Learning Deep Movement Primitives using Convolutional Neural Networks

Affan Pervez, Yuecheng Mao and Dongheui Lee

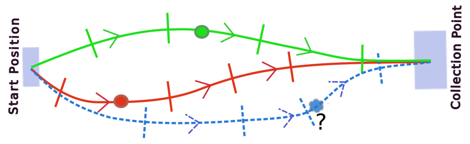


Fig. 1: In a sweeping task, the position of the trash (colored circles) can be considered as the task parameter, governing variations in the demonstrations.

***Abstract*— Dynamic Movement Primitives (DMPs) are widely used for encoding motion data. Task parameterized DMP (TP-DMP) can adapt a learned skill to different situations. Mostly a customized vision system is used to extract task specific variables. This limits the use of such systems to real world scenarios. This paper proposes a method for combining the DMP with a Convolutional Neural Network (CNN). Our approach preserves the generalization properties associated with a DMP, while the CNN learns the task specific features from the camera images. This eliminates the need to extract the task parameters, by directly utilizing the camera image during the motion reproduction. The performance of the developed approach is demonstrated through a trash cleaning task, executed with a real robot. We also show that by using the data augmentation, the learned sweeping skill can be generalized for arbitrary objects. The experiments show the robustness of our approach for several different settings.**

I. INTRODUCTION

Programming by Demonstration (PbD) is an active research area in robotics where a skill is acquired through human demonstrations. The aim of learning is not only the exact reproduction of the demonstrations, but also the generalization to unseen scenarios. In PbD the teacher is not assumed to have expert knowledge in robotics or programming. This provides a great potential for industrial applications, as it can help to reduce the setting up time of an assembly line.

A Dynamic Movement Primitive (DMP) is a way to represent motor actions [15]. It can encode a discrete/rhythmic movement. The only input to a DMP is a clock signal, for which it generates the corresponding acceleration command, for motion execution. A skill encoded by a DMP can be further refined by using Reinforcement Learning (RL) [5]. External environmental variables can affect a motion. For example in a trash cleaning task, the position of the trash can be considered as an important factor, which can modify

All authors are with Human-centered Assisitve Robotics, Technical University of Munich (TUM), Germany. Dongheui Lee is also with the Institute of Robotics and Mechatronics, German Aerospace Center (DLR). {affan.pervez,yuecheng.mao,dhlee}@tum.de

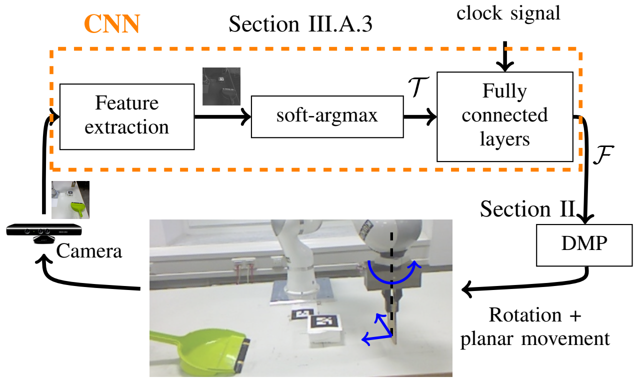


Fig. 2: An overview of the proposed Deep-DMP (**D-DMP**) architecture.

the behavior of the generated motions, as shown in Fig. 1. DMP in its original form, does not consider such external factors, which are referred as *task parameters* ($\mathcal{T}$) in this work. For task specific learning, we firstly want to learn from multiple demonstrations executed for different task parameters. Secondly, for adapting the motion to a new task, the task parameters should also be passed as an input along with the clock signal.

Task specific variations of DMPs are considered in [2], [9], [12], [16], [17]. A two-step learning process is adapted in [2], [17]. In the first step, a mapping from task parameters to the DMP parameters is learned. In the second step, the inferred DMP parameters are used for motion generation. The mapping in the first step can either be learned by locally weighted regression [17], or by a function approximator such as Gaussian process regression [2]. In a one-step learning process, the mapping from task parameters to the forcing terms of the DMPs is learned directly by using a suitable function approximator, eliminating the need to follow the two-step learning procedure [16]. In the task parameterized DMP (TP-DMP) approach [12], learning can be performed with a few demonstrations by using the Gaussian Mixture Model (GMM) based direct encoding of the forcing terms.

The main limitation of the task parameterized DMP approaches is that the task parameters have to be provided during the motion reproduction. This is usually done by using the dedicated vision systems, specifically designed for the targeted problems. For instance a blob detection algorithm is used for tracking the position of a ball [5]. In [12] markers are used for extracting positions of objects in the environment. The use of such dedicated systems limits the use of these approaches for real world scenarios. Levine et al. have shown that Convolutional Neural Networks (CNNs) can be used for generating motor actions, by extracting useful features directly from camera images [8]. Their experiment consists of learning robotic grasping from monocular images by using RL. They used 800,000 grasp attempts executed over 14 robotic manipulators. The amount of resources needed and the learning time makes the applicability of such an approach infeasible for real world scenarios. If an optimizer can provide trajectories for solving the manipulation task, then it can be used to guide the policy search of a CNN to a good local optima and to speed up the learning procedure [1], [7]. A drawback of [7] is that the task has to be first formulated as an optimization problem and requires the user to define a cost function for the executed actions. These constraints limit the applicability of their approach to complex tasks. A PbD approach based on a combination of convolutional auto-encoders and a fully connected neural network is presented in [18]. A shortcoming of their approach is that it cannot achieve the generalization properties which are typically associated with a DMP, for instance temporal rescaling of the motion and the variation of the goal position.

In this work, we present a PbD approach which combines a DMP and a CNN. By doing so, we preserve the generalization aspects of the DMP model, while the task specific features are automatically learned by the CNN. A brief overview of the proposed approach can be visualized in Fig. 2. In our approach, camera images are directly used during motion reproduction. This eliminates the need to provide task parameters during the motion reproduction. We also show the robustness of our approach by evaluating its performance for different objects, for extrapolating beyond the demonstrated region, for temporal rescaling of the motion, for changing the goal position, for unseen object and against disturbance in the images. Our approach can learn from relatively few number of demonstrations and as the model is learned through demonstrations, we do not need to formulate the task as an optimization problem, as in [7].

II. Movement Primitives

A. Dynamic Movement Primitive

In this work we consider the DMP formulation presented in [10]. Each degree of freedom (DoF) is encoded by a separate DMP. A canonical system acts as a clock. The different DoFs are synchronized by using the common clock signal

$$\tau \dot{s} = -\alpha_s s \tag{1}$$

The parameter s is usually initialized to one and it monotonically decays to zero, τ is the temporal scaling factor while α_s determines the duration of the movement. The canonical system drives the second order transformed system:

$$\begin{aligned} \tau \dot{v} &= k(g-x) - dv - k(g-x_0)s + sk\mathcal{F}(s) \qquad (2) \\ \tau \dot{x} &= v \end{aligned}$$

where x_0 and g are start and goal positions respectively. The damping term d is set such that the system is critically damped, while k acts like a spring constant. The learning of forcing term $\mathcal{F}(s)$ allows arbitrarily complex movements. $\mathcal{F}(s)$ is defined as $\frac{\sum_{i=1}^{K} \psi_i(s)\omega_i}{\sum_{i=1}^{K} \psi_i(s)}$ where $\psi_i(s) = \exp(-h_i(s-c_i)^2)$ are Gaussian basis functions with spread h_i, centers c_i and adjustable weights ω_i. To encode a movement, we first register $x(t)$ and its first and second derivatives $v(t)$ and $\dot{v}(t)$ respectively at each time step $t = 0, \ldots, T$. Then, for a suitable value of τ, we integrate the canonical system and calculate the target value $\mathcal{F}_{tar}(s)$ for each time step

$$\mathcal{F}_{tar}(s) = \frac{\dot{v}\tau - k(g-x) + dv + k(g-x_0)s}{sk}$$

Learning is performed to minimize the error criterion $J = \sum_s (\mathcal{F}_{tar}(s) - \mathcal{F}(s))^2$ which is a linear regression problem and the weights ω_i are learned with weighted least squares.

B. Task Parametrized (TP)-DMP

In the TP-DMP, the task specific variables are also passed as an input along with the clock signal [12]. This means that the forcing terms are now a function of clock signal and the task variables, i.e. $\mathcal{F}(s, \boldsymbol{\mathcal{T}})$. The dependency between the clock signal, the task parameters and the forcing terms is encoded by using a Gaussian Mixture Model (GMM). A GMM with K components is parameterized by $\boldsymbol{\theta}_{(K)} = \{\boldsymbol{\pi}_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K}$, where $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_K$ are mixing coefficients with constraints $\boldsymbol{\pi}_k > 0$ and $\sum_{k=1}^{K} \boldsymbol{\pi}_k = 1$, $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ are means and $\boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_K$ are covariance matrices. A data scarcity problem arises when fitting the GMM in TP-DMP. This problem is solved by using an Expectation Maximization approach presented in [12]. After encoding data with a a GMM, the forcing terms for the given clock signal and task parameters is synthesized by using Gaussian Mixture Regression.

III. Proposed approach

A. Deep-DMP

As stated earlier, the forcing term of a DMP can be modeled with any suitable function approximator. In this work, we model it with a CNN. By doing so we preserve all the useful properties associated with a DMP model, i.e. temporal and spatial rescaling properties as well as

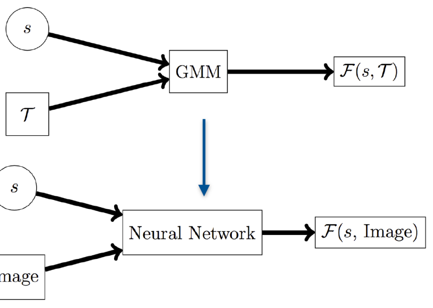


Fig. 3: Schematic of TP-DMP (above) and **D-DMP** (below).

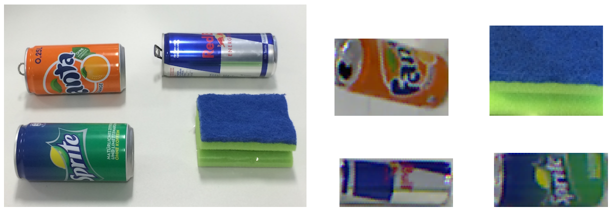


Fig. 4: Selected real objects (left) for generalization and their cropped images (right) for data augmentation .

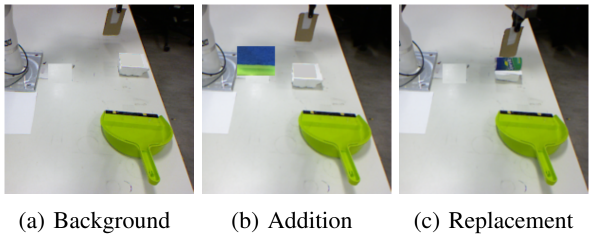


Fig. 5: Data augmentation process.

the guaranteed convergence to a goal position, while the generalization capabilities associated with feature learning are exploited by using a CNN. Since we employ deep learning, we call this formulation Deep-DMP (**D-DMP**). Compared with TP-DMP, the camera images are directly processed by the **D-DMP**, eliminating the need to extract the task parameters during motion reproduction, as shown in Fig. 3.

1) Data collection: Kinesthetic teaching is often used for data collection in PbD [6], [13]. In kinesthetic teaching, a teacher physically holds the robot's end-effector for generating the required motion. Since we want to use camera images, if a human is always present in the images during demonstrations, a CNN can learn human specific feature. Now during motion reproduction, if the human is not present in the image, then it can result in a failure of the task during reproduction phase [7], [8], [18]. Alternatively a human can provide teleoperated demonstrations as in [11], [18]. Since generating teleoperated demonstrations can be a time consuming and a tedious task for a human operator, we use an already trained TP-DMP model for collecting the demonstrations. For the task considered in this paper, see [12] for details about data collection and training process of the TP-DMP. The dataset consists of 50 motions executed for different task settings. The entire data collection process by using TP-DMP took less than one hour. Each motion has 480 samples, which contain the clock signals, the forcing terms of the DMPs, the RGB images and the task parameters. We select 45 motions (21600 data-points) for the training set while the remaining 5 motions (2400 data-points) are used for validating the learned model. This is lower than $800{,}000$ grasp attempts in [7], at least 156 execution trails in [8] and $24,500$ and 3500 data-points in training and validation sets respectively in [18]. RGB images were captured with a Kinect Xbox 360 camera. It has a resolution of 480×640 pixels (height$\times$width). The left parts of the images contain the robot's base and are irrelevant for the task. We keep the right 480×480 pixels of the images, as they are relevant for the task and then resize them to 200×200 pixels for computational efficiency.

2) Data augmentation: In this work, we consider the position of an object in an image as a task parameter. Position of objects are often important when performing manipulation or reaching tasks. Here we show that if the task parameters are defined as the position of an object, then by using the proposed data augmentation approach, the learned CNN can generalize for various different objects. For TP-DMP, this position is extracted by placing a marker on the object. Our data augmentation process consists of first removing the marker from the image. This is done by covering the marker with a white patch of similar color as that of the table, as shown in Fig. 5a. These marker-less images are now used as the background images for data augmentation. Now the data augmentation process is used for two purposes. Firstly for pretraining the network to predict the position of the objects in the images, by inserting the RGB image of a randomly selected real object, at a randomly selected position, as shown in Fig. 5b. Secondly for learning to generate the motions for the real objects, by replacing the marker with a real object in the image, as shown in Fig. 5c. The images of the four objects that are being used for data augmentation can be visualized in Fig. 4. With data augmentation, the learned model generalizes for multiple real objects, without any need of detecting their positions. The objects positions have to be only provided during the training phase while the learned CNN directly uses the camera images during the motion reproduction, eliminating the need to extract the task parameters.

3) Network architecture: The architecture of our CNN is illustrated in Fig. 6. It consists of nine layers, namely the input layer for image, three convolutional layers, a reduced layer of feature maps, input to fully connected layers, two fully connected layers and a linear output layer. The neural network takes a RGB image and a clock signal as inputs and predict the forcing terms of the DMPs. The input image has 120000 dimensions ($200 \times 200 \times 3$) as compared with a single value of the clock signal. The image vector can dominate over the clock signal due to its high dimensionality. To avoid this domination, the clock signal is concatenated with the extracted task parameters and then passed as an input to the fully connected layers. This is similar to what Levine et al. did with robot's state in their approach [7]. In order to avoid the vanishing gradient problem and for learning a good feature representation, the convolutional layers are pretrained to predict the object's position in the image.

Smaller sized kernels as in [3] can capture fine details from the image. Hence we use smaller kernels of size 3×3, with the stride length of 1 and SAME padding, as shown in Fig. 6. After convolutional layers we use a variant of soft attention mechanism called soft-argmax, which is introduced in [7]. The soft-argmax 'softpicks' out the position or indices of the maximum value within a matrix. It transforms a feature

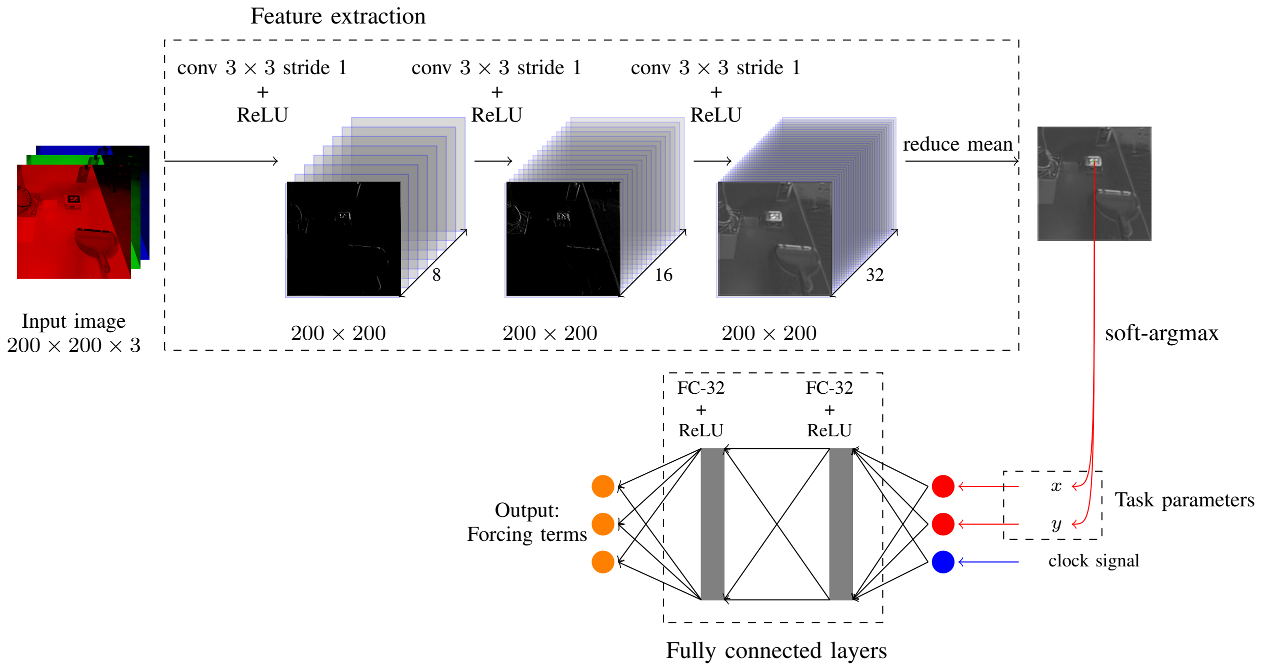


Fig. 6: Architecture of CNN.

map into a probability matrix, then the probability matrix is element wise multiplied with the position matrices. The position matrices for x and y directions are defined by Eq. (3) and (4)

$$M_x = (\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 199 & 199 & \cdots & 199 \end{bmatrix} - 100.0)/100.0 \tag{3}$$

$$M_y = (\begin{bmatrix} 0 & 1 & \cdots & 199 \\ 0 & 1 & \cdots & 199 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \cdots & 199 \end{bmatrix} - 100.0)/100.0 \tag{4}$$

Now the output positions are calculated as $x = \sum_{k,l}(softmax(F) \circ M_x)_{k,l}$, $y = \sum_{k,l}(softmax(F) \circ M_y)_{k,l}$ 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 [7], we add an additional reduce_mean operator before soft-argmax. 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 as $F(i, j) = (\sum_n F_n(i, j))/n$ where F denotes the reduced feature map, F_n is a single feature map in the last convolutional layer, $n = [1...32]$, and $i, j = [0...199]$ are the indices of pixels. All activation functions are Rectified Linear Unit (ReLU) in our neural network. The fully connected layers are pretrained with the positions of the marker and the clock signals as inputs and with the forcing terms as outputs. Once both the convolutional layers and the fully connected layers are pretrained, the pretrained weights are updated further with end-to-end training.

We use Adam optimizer for updating the weights of the CNN [4]. 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 pixels intensities and the predicted task parameters are rescaled within the interval $[-1, 1)$. The learning rate for pretraining convolutional layers of both object with marker and the real objects is 0.001. The learning rate for pretraining fully connected layers starts at 0.01 and ends at 0.0001, and is decayed by half after every 20 epochs. The learning rate for doing end-to-end training starts at 0.001 and ends at 0.0001. It is decayed by half after every 3 epochs. The loss function for all training steps is the mean squared error and the batch size for updating the weights is 100. We trained our CNN by using TensorFlow v1.0 on GPU GTX1060 6G and it takes about 4 hours to train one model. The forward pass of the CNN takes about 20ms with python 2.7 on GTX1060 GPU. For avoiding the overfitting problem, we validate our model after every 50 steps.

IV. Experiments and Results

The proposed method is tested in a sweeping task. The goal of this task is to generate robot motion for moving a trash into the dustpan. The task parameters are defined as the position of the trash in the image. The position of the trash governs the movement of the robot's end-effector. For a new trash position, the robot should be able to generate a trajectory for moving the trash to the collection point. Three DMPs are learned, two for generating a planar motion and

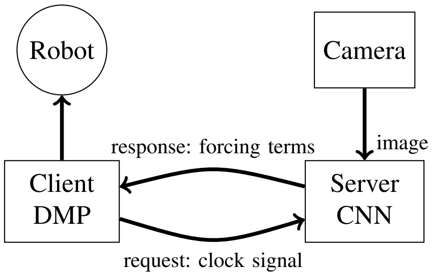


Fig. 7: Communication interface: interaction between DMP model and CNN using Server-Client pattern on ROS.

one for encoding the rotary motion about the z-axis of the end-effector, as shown by the blue arrows in Fig. 2.

A. *Experimental setup*

The experiments are conducted by using a KUKA light weight robot IV+. An already trained TP-DMP is used for collecting demonstrations for the **D-DMP**. For learning the TP-DMP, four demonstrations are collected via Kinesthetic teaching, by setting the robot to gravity compensation mode. For recording the position of the trash, a marker is attached on the object. The marker is tracked with Kinect RGB-D camera by using Robot Operating System (ROS) wrapper for Alvar, an open source augmented reality tag tracking library[1]. By using ROS a Server-Client communication interface is built between the CNN and the DMP as shown in Fig. 7. The client PC (Personal Computer) on which the DMP model is running has to control the robot in real time. In order to avoid the computational burden on the client PC, the CNN is executed on another high performance server PC. The server PC receives the clock signal from the client PC. It then passes the current image and the received clock signal to the learned CNN, which then calculates the required forcing terms. The predicted forcing terms are then sent back to the client PC. Finally, the DMP uses the received forcing terms for the motion execution.

B. *Feature maps*

An example of the feature maps after ReLU operation of the three convolutional layers is shown in Fig. 8. Since we do not decrease the size of feature maps, the feature maps of each convolutional layers are human readable. Figure 8b shows one of the feature maps after the third convolutional layer. The target object (marker) has a strong activation in the reduced feature map. Visually all feature maps look very similar after the third convolutional layer and have only minor differences. Although we can directly pass one of these feature maps into the soft-argmax operator for predicting the position of the object, we use the mean value of these feature maps for making the prediction more robust.

C. *Evaluation*

Now we calculate the errors and accuracies of predicted object's positions in the training and the validation set. The

[1]`http://wiki.ros.org/ar_track_alvar`

		Marker		**Real Objects**	
		horizontal	vertical	horizontal	vertical
Training	Error	1.87	4.56	3.23	8.28
	Accuracy	99.07	97.72	98.39	95.86
Validation	Error	1.83	4.39	3.44	8.36
	Accuracy	99.09	97.81	98.28	95.82

TABLE I: Errors (in pixel) and accuracies (%) of predicted positions in image coordinate system.

unit of error is pixel and the accuracy is stated in percentage. The (x, y) values are rescaled from range $[-1, 1)$ to $[0, 200)$ for calculating the error in pixels. Mean absolute error is calculated for the training set and the validation set. Errors in horizontal and vertical directions are evaluated separately. The resolution of the reduced feature map is 200×200, so we use 200 pixels as reference value and calculate the accuracy of predicted position by using Eq. (5). The errors and accuracies of the pretrained networks for predicting the positions of the marker and the objects are given in Tab. I.

$$\text{Accuracy} = \big(1 - \text{error}/(200\text{pixels})\big) \times 100 \tag{5}$$

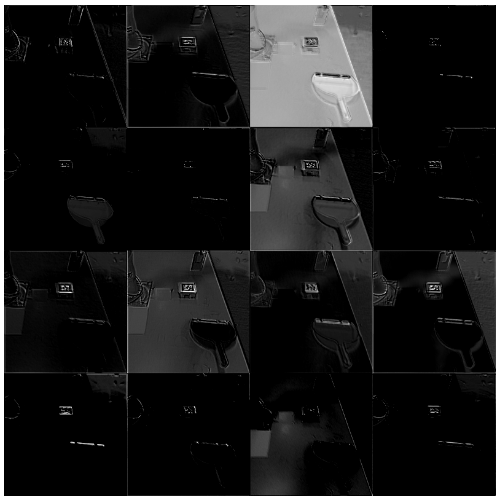
In order to check the performance of our CNN on real robot, the trained model is evaluated in different scenarios. At first, we tested that our CNN has learned the correct behavior from the collected dataset and can reproduce the results in [12]. We evaluate our model with the marker placed over the object. The successfully executed motion with images at different time steps and their corresponding reduced feature maps are visualized in Fig. 9a. The feature maps have high activation at the marker. The red dot in the feature maps represent the predicted task parameters, which in our case is the object's position in the image. Since the TP-DMP can be used for generating the forcing terms for an object with the marker, they are also being generated with the TP-DMP, while the motion is being executed by the **D-DMP**. Figure 9b shows the generated forcing term from the two models, for motion along x-axis. We can see that the graph of the predicted forcing terms from the **D-DMP** is smooth and closely matches to the graph of the TP-DMP.

Now we run the robot with objects without markers. We evaluate the performance of our approach for an object, which is not present in the training data. Since the object specific features are learned by the CNN, the learned skill can be generalized for the objects that look similar to the objects in the training data. The successfully executed motion for a new object can be visualized in Fig. 10. The red rectangle in the left most image in Fig. 10 depicts the range of the starting positions of the trash in the collected dataset. It can be seen in image sequences that the learned CNN can generalize for real object, even when they are placed outside the range of the training data.

By combining DMP with a CNN, we inherit the desirable attributes associated with a DMP model. For instance Fig. 11 shows that the motion can adapt for a change in the dustpan position, by changing the goal value g in Eq. (2). The speed of the executed motion can be changed, by changing the

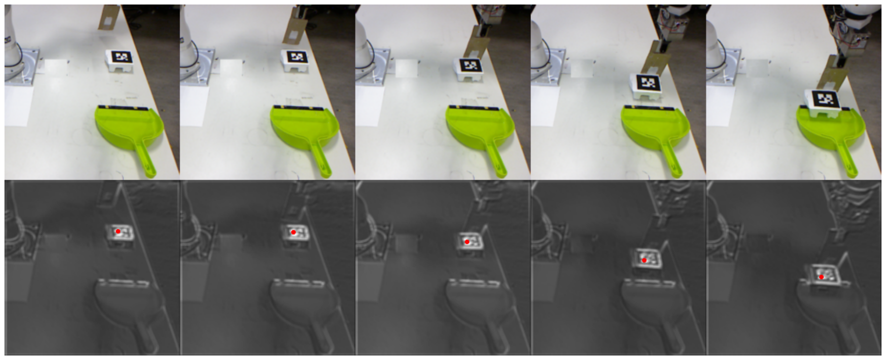
(a) Feature maps after first convolutional layer.

(b) One feature map after third convolutional layer.

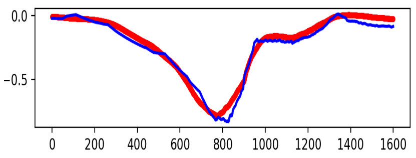


(c) Feature maps after second convolutional layer.

Fig. 8: Feature maps from the three convolutional layers.



(a) The second row represents the reduced mean feature map, with the red dots showing the predicted marker position by the CNN.



(b) Generated forcing term for motion in x-axis by the TP-DMP (red curve) and the **D-DMP** (blue curve).

Fig. 9: Motion execution with a marker on the object.

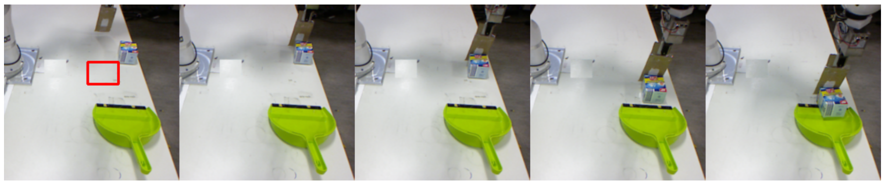


Fig. 10: Motion execution for an object not used during the training. The red rectangle represents the bounding box of the initial trash positions in the collected dataset. The object is placed outside the range of training data for evaluating the extrapolation performance.

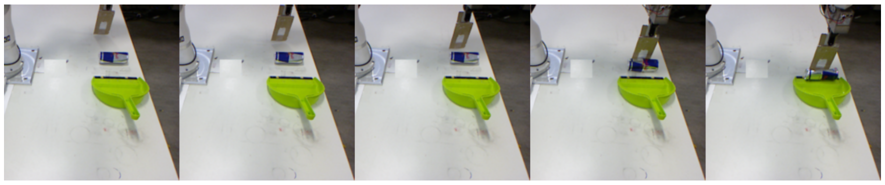


Fig. 11: Motion execution for a different goal position.

decay rate α_s in Eq. (1) (see attached video for results).

We also evaluate the performance of our model against the disturbance in the image, while changing the position of the sponge in real time. The results are shown in Fig. 12. The sponge is pulled and pushed by a human hand during the task execution. The hand was never observed by the CNN in the

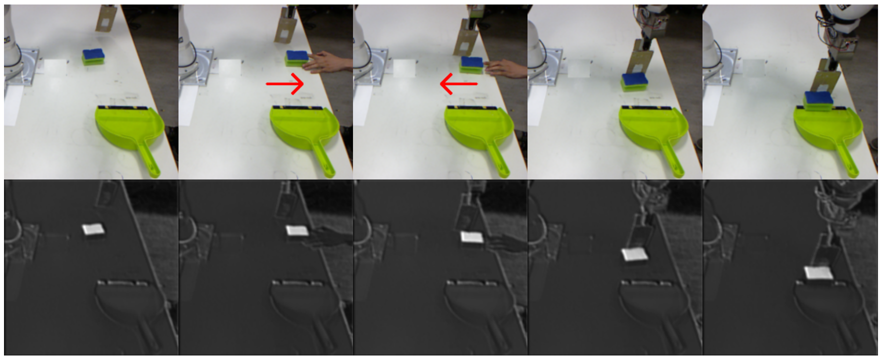


Fig. 12: Robust motion execution under perturnation (above). Corresponding reduced mean feature map (below).

training data and can be termed as a disturbance in the image. The end-effector of the robot follows the changing position of the sponge and neglects the disturbance in the image, by successfully pushing the sponge into the dustpan. A strong activation of the object of interest (sponge) can be visualized in the extracted feature maps. Since the shape and color of the hand is very different from the objects in the training data, the performance of the CNN remains unaffected by its introduction in the image.

V. Conclusion and Future work

This work proposes **D-DMP**, which is learned by modeling the forcing terms of a TP-DMP with a CNN. Existing approaches for learning TP-DMP usually require specialized vision systems or markers for extracting task specific variables. In contrary, task specific features are learned by supervised learning in our approach. In our approach the task parameters are only required during the training phase. After learning, the camera image is directly used for motion reproduction. We also show the pretraining of the convolutional layers for extracting the task specific variables. This reduces the effect of vanishing gradient problem as compared with directly doing end-to-end learning of the CNN. Since the features are learned, this eliminates the need to use any markers. Additionally by applying the data augmentation, our approach can easily generalize the learned skill for various different objects. The generalization capability of our approach is demonstrated by executing the task for a novel object and against the background perturbations.

Our approach requires relatively fewer training data for learning, as compared with other CNN based motor learning approaches [7], [8], [18]. Applying RL for learning a motor skill can require a lot of trials [8]. Thus the imitation strategy proposed in this work can be used to initialize the network parameters and for rapidly acquiring a motor skill, while the skill refinement can subsequently be done by using RL [14]. In this work the goal position g is provided by the user. Alternatively, it can be also predicted by the learned network, along with the forcing terms of the DMPs.

References

[1] Y. Chebotar, M. Kalakrishnan, A. Yahya, A. Li, S. Schaal, and S. Levine, "Path integral guided policy search," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3381–3388.

[2] D. Forte, A. Gams, J. Morimoto, and A. Ude, "On-line motion synthesis and adaptation using a trajectory database," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1327–1339, 2012.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[4] D. Kinga and J. B. Adam, "A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[5] J. Kober and J. Peters, "Imitation and reinforcement learning," *IEEE Robotics & Automation Magazine*, vol. 17, no. 2, pp. 55–62, 2010.

[6] D. Lee and C. Ott, "Incremental kinesthetic teaching of motion primitives using the motion refinement tube," *Autonomous Robots*, vol. 31, no. 2, pp. 115–131, 2011.

[7] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[8] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, 2016.

[9] T. Matsubara, S.-H. Hyon, and J. Morimoto, "Learning parametric dynamic movement primitives from multiple demonstrations," *Neural Networks*, vol. 24, no. 5, pp. 493–500, 2011.

[10] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *IEEE International Conference on Robotics and Automation, ICRA*, 2009, pp. 763–768.

[11] A. Pervez, A. Ali, J.-H. Ryu, and D. Lee, "Novel learning from demonstration approach for repetitive teleoperation tasks," in *IEEE World Haptics Conference (WHC), 2017*, pp. 60–65.

[12] A. Pervez and D. Lee, "Learning task-parameterized dynamic movement primitives using mixture of gmms," *Intelligent Service Robotics*, pp. 1–18, 2017.

[13] M. Saveriano, S. i. An, and D. Lee, "Incremental kinesthetic teaching of end-effector and null-space motion primitives," in *IEEE International Conference on Robotics and Automation (ICRA), 2015*, pp. 3570–3575.

[14] M. Saveriano, Y. Yin, P. Falco, and D. Lee, "Learning control policies using a simplified robot model," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[15] S. Schaal, "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics," in *Adaptive Motion of Animals and Machines*. Springer, 2006, pp. 261–280.

[16] F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud, "Learning compact parameterized skills with a single regression," in *13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2013, pp. 417–422.

[17] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *Robotics, IEEE Transactions on*, vol. 26, no. 5, pp. 800–815, 2010.

[18] P.-C. Yang, K. Sasaki, K. Suzuki, K. Kase, S. Sugano, and T. Ogata, "Repeatable folding task by humanoid robot worker using deep learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 397–403, 2017.