chapter 2 describes the main technical approach for solving our problem. In Chapter 3, evaluation results are shown and interpreted. The important aspects and problems of proposed approach are discussed in Chapter 4. In the last Chapter, this thesis is concluded and an overview of future work is imagined.

Chapter 2

Technical Approach

2.1 Overview of Design

The whole project is divided into three stages as follows:

- collecting dataset
- implementing and training model
- evaluating trained model on real robot

Firstly, a dataset is needed for training neural network. The dataset is collected with one Kinect camera and two computers on Ubuntu system by running the

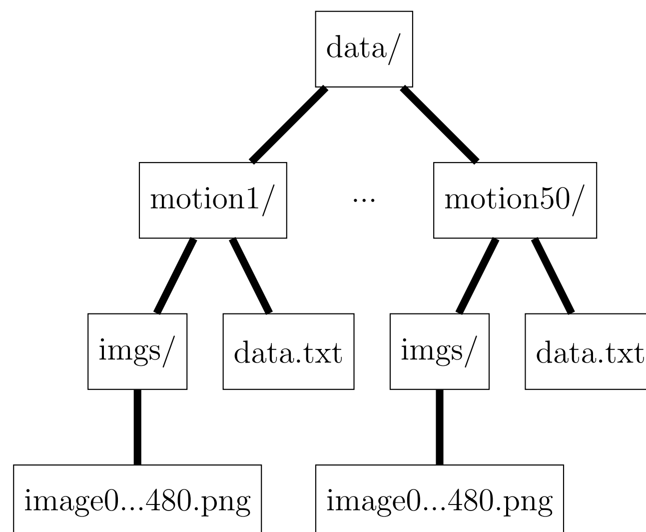


Figure 2.1: Dataset structure: the data folder contains 50 motion folders, each motion folder contains a 'imgs' folder and text file and each *imgs* folder contains about 480 RGB images

Columns description	clock signal	joint angles	end effector position	trash position	forcing terms
Number of values	1	7	2	2	3

Table 2.1: Columns description in *data.txt* file.

existing TP-DMP model, which is implemented in [PL17]. One computer runs DMP program for generating data and the program of collecting data is executed on the other computer. The communication interface between the two computers works under Robot Operating System (ROS) and all of this are implemented in C++. Figure 2.1 shows the structure of collected dataset. The dataset consists of about 24 000 samples within 50 motions. Each image has a resolution of 480×640 pixels. In a single *data.txt* file, the rows correspond number of samples and the columns correspond features or labels. The detailed structure of a single row in *data.txt* is given in Tab. 2.1. Its delimiter is comma (,) for each row. As shown in Tab. 2.1, the last three columns are forcing terms i.e. outputs, which need to be predicted. Therefore, any other columns and images could be the inputs, which are also called features in machine learning.
At the final stage, the well trained model is evaluated on real robot.

2.2 Relevant Methods

2.2.1 Discrete Convolution

Image convolution is widely used linear transformation in Convolutional Neural Network. A weights matrix also called kernel is element wise multiplied with image patch over the whole image. The resulting matrix is summed up for each image patch respectively. Usually, the output matrix is referred to as the feature map. The convolution formula [GBC16] is given by (2.1).

$$F(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.1}$$

where n is the kernel width, m the kernel height, K the kernel matrix and I input image matrix.
For multichannel images e.g. RGB image the formula is similar to (2.1). An additional summation is performed over image channels. The 2-dimensional discrete convolution on an multichannel image is given by (2.2).

$$F_k(i,j) = \sum_c \sum_m \sum_n I_c(i+m, j+n)K_{ck}(m,n) \tag{2.2}$$

where n is the kernel width, m the kernel height, c the number of input channel (e.g. for RGB image, c = 3) and k the number of output channels or feature maps.

2.2.2 Activation Function

Activation functions are nonlinear transformation in deep neural network. This gives neural network ability to approximate nonlinear complex functions. There are two activation functions described here, i.e., *tanh* and *ReLU*, which are mainly used in this work. Sigmoid function is included in our model as well, but it is a built in function in LSTM cell. It is similar to *tanh* and therefore not explained here.

tanh

The *tanh* activation function is given by (2.3). Its plot is shown in Fig. 2.2. The advantage of the *tanh* function is that its outputs are zero-centered and located at range $(-1, 1)$. However, it saturates at either tail of -1 or 1, because the gradient of *tanh* is almost equal to zero when its output is nears to -1 or 1 [GBC16].

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.3}$$

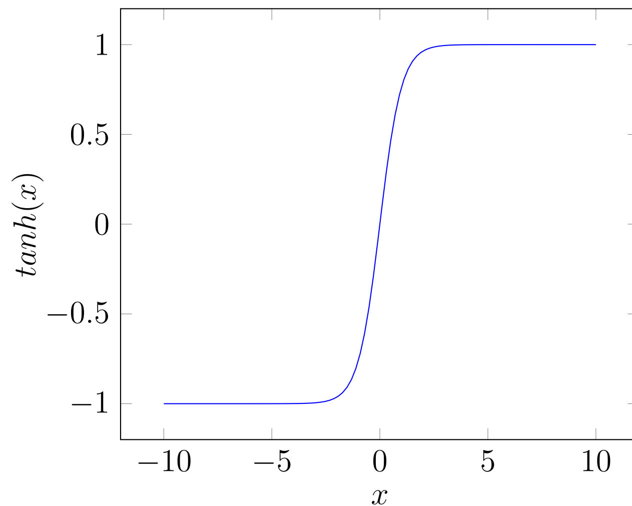


Figure 2.2: tanh activation function

ReLU

The Rectified Linear Unit (ReLU) is another popular activation function and its formula is given by (2.4). Figure 2.3 shows the ReLU activation function. The activation is simply thresholded at zero when its inputs are negative values. For

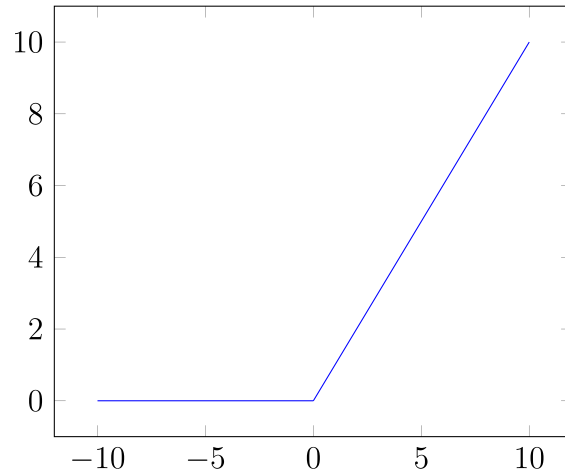


Figure 2.3: ReLU

values at positive range, it has no saturation. Unfortunately, a ReLU neuron can be "dead" during training. If a large gradient flows through a ReLU neuron and its weights are over-updated so that the inputs to ReLU are all negative values. The outputs of ReLU are all zeros if its inputs are all negative. It cause the neuron to be "dead" [GBC16].

$$f(x) = max(0, x) \tag{2.4}$$

2.2.3 Soft Attention

softmax

The *softmax* function transforms a score vector to a probability vector associated with a multinoulli distribution. The *softmax* function is defined to be (2.5). Since its output is a probability vector, the summation over all elements results 1.0 [GBC16].

$$softmax(\mathbf{x})_i = \frac{exp(x_i)}{\sum_{j=1}^{n} exp(x_j)} \tag{2.5}$$

soft-argmax

The soft-argmax is an *argmax* operation with soft attention mechanism [LFDA16]. Let $I(x, y)$ be a single channel image or a feature map, the soft-argmax on 2D image is given by 2.6.

$$f_x = \sum_{i,j} softmax(I)_{ij} x_{ij},$$
$$f_y = \sum_{i,j} softmax(I)_{ij} y_{ij} \quad (2.6)$$

Where (x_{ij}, y_{ij}) is the image-space position of the point (i, j) in the image, I is the 2D image matrix and (f_x, f_y) is the expected feature point in the image. I has to be flattened into vector before $softmax$ and is reshaped back after $softmax$. In other words, this "softpicks" out the pixel position in image coordinate system with the largest pixel intensity. It is called "soft", because its position is multiplied by a probabilistic vector. The soft mechanism make this operation differentiable. It can be integrated in neural network directly compared to simple $argmax$, which is not differentiable and has no derivatives.

2.2.4 Long Short Term Memory

Long Short Term Memory (LSTM) is a popular gradient-based Block Cell, which is mostly used in Recurrent Neural Network for dealing with vanishing gradient problem. Hence, it is suited for long time dependency prediction. The used LSTM cell in this work is based on [ZSV14] according to TensorFlow's reference.

2.3 Implementation

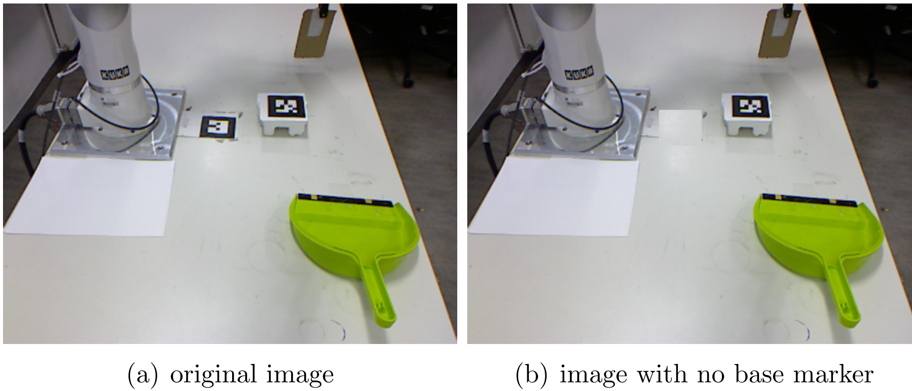
2.3.1 Initial Researched Approach

The initial research purpose is trying to solve the problem of sweeping task by using RNN. The following sections describe a mathematical model of combination of CNN and RNN.

Data Preprocessing

As described in Sec. 2.1, the outputs of dataset are forcing terms and the inputs consist of images, clock signal and additional robot configurations. As the images are the only inputs of CNN, normalization of pixel value is not needed here, i.e., the intensity of images is at range [0, 255]. Since the dataset is collected with TP-DMP model [PL17]. In their model, a base marker is needed for localization of objects, as shown in (a) of Fig. 2.4 (left marker near the robot base). It can not be removed during collecting data. When the convolution operator is executed on a image, each kernel (also called filter) is shared over every image patch. The base marker looks similar as the trash marker (see the right marker of (a) in Fig. 2.4). Hence, the base marker is replaced with a small image patch at training phase, which is cropped from the background of desk area, as shown in (b) of Fig. 2.4. If the base marker

were not removed, then the robot could oscillate between base marker and trash marker, even though the base marker doesn't affect the training error at training phase evidently. As model is evaluated on real robot, it is removed for robot safety. This problem will be elaborated in Chap. 4. The second preprocessing of image is a trivial resizing operation. The original image (480×640) is resized to 200×200 for avoiding out of memory and computational efficiency.



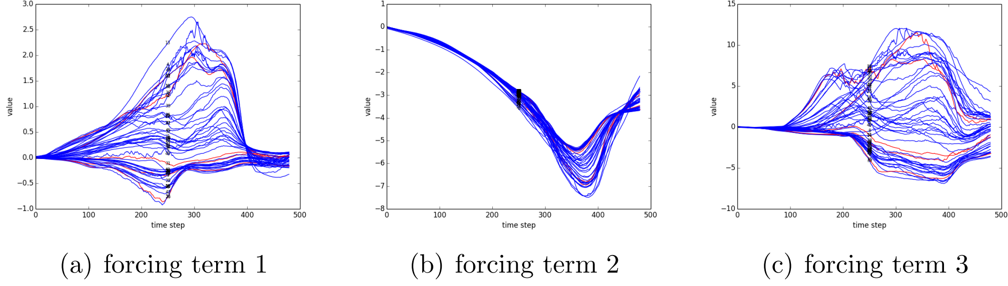
(a) original image (b) image with no base marker

Figure 2.4: Image preprocessing: (a): original image, (b): Base marker is covered

In machine learning as well as deep learning, it is necessary to split data into training and test set. Since the evaluation or testing is performed on real robot, the dataset is divided into training set and validation set for avoiding overfitting. The collected dataset contains 50 motions, as described in Sec. 2.1. The expected moving behavior of learned robot is that the robot can generating new motions by imitating the motions within training set. Hence, the dataset is split with 45/5 ratio across motions i.e. 45 motions for training and 5 motions for validation. As only 50 motions are available, random splitting is not suited to this situation. The dataset is split according to the distribution of forcing terms, as shown in Fig. 2.5. The 5 red colored motions are picked out as validation motions and the remaining motions are moved into training set. The criteria of choosing testing motions are shape of curve and different value ranges. After splitting data, the training set contains 21 584 samples and the validation set has 2401 samples.

Building and Training Model

The designed model is shown in Fig. 2.6. It contains CNN, RNN and fully connected layers. It is named to be CONVRNN here for ease of interpretation. CONVRNN has a total of 9 layers. The first layer is a colored image with three channels i.e. red, green and blue followed by three convolutional layers. The outputs of last convolutional layer are transformed into feature points by using soft-argmax operator, as described in Sec. 2.2.3. The feature points are not be passed through fully connected layers directly compared to Levine et al. [LFDA16]. A bidirectional RNN is



(a) forcing term 1 (b) forcing term 2 (c) forcing term 3

Figure 2.5: Splitting dataset: red curves for testing and blue curves for training

inserted between CNN and fully connected layers, i.e., the feature points (x_i, y_i) are considered as a sequence of two dimensional points. This makes training faster and it achieves lower training and validation error than only using CNN for our case (see Fig. 2.7). The outputs of RNN are concatenated with clock signal from robot as the inputs of fully connected layers. The bidirectional RNN contains a forward layer and a backward layer. For both directions, LSTM cell is used as memory cell. The LSTM cell is shared within either forward or backward RNN, but not shared across the two directions of RNN. The red and blue circles in Fig. 2.6 show the LSTM cell for both directions. The red cells or blues cell are the same cell. This bidirectional RNN is an unfolded version, as shown in brown square of Fig. 2.6. This is to say, each feature point (x_i, y_i) is passed into the same cell but at different time step. The LSTM cells for both directions have 16 hidden units. The outputs at last time step of both directional RNN are chosen for further computing (double lined red and blue arrows in Fig. 2.6), because only the output of last time step contains all the informations of all previous time steps. A bidirectional RNN is used here, because the feature points are not a time series but only a sequence. More information can be extracted theoretically in two directions compared to one direction. The two fully connected layers have 24 neurons respectively. The *tanh* activation function is used in fully connected layers. The reason for this is that *tanh* outputs a zero-centered values as described in Sec. 2.2.2 and the forcing term1 and forcing term 3 are zero-centered approximately as well (see Fig. 2.5).The forcing term 2 stays almost at the same distribution for all motions and is easy to learn. Also, the *tanh* outperforms *sigmoid* and ReLU after several experiments.
As the soft-argmax function is described in 2.2.3, it transforms feature maps into feature points. For CONVRNN, the last ReLU activation results a size of 95×95 feature map for each channel. To illustrate it clearly, the x position in image space of feature point (x_i, y_i) is calculated by (2.7). The y position is given by (2.8).

$$x_i = \sum_{k,l} (softmax(featuremap_i) \circ \begin{bmatrix} 0 & 1 & \cdots & 94 \\ 0 & 1 & \cdots & 94 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & \cdots & 94 \end{bmatrix})_{k,l} \tag{2.7}$$

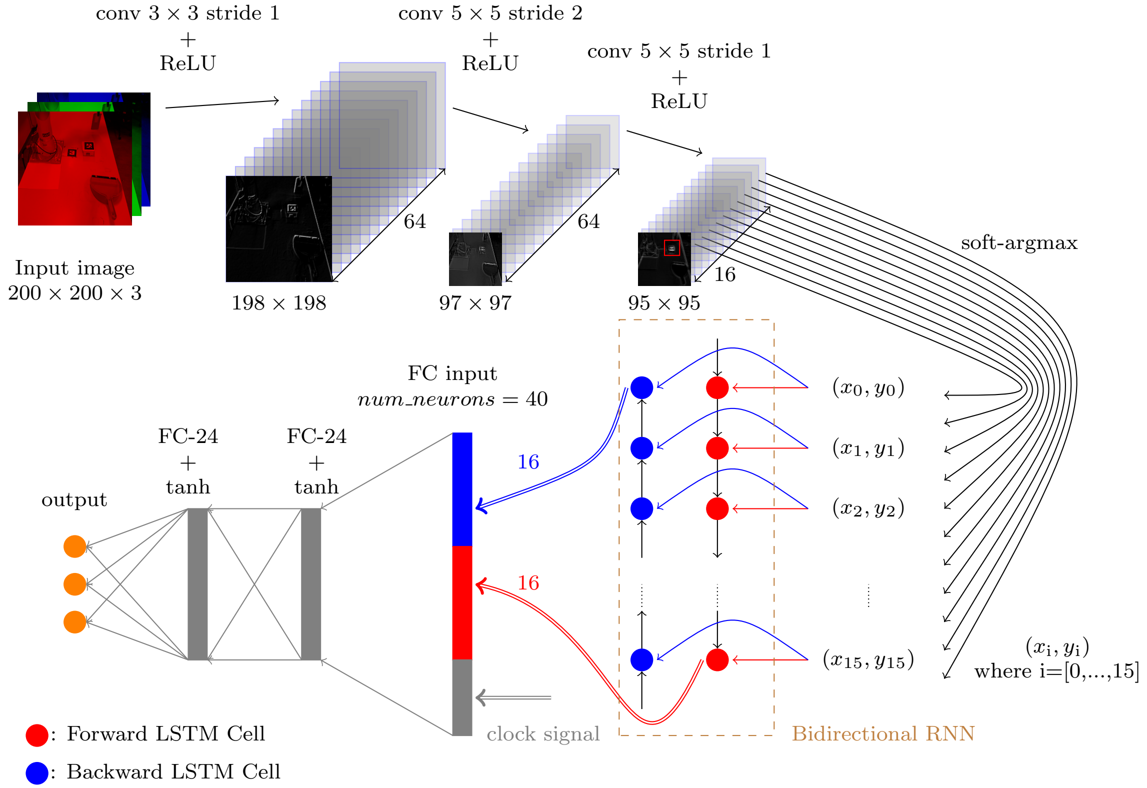


Figure 2.6: Architecture of CONVRNN: The input is a RGB image, followed by three convolutional layers, i.e., conv1: kernel size: 3×3, stride 1, ReLU activation, depth 64, conv2: kernel size: 5×5, stride 2, ReLU activation, depth 64, conv3: kernel size: 5×5, stride 1, ReLU activation, depth 16. All convolution operation are performed with VALID Padding. A soft-argmax operation is performed after last activating and results 16 feature points over 16 feature maps. The 16 feature points go through a bidirectional RNN with 16 hidden units for both direction. The last outputs of of forward and backward RNN are concatenated with additional robot parameters (40 neurons), then passed through two fully connected layers with tanh activation, which both have 24 neurons. The last layer is output layer i.e. the forcing terms. Hence, the architecture contains a total of 9 layers including input layer and output layer.

$$y_i = \sum_{k,l}(softmax(featuremap_i) \circ \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 94 & 94 & \cdots & 94 \end{bmatrix})_{k,l} \tag{2.8}$$

Where $\circ$ denotes element wise product or Hadamard product, $i = [0...15]$, $k, l =$

[0...94] and feature maps have to be flattened before $softmax$, the result of $softmax$ has to be reshaped back to 95×95.

As CONVRNN has 9 layers, the gradients are hard to flow through the convolutional layers if the neural network is too deep (vanishing gradient problem [Hoc91]). A large learning rate can cause the ReLU units to be "dead" easily. Another problem is that the inputs of robot configurations are inserted after RNN, i.e., the goal of CNN and RNN is to extract the trash position from images. Hence, pretraining is needed. CONVRNN is sliced into two parts and the break point is at the outputs of RNN. The CNN and RNN are pretrained for predicting the trash position. The trash position neurons and the RNN outputs (32 neurons) are connected with an additional trainable weight matrix and biases vector. This weight matrix transforms RNN outputs to trash position, i.e., $\mathbb{R}^{32} \rightarrow \mathbb{R}^2$. The first convolutional layer is initialized with the weights and biases of the first layer of VGGnet [SZ14]. The loss function is defined to be Mean Squared Error (MSE) and given by (2.9).

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{2.9}$$

After pretraining, this additional weight matrix and bias vector are removed. All the outputs of RNN are computed by using pretrained CNN and RNN and saved into text file. The predicted outputs of RNN are assumed as a representation of trash positions. The fully connected layers are pretrained as well by using the saved prediction of RNN outputs, clock signal. In the final step, all pretrained weights and biases are loaded into CONVRNN for doing End-to-End training. At this training phase, a smaller learning rate is selected for fine tuning the model. The loss function is MSE for both pretraining and fine tuning. Adam optimizer is used for backpropagation [KB14].

Since the training is not an one-step End-to-End training, regularization is done by checking validation error and early stopping. This makes the pretraining easier to be controlled by human.

Overall, the training steps are summarized as follows,

1. pretraining CNN and RNN for predicting trash position
2. save trained parameters of CNN and RNN
3. save predicted outputs of RNN
4. pretraining fully connected layers by using predicted outputs of RNN, clock signal and robot configurations
5. save trained parameters of fully connected layers
6. load trained parameters from the pretraining steps into CONVRNN
7. doing End-to-End training

Hyperparameter

The used hyperparameters for this model are given in Tab. 2.2. The final model i.e. CONVRNN contains **117579** trainable parameters. The hyperparameters can be varying and are not defined restrictedly for reproducing the results.

Hyperparameter		
Pretraining of trash position	**Pretraining of fully connected layer**	**End-to-End training**
batch size: 100	batch size: 100	batch size: 100
learning rate: 0.001 decayed by half every 8 epochs	learning rate: switch from 0.01 to 0.001 and then to 0.0001 about every 20 epochs	learning rate: 0.0001 decayed by half every 5 epochs
numpy seed number: 1	no seed: easy to train	numpy seed number: 1
TensorFlow seed number: 1	no seed: easy to train	TensorFlow seed number: 1

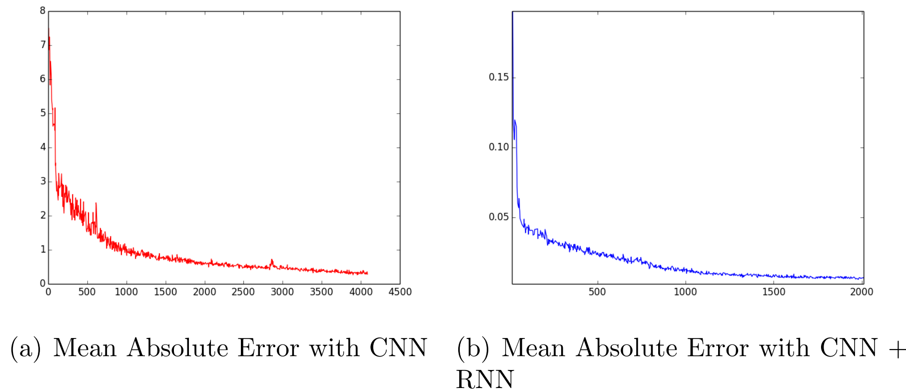
Table 2.2: Hyperparameters for training CONVRNN.

2.3.2 Switch Approach from CNN + RNN to CNN

Comparison of CNN and CNN + RNN Figure 2.7 shows the Mean Absolute Error (MAE) while model is at training phase. The horizontal axis indicates the training steps. These are only examples not the final loss of the well trained model. The left red curve illustrates the training loss with single CNN. Its loss converges to

a higher value than the right blue curve, which is a combination of CNN and RNN. After few Training steps or even at the beginning of the two curves, the right one outperforms the left one.

The reason for this is that the position matrices used in (2.7) and (2.8) are not normalized. The model with combination of RNN and CNN does not have to normalize the position matrices, because the output of LSTM has the activation function tanh and its outputs are zero-centered at range $(-1, 1)$. The trash positions in collected dataset are zero-centered at range $(-1, 1)$ as well. Since the problem of the approach with a single CNN is found, a second approach with a single CNN is developed and described in the section. It has less trainable parameters than initial approach and can be easily generalized for multiple real objects instead of a single object with special marker.



(a) Mean Absolute Error with CNN (b) Mean Absolute Error with CNN + RNN

Figure 2.7: CNN vs. CNN + RNN: left red curve CNN, right blue curve CNN + RNN

2.3.3 Main Approach: Convolutional Neural Network

The basic idea of this approach is highlighting. If the position of expected object within an image can be highlighted, then the object position in image coordinate system can be extracted and processed for further use. Figure 2.8 shows an example overview of this idea. The input can be either a RGB image or a gray scale image. The model in between should be able to transform the input image to an image with highlighted object. Once the object is highlighted, the approximate position of this object can be easily extracted. For a single image, this can be done with a simple linear transform on the input image. As there are infinite different images for this work, a CNN can be used for highlighting the expected object in a complex environment.

The following section describes how the sweeping problem for the trash glued with a special marker can be easily solved with CNN. Also, the CNN is generalized for multiple real objects as an extended task.

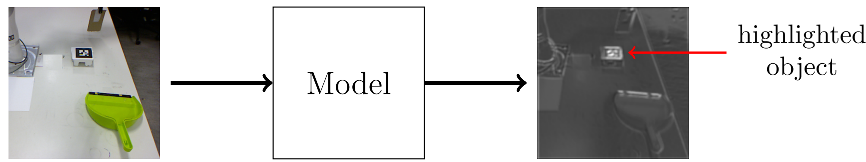
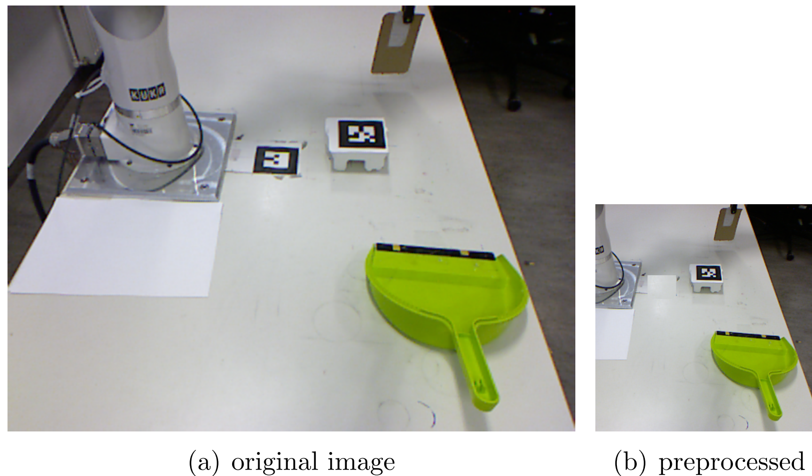


Figure 2.8: Highlighting object.

Data preprocessing

The dataset is split into training set and validation set in the same way to previous work with CONVRNN. The original image has the resolution of 480×640 (height$\times$width). The right 480×480 part of images is cropped, because this can keep the object height-width-ratio within image, if the image is resized as next step. Then cropped images are resized to 200×200 pixels for computational efficiency. The base marker is covered as well, which is also performed in previous research work. An example of the original image and the preprocessed image can be seen in Fig. 2.9.



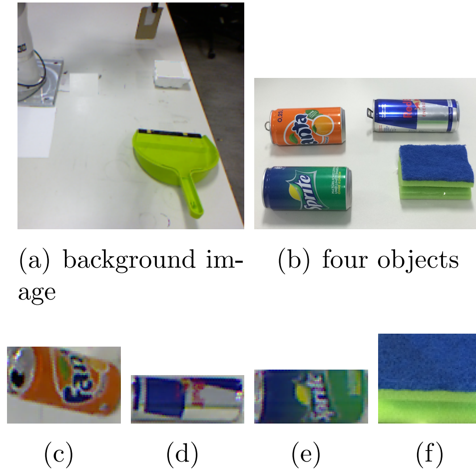
(a) original image (b) preprocessed

Figure 2.9: Image preprocessing for CNN

Data augmentation

Doing the same task, which TP-DMP can already achieve do not highlight the benefits of this approach. Therefore, the CNN is generalized for multiple real objects instead of using a special marker object, as the existing model can only be applied with a marker glued on target object. And doing a lot of demonstrations with real objects by a human is a boring and inefficient process, the collected dataset is augmented by removing the marker of the target object from the images and inserting one of the small new image of real objects into the images. The markers in the images within training set and validation set are removed and replaced with a special

patch, which has the similar color as desk in the background has. These marker-less images are used as background images for data augmentation. An example of background images is shown in Fig. 2.10(a). A small random patch of the background image will be replaced with one of the images of real objects, which are shown in Fig. 2.10(c), 2.10(d), 2.10(e) and 2.10(f). The full view of the four objects is shown in Fig. 2.10(b).



(a) background image (b) four objects

(c) (d) (e) (f)

Figure 2.10: Data augmentation

Network architecture

The architecture of CNN is illustrated in Fig. 2.11. It consists of nine layers, namely the input layer of image, three convolutional layers, a reduced layer of feature maps, input of fully connected layers, two fully connected layers and a linear output layer. The inputs to the neural network are a RGB image and clock signal. The outputs to be predicted are the nonlinear forcing terms of DMP model [PL17]. Since in order to preserve the good properties of the DMP model, forcing terms are preferred to be predicted instead of directly predicting the motor torque. The input image has a dimension of 120000 ($200 \times 200 \times 3$) and the clock signal is a single scalar. Direct concatenation of image and clock signal as input to a neural network cause the input features hard dominated by image. Hence, the clock signal is attached into neural network as an input neuron of the dense layer, as Levine et al. [LFDA16] did this similar on the robot parameters in their approach.
In order to avoid vanishing gradient problem [Hoc91] and make convolutional layer learning correct features fast, the convolutional layers are pretrained with the object position in image coordinate system as output and image as input. This object position differs to the predicted object position in CONVRNN. The used object position as output for pretraining in CONVRNN is collected object position, which is

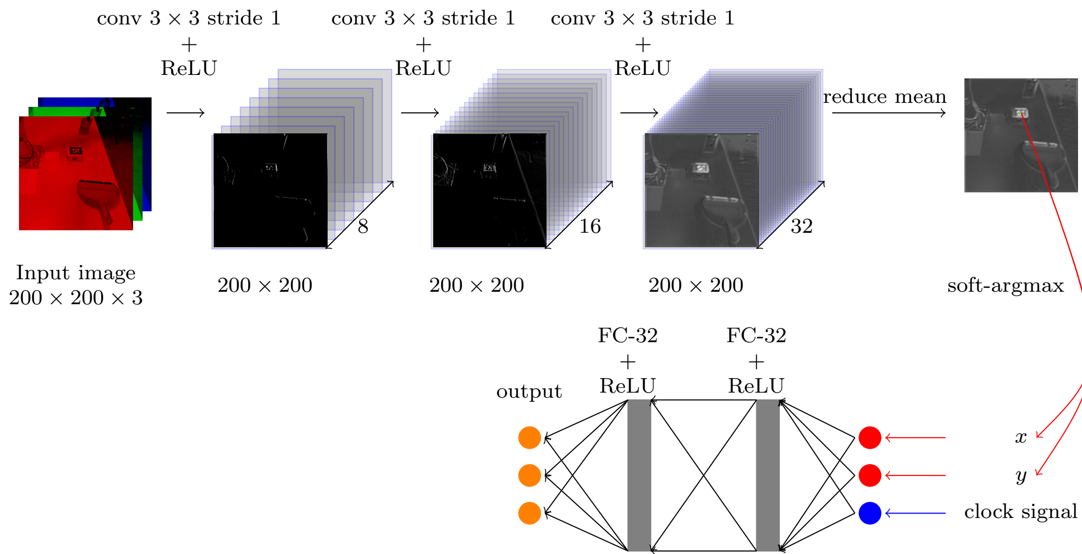


Figure 2.11: Architecture of CNN: The input is a RGB image, followed by three convolutional layers, i.e., conv1: kernel size: 3×3, stride 1, ReLU activation, depth 8, conv2: kernel size: 3×3, stride 1, ReLU activation, depth 16, conv3: kernel size: 3×3, stride 1, ReLU activation, depth 32. All convolution operations are performed with SAME Padding. The outputs of last convolutional layer are reduced using mean value over its last dimension (the channel dimension). A soft-argmax operation is performed after reduce_mean operator and results a 2D position (x,y) in image coordinate system. The position is concatenated with clock signal and go through the fully connected layers. Both fully connected layers have 32 neurons and ReLU as nonlinear activation function. The last layer is a linear layer, which outputs the predicted forcing terms. Hence, the CNN contains a total of nine layers including input layer and output layer.

defined in real world coordinate system. Both input (image) and output (position) are rescaled to zero-centered at range $[-1, 1)$. For the single object with special marker, the output position of the marker is calculated by a simple marker detection algorithm. The data for multiple real objects are augmented by replacing the patch from background image with new objects. For ease of processing, data are augmented during training phase instead of doing it off-line. One of the four real object images is randomly selected and inserted into a randomly selected background image from the training set. The validation set is augmented in the same way as well for validating model.

Recently, a lot of researchers tend to choose smaller kernel size e.g. VGGnet [SZ14] and Resnet [HZRS16] rather than big kernel size e.g. used in Alexnet [KSH12]. As depicted in Fig. 2.11, all convolutional kernels have a small size of 3×3 and its stride is 1. The stride is set to be 1 and SAME padding is used for keeping the size of feature maps as same as the original image. The reason for this is that the feature maps

of the last convolutional layer are not concatenated and feed into fully connected layer as usual, e.g. Alexnet [KSH12]. Instead, a variant of soft attention mechanism called soft-argmax is applied here, which is introduced in [LFDA16]. The soft-argmax 'softpicks' out the position or indices of the maximum value within a matrix (see Sec. 2.2.3). It transforms a feature map into a probability matrix, then the probability matrix is element wise multiplied with position matrices. The position matrices for x and y directions are defined by (2.10) and (2.11) and its increasing direction of the values within the matrix differs to (2.7) and (2.8). The increasing direction depends on the order of predicted positions, i.e., either (horizontal, vertical) or (vertical, horizontal).

$$M_x = (\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 199 & 199 & \cdots & 199 \end{bmatrix} - 100.0)/100.0 \tag{2.10}$$

$$M_y = (\begin{bmatrix} 0 & 1 & \cdots & 199 \\ 0 & 1 & \cdots & 199 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \cdots & 199 \end{bmatrix} - 100.0)/100.0 \tag{2.11}$$

Hence, the output positions have higher precision if the matrix or feature map is bigger. In this work, the output positions are calculated by (2.12) and (2.13), which are the task parameter and defined in image coordinate system.

$$x = \sum_{k,l} (softmax(F) \circ M_x)_{k,l} \tag{2.12}$$

$$y = \sum_{k,l} (softmax(F) \circ M_y)_{k,l} \tag{2.13}$$

Where $\circ$ denotes element wise product or Hadamard product, $k, l = [0...199]$ and F is a single feature map, which has to be flattened before $softmax$, the result of $softmax$ has to be reshaped back to 200×200.

Unlike [LFDA16], an additional reduce_mean operator is added before soft-argmax is performed. The soft-argmax operator is executed on the single reduced feature map, so that it outputs only one position pair (x, y), which can be concatenated with clock signal for further training. The reduce_mean operator is defined by (2.14).

$$F(i, j) = (\sum_n F_n(i, j))/n \tag{2.14}$$

Where F denotes the reduced feature map, F_n is a individual feature map in the last convolutional layer, $n = [1...32]$, and $i, j = [0...199]$ are the indices of pixels.

The fully connected layers are pretrained as well for fast convergence of subsequent end-to-end training. The fully connected layers are pretrained with the true positions of the marker in images and clock signal as inputs. The outputs of fully connected layers are the final outputs. All activation functions are Rectified Linear Unit (ReLU) in the neural network. Once both convolutional layers and fully connected layers are pretrained, the pretrained weights are loaded into CNN for doing end-to-end training. The weights of convolutional layers are separately pretrained for single object with special marker and multiple real objects, but the pretrained weights of fully connected layers are pretrained only once and used in both marker object and real objects for end-to-end training CNN.

Training Details

As Figure 2.11 shows, the CNN has nine layers totally. It is not extremely deep but also not that shallow. Pretraining provides a way to train deep neural network. The whole training phase is divided in five stages for two different models. One is used for sweeping marker object and the other one is able to handling multiple real objects. Some stages can also be finished before other stages are completed and are not restrictedly sequential. The five stages are described in following paragraphs. We trained our CNN by using TensorFlow v1.0 on GPU GTX1060 6G and it takes about 4-6 hours to train one model. The forward pass of the entire CNN takes about 20ms with python2.7 on GPU GTX1060.

Pretraining fully connected layers: The fully connected layers (FC layers) have to be pretrained. The inputs to FC layers are task parameters and clock signal. The tasks parameters are defined for CNN as normalized object position (x, y) in image coordinate system. The objects positions in collected dataset are not the expected positions in image space. It denotes the marker positions in real world. Hence, a simple marker detection program is used to calculate the marker in image space. Since the original image has a resolution of 480×640 and the inputs image to CNN is 200×200, the calculated marker positions should be normalized to range $[-1, 1)$. The positions for both x (horizontal) and y (vertical) directions are normalized with (2.15). The -160.0 for y direction is the offset after cropping images in image preprocessing, which can be seen in Fig. (2.9). The normalized positions are concatenated with clock signal and passed to fully connected layers. The outputs of fully connected layers are the final forcing terms that needs to be predicted in the entire model.

$$
\begin{aligned}
x &= (x_{raw} - 240.0)/240 \\
y &= (y_{raw} - 160.0 - 240.0)/240
\end{aligned}
\tag{2.15}
$$

Where x_{raw} and y_{raw} denote the raw positions, which are located at range $[0, 479)$ and $[0, 639)$. x and y are the normalized positions in horizontal and vertical direc-

tions.

The used gradient descent optimizer is Adam Optimizer [KB14]. The first moment is set to be 0.9 and the second moment is set to be 0.999. The epsilon is set to be 1e-8 for numerical stability. The batch size is always defined as 100 is. The learning rate starts at 0.01, switches to 0.001 and ends at 0.0001 in every 20 epochs. If the validation error stops decreasing or even starts increasing, then the training process could be early stopped. The training set is random shuffled after every epoch. The loss function is defined as mean absolute error. All weights and biases are initialized with Xavier initializer [GB10]. The trained weights and biases are saved as a 'npz' file for later use. The 'npz' file is a type of binary file, which can be used in python for storing arrays.

Pretraining convolutional layers for marker object: Pretraining convolutional layers is kind of slower and harder than pretraining fully connected layers, because it has high dimensional input, which is a RGB image. The pixel intensity is normalized by using (2.16). The output of pretraining the convolutional layers is the normalized positions in image space, which are used as part of input for pretraining fully connected layers.

$$I_{ijc} = (x_{ijc} - 128.0)/128.0 \tag{2.16}$$

Where I denotes a digital image, c is the channel index and ij are the indices of horizontal and vertical direction.
The used batch size and optimizer are the same as in pretraining fully connected layers. The first and second convolutional layers are initialized with Xavier initializer and the last convolutional layer is initialized with constant 1.0. The Xavier initializer is also used to initialized the bias of the first convolutional layer. The remaining biases are initialized with constant 0.0. Mean Squared Error is the loss function and the training set is shuffled after every epoch as training fully connected layers. The learning rate starts at 0.001 and ends at 0.0001 decayed by half every 3 epochs. The training process takes long time. To check if training process goes in the right direction, feature maps can be observed by human for making decision whether the training process can be continued or start a new training with another hyperparameters. The training error is a indication as well. Once the convolutional layers are well trained, its weights and biases are saved into a 'npz' file.

End-to-End training CNN for marker object: The end-to-end training for CNN is trivial. It connects the convolutional layers and fully connected layers and loads the pretrained weights and biases from the previously saved 'npz' files. Then the model is trained end-to-end with a RGB image and clock signal as input and the forcing terms as output. The settings of hyperparameters is same as pretraining convolutional layers. The only difference is that in this case the weights and biases are loaded into neural network instead of random initialized.

Pretraining convolutional layers for multiple real objects: Since there is no dataset for real objects available, the dataset of marker object is augmented, which is described in Sec. 2.3.3. The dataset is augmented during training one by one. One marker-less image is and randomly selected from training set as background image and a real object is randomly selected from prepared four objects as well. A random patch of the background image is replaced with the real object image associated with center position of the patch. The center position of the patch is randomly generated in batch size of 100 and is bounded at range $[50, 150]$ for avoiding out of edge of background image while replacing. An example of augmented pretraining convolutional layers is shown in Fig. 2.12. The augmented image is passed into convolutional layers for pretraining. The other settings and preprocessing steps are the same to pretraining convolutional layers for marker object. The pretrained weights and biases are saved into a 'npz' file.

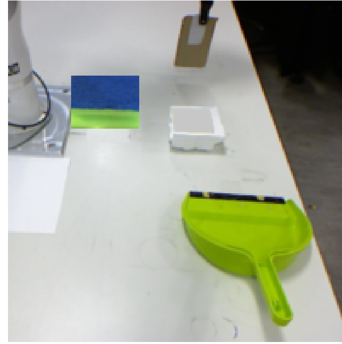


Figure 2.12: Augmented image for pretraining

End-to-End training CNN for multiple real objects: Unlike pretraining step, the forcing terms are the outputs of CNN. The available dependency to forcing terms is the image with marker object, which is collected by TP-DMP model [PL17]. Hence, a random selected patch from background image can not be used in this case. Instead, the extracted marker positions are used for replacing patch with real objects. This is approximately equal to that the markers in the collected dataset are replaced with new real objects. An example of augmented pretraining convolutional layers can be seen in Fig. 2.13. As it shows, the marker is replaced with the new object approximately. The background images inside of validation set are also randomly replaced with real object for validating model during training. In this training step, the same pretrained weights and biases of fully connected layers are loaded for initialization of fully connected layers of CNN. Also, the pretrained weights and biases of convolutional layers for real objects are loaded into CNN. The other settings are the same to end-to-end training CNN for marker object.

Overall, the training steps are summarized as follows,

Figure 2.13: Augmented image for end-to-end training

1. pretraining fully connected layers by using normalized true positions in image space and clock signal for predicting forcing terms
2. pretraining Convolutional layers for predicting object position in image space
3. load pretrained weights into whole architecture
4. doing End-to-End training

Optimization after training

Smoothing: In order to avoid oscillation of predicted positions (x, y) shown in Fig. 2.11, a low pass filter is used for smoothing the positions on the trained CNN. The filter is defined by (2.17). The smooth coefficient for this work is set to be 0.3.

$$(x, y) = \alpha(x, y)_{current} + (1 - \alpha)(x, y)_{prev} \tag{2.17}$$

Where $(x, y)_{current}$ is the current prediction of position, $(x, y)_{prev}$ is the previous filtered position and α denotes the smooth coefficient, which is at range [0, 1].

Filtering: During training phase, all mathematical operations must be differentiable if gradient descent based optimization algorithm is used. In order to make the CNN more robust after training, the soft-argmax operator is performed on a small window of the reduced feature map instead the whole reduced feature map (see the gray scale image of the 5th layer, top-right in Fig. 2.11), which is used in [FTD+16] in a similar situation. The center position of the small window is the position of strongest activation in the reduced feature map. This avoids the oscillation of predicted positions in image coordinate system while robot is moving or some other objects is appearing in the camera view. The predicted position could be 'pulled' away by some pixels without this filtering window. As long as the strongest activa-

tion is located at the target object, the CNN should be able predicting the position robustly.

Avoiding jumping: If the strongest activation in the reduced feature map is changed due to some other noises e.g. human's clothes during executing motion, it could completely destroy the model prediction. To avoid this negative effect, the predicted position is compared to the previous prediction. The absolute difference between them is bounded. If the current prediction is out the boundary, then it will be discarded and the previous prediction is used for predicting forcing terms. To ensure the first prediction of position is correct, a program for checking strongest activation in reduced feature map can be used before executing motion.

Hyperparameter

The used hyperparameters in this work are given in Tab. 2.3. The final model i.e. CNN contains **7315** trainable parameters. The hyperparameters can be different and are not defined restrictedly for reproducing the results.

Hyperparameter		
Pretraining of convolutional layers	**Pretraining of fully connected layer**	**End-to-End training**
batch size: 100	batch size: 100	batch size: 100
learning rate: 0.001 decayed by half every 3 epochs until 0.0001	learning rate: from 0.01 to 0.001 and then to 0.0001 about every 20 epochs	learning rate: 0.001 decayed by half every 3 epochs until 0.0001
numpy seed number: 1	no seed: easy to train	numpy seed number: 1
TensorFlow seed number: 1	no seed: easy to train	TensorFlow seed number: 1
Smooth coefficient: $\alpha = 0.3$		
Filtering window size: 41×41		
Allowed jumping pixels: $<= 5$		

Table 2.3: Hyperparameters for training model and optimization after training.

Chapter 3

Evaluation

3.1 Experimental Setup

Communication Interface: The well-trained model needs to be evaluated on real KUKA robot. Since the robot computer has a slow GPU, the deep learning model has to be executed on high performance PC. A Server-Client communication interface is implemented between neural network and robot PC by using ROS (Robot Operating system) library. An overview of the communication interface is illustrated in Fig. 3.1. The DMP model is running on robot pc and a client program is integrated into DMP model. The server PC builds the deep learning graph (model) and is listening for request. The client sends clock signal as request to deep neural network, then the server captures the current image from camera and passes the received clock signal and captured image to neural network for computing forcing terms. The predicted forcing terms are sent back to client side. Finally, the DMP uses the forcing terms and sends command to robot controller for executing motions.

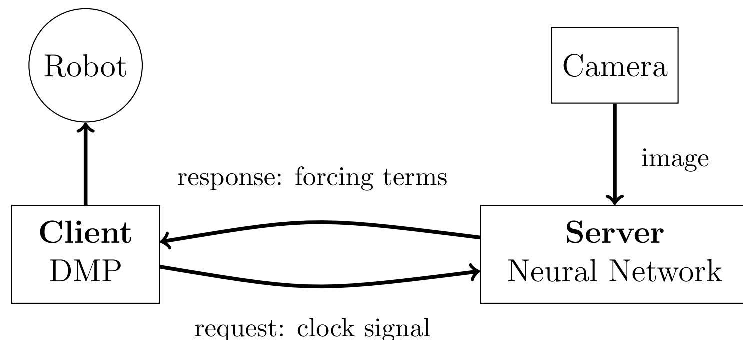


Figure 3.1: Communication interface: interaction between DMP model and deep neural network using Server-Client pattern on ROS (Robot Operating System).

Set up camera position: The camera position is fixed and has to be set at the same position as data collecting phase. Figure 3.2 shows where the camera should be located at. Once the green square is around the base marker, the camera is about to be set up. The camera position is very important before executing any motions. A wrong camera position directly affect the performance of the deep learning model, because the model has to predict object position.

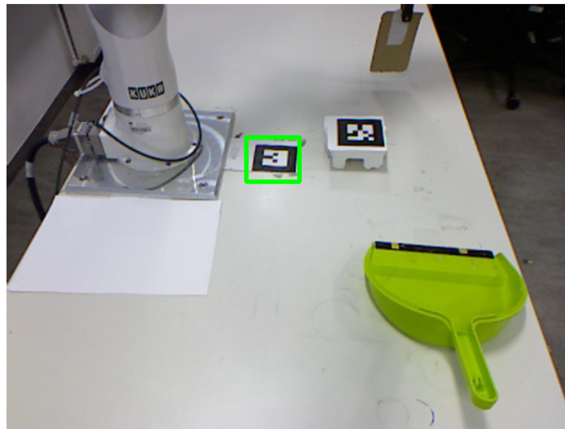


Figure 3.2: Check camera position: the green square is indication of camera pos.

3.2 Evaluation with CONVRNN

3.2.1 Evaluation on Dataset

Trash Position of Training Set

As CONVRNN are pretrained for predicting the trash position. The true position and predicted position in training set are shown in Fig. 3.3 and the motions are separated by small gap. The horizontal axis is the number of samples and the vertical axis indicates the trash position in real world coordinate system. The training set has 45 motions, as described in Sec. 2.1. The top diagram shows the position of horizontal direction (x) and the bottom diagram shows the vertical direction (y) (see Fig. 2.4). The Mean Absolute Error in both directions is given as follows:

x : **0.00395642772795**
y : **0.00490720878207**

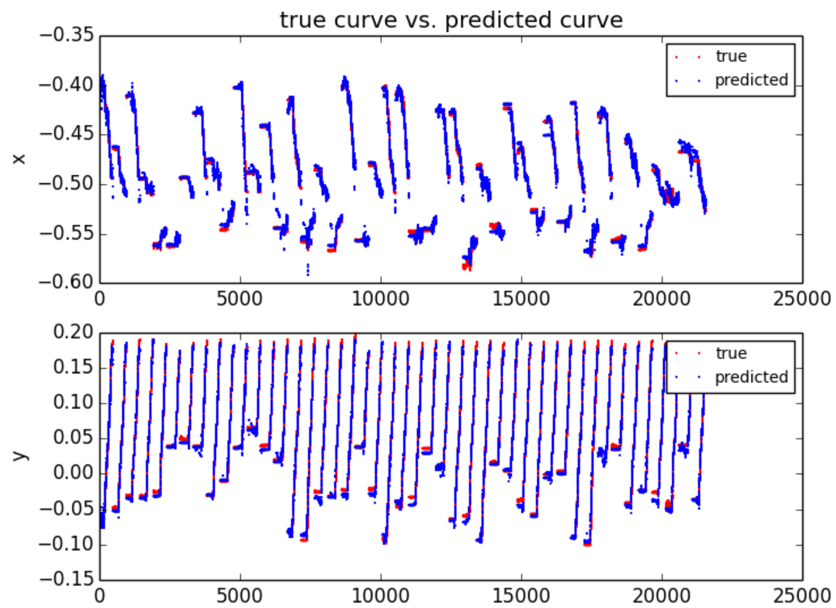


Figure 3.3: Curve fitting of marker position in training set

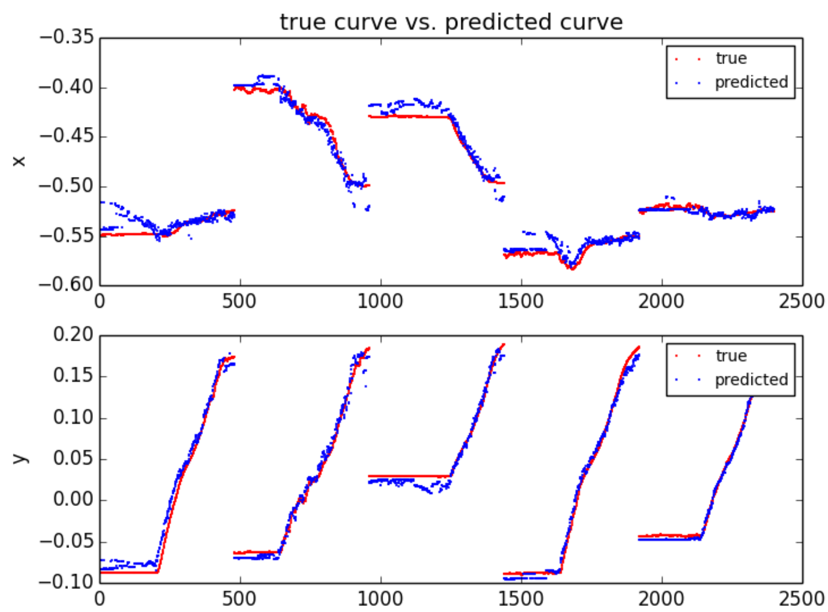


Figure 3.4: Curve fitting of marker position in validation set

Trash Position of Validation Set

Figure 3.4 shows the true trash position and predicted trash position in validation set. Since the trash is not a single point within images, the predicted distribution of trash position is noisy. Overall, the true distribution is well fitted by the predicted curves. The mean absolute differences in both directions are given as follows:

x : **0.00641358980841**
y : **0.0067520192803**

Feature Maps

As convolutional layers are used in CONVRNN, it is necessary to evaluate the feature maps (also called response map) after each convolutional layers. Figure 3.5 shows the feature maps of the first convolutional layer. Each map has the shape of 198×198 and 64 maps are shown as a merged 8×8 image. Figure 3.6 shows the feature maps of the second convolutional layer. Each map has the shape of 97×97 and 64 maps are shown as a merged 8×8 image. These can be seen in Fig. 2.6 as well. In the first two convolutional layers, the model can already recognize the trash marker. The goal position i.e. the dustpan is recognized as well, because it could be a relevant information for predicting the trash position. The feature maps of third convolutional layer are shown in Fig. 3.7. These feature maps have the resolution of 95×95 and 16 maps are merged into a single 4×4 image. The function soft-argmax picks out the strongest activation (largest pixel intensity) at each map. These feature points are marked with small red circle in Fig. 3.7. In the most feature maps, the strongest point is located at the trash. But three feature maps are different here, i.e., the 10th one, the 13th one and the last one (counted from left to right and top to bottom). The 10th neuron is "dead". This is the side effect of ReLU function, as described in Sec.2.2.2. It is hard to prevent this side effect, even the model is trained with a small learning rate or uses another variant version of ReLU e.g. Leaky ReLU. The 10th and 13th feature maps are stable, i.e., the strongest activation are almost located at the same point during whole motion. The red point in the last feature map oscillates a little bit around the end effector of robot. Not all the feature points have to be located at the trash marker, because these feature points are the inputs of RNN (see Fig. 2.6) and RNN treats them as a sequence. The memory cell inside the RNN can learn to store important information from its inputs. Figure 3.8 shows the first feature map of the last convolutional layer of a moving scenario. The model is able to tracking the trash in real time (marked red circle in Fig. 3.8). The final predicted distribution of trash position against the true distribution (see Fig. 3.3 and Fig. 3.4) indicate the successful prediction of trash position.

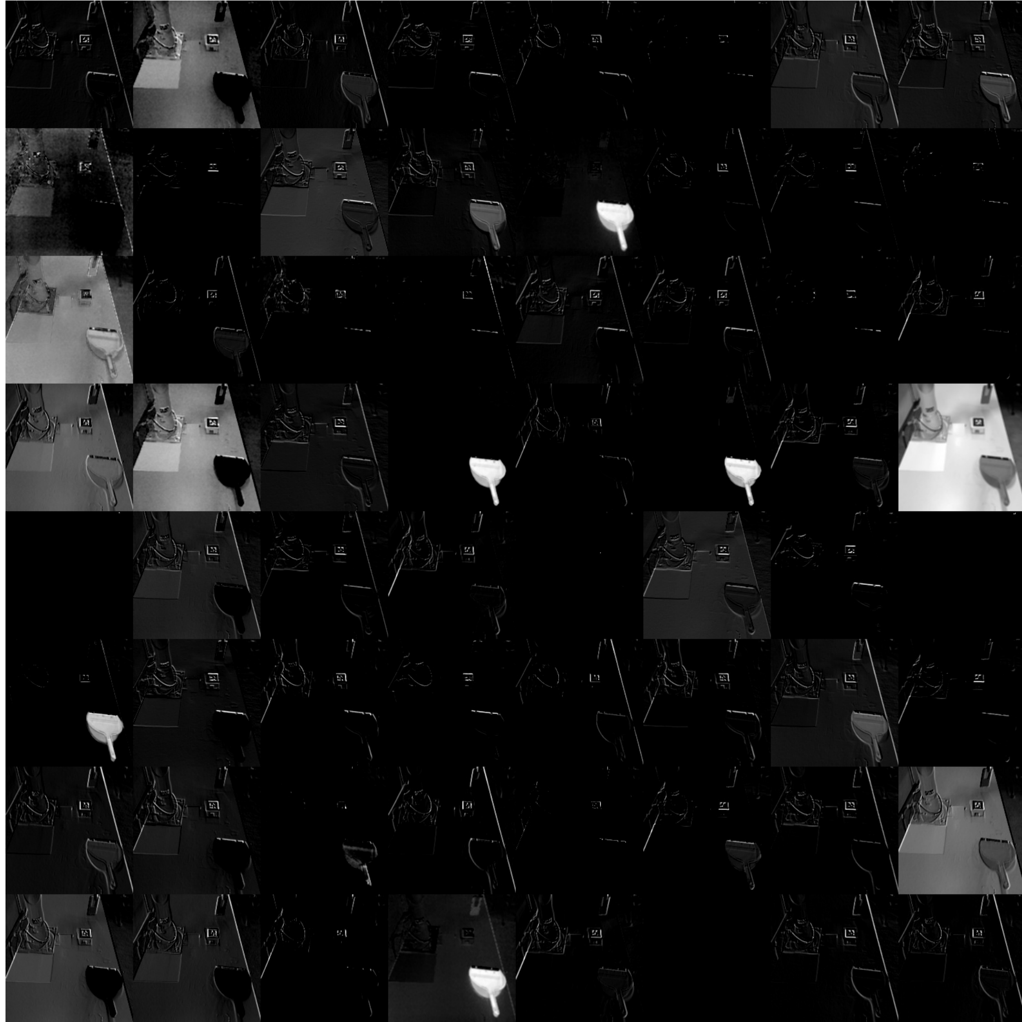


Figure 3.5: Feature maps after first convolutional layer

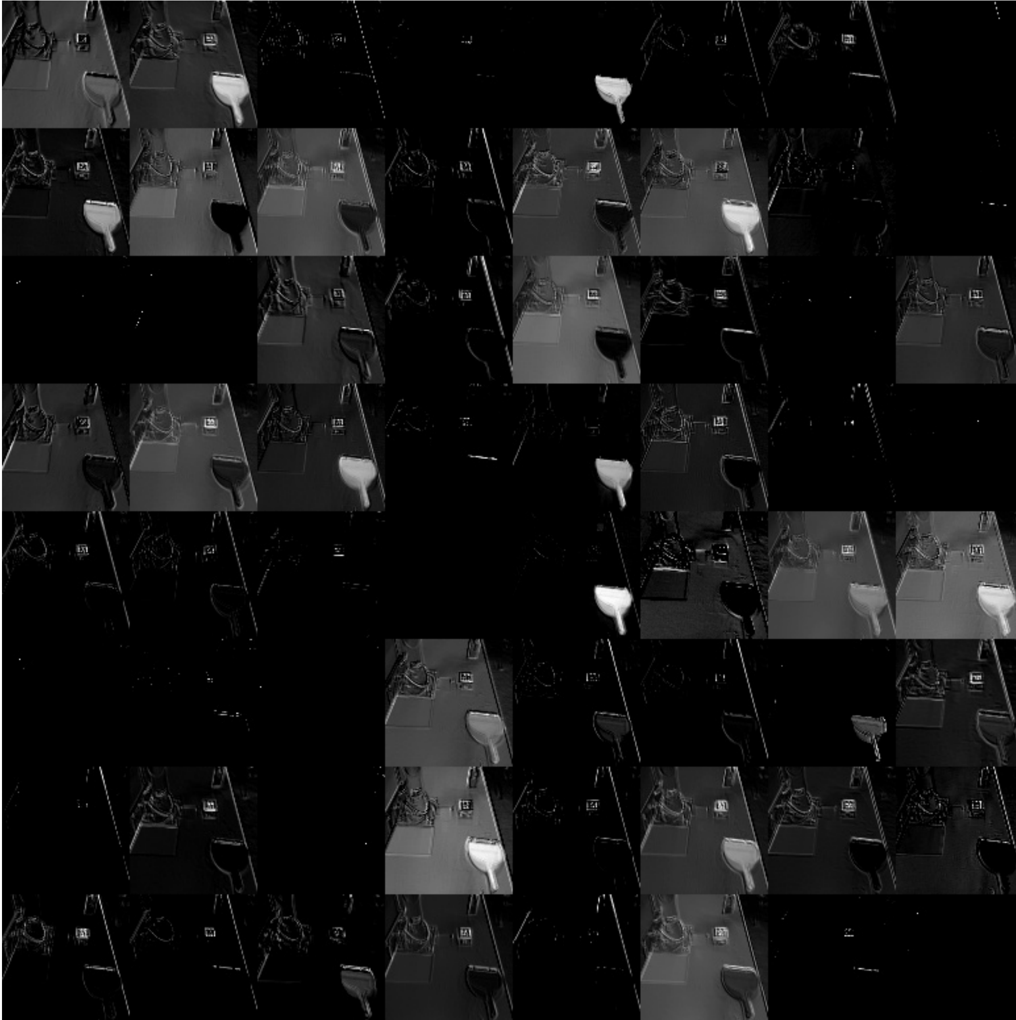


Figure 3.6: Feature maps after second convolutional layer

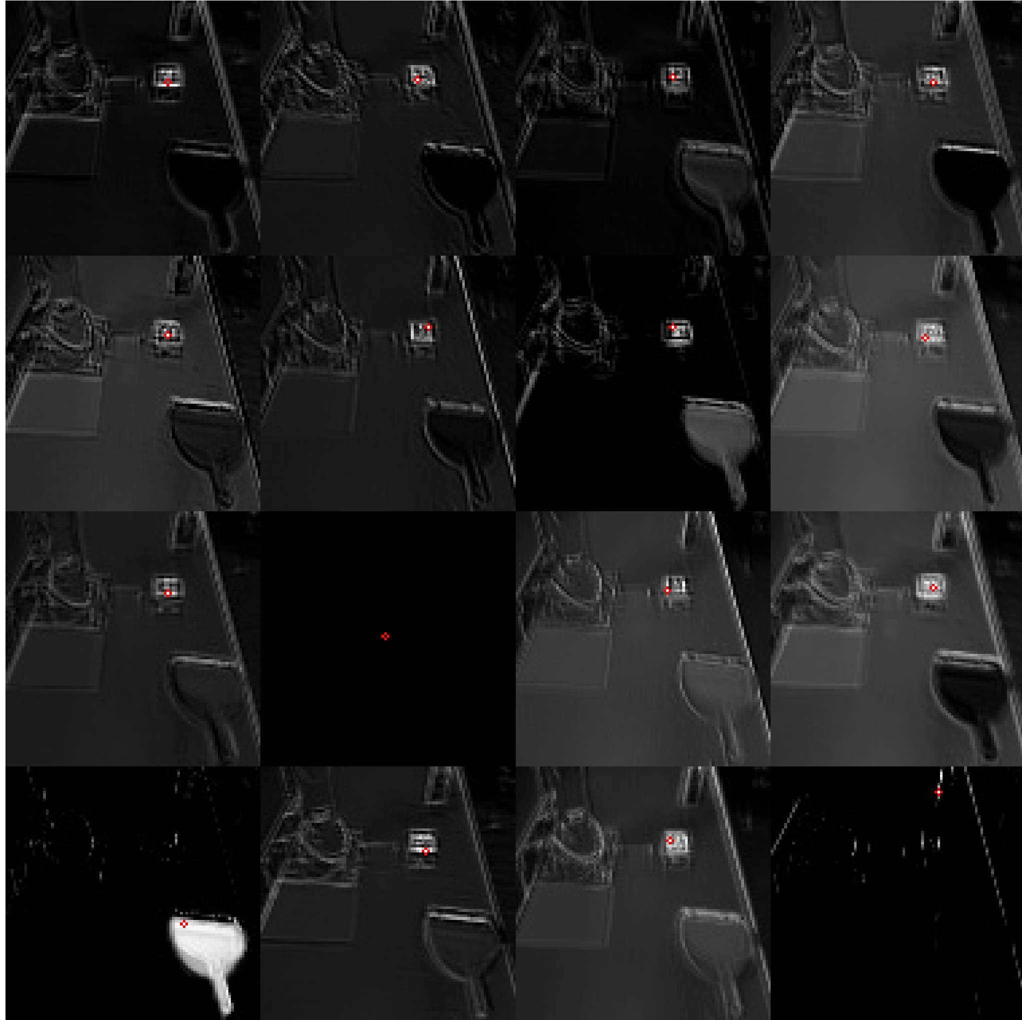
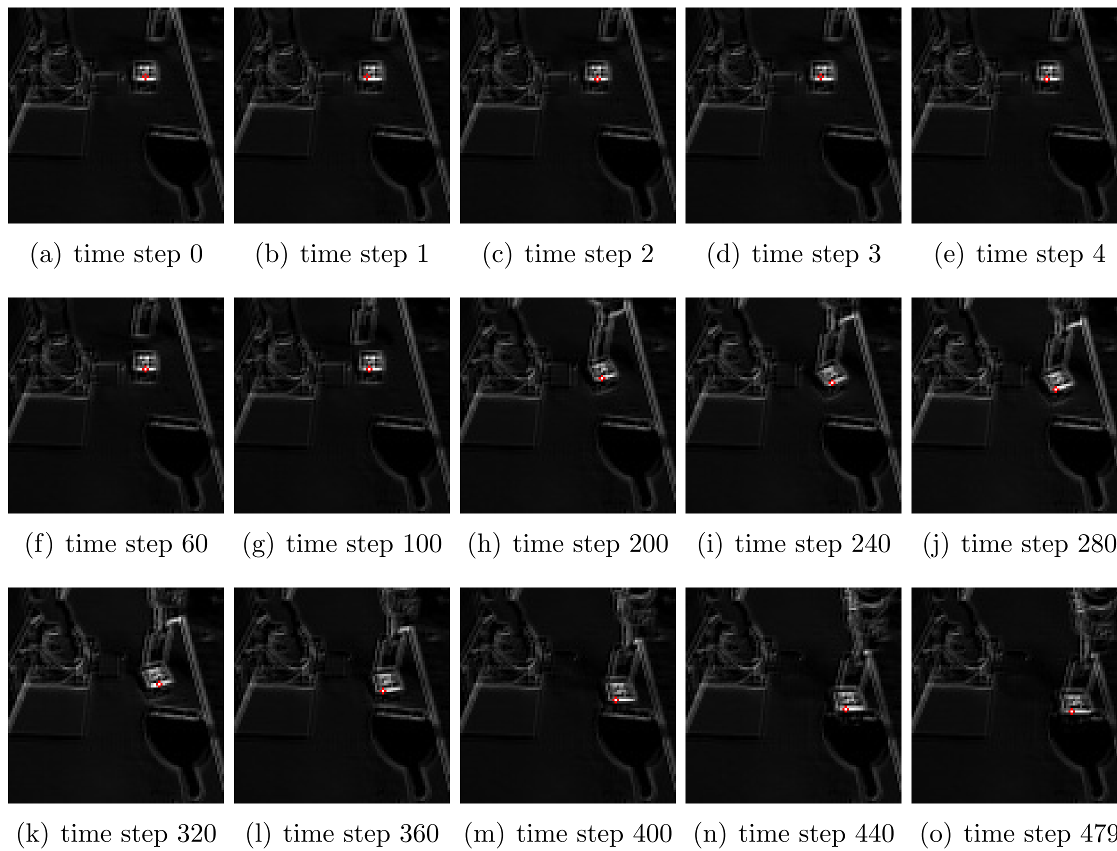


Figure 3.7: Feature maps after third convolutional layer



(a) time step 0 (b) time step 1 (c) time step 2 (d) time step 3 (e) time step 4

(f) time step 60 (g) time step 100 (h) time step 200 (i) time step 240 (j) time step 280

(k) time step 320 (l) time step 360 (m) time step 400 (n) time step 440 (o) time step 479

Figure 3.8: First feature map after third convolutional layer of a motion at different time step

Forcing Terms

The performance of trained model can be seen on the evaluation of forcing terms. The predicted forcing terms and its true values of training set and validation set are shown in Fig. 3.10 and Fig. 3.9. The predicted forcing terms are not filtered and oscillate a bit. The Mean Absolute Error is given as follows:

Training set: 0.0832534965982
Validation set: 0.126778447162

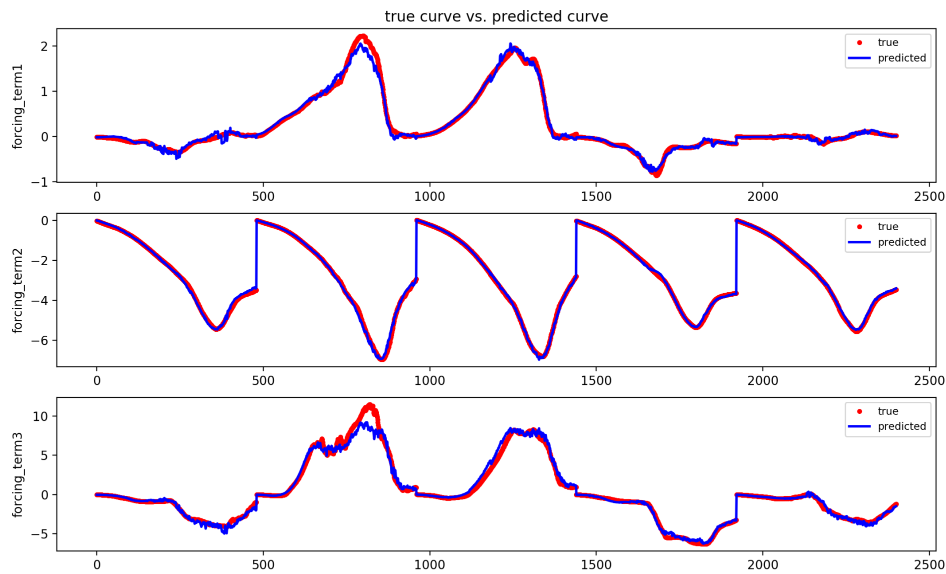


Figure 3.9: Curve fitting of forcing terms in validation set

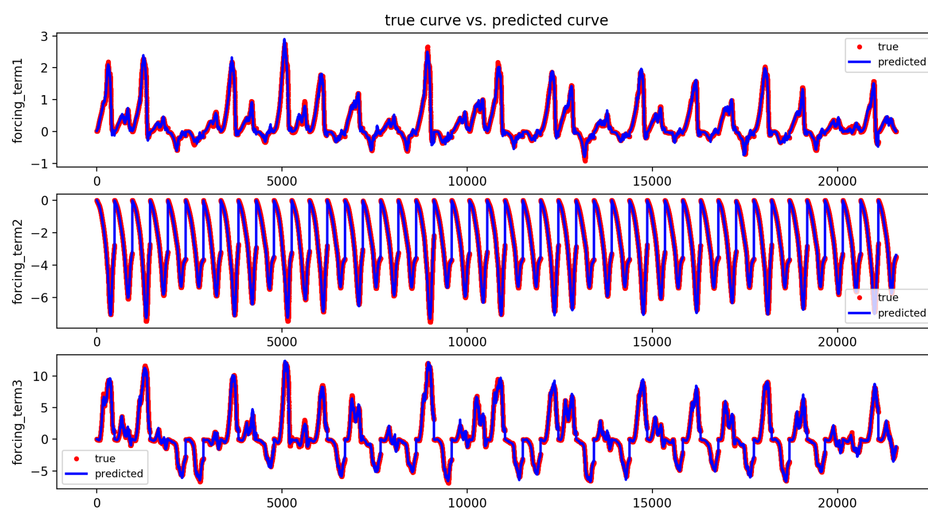


Figure 3.10: Curve fitting of forcing terms in training set

3.2.2 Evaluation on Robot

The model is evaluated with a lot of executed motions on real robot. Two example motions are shown in Fig. 3.11. The blue curves indicate the predicted forcing

terms from CONVRNN and the red curves are the predicted curves from GMM. The target marker object is first located at the left boundary of predefined region and the predicted forcing terms can be seen in the left plotting of Fig. 3.11. As the curve shows, GMM and CONVRNN generate similar motions, which means the CONVRNN has learned the behavior from GMM. Since the predicted forcing terms from CONVRNN is unfiltered, the curve seems not smooth. The right plotting shows the motion where the marker object starts at the right boundary. The curves have some spiking point (forcing term3 at about time step 850), but the model correct itself back to the right state and finishes the motion successfully. The recorded videos are stored in DVD.

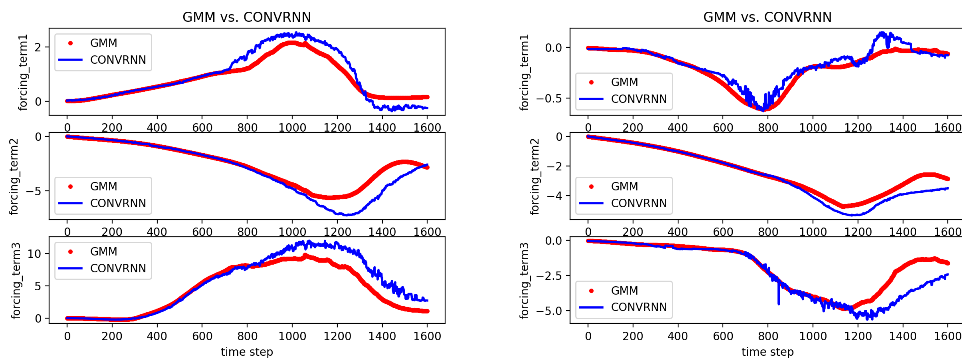


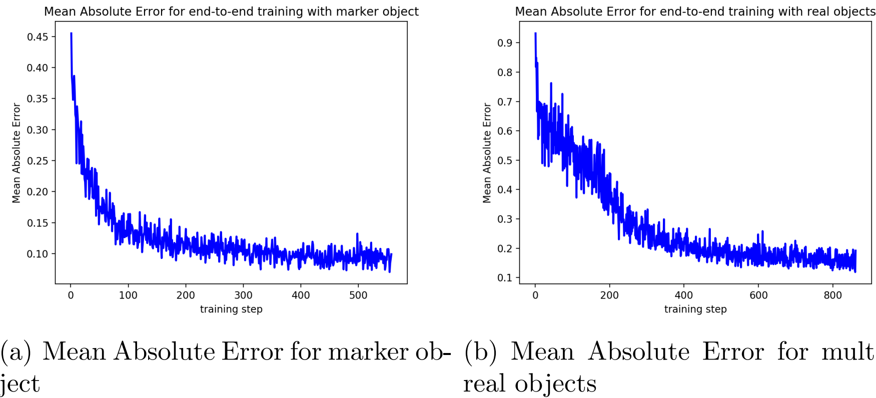
Figure 3.11: GMM vs. CONVRNN, left: marker at left boundary, right: marker at right boundary

3.3 Evaluation with CNN

3.3.1 Error of End-to-End Training

The loss function of end-to-end training for both marker object and multiple real objects is MSE. The difference between predicted value and true value in MSE equation is powered of 2. But MAE computes the direct mean value of the differences. So the MAE is better to be used for visualization of the error.

Figure 3.12 illustrates MAE during the end-to-end training for marker object (left plotting) and real objects(right plotting). The error for marker object converges to a lower value than for real objects, i.e., the model for multiple real objects is harder to learn than for a single marker object. It can also be seen that the error of end-to-end training still goes down after pretraining convolutional layers and fully connected layers. This is to say, doing end-to-end training as last training step improves the pretrained model significantly.



(a) Mean Absolute Error for marker object

(b) Mean Absolute Error for multiple real objects

Figure 3.12: End-to-End training loss, left: marker object, right: real objects

3.3.2 Evaluation on Dataset

This section describes and shows the evaluation results on collected dataset and preprocessed dataset. The part of training error is illustrated as well.

Object Position in Image Space

Since the predicted positions are some values in image coordinate system, it is meaningful to evaluate the errors and accuracies of positions in training set and validation set. The predicted position corresponds the (x, y), which is shown in Fig. 2.11. The unit of error is pixel and the accuracy is stated in percentage. Hence, the task parameters (x, y) are de-normalized from range $[-1, 1)$ to $[0, 200)$ for adjusting the unit of error in pixel. Both horizontal and vertical directions of position are evaluated separately. The error is computed by averaging the differences between predicted positions and true positions either over whole training set or validation set. The resolution of the reduced feature map is 200×200, so 200 pixels is used as reference value. The equation of the accuracy of predicted position is defined by (3.1).

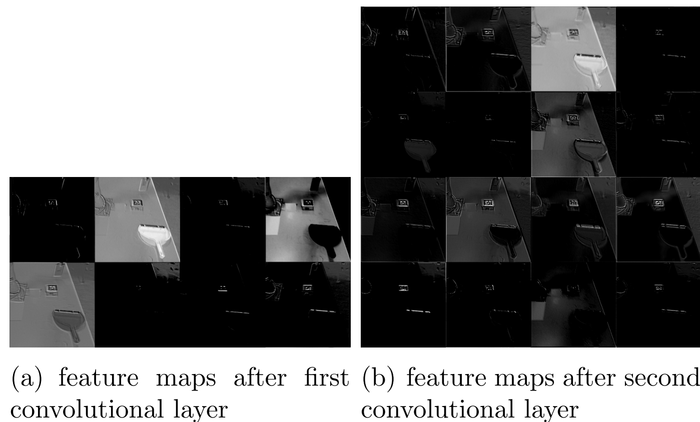
$$Accuracy = (1 - \frac{error}{200pixels}) \times 100 \tag{3.1}$$

Position		**Marker**		**Real Objects**	
		horizontal	vertical	horizontal	vertical
Training set	Error in pixel	1.36	4.53	2.28	12.12
	Accuracy in %	99.32	97.74	98.86	93.94
Validation set	Error in pixel	1.42	4.27	2.27	11.96
	Accuracy in %	99.29	97.87	98.86	94.02

Table 3.1: Errors and accuracies of predicted positions in image coordinate system

The results after pretraining convolutional layers are given in Tab. 3.1. Horizontal and vertical correspond to de-normalized x and y in Fig. 2.11. It can be seen, the accuracy for a single marker object has is higher than for real objects and the vertical direction is harder to learn than horizontal direction.

Feature Maps

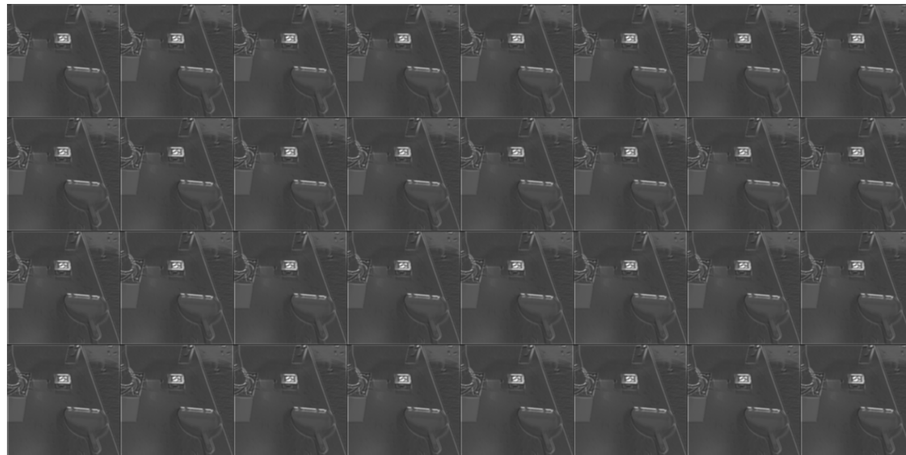


(a) feature maps after first convolutional layer (b) feature maps after second convolutional layer

Figure 3.13: Feature maps

As convolutional layers are used in CNN, it is necessary to evaluate the feature maps (also called response map) after each convolutional layers. An example of the activated feature maps after ReLU operation for three convolutional layers and the reduced feature map after convolutional layers are shown in Fig. 3.13 and 3.14. Since the size of feature maps is not decreased at every layer, all the feature maps are human readable. The model starts learning some overall features in the first convolutional layer and the target object already get strong activated in the most feature maps in the second layer, which can be seen in Fig. 3.13(a) and Fig. 3.13(b). Figure 3.14(a) shows the feature maps after the third convolutional layer. The target object (marker) is strong activated and all feature maps look similar. There are only minor difference between these feature maps. Hence, it is also possible to directly pass one of these feature maps into soft-argmax operation (see. Fig. 2.11) for computing expected position. In order to make the final reduced feature map more robust, the mean value of these feature maps is used, which can be seen in Fig. 3.14(b). Another alternative is passing 32 extracted positions from the 32 feature maps to the soft-argmax, but the position of strongest activated pixel with respect to the 32 feature map varies 1-2 pixel to each other. It doesn't bring much more useful features compared to use only one feature map as experimented and makes the dense layer converging slowly. Each layer is a representation of input in neural network, but the top layers provide more informative features than bottom layers. So the extracted features of first convolutional layer and second convolutional

layers are not considered to be 'soft-argmaxed' and concatenated with clock signal for further end-to-end training.



(a) feature maps after last convolutional layer

(b) reduced feature map

Figure 3.14: Feature maps

Forcing Terms

The final output of CNN is the forcing terms. The predicted forcing terms are first evaluated on collected validation set and training set. The results are shown in Fig. 3.15 and Fig. 3.16. The validation set consists of 5 motions, which can be known by looking at the broken curve of 'forcing term2' in the middle plotting. The training set has 45 motions. The red curve indicates the forcing terms in collected dataset and the blue curve is predicted by CNN. The horizontal axis is just a sample counter, which means e.g. the validation set has about 2400 observations totally and each motion contains about 480 samples. As the figure shows, the curves of predicted forcing terms fit the true curves well in both validation set and training set.
The corresponding quantitative results of Fig. 3.15 and Fig. 3.16 are given in Tab. 3.2. The errors are calculated by averaging the absolute difference of each observation for

three forcing terms respectively. The approximation of accuracies is defined by (3.2).

$$Accuracy = (1 - \frac{error}{Denominator}) \times 100 \tag{3.2}$$

Where the denominators is computed by averaging the absolute value of each forcing term in training set respectively.

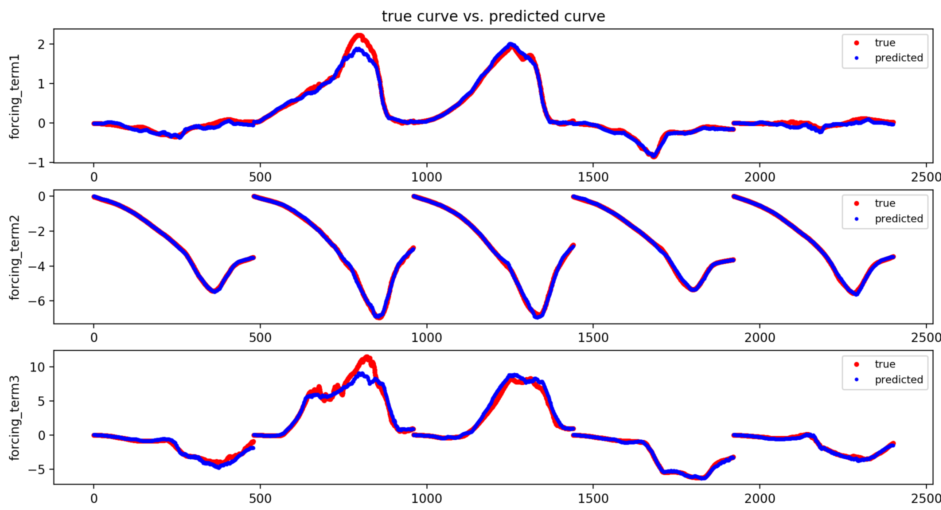


Figure 3.15: Curve fitting of forcing terms in validation set

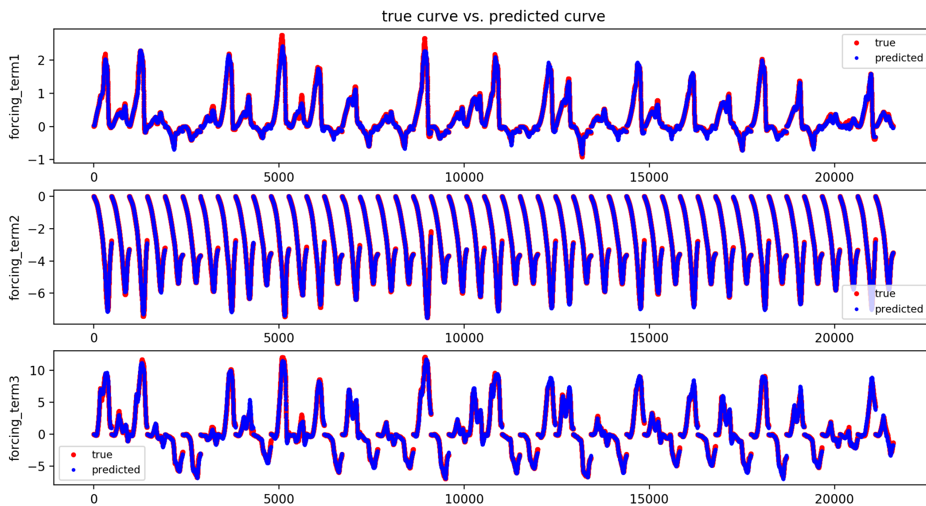


Figure 3.16: Curve fitting of forcing terms in training set

Forcing terms		Marker			Real Objects		
		1	2	3	1	2	3
Training set	Error	0.0375	0.0429	0.1988	0.0793	0.0917	0.5115
	Accuracy in %	90.21	98.47	91.73	79.29	96.72	78.73
Validation set	Error	0.0474	0.0384	0.2791	0.0883	0.0933	0.5851
	Accuracy in %	87.62	98.63	88.40	76.94	96.67	75.67
Denominator		0.3829	2.7950	2.4051	0.3829	2.7950	2.4051

Table 3.2: Errors and accuracies of predicted forcing terms

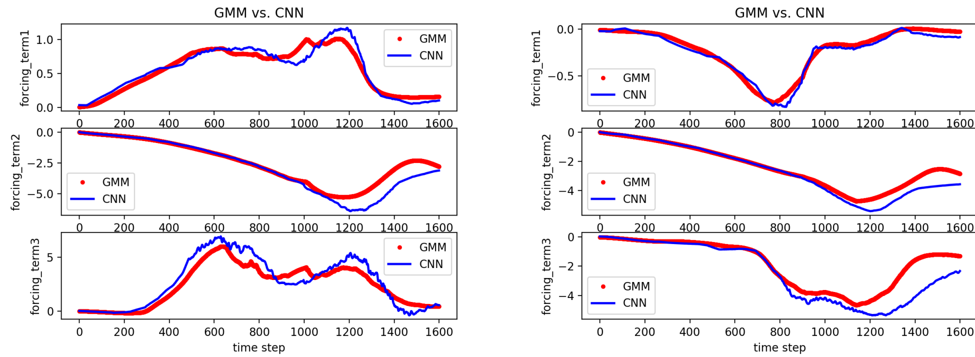


Figure 3.17: GMM vs. CNN, left: marker at left boundary, right: marker at right boundary

3.3.3 Evaluation on Robot

In order to check the performance of trained CNN on real robot, the model is evaluated in different scenarios.

Firstly, it is necessary to make sure that our CNN has learned the correct behavior from collected data set such that it can reproduce the results of TP-DMP in [PL17]. Both GMM model and CNN model are used for executing motions on Alvar marker object. The marker object is first located at the left boundary of predefined region and then at the second running it is moved to the right boundary. The predicted forcing terms are shown in Fig. 3.17. The blue curves indicate the predicted forcing terms of CNN and the red curves are from GMM model. The prediction of CNN is not filtered with (2.17) in this experiments, because it is good to see how robust the CNN can perform. As the figure shows, the predicted curves oscillate a little bit but at the right range, i.e. the predicted task parameters (x, y) are located at the marker object in reduced feature map but not that robust (in fact, the human eyes can not recognize, whether the robot end effector is oscillating or not). An example of the executed motions is shown in Fig. 3.18. Images of different time steps and its corresponding reduced feature maps are picked out and shown in Fig. 3.18.

Secondly, the CNN is evaluated on real objects. Figure 3.19 shows the executed

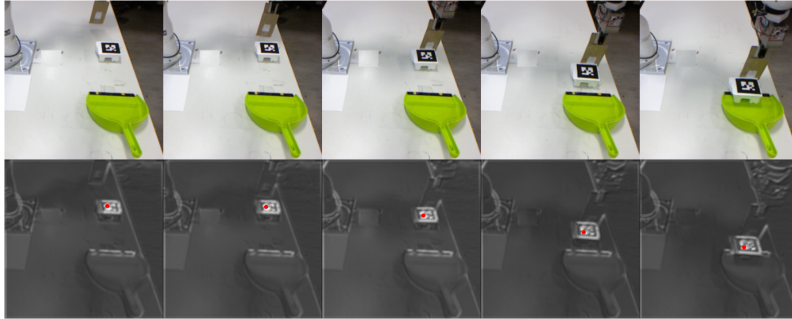


Figure 3.18: Executing motion with marker object: The red circle denotes the expected position of marker object in image coordinate system

motion with one of the trained real object, in this case a wash sponge. The red rectangle at the top left image in Fig. 3.19 depicts the approximated predefined region of target object in collected data set. As the images show a successful trajectory, our CNN has good performance on real object even at out of predefined range as well.

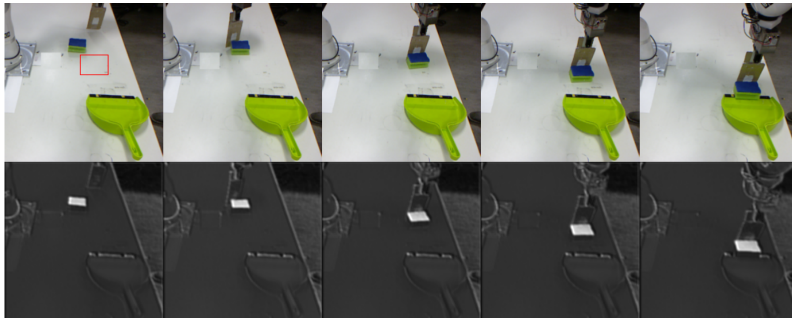
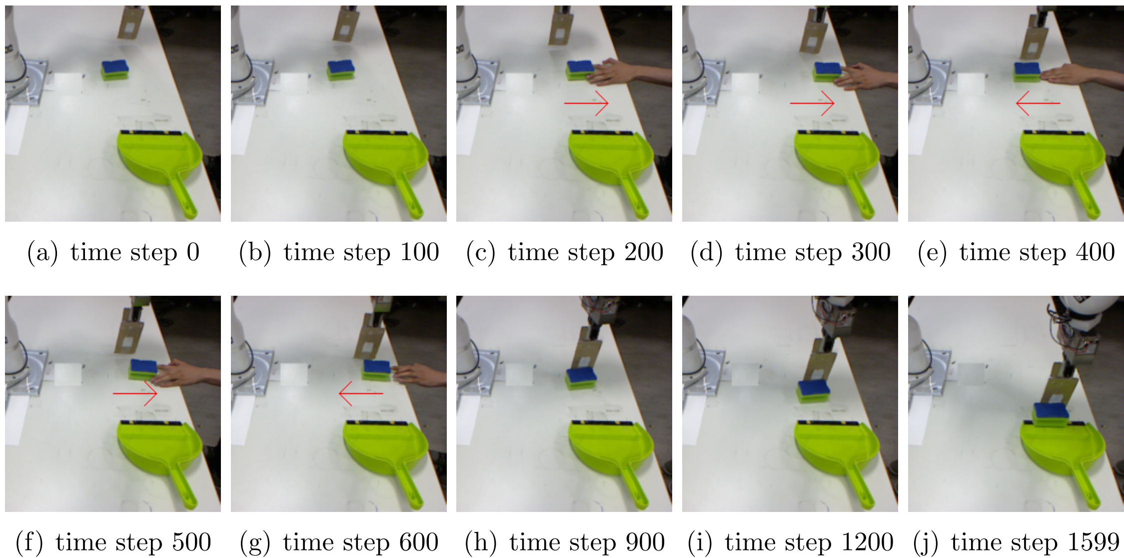


Figure 3.19: Executing motion with real object and the start position of object is out of range of collected data set. The red rectangle indicates the approximate range of start positions of target object in collected data set.

As next step, some disturbance with human hands is added into camera view and the position of wash sponge is changed in real time. The result is shown in Fig. 3.20. The wash sponge is pull and pushed by human hand after starting execution. The directions of pulling and pushing are depicted with red arrows in Fig. 3.20. The end effector of robot can follow the changing direction of the wash sponge and push it into dustpan successfully. The predicted corresponding forcing terms to Fig. 3.20 is illustrated in Fig. 3.21. The forcing terms are displayed at vertical axis and the horizontal axis shows the time steps during executing motion. At about time step 200, the first forcing term and the third forcing term begin to change its trend that is caused by changing the sponge position, which also can be seen in Fig. 3.20. At this time, the low pass filter defined in (2.17) is added into CNN compared to previous evaluation on marker object (see Fig. 3.17) before passing the predicted position into

fully connected layers. The curve of predicted forcing terms becomes stable again after about time step 650, because the position of the object is not pushed or pulled anymore by human. After the experiment, the curve of predicted forcing terms are still smooth and has no spiking points (no jump points), even human moves the wash sponge in real time.



(a) time step 0 (b) time step 100 (c) time step 200 (d) time step 300 (e) time step 400

(f) time step 500 (g) time step 600 (h) time step 900 (i) time step 1200 (j) time step 1599

Figure 3.20: Moving object with disturbance at different time steps. The red arrows indicate the pull or push direction during executing motion.

More evaluations are available in attached DVD.

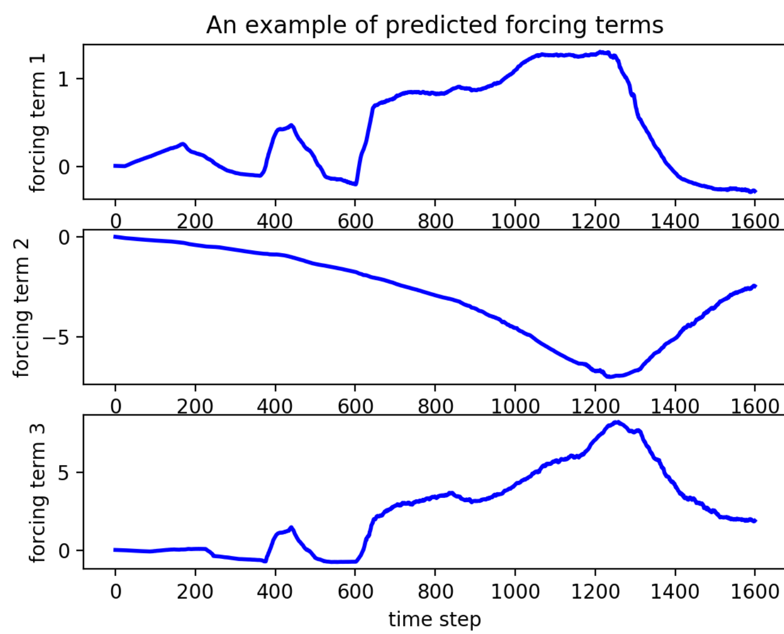
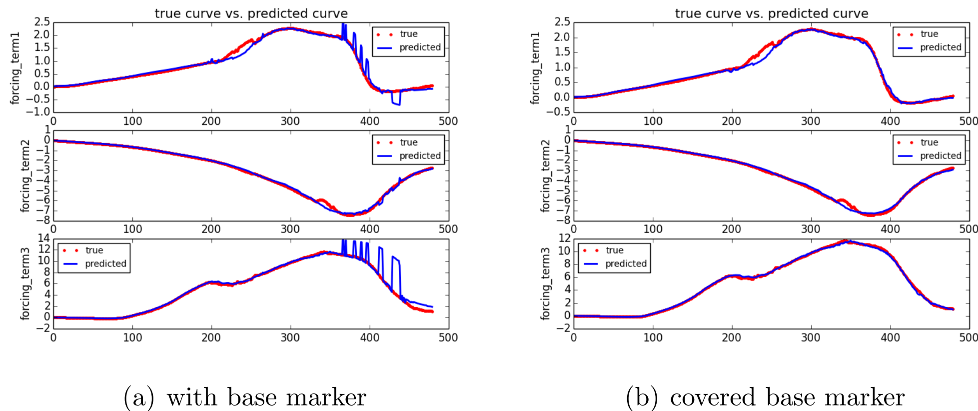


Figure 3.21: Forcing terms of moving object with disturbance

Chapter 4

Discussion

Covering base marker: The base marker is used in [PL17] for localization of target object during collecting data. It was glued on the desk and can be removed when CNN is applied for permanent use on the robot. In the image preprocessing, the base marker is removed from images because of high similarity to target marker. Figure 4.1 shows the impact of base marker of an example motion from collected dataset. The predicted forcing terms can jump at some time step, which can be seen in Fig. 4.1(a). But the prediction in Fig. 4.1(b) has no jump compared to Fig. 4.1(a), if the base marker is covered. This is to say, the predicted position jumps from target marker object to base marker when the base marker is not covered. The base marker doesn't affect the training error significantly, if the base marker is covered during training, because the base marker leads to wrong prediction only at few time steps, as it can be seen in Fig. 4.1(a). But it is not safe for evaluation on real robot. Hence, it is necessary to cover the base marker.



(a) with base marker

(b) covered base marker

Figure 4.1: Impact of base marker: comparison predicted motion with base marker and without base marker

Clock signal vs. clock signal + robot configurations: During collecting data, a lot of additional parameters e.g. joint angles are also collected. After research work, the only used parameter is clock signal. This is same as it used in [PL17] and the dataset is collected by using [PL17] as well. At research phase, several models are trained with joint angles or end effector positions. These models work well on training set and validation set but not on real robot. It seems like that the models are overfitted. As the trained models are tested by varying the input features e.g. images, clock signal, joint angle. The result is that these models are hard dominated by joint angles. The models can still predict the correct forcing terms approximately even with wrong images as input. This is to say, the prediction depends more on joint angles while robot is moving, which cause that robot can not find the object position in image. But the object position is a important dependency of forcing terms, because the collected forcing terms are calculated with clock signal and task parameters [PL17]. Hence, besides images only clock signal is used as input to CNN.

CNN vs. CONVRNN: The CONVRNN is also running on real robot successfully but only for the marker object. The initial researched CONVRNN model is discarded when the main approach CNN is successfully developed. Firstly, the CNN has only 7315 trainable parameters, which is much less than CONVRNN that has 117579. The program speed is improved with CNN. Secondly, the feature maps after last convolutional layer in CONVRNN are different to each other and also have a 'dead' feature map. In contrast to CONVRNN, the CNN has only one reduced feature map at the end. The predicted positions can be easier controlled with a single feature map by adding some conditions, e.g., bounding the predicted position in reduced feature map. Hence, The second developed approach is much better than CONVRNN.

Reduce feature map vs. 32 feature maps: Levine et al. [LFDA16] have shown that the model performs better by using extracted feature points than directly using predicted position. Since in this work, a different pretraining of convolutional layers method is used. The feature maps in the last convolutional layers look all similar to each other. A model with extracted 32 positions is also trained during research. It worked on real robot, but has less accuracy. Also, the 32 predicted positions are harder controlled, e.g. smoothing positions compared to a single position for robot safety. The soft-argmax costs more binary operations on 32 feature maps than one reduced map, which slows the forward pass of CNN. Hence, the reduced map is passed into soft-argmax operation instead 32 feature maps.

Number of objects: Two different models are developed. One is for single marker object. The other one is for multiple human selected real objects. As Tab. 3.2 shows, the accuracy of a single object is much higher than multiple object, even though the marker object has similar color to the environment color. The convolutional kernel

in a Convolutional Neural Network is shared over each feature map. So the same intensity of several different pixels should be activated into same value in feature maps. As all pixels of a object can not have the same intensity value, CNN is able to learn some useful pixels and discard noisy pixel. Hence, the marker object can also be learned. But increasing the number of different objects makes this learning procedure more harder.

Chapter 5

Conclusion

This thesis shows how Task Parametrized Dynamic Movement Primitives can be modeled by using Convolutional Neural Networks. Even out of value space of collected dataset, the developed model still performs well. More specifically, the model is generalized for multiple real objects by using data augmentation. The power of Convolutional Neural Networks for visual task and data augmentation are way confirmed in this work.
As the proposed approach performs much robust on object positions, the vision part can be used for doing different tasks in the future, e.g. grasping object. It can be also extended for more difficult sweeping tasks, e.g. classifying different multiple real objects and pushing into different collection points.

Appendix A

DVD

List of Figures

1.1 Sweeping task 6
1.2 Research purpose 7

2.1 Dataset structure 9
2.2 tanh 11
2.3 ReLU 12
2.4 Image preprocessing for CONVRNN 14
2.5 Splitting dataset 15
2.6 Architecture of CONVRNN 16
2.7 CNN vs. CNN + RNN 19
2.8 Highlighting object 20
2.9 Image preprocessing for CNN 20
2.10 Data augmentation 21
2.11 Architecture of CNN 22
2.12 Augmented image for pretraining 26
2.13 Augmented image for end-to-end training 27

3.1 Server-Client 29
3.2 Check camera position 30
3.3 Curve fitting of marker position in training set 31
3.4 Curve fitting of marker position in validation set 31
3.5 Feature maps after first convolutional layer 33
3.6 Feature maps after second convolutional layer 34
3.7 Feature maps after third convolutional layer 35
3.8 First feature map after third convolutional layer of a motion 36
3.9 Curve fitting of forcing terms in validation set 37
3.10 Curve fitting of forcing terms in training set 37
3.11 GMM vs. CONVRNN 38
3.12 End-to-End training loss 39
3.13 Feature maps 40
3.14 Feature maps 41
3.15 Curve fitting of forcing terms in validation set 42
3.16 Curve fitting of forcing terms in training set 42

3.17 GMM vs. CNN 43
3.18 Executing motion with marker object 44
3.19 Executing motion on sponge 44
3.20 Moving object with disturbance 45
3.21 Forcing terms of moving object with disturbance 46

4.1 Impact of base marker 47

Acronyms and Notations

CNN Convolutional Neural Network

CONVRNN A combination of CNN and RNN

DMP Dynamic Movement Primitive

MAE Mean Absolute Error

MSE Mean Squared Error

RNN Recurrent Neural Network

ROS Robot Operating System

Bibliography

[FTD+16] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 512–519. IEEE, 2016.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[Hoc91] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91, 1991.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[INS03] Auke J Ijspeert, Jun Nakanishi, and Stefan Schaal. Learning attractor landscapes for learning motor primitives. In *Advances in neural information processing systems*, pages 1547–1554, 2003.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[LFDA16] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[Mun14] Andres Munoz. Machine learning and optimization. *URL: https://www. cims. nyu. edu/˜ munoz/files/ml_optimization. pdf [accessed 2016-03-02][WebCite Cache ID 6fiLfZvnG]*, 2014.

[PL17] Affan Pervez and Dongheui Lee. Learning task parameterized dynamic movement primitives using mixture of gmms. *Intelligent Service Robotics*, 2017.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[ZSV14] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

License

This work is licensed under the Creative Commons Attribution 3.0 Germany License. To view a copy of this license, visit http://creativecommons.org or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.