Modeling action-conditional dynamics of objects from video

Arash Khodadadi

Indiana University, Department of Statistics, Bloomington, IN, United States

Abstract

In this report, we consider the problem of extracting states for modeling the dynamics of moving objects in videos. First, with a simple example, we show that a previously proposed architecture, has an undesirable property: it does not preserve the spatial information of the features in a frame. To mitigate this, we propose a new architecture. Like previous work, this network takes the form of an autoencoder. The novel property of this network is that, unlike the previous networks, here the input image is reconstructed at the output of the network using the weighted sum of radial basis kernels. In contrast to the conventional radial basis function (RBF) networks the weights are not trainable parameters. Instead, they are computed for each frame as the output of a network consisted of convolutional and fully connected layers. In this network, since the center of the RBFs are fixed, the spatial information is preserved.

1. Introduction

A dynamical system can be specified by a set of difference or differential equation (see for example Slotine & Li (1991)):

$$\begin{aligned} x_{t+1} &= f(x_t, u_t) + w_t \\ y_t &= g(x_t) + e_t \end{aligned} \tag{1}$$

In these equations, x_t is the state of the dynamical system which is usually not observable, y_t are observable variables which are assumed to be non-linear and corrupted version of the states, f and g are generally non-linear functions which determine the dynamic of the system, u_t is the control signal, and w_t and e_t are stochastic processes that model the noise in the system. It is usually desired to compute the control signal such that the states of the system take a specific profile through time. If the dynamic of the system was known (i.e., the function f, g were known) one could use for example *dynamic programming methods* to compute the control signal to achieve the desired behavior. However, usually the dynamic of the system is unknown. In addition, the states of the system are not observable directly. As an example, consider the problem of controlling a robotic arm based on the videos taken from a camera mounted on the robot. In this example, the observations y_t are the video frames at each time step.

One traditional approach to tackle this control problem is to build and estimate a model of the dynamical system and then used the estimated model to design the controller. The process of building a model of the dynamical system is known as *system identification* (Ljung (1998)). Identifying nonlinear systems is a hard problem and has been an active research area in the past decades (Ljung (2010)).

In recent years, with the advances in the area of the deep neural networks, the problem of vision-based control has gained attention (Wahlstrm et al. (2014, 2015); Assael et al. (2015); Finn et al. (2015); Gu et al. (2016); Agrawal et al. (2016); Finn & Levine (2016)). The main idea here is not to identify the system directly. Instead, given the set of observations (video frames), these method learn a compact representation of them. These lower dimensional signals are then considered as the states of a dynamical system and their dynamics are learned. The controller is design to achieve a desired profile on these new set of states. More precisely, given a set of observations $\{y_t, u_t\}, t = 0, 1, ..., T$, these algorithms extract a set of new states $z_t = \psi(y_t)$ and then learn the dynamics $z_{t+1} = \phi(z_t, u_t)$. The key point is that the functions ψ and ϕ are constructed using the deep neural network architectures.

A challenge in building such representations is that most of the deep architectures consist of convolutional layers. These layers are invariant to spatial information which is a desirable property for many computer vision task like recognition. However, the spatial information are important for control and so it is important to somehow preserve these information during building the compact representations. The aim of this report is to investigate this issue. First, we will consider a recently proposed network by Finn et al. (2015). We show that this network cannot preserve the spatial information if there are more than one salient features in a frame with similar appearance. Next, we propose a new architecture which does not suffer from this drawback. The performance of the networks are compared on a dataset recently released by Google called "Google's push dataset".

2. Related work

Several recent works have proposed different architectures for extracting states for modeling the dynamics of moving objects in videos. To the best of our knowledge, Wahlstrm et al. (2014) is the first considered this problem. They used an autoencoder network which first reduces the dimensionality of the input image using and encoder and then uses the inverse of the encoder as a decoder to reconstruct the input frame. The activation at the last layer of the encode, which has the smallest dimensionality in the network, is considered as the states. This architecture was used in a later work, to provide states for designing an optimal controller (Wahlstrm et al. (2015)).

Although the approach is appealing due to its simplicity, one major drawback of it is that all the spacial information is lost during the encoding stage. To mitigate this problem, Finn et al. (2015) proposed a network which consists of several convolutional layers followed by a spatial softmax layer. This network is

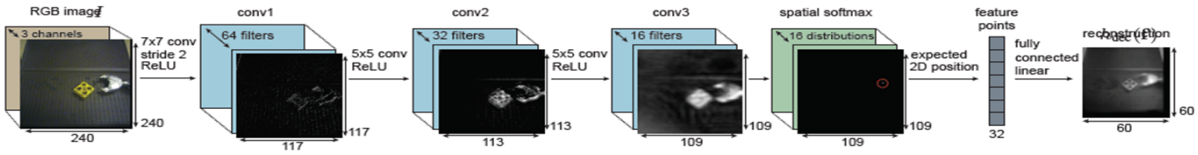

Figure 1: **SS network**. re-printed from Finn et al. (2015).

shown in Figure 1. The input frame is passed through 3 convolutional layers. The softmax probabilities of the last feature maps are then computed as follows:

$$S_{cij} = \frac{e^{a_{cij}/\alpha}}{\sum_{i'j'} e^{a_{ci'j'}/\alpha}} \tag{2}$$

In this equation, a_{cij} is the activation value of the feature map c at position (i, j) and α is the *temperature parameter*. After this, the expected 2D position of each softmax distribution is computed as follows:

$$f_c = \big(\sum_i i \cdot s_{cij}, \sum_j j \cdot s_{cij}\big) \tag{3}$$

If each of the feature maps, capture one dominant feature in the frame, this will provide an estimate of the position of each feature. These proposed features are then augmented to form a 32D feature vector which is then combined linearly to reconstruct a down-sampled version of the frame at the output. As we will show in the next section, if there are more than one dominant feature with similar appearance, then the spacial information will be lost in this network.

Another approach for modeling dynamical systems from video data is to model the dynamic of the pixels directly and without extracting any states. This approach is taken for example by Finn & Levine (2016). Their proposed network consists of a series of convolutional layers followed by a series of LSTM layers. The latter layers are responsible for learning the dynamical changes in the pixel values. The network is trained to predict the value of the pixels in the next time step given the pixel values in the current time step and the control signal.

3. Preservation of spacial information in SS network: or its lack thereof

The main purpose for the averaging operation in the SS network is to provide the spacial information of the dominant features in an image. Our goal in this section is to show that the spacial information will be lost, however, in this network if there are more than one dominant features in the input image.

To this end, we trained the SS network on a dataset (the details of the dataset and the training procedure is provided in Sections 5 and 6). After training, we gave the image shown in panel (a) of Figure 2 as the

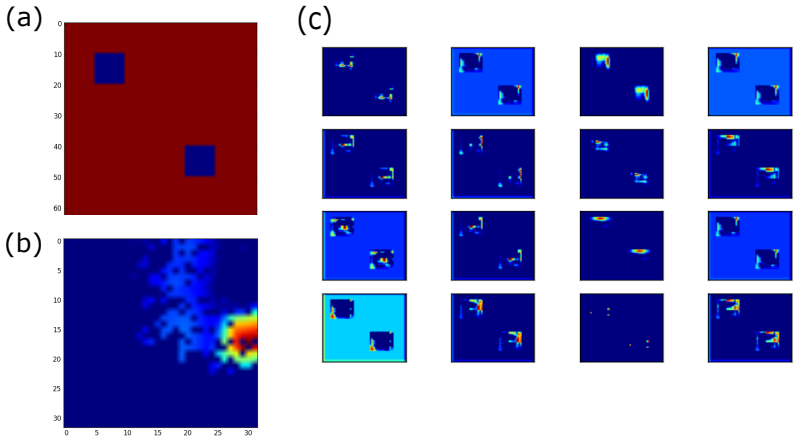

Figure 2: **Lack of spacial information preservation in the SS network**. (a) input image, (b) reconstructed output, (c) the response of the feature maps in the fourth layer of the SS network.

input to the network. Panel (c) of this figure, indicates the response of the 16 feature maps in the fourth layer of the network to this image. As it can be seen, each feature map is detecting a specific aspect of the two squares in the image. The critical point is that each filter has responded exactly the same to the two squares. Now, if we apply the averaging operation in Equation 2 to each feature map, the average will be at the center of the image. Therefore, the output of the network indicates that there is an important feature at the center of the screen. Obviously, this is not the case and so the spacial information of the two squares is lost in the output of the network. Panel (b) of Figure 2 shows the reconstructed image at the output of the SS network. As it can be seen, the network totally ignores the left square.

This simple example shows that the SS network cannot preserve the spacial information when there are more than one dominant features with similar characteristics in an image.

4. RBF-CNN architecture

In this section, to mitigate the problem illustrated in the previous section, we propose a new architecture. This architecture is depicted in Figure 3. Similar to previous architectures propose for state extraction (see Section 2), this architecture acts as an autoencoder: it tries to reconstruct its input at its output. The input frame is first passed through several convolutional layers. The output of the last convolutional layer is then fed to a fully connected module (consisted of one or more fully connected layers). The output of this module is then used as the weights of a RBF module. The output of the network is a weighted sum of the RBF kernels in the module named *RBF bank*. We consider a grid over the input image. At each grid point, several RBF filters are considers. These filters differ in their covariance matrices. More precisely, at each position

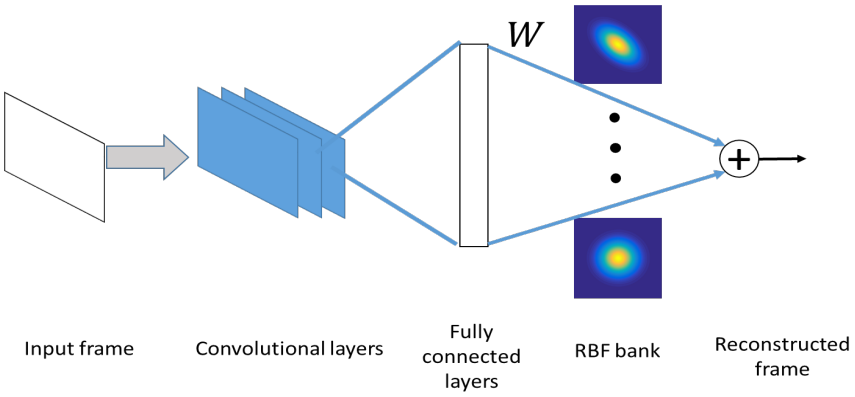

Figure 3: **RBF-CNN architecture.** See the text for more details.

(i, j) on the grid, we consider k RBF filters where the u^{th} filter $K_u(i, j)$ is specified by its covariance matrix Λ_u. The output of the network is the weighted sum of all these filters:

$$\hat{y} = \sum_{i,j,u} W_{i,j,u} K_u(i, j) \tag{4}$$

We used $k = 3$ with the following covariance matrices:

$$\Lambda_1 = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}, \Lambda_2 = \begin{bmatrix} 10 & -5 \\ -5 & 10 \end{bmatrix}, \Lambda_3 = \begin{bmatrix} 10 & 5 \\ 5 & 10 \end{bmatrix} \tag{5}$$

Reconstructing the input image using a limited number of RBF filters enforces the sparsity in the proposed autoencoder architecture. The spacial information in this network is preserve due to the fact that the position of each filter remains the unchanged. Assume that in a given input image there is a dominant object at the center. We expect that in the trained RBF-CNN, the weights corresponding to the filters positioned at the center of the image have larger values. As the object moves away from the center, these weights decrease and the weights in the neighboring areas increase. Therefore, if we consider the weights (denoted by W in Figure 3) as the sates, the spacial information is preserved. Moreover, the dynamic of these states will correspond to the dynamic of the moving objects in the video.

It is import to note that in contrast to the conventional RBF networks, in this architecture the weights of the RBF kernels are not trainable parameters. Instead, these weights are computed for each given input image using trainable convolutional and fully connected layers.

5. Training networks

We trained each network in two ways. This section provides the details on these procedures for training the networks.

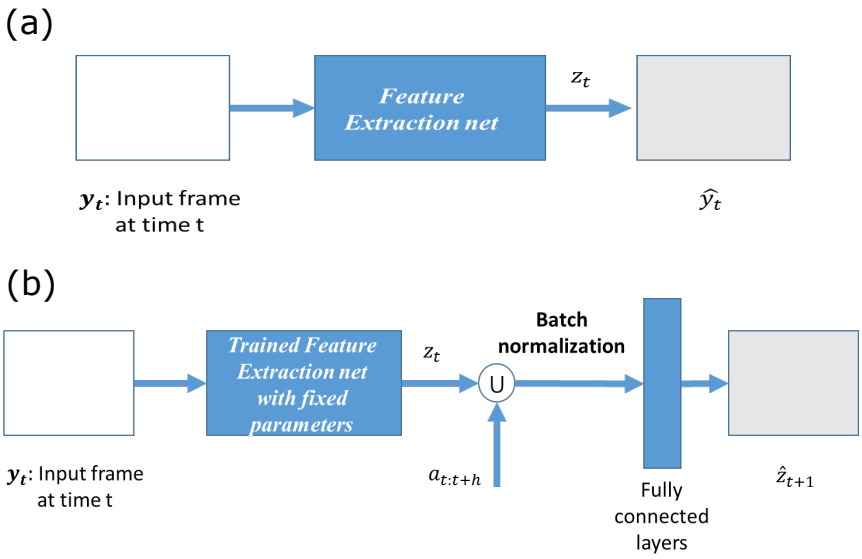

Figure 4: **Separate state extraction and prediction.**

5.1. Separate training for state extraction and prediction

This procedure for training the networks is depicted graphically in Figure 4. In this method, training is performed in two steps. In the first step (panel (a) of the figure), the network is trained to reconstruct a frame at its output. After training, in the second step (panel (a) of the figure), the weights are kept fixed and for each given frame the network is used to extract the states z_t. These states are then used to train another fully connected network to predict the next state.

Specifically, consider two consecutive frames y_t and y_{t+1}. The trained network in hte first step is used to extract z_t and z_{t+1}. Then, z_t is augmented with the control command $a_{t:t+1}$. The resulting vector is used as the input to the fully connected network. The output of this network is predicted value of the state in the next time step $\hat{z}_{t+1}$. The error $\hat{z}_{t+1} - z_{t+1}$ is used to adjust the weights of the fully connected network.

As it can be seen in panel (b) os Figure 4, after augmenting the a_t and the control command, we batch normalize the resulting vector. We found this step very critical for training the network. The reason is that the range of the values of the states and the control commands differ by an order of magnitude. Therefore, without batch normalization, the network would ignore the control command and so will perform poorly in predicting the next state.

5.2. End-to-end training for state extraction and prediction

This method is illustrated in Figure 5. In this method, for a given frame y_t at time t, the network's output (z_t) is computed and augmented with the control command $a_{t:t+1}$. The resulting vector is batch-normalized

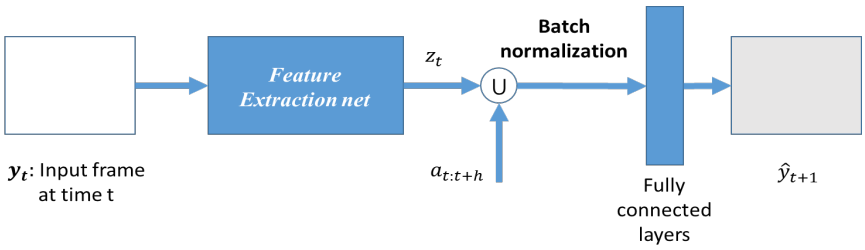

Figure 5: **End-to-end training for state extraction and prediction.**

and fed to another fully connected module. The output of this is considered as the reconstructed version of the next frame $\hat{y}_{t+1}$. The error in this case will be $\hat{y}_{t+1} - y_{t+1}$ which is used to adjust the weights of the network and the last fully connected module together.

6. More details on training

6.1. Input frames

As it was mentioned before, each network will learn to reconstruct its input at the output. In Finn et al. (2015) the original RGB frames were used as the input. In this work, we used the Euclidean norm of the optical flow of each frame as the input of each network. One property of the dataset is that many of the pixels do not change from frame to frame and therefore the optical flow is zero for them. This property has the advantage that the network will learn to focus only on moving objects. However, the fact that many pixels are zero will cause the network to be biased to produce zero output as we explain in the next section.

In addition, the push dataset contains a frame which shows the scene before the robot arm enters. We subtracted each frame in each video from this initial frame. In addition to the optical flows, we trained RBF-CNN once with these subtracted frames as the input.

6.2. Masking errors

When we first trained the network using the sum of square errors between the input frame and the output of the network, only after a few iterations of training, all the weights would go to zero and the network would predict zero for all input frames. The reason for this bias toward producing zero output is that in the training set, many pixels in the optical flow of each frame is zero (more than 95% of the pixels in total). Therefore, by producing 0 at the output, the network can achieve relatively low error value.

To overcome this issue, we first computed the 95% percentile of the pixel values in the training set. Let p denote this value. Next, for each frame we created a mask frame with the same size as the input frame (64×64). The (i,j) entity of this matrix, e_{ij}, was set equal to 1 if $I(i,j) > p$. The value of 10% of randomly selected of the remaining pixels of this matrix was also set to 1 and all remaining pixels were set to 0. A

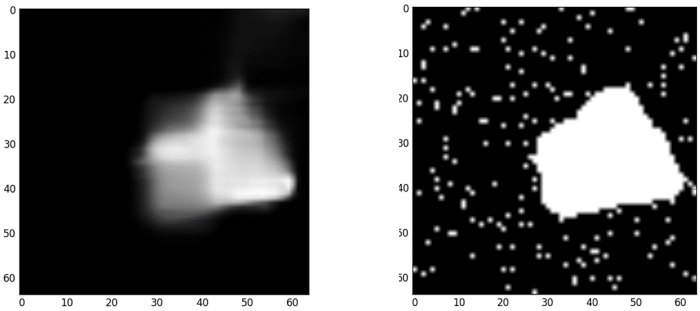

Figure 6: **An example of the optical flow of a frame and the corresonding mask.**

sample of the optical flow of a frame and its corresponding mask are shown in Figure 6.This mask was multiplied by the reconstruction error.

6.3. Training procedure

All networks were trained using stochastic gradient descent with Adam optimizer (Kingma & Ba (2014)). Unfortunately, we did not have access to any GPU and so all the networks were trained on a PC with Intel(R) core TM i7-2600 CPU, 3.4 GHz with 16 GB of RAM.

7. Experiments

7.1. Dataset

We used Google's push dataset available publicly at: `https://sites.google.com/site/brainrobotdata/home/push-dataset`. This dataset contains roughly 59,000 videos each about 3-5 secs long of a robotic arm pushing objects (See Figure 7 for two sample frames). The data was collected using 10 different robots with different camera orientations. The dataset also contains the commanded pose and the robot's state at each time step.

Due to limited computational resources, we only used about 5% of these data for training the networks. The commanded pose changes every 3 time-steps. Therefore, we sub-sampled the videos by this rate and computed the optical flow based on I_t and I_{t+3}. The commanded pose was used as the control signal for predicting the next frame.

7.2. Results of separate training

As it was explained before, in this method we first trained the network to predict a frame itself and then used the states extracted from the trained network to train another fully connected network which predict

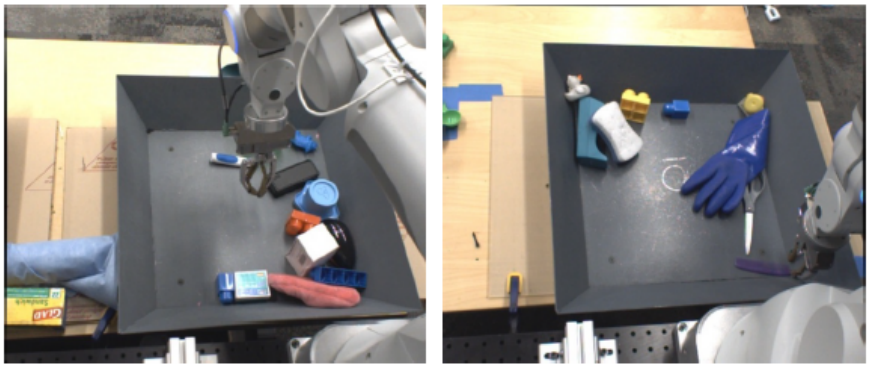

Figure 7: **Two sample frames from the Google's push dataset.** Re-printed from Finn & Levine (2016).

the state in the next time step. The aim of this experiment was to investigate the "predictability" of the states provided by each network.

To provide a baseline, we performed PCA on the optical flows of the training set and used the first 50 principal components as the states. Then we trained the fully connected network to predict the first 50 PCs of the next frame.

We also trained RBF-CNN and the SS networks both on the optical flows as their input. Since the networks are trained to reconstruct the frame, this is a regression problem. Therefore, one reasonable method for comparing the performance is to compare the R^2 values defined as follows:

$$R^2 = \frac{SS_{reg}}{Syy} \tag{6}$$

where $SS_{reg} = \sum(y - \hat{y})^2$ and $Syy = \sum(y - \bar{y})^2$. Here, $\bar{y}$ is the mean pixel values and so R^2 measures the improvement obtained by using the regression method over using the mean as the prediction.

The R^2 values obtained from 1 step ahead prediction of the states obtained from the PCA, RBF-CNN and the SS network are shown in Figure 8. As it can be seen, there is a significant improvement in R^2 when using the two networks compared to using the states from PCA. There is no significant difference between the two networks.

7.3. Results of end-to-end training

In this method of training, each network is augmented with another fully connected module. The goal here is to reconstruct the frame in the next time step given the frame at the current time step and the commanded pose.

We trained both RBF-CNN and the SS network with optical flows as the frames, using this method. In addition, we trained RBF-CNN with the subtracted frames (see Section 6.1) as input. The R^2 values are reported in Figure 9. Each network was trained for reconstructing the frame at its input (0 step horizon) and the frame at the next time step (1 step horizon).

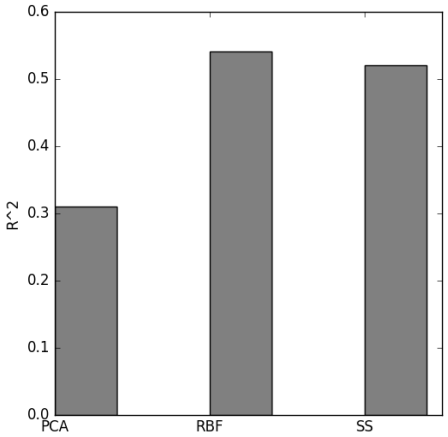

Figure 8: R^2 **for 1-step ahead prediction of the states obtained from the PCA, RBF-CNN and the SS network.**

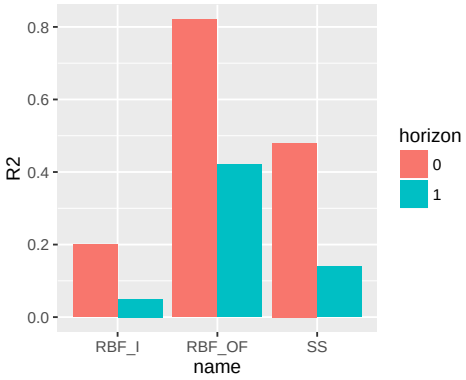

Figure 9: R^2 **for 0-step and 1-step ahead prediction of the frames using the the SS network (SS), RBF-CNN with optical flow input (RBF_OF) and RBF-CNN with subtracted frames (RBF_I)**.

The results for RBF-CNN with subtracted frames (labeled RBF_I in the figure) are not reliable. It seems that the network was not trained well. During training, the test error would reduce quickly in the first few iterations and remained constant afterwards.

For RBF-CNN with optical flow input, the value of R^2 drops from about 80% for 0 step to about 40% for 1 step. For the SS network these values are 50% and 10%, respectively. Based on these results, it seems that the states provided by RBF-CNN are more predictable than those provided by the SS network.

Any conclusion based on these results should be made with caution. In the original implementation of the SS network (Finn et al. (2015)), the temperature parameter of the softmax (Equation 1) is considered as a trainable parameter. However, we set the value of this parameter to 1. Two sample frames and the 1 step ahead reconstruction of then using the SS network are shown in Figure 10. As it can be seen, the prediction

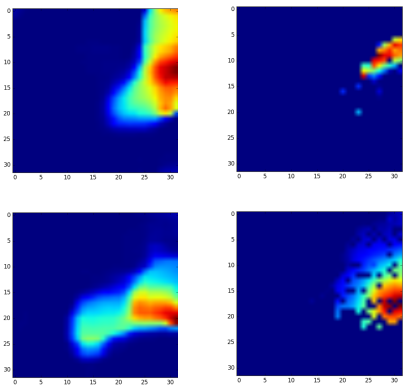

Figure 10: **Two samples of the groud truth frame (left column) and the 1 step ahead prediction of the SS network (right column)**.

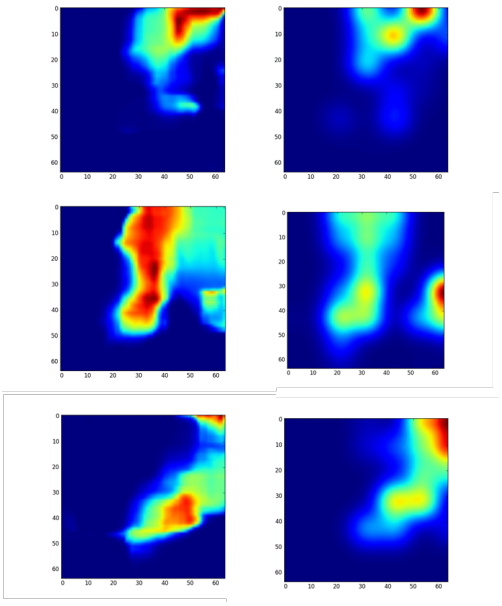

Figure 11: **Three samples of the groud truth frame (left column) and the 1 step ahead prediction of RBF-CNN (right column)**.

frame has more zero pixels than the ground truth. One reason for this could be that the temperature parameter is large and so the softmax distribution is set to zero at a smaller than optimal threshold. By setting the temperature as a trainable parameter, the quality of the reconstruction in the SS network could increase and so the R^2 values will improve.

Some examples of the ground truth frames and the 1 step ahead reconstruction of RBF-CNN are shown in Figure 11. As we can see, the reconstructed frames have generally a good quality.

8. Limitations and future work

This work has several limitations. First, the networks are trained on only a small portion of the dataset. The main for this was the lack of computational resources. Training on the whole dataset may lead to different results than what were achieved in this report. Second, we only considered the one step ahead predictions of the models. To extend to longer horizons it is necessary to augment each network with a recurrent network such as LSTM cells. Third, as we explained in the previous section, our implementation of the SS network is not complete. Specifically, we did not set the temperature parameter as trainable. The results showed that this has a major effect on the performance of this network in reconstructing a frame. Fourth, we have compared the networks only based on the R^2 values of the reconstruction errors. A complete comparison, however, is not possible except if the states extracted from each network is used for control. We plan to to this comparison using simulated control problems.

In conclusion, the aim of this work was not to provide a through comparison between the methods. The goal here was to introduce the idea of using kernels with fixed position as a solution to the problem of invariance to spatial information in convolutional layers.

References

Agrawal, P., Nair, A., Abbeel, P., Malik, J., & Levine, S. (2016). Learning to Poke by Poking: Experiential Learning of Intuitive Physics. *arXiv:1606.07419 [cs]*, .

Assael, J.-A. M., Wahlstrm, N., Schn, T. B., & Deisenroth, M. P. (2015). Data-Efficient Learning of Feedback Policies from Image Pixels using Deep Dynamical Models. *arXiv:1510.02173 [cs, stat]*, .

Finn, C., & Levine, S. (2016). Deep Visual Foresight for Planning Robot Motion. *arXiv:1610.00696 [cs]*, .

Finn, C., Tan, X. Y., Duan, Y., Darrell, T., Levine, S., & Abbeel, P. (2015). Deep Spatial Autoencoders for Visuomotor Learning. *arXiv:1509.06113 [cs]*, . URL: `http://arxiv.org/abs/1509.06113`.

Gu, S., Lillicrap, T., Sutskever, I., & Levine, S. (2016). Continuous Deep Q-Learning with Model-based Acceleration. *arXiv:1603.00748 [cs]*, .

Kingma, D., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, . ArXiv: 1412.6980.

Ljung, L. (1998). *System Identification: Theory for the User*. Pearson Education. Google-Books-ID: fYSrk4wDKPsC.

Ljung, L. (2010). Perspectives on system identification. *Annual Reviews in Control*, *34*, 1–12.

Slotine, J.-J., & Li, W. (1991). *Applied Nonlinear Control*. Englewood Cliffs, N.J: Pearson.

Wahlstrm, N., Schn, T. B., & Deisenroth, M. P. (2014). Learning deep dynamical models from image pixels. *arXiv:1410.7550 [cs, stat]*, .

Wahlstrm, N., Schn, T. B., & Deisenroth, M. P. (2015). From Pixels to Torques: Policy Learning with Deep Dynamical Models. *arXiv:1502.02251 [cs, stat]*, . URL: `http://arxiv.org/abs/1502.02251`. ArXiv: 1502.02251.