Convolutional Encoder-Decoder Networks for Robust Image-to-Motion Prediction

Barry Ridge[1,2], Rok Pahič[2], Aleš Ude[2], and Jun Morimoto[1]

[1] ATR Computational Neuroscience Laboratories, Kyoto, Japan,
barry.ridge@atr.jp, xmorimo@atr.jp,
[2] Dept. of Automatics, Biocybernetics, and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia,
rok.pahic@ijs.si, ales.ude@ijs.si

Abstract. A deep encoder-decoder network was previously proposed for learning a mapping from raw images to dynamic movement primitives in order to enable a robot to draw sketches of numeric digits when shown images of same. In this paper, the network architecture, which was previously constructed entirely with fully-connected linear layers, is modified to include convolutional layers in order to improve the image encoder component and make the network more robust to noise. The convolutional layers are pre-trained as part of an MNIST digit classifier and adapted for use in the encoder-decoder network, before the network is trained using a dataset composed of digit images and corresponding writing trajectories. This architecture was tested on several challenging noisy digit datasets and the use of convolutional layers is shown to provide a robust improvement in results.

Keywords: deep neural networks, dynamic movement primitives

1 INTRODUCTION

Effectively learning to predict action mappings directly from perceptual input is a highly challenging problem in robotics research that has seen a broad variety of approaches attempting to solve it in different settings. One example is that of object-action complexes (OACs) [8], which is a grounded representation that binds objects, actions, and attributes in a causal model. The particular setting under consideration in this work is depicted in Fig. 1, in which a robot must learn direct mappings between handwritten characters in input images and the motion trajectories needed to draw them. Our previously proposed fully-connected encoder-decoder network architecture [9] used dynamic movement primitives (DMPs) [6] for movement representation and this proved to be an effective choice both for representation and learning with the neural network and ultimately for control of the robot when drawing the actual digits. The fully-connected architecture, however, was not ideal for image representation.

In this paper, we investigate a different architecture that combines the benefits of convolutional layers for image encoding with those of a fully-connected

encoder-decoder architecture for DMP parameter prediction and image-to-motion representation in a low-dimensional latent space. This combination allows for relatively robust prediction compared to the previously proposed architecture, even when the input images are heavily corrupted by noise. The use of convolutional layers has the added benefit of significantly reducing the number of network parameters and by pre-training these layers on images from a similar image domain, the learning process is further improved.

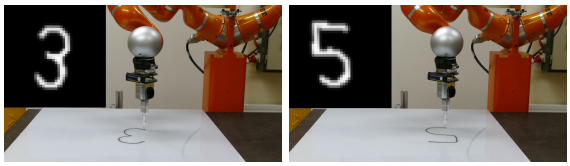

Fig. 1. Writing digits with a robot using image-to-motion encoder-decoder network prediction. The movements are generated using DMPs predicted by the network from the images shown in upper left corners.

Autoencoders [4], as well as variational autoencoders [5], have been demonstrated to be quite effective when it comes to calculating DMP-based representations of human motion. Since our focus is on learning direct mappings between images and actions, instead of using such autoencoder networks in which the DMP encoding occurs in the latent space, we use an encoder-decoder architecture in which the image is encoded from the input layer, the DMP parameters are predicted at the output layer and the transformation and generalization of the image-to-motion representation occurs in the low-dimensional latent space. Encoder-decoder networks in combination with convolutional layers have proven to be useful in computer vision. A well-known example is SegNet [2], in which pre-trained convolutional layers from a convolutional neural network (CNN) were adapted to form a fully-convolutional encoder-decoder architecture for semantic pixel-wise segmentation. We also use pre-trained convolutional layers from a CNN in this work, although the architecture differs in that the convolutional layers are only used to form the encoder part of the network. The CNN that we use here is also comparatively basic, but the use case is a very different one in which a more limited image recognition approach is sufficient and the architecture would nonetheless be extensible for more advanced scenarios.

Prior to the adoption of deep learning methods for such tasks, an effort was made by Ali [1] at modeling individual brush strokes of calligraphic characters using Gaussian Mixture Models, combining the brush strokes using Gaussian Mixture Regression and reproducing brush stroke trajectories using DMPs. The reproduced stroke trajectories were iteratively refined using reinforcement learning for learning examples in the database, but each reproduction started from scratch with no generalization between different examples or to new unseen cases. Usually when CNNs are used for supervised learning of perception-action couplings, they are used in combination with another neural network in two separately trainable parts. In [13], Yang et al. first used a deep convolutional

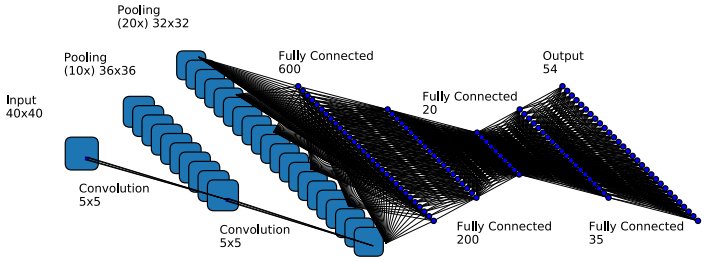

Fig. 2. The CIMEDNet architecture.

autoencoder for finding camera image features and then in combination with recorded robot angles, formed sequences for the learning task dynamics with a time delay neural network. Pervez et al. [11] used a pre-trained CNN for finding task parameters from input images, while using a another fully-connected neural network to learn to generate forcing terms from the clock signal and task parameters, before combining both networks in an end-to-end training scheme. Both of these two examples produce the next step from the image of the current step while working in online loop, whereas our method, by contrast, uses just single images for generating entire trajectories.

2 CONVOLUTIONAL IMAGE-TO-MOTION ENCODER-DECODER NETWORKS

The structure of the data under consideration in this work is the same as in [9] where the input and output data pairs take the form $\mathbf{D} = \{\mathbf{C}_j, \mathbf{M}_j\}_{j=1}^{M}$ where M is the number of input and output training pairs, $\mathbf{C}_j \in \mathbb{R}^{H \times W}$ are the input images of width W and height H, and $\mathbf{M}_j$ the corresponding movements associated with each image, i. e. $\mathbf{M}_j = \{\boldsymbol{y}_{i,j}, t_{i,j}\}_{i=1}^{T_j}$. Here $\boldsymbol{y}_{i,j} \in \mathbb{R}^{\mathrm{d}}$ are the vectors describing the movement's degrees of freedom, e. g. Cartesian positions or joint angles, $t_{i,j} \in \mathbb{R}$ the measurement times for the j-th movement, and d is the number of degrees of freedom. However, it should be noted that in this paper, we convert the movements $\mathbf{M}_j$ to DMPs and construct all of the datasets used to train the network models as follows: $\mathbf{D}' = \{\mathbf{C}_j, \mathbf{k}_j\}_{j=1}^{M}$, where $\mathbf{k}_j$ are the DMP parameters calculated for each movement $\mathbf{M}_j$ and are represented as

$$\mathbf{k}_j = \left\{ \{\boldsymbol{w}_k\}_{k=1}^{N},\ \tau,\ \boldsymbol{g},\ \boldsymbol{y}_0 \right\}. \tag{1}$$

The construction of DMPs and the nature of the parameters $\{\boldsymbol{w}_k\}_{k=1}^{N}$, τ, $\boldsymbol{g}$ and $\boldsymbol{y}_0$ are explained in detail in the following subsection.

2.1 Motion Representation with DMPs

Letting a time-dependent movement trajectory be denoted as $\boldsymbol{y}(t) \in \mathbb{R}^d$, a DMP specifying this trajectory is given by the following system of differential equations

$$\tau \dot{\boldsymbol{z}} = \alpha_z(\beta_z(\boldsymbol{g} - \boldsymbol{y}) - \boldsymbol{z}) + \mathrm{diag}(\boldsymbol{g} - \boldsymbol{y}_0)\mathbf{F}(x), \tag{2}$$

$$\tau \dot{\boldsymbol{y}} = \boldsymbol{z}, \tag{3}$$

where $\boldsymbol{y}_0 \in \mathbb{R}^d$ is the initial position on the trajectory, $\boldsymbol{g} \in \mathbb{R}^d$ the final position on the trajectory, $\mathrm{diag}(\boldsymbol{g} - \boldsymbol{y}_0) \in \mathbb{R}^{d \times d}$ a diagonal matrix with components of vector $\boldsymbol{g} - \boldsymbol{y}_0$ on the diagonal, $\mathbf{F}(x) \in \mathbb{R}^d$ a nonlinear forcing term, $\boldsymbol{z} \in \mathbb{R}^d$ a scaled velocity of motion, and $x \in \mathbb{R}$ the phase defined by the following equation

$$\tau \dot{x} = -\alpha_x x. \tag{4}$$

The phase x is used instead of time to avoid explicit time dependency. It is fully defined by setting its initial value to $x(0) = 1$. Eq. system (2) – (4) constitutes a *dynamic movement primitive* (DMP). If the parameters $\tau, \alpha_x, \alpha_z, \beta_z \in \mathbb{R}$ are defined appropriately, e. g. $\tau, \alpha_x > 0$ and $\alpha_z = 4\beta_z > 0$, then the linear part of equation system (2) – (3) becomes critically damped and $\boldsymbol{y}$, $\boldsymbol{z}$ monotonically converge to a unique attractor point at $\boldsymbol{y} = \boldsymbol{g}$, $\boldsymbol{z} = 0$. The forcing term $\mathbf{F}(x)$ is usually defined as a linear combination of radial basis functions

$$\mathbf{F}(x) = \frac{\sum_{k=1}^{N} \boldsymbol{w}_k \Psi_k(x)}{\sum_{k=1}^{N} \Psi_k(x)} x, \tag{5}$$

$$\Psi_k(x) = \exp\left(-h_k \left(x - c_k\right)^2\right), \tag{6}$$

where c_k are the centers of Gaussians distributed along the phase of the trajectory, and h_k their widths. The role of $\mathbf{F}$ is to adapt the dynamics of (2) – (3) to the desired trajectory $\boldsymbol{y}(t)$, thus enabling the system to reproduce any smooth movement from the initial position $\boldsymbol{y}_0$ to the final configuration $\boldsymbol{g}$. This can be accomplished by computing the free parameters $\boldsymbol{w}_k \in \mathbb{R}^d$ using regression techniques. See [12] for more details.

α_z, β_z, and α_x are usually constants that do not change between movements. Thus the neural network needs to learn the other parameters of differential equation system (2) – (4) to fully specify a DMP as defined in Equation (1).

2.2 Network Architecture

In our improved architecture, images are encoded via convolutional layers that are pre-trained as part of a basic CNN classifier that was trained on the original MNIST dataset. The input is a $40 \times 40 \times 1$ grayscale pixel image, followed by a convolutional layer with 5×5 kernel size and 10 feature maps, a convolutional layer with 5×5 kernel size and 20 feature maps, a 0.5 dropout layer, a fully-connected layer of size 320, a fully-connected layer of size 50 and the output layer of size 10 matching the number of digits. After training the classifier, the

fully-connected layers are removed and the convolutional layers are retained and are used to form the first layers of the encoder in our proposed architecture. These convolutional layers are illustrated on the left side of Fig. 2. In different variations of the architecture, the encoder is either composed entirely by the convolutional layers or the convolutional layers are followed by additional fully-connected layers before reaching the decoder. The latter case is illustrated in Fig. 2 which shows two fully-connected layers with sizes of 600 neurons and 200 neurons respectively in the encoder following the two convolutional layers. These two variations of the encoder structure as well as their differences in terms of performance are described further in Section 3.

Following the final convolutional layers or fully-connected layers of the encoder depending on the precise network architecture, at the bottleneck of the network that forms the latent space representation, a decoder is formed via a number of fully-connected layers that gradually expand the number of units in each layer until the final output layer which has a size set to 55 units in order to match the DMP parameters $\{\boldsymbol{w}_k\}_{k=1}^{N}$, τ, $\boldsymbol{g}$ and $\boldsymbol{y}_0$. The layers of the decoder are illustrated on the right side of 2 starting with the bottleneck of size 20, followed by a layer of size 35 and finishing with the output layer. This is the same decoder structure as used [9] and we retain it here as-is, having found it to be effective throughout our experiments for this particular use case. The cost function used to evaluate the output of the network is the same as that of Equation (9) in [9], which is defined for the j-th DMP as follows:

$$\mathrm{E}_p(j) = \frac{1}{2}\left(\sum_{k=1}^{N} \|\boldsymbol{w}_k - \boldsymbol{w}_{k,j}\|^2 + (\tau - \tau_j)^2 + \|\boldsymbol{g} - \boldsymbol{g}_j\|^2 + \|\boldsymbol{y}_0 - \boldsymbol{y}_{0,j}\|^2\right), \quad (7)$$

where $\{\{\boldsymbol{w}_k\}_{k=1}^{N},\ \tau,\ \boldsymbol{g},\ \boldsymbol{y}_0\}$ denotes the output of the neural network and $\{\{\boldsymbol{w}_{k,j}\}_{k=1}^{N},\ \tau_j,\ \boldsymbol{g}_j,\ \boldsymbol{y}_{0,j}\}$ the DMP parameters from the training data $\mathbf{k}_j \in \mathbf{D}'$. For further details on the gradient calculations required for minimizing the cost function via backpropagation we refer the reader to [9].

3 EXPERIMENTS

In our experiments, we trained both the fully-connected image-to-motion encoder-decoder architecture (IMEDNet) and the convolutional architecture (CIMEDNet) on various digit image and motion trajectory datasets. We experimented with two different versions of the new architecture, one of which used exclusively pre-trained convolutional layers in the encoder CIMEDNet used pre-trained convolutional layers followed by some additional fully-connected layers in the encoder with the decoder constructed entirely with fully-connected layers. The IMEDNet architecture was the same as described in [9] with fully-connected hidden layer sizes of 1500, 1300, 1000, 600, 200, 20, and 35 neurons, respectively. The CIMEDNet architecture was as described in Section 2.2 and as illustrated in Fig. 2 where the two additional fully-connected layers in the encoder are shown to have sizes of 600 and 200 respectively. We used PyTorch [10] in order to implement all networks and trained our models on NIVIDIA GTX 1080 and 1080Ti

GPUs. When pre-training the MNIST CNN classifier, we used a stochastic gradient descent optimizer, a negative log-likelihood loss, a batch size of 64, a learning rate of 0.01 with momentum 0.5, and trained for 10 epochs. This achieved a 98% accuracy which we deemed sufficient for our purposes in extracting the trained convolutional layers for use in the encoder-decoder networks.

When training the encoder-decoder networks, we used the Adam optimizer [7] with a learning rate of 0.0005 and the epsilon parameter for numerical stability set to 0.001. In order to avoid learning plateaus, the optimizer parameters were periodically reset to initial values every 500 epochs. A mean squared error loss was used to evaluate the cost function in Equation (7) and the batch size was set to 128 for weight updates. As a stopping criterion, if the best validation loss was unchanged after 60 epochs, training was halted. The above training procedure was used for both the IMEDNet and CIMEDNet architectures. However, in the case of CIMEDNet, we also experimented with either freezing the convolutional layer weights or training the entire network end-to-end. The results for these different training regimes are cataloged in Table 1.

3.1 Datasets

In order to construct $\mathbf{D}$, we employed the same scheme described in [9] to generate 40×40 images of synthetically written digits and associated two-dimensional artificial writing trajectory movements. Briefly, the synthetic trajectory data was generated using a combination of straight lines and elliptic arcs. These geometric elements were used to generate grayscale digit images and their paramaters were varied according to a uniform distribution. The resulting images were processed with a Gaussian filter and some moderate salt-and-pepper noise was added to the foreground pixels. Finally, both the generated trajectories and the resulting images were transformed using affine transformations composed of translation, rotation, scaling, and shearing. These parameters were again taken from a uniform distribution. For the DMP representation of the trajectories, 25 radial-basis functions were selected for every dimension. The weights of these basis functions form together with the common time constant (1 parameter) and the start and the goal values of a planar movement (2×2 parameters), the full set of 55 DMP parameters that represent the motion. Using this procedure, several datasets were generated both with and without similar noise as used in the noisy MNIST (n-MNIST) datasets [3] as follows:

- **s-MNIST:** 2000 pairs of images and trajectories without any added noise were generated for each digit, for a total of 20000 samples that were split in a 70%/15%/15% ratio between training/validation/test data,
- **s-MNIST-AWGN-19.0:** 300 samples per digit/3000 total samples, using additive white gaussian noise with a signal-to-noise ratio of 19.0,
- **s-MNIST-AWGN-9.5:** 300 samples per digit/3000 total samples, using additive white gaussian noise with a signal-to-noise ratio of 9.5,
- **s-MNIST-MB:** 300 samples per digit/3000 total samples, using a motion blur filter emulating a linear motion of the camera of 5 pixels and a 15 degree motion in the counterclockwise direction,

– **s-MNIST-RC-AWGN:** 300 samples per digit/3000 total samples, using a contrast range scaled down to half as well as additive white gaussian noise with a signal-to-noise ratio of 9.5.

It should be emphasized that in the results that follow, only the s-MNIST dataset was used for training the presented models.

Table 1. DMP reconstruction statistics. The results are in pixels. The best result for each dataset is highlighted in boldface.

	s-MNIST	**s-MNIST-AWGN-19.0**	**s-MNIST-AWGN-9.5**	**s-MNIST-MB**	**s-MNIST-RC-AWGN**
IMEDNet (End-to-End)	0.22 ± 0.08	0.56 ± 0.20	1.66 ± 0.60	$\mathbf{0.35 \pm 0.15}$	2.32 ± 0.77
CIMEDNet (Frozen Conv.)	0.26 ± 0.10	0.54 ± 0.20	1.48 ± 0.55	0.47 ± 0.25	2.19 ± 0.76
CIMEDNet (End-to-End)	$\mathbf{0.19 \pm 0.08}$	$\mathbf{0.36 \pm 0.14}$	$\mathbf{1.02 \pm 0.45}$	0.36 ± 0.12	$\mathbf{1.93 \pm 0.66}$

3.2 Results

The main quantitative results are presented in Table 1 while qualitative results for selected samples are presented in Fig. 3. After training on the noiseless s-MNIST dataset each of the models were tested on all five of the noiseless and noisy s-MNIST datasets described in the previous section. The CIMEDNet architecture was trained with two separate training regimes in which the convolutional layer weights were frozen and the models were trained end-to-end respectively. For the quantitative evaluation, dynamic time warping was used to measure the mean pointwise pixel distance between the trajectories generated by the DMPs predicted by the networks from the digit images and the actual digit trajectories.

As can be seen in Table 1, the CIMEDNet model that is trained end-to-end significantly out-performs the IMEDNet model on both the noiseless s-MNIST dataset and on most of the noisy s-MNIST datasets, apart from the dataset featuring motion blur noise. We reason that this may be due to the fact that motion blur can significantly distort overall object shape and edge profiles and given that convolutional neural networks function the basis of exploiting hierarchies of image filters often heavily represented by edge detectors, this may impact on their effectiveness in such circumstances. The CIMEDNet that was trained with frozen convolutional layers also fared well, beating the IMEDNet model on the same noisy datasets despite not scoring as well on the noiseless dataset. This indicates that the feature detectors in the convolutional layers allow for more robust generalization whereas fully-connected layers are more inclined to overfit.

The qualitative result samples in Fig. 3 are also interesting. Results using the s-MNIST-RC-AWGN dataset are omitted as the noise levels are so pathologically difficult that the qualitative results are comparatively worthless. However, the CIMEDNet model often performs surprisingly well given that it was not trained or fine-tuned on the noisy data. Both models appear to produce highly legible

writing trajectories that closely match the actual trajectories in the case of the s-MNIST-MB dataset, but the CIMEDNet model is demonstrably superior to IMEDNet in many cases with the s-MNIST-AWGN-19.0 and s-MNIST-AWGN-9.5 data, producing much more legible results and demonstrating the robustness of the convolutional layers in dealing with even high noise levels.

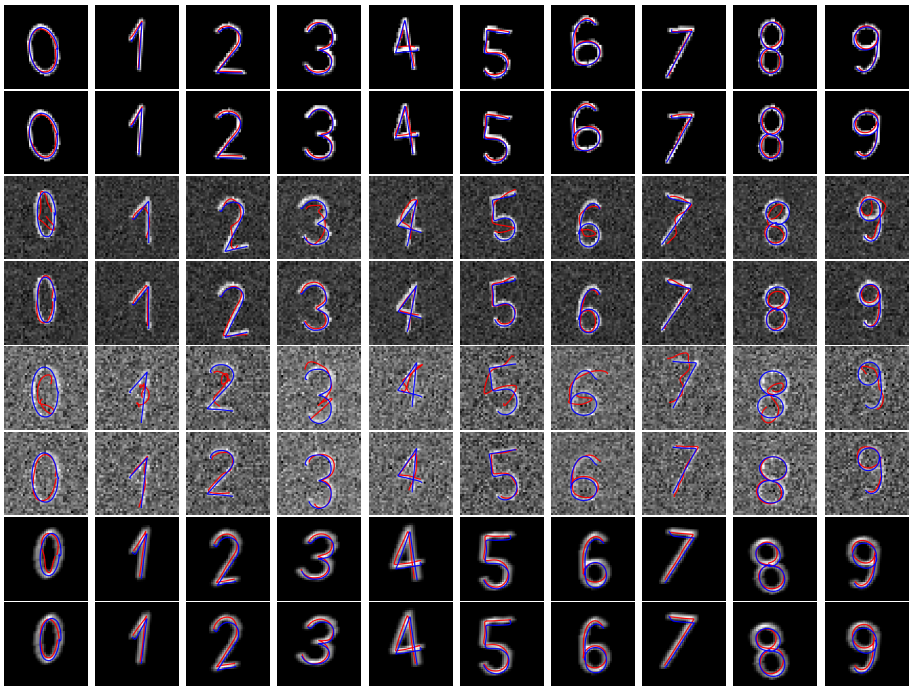

Fig. 3. Example results for IMEDNet (rows 1, 3, 5 & 7) & CIMEDNet trained end-to-end (rows 2, 4, 6 & 8). Rows 1 & 2: s-MNIST, rows 3 & 4: s-MNIST-AWGN-19.0, rows 5 & 6: s-MNIST-AWGN-9.5 and rows 7 & 8: s-MNIST-MB. Original trajectories are shown in blue, trajectories calculated by the neural networks are shown in red. Samples in matching dataset rows are identical.

4 CONCLUSIONS AND FUTURE WORK

We have presented an extended form of an encoder-decoder neural network for image-to-motion prediction that employs convolutional layers in the encoder in order to make the image recognition component more robust to noisy input. We have demonstrated that this architecture outperforms its predecessor on a variety of different kinds of noise. Regarding future work, we intend to further expand the capabilities of this model by incorporating layers from more powerful pre-trained CNN models into the encoder and training the network on more challenging image sets. One challenge here lies in either finding suitable image datasets that include trajectory information in their target outputs or in finding other means of producing images with corresponding motion trajectories, e.g. by gathering both in a robot simulation environment.

Acknowledgement: This work has received funding from the EU's Horizon 2020 RIA AUTOWARE (GA no. 723909); the Slovenian Research Agency under GA no. J2-7360; JSPS KAKENHI JP16H06565; NEDO; the Commissioned Research of NICT; the NICT Japan Trust (International research cooperation program); and JST-Mirai Program Grant Number JPMJMI18B8, Japan. The authors also wish to thank Marcel Salmič for his significant contribution to the PyTorch network implementations.

References

1. Ali, O.: Robotic Calligraphy: Learning From Character Images. Ph.D. thesis (2015)
2. Badrinarayanan, V., Kendall, A., Cipolla, R.: SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence **39**(12), 2481–2495 (2017)
3. Basu, S., Karki, M., Ganguly, S., DiBiano, R., Mukhopadhyay, S., Gayaka, S., Kannan, R., Nemani, R.: Learning Sparse Feature Representations Using Probabilistic Quadtrees and Deep Belief Nets. Neural Processing Letters **45**(3), 855–867 (2017)
4. Chen, N., Bayer, J., Urban, S., van der Smagt, P.: Efficient movement representation by embedding Dynamic Movement Primitives in deep autoencoders. In: 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), pp. 434–440 (2015)
5. Chen, N., Karl, M., van der Smagt, P.: Dynamic movement primitives in latent space of time-dependent variational autoencoders. In: 2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids), pp. 629–636 (2016)
6. Ijspeert, A.J., Nakanishi, J., Hoffmann, H., Pastor, P., Schaal, S.: Dynamical movement primitives: Learning attractor models for motor behaviors. Neural computation **25**(2), 328–373 (2013)
7. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. In: 3rd International Conference for Learning Representations (ICLR). San Diego (2015)
8. Krüger, N., Geib, C., Piater, J., Petrick, R., Steedman, M., Wörgötter, F., Ude, A., Asfour, T., Kraft, D., Omrčen, D., Agostini, A., Dillmann, R.: Object–Action Complexes: Grounded abstractions of sensory–motor processes. Robotics and Autonomous Systems **59**(10), 740–757 (2011)
9. Pahič, R., Gams, A., Ude, A., Morimoto, J.: Deep Encoder-Decoder Networks for Mapping Raw Images to Dynamic Movement Primitives. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 5863–5868. Brisbane, Australia (2018)
10. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: NIPS 2017 Autodiff Workshop (2017)
11. Pervez, A., Mao, Y., Lee, D.: Learning deep movement primitives using convolutional neural networks. In: 2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids), pp. 191–197 (2017)
12. Ude, A., Gams, A., Asfour, T., Morimoto, J.: Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives. IEEE Transactions on Robotics **26**(5), 800–815 (2010)
13. Yang, P.C., Sasaki, K., Suzuki, K., Kase, K., Sugano, S., Ogata, T.: Repeatable Folding Task by Humanoid Robot Worker Using Deep Learning. IEEE Robotics and Automation Letters **2**(2), 397–403 (2017)