Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische
Friedrich-Wilhelms-Universität Bonn

Master thesis

Video Prediction at Multiple Levels with Hierarchical Recurrent Networks

Author:
Ani Karapetyan

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Prof. Dr. Joachim K. Anlauf

Advisor:
M. Sc. Angel Villar-Corrales

Submitted: June 6, 2022

Declaration of Authorship

I declare that the work presented here is original and the result of my own investigations. Formulations and ideas taken from other sources are cited as such. It has not been submitted, either in part or whole, for a degree at this or any other university.

Location, Date

Signature

Abstract

Effective human-robot collaboration requires autonomous agents not only to perceive the dynamic environment they are interacting with, but also to have the ability of making predictions about the behavior and future state of the nearby agents. The problem of predicting a plausible continuation for the given sequence of video frames is known in the literature as *video prediction*. Despite the recent advances in deep-learning based video prediction, existing methods lack the ability to make reliable long-term predictions suitable for perceptual inference and spatial reasoning applications. Although for relatively longer time horizons it is not even feasible to forecast pixel-level details, reliable prediction of more abstract representations, such as human poses or scene semantics, can be more beneficial than blurry pixel-level predictions.

In this thesis we propose and investigate *MSPred* (**M**ulti-**S**cale Hierarchical **Pred**iction), a novel hierarchical video prediction model able to simultaneously predict future possible outcomes of different levels of granularity at distinct time resolutions, conditioned on the given video frames. We evaluate the performance of our model on three datasets, namely Moving-MNIST [95], KTH-Actions [88] and SynPickVP [46, 79]. We empirically demonstrate that MSPred achieves quite competitive results for future frame prediction, outperforming several popular video prediction baselines, while simultaneously making plausible future predictions at distinct abstraction levels and time-scales. In addition, we conduct an extensive ablation study and analysis to investigate the importance and effect of different components of our model.

Contents

1 Introduction

To achieve effective human-robot collaboration, autonomous systems, such as industrial or domestic robots, should not only recognize the dynamic environment they are interacting with, but also be able to make predictions about the behavior and future state of the nearby agents. The problem of video prediction is formally defined as follows. Given C input (or *context*) frames, the goal is to generate next N frames that make up a plausible continuation for the given sequence (Figure 1.1). This is a very challenging task, since the model should be able to estimate the spatio-temporal dynamics of the environment, striving to capture its inherent uncertainty. The problem of future frame prediction complies with the self-supervised learning paradigm, since no expensive annotations are needed for training. Moreover, video prediction is a particularly promising task, due to its representation-learning nature, since rich internal representations are required for making coherent predictions.

The necessary level of abstraction of the predicted representations depends on the specific scenario and the desired time horizon. For example, highly detailed pixel-level predictions are desirable when forecasting the immediate future. Although for relatively longer time horizons it is no longer feasible to forecast pixel-level details, it can be beneficial to predict more abstract structures, such as object poses or scene semantics. Moreover, in order to plan longer into the future, reliable prediction of representations of higher abstraction level, such as actions or locations, can be more beneficial than blurry pixel-level predictions.

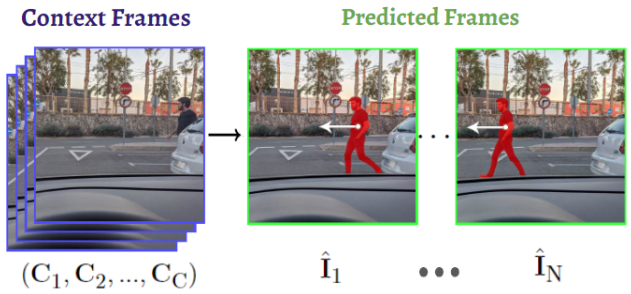


Figure 1.1: Illustration of the future frame prediction task for C context and N predicted frames. Figure adapted from Oprea et al. [77].

In recent years, many deep-learning-based approaches [15, 102, 54, 87, 77] have been designed for future frame prediction. However, existing models mainly specialize on predicting detailed future frames for the next few time-steps, and suffer from blurry and poor generations for longer time-horizons, which makes it not feasible to apply them for perceptual inference and spatial reasoning scenarios. Moreover, most of these methods employ an autoregressive flow in pixel-space, hence requiring a lot of iterations to make predictions for larger time-steps, and therefore are prone to fast error accumulation.

Most of the successful efforts towards long-term video prediction rely on making predictions in a more tractable structured space, such as the space of human poses [103, 104, 71, 27] or semantic maps [68, 96, 67, 58]. A number of these models [103, 104, 27, 58] concatenate multiple networks making predictions in an intermediate structured space, prior to the actual future frame generation. However, most of these models are not trained in an end-to-end manner, and are often constrained to specific domains. A few other approaches [68, 67, 71, 96] attempt to directly make predictions into a high-level space from raw image frames. However, these models are limited to making predictions of specific structure, such as segmentation maps, which can result in a loss of relevant information for other tasks. Furthermore, these models lack the flexibility to simultaneously make predictions in different levels of abstraction.

In this thesis, we propose and investigate a novel hierarchical video prediction model *MSPred* (**M**ulti-**S**cale Hierarchical **Pred**iction), addressing the above-mentioned issues. MSPred is a convolutional and recurrent neural network designed to predict future plausible outcomes of different levels of abstraction at different temporal resolutions in parallel, conditioned on the given video frames. In order to better model the world dynamics, MSPred leverages a hierarchical predictor module, with different levels operating at different spatial and temporal resolutions.

The thesis has the following structure:

- *Chapter 2: Theoretical Background*

  Summarizes deep-learning prerequisites and popular architectures that make up the main building blocks of video prediction models.

- *Chapter 3: Related Work*

  Reviews the related work on video prediction problem, the existing efforts towards long-term video prediction, as well as hierarchical architectures for improved future frame prediction.

- *Chapter 4: Approach*

  Outlines the overall architecture and design choices of different components of the proposed MSPred model, and describes some implementation details.

- *Chapter 5: Evaluation and Results*

  Specifies the datasets and metrics used for evaluation of the model, reports the quantitative and qualitative results on each dataset, and comparison with existing video prediction models. Presents the conducted ablation study and analysis of our model.

- *Chapter 6: Conclusion*

  Encloses the thesis with a conclusion on the presented method and results, as well as some notes on the future work.

The thesis has been partially submitted as a paper to IROS 2022 (International Conference on Intelligent Robots and Systems), as well as published as an arXiv preprint by Karapetyan et al. [46].

2 Theoretical Background

In this chapter we briefly summarize the essential theoretical background and concepts for the thesis.

2.1 Deep Learning Essentials

Traditional machine learning and computer vision algorithms rely on hand-crafted features (e.g. SIFT [64], SURF [3]) designed by domain-experts, for processing high-dimensional data. In contrast, deep neural networks are able to extract powerful representations from unstructured data (such as images and videos) and learn function approximations based on those features, within the same model.

Deep learning encompasses the methods and paradigms enabling deep neural networks to learn hierarchical representations from data [55]. The standard fully-connected feedforward neural network, also known as a multilayer perceptron (MLP), defines a computational graph that stacks multiple layers of neurons (computational units), where each neuron is connected to all the neurons on its previous and the next layers. In a broad sense, the purpose of a neural network is to approximate an unknown function f^* by a category of functions $y = f(x; \Theta)$ defined by the computational graph of the network, mapping input x to output y [28]. The parameters (or weights) Θ of the network are learned from the observed data. Figure 2.1 illustrates an example of a simple feedforward neural network with two hidden layers. Each neuron first computes a *pre-activation* as a linear combination of its inputs and the respective incoming edge *weights*, then applies a non-linear *activation function* and passes the output to the next layer.

2.1.1 Popular Activation Functions

The most popular activation functions used in the modern deep networks, especially in the area of computer vision, include rectified linear unit [41, 98, 28] (ReLU), logistic sigmoid [116], hyperbolic tangent (tanh) [69], softmax [28], and their different variants, such as Leaky ReLU [44] among others. ReLU [28] represents a default choice for the activation function in many scenarios, and is simply

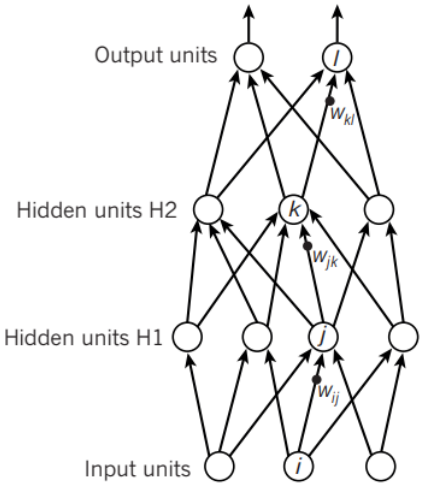


Figure 2.1: An example of a simple MLP. At each layer, the output of the previous layer (or the input in case of the first hidden layer) is linearly transformed and passed through an element-wise non-linear activation function to produce the input for the next layer. Parameters (or weights) of the network are denoted by w, whereas bias terms are omitted in the visualization. Figure extracted from LeCun et al. [55].

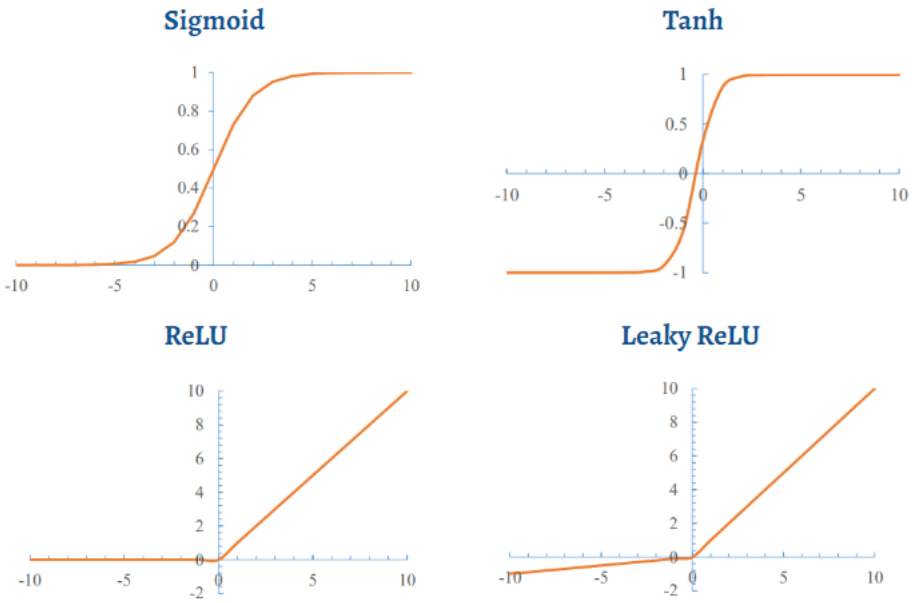


Figure 2.2: Visualization of some popular activation functions: Sigmoid, Tanh, ReLU and Leaky ReLU. Figure extracted and modified from J. Yang et al. [113].

defined as $g(z) = \max\{0, z\}$. Due to its piecewise-linearity, ReLU makes the networks easier and faster to optimize [28]. Despite being widely used, ReLU has the drawback of being prone to "dying" [31] during training. This can happen when, for instance due to very large learning rate, the weights of the network arrive at a state where a ReLU unit always outputs the same score (zero) for any input. In this case it will forever stay in that state, since there is no gradient signal to help it recover. The Leaky ReLU [44] activation function adds a small positive gradient for the negative range of inputs, in order to address the "dead" ReLU problem. It is defined as $g(z) = \max\{\alpha \cdot z, z\}$ where α is set to a small number, such as 0.01. Two reknown sigmoidal activation functions are the logistic sigmoid (2.1) and hyperbolic tangent (2.2), which are closely related through the $\tanh(z) = 2 \cdot \sigma(2z) - 1$ formula [28].

$$\sigma(z) = \frac{1}{1 + \mathrm{e}^{-z}} \qquad (2.1) \qquad\qquad \tanh(z) = \frac{\mathrm{e}^{z} - \mathrm{e}^{-z}}{\mathrm{e}^{z} + \mathrm{e}^{-z}} \qquad (2.2)$$

The logistic *sigmoid* is usually applied at the final layer to map the output to the $(0, 1)$ range, for example when predicting the probability that an input belongs to a certain class in binary classification problems. However, hyperbolic tangent is relatively more preferable for hidden units in terms of optimization [28], due to its resemblance to the identity function, and it is more common in recurrent networks and autoencoders. The problem with sigmoidal functions resides in their highly saturating behaviour [28] on most of the input range, i.e. both at very positive and very negative input values. Therefore, when using these functions on the output layer, it is recommended to employ a loss function (e.g. negative log-likelihood) that can cancel this saturation behavior [28]. While logistic sigmoid units model Bernoulli distributions, *softmax* [28] units model multinoulli distributions, such as the output of a multi-class classification model. A vector z of length n is transformed by the softmax operation to another vector of the same length, where each component is defined as:

$$\mathrm{softmax}(z)_i = \frac{\mathrm{e}^{z_i}}{\sum_{j=1}^{n} \mathrm{e}^{z_j}}. \qquad (2.3)$$

Figure 2.2 visualizes the four activation functions discussed above.

2.1.2 Learning Paradigms

The most widely applied paradigms for learning of neural networks are the supervised and unsupervised learning. *Supervised learning* refers to the scenarios

when the data is labeled according to the task requirements, whereas *unsupervised learning* implies no available annotations of the data [28]. In contrast, *semi-supervised learning* [28, 18] methods make use of large amount of unlabeled data to learn effective feature representations that are then used to train the model on a smaller number of labeled samples. *Self-supervised learning* [6] entirely relies on unlabeled data, and in this sense can be considered as a special case of *unsupervised learning*. However, while *unsupervised learning* problems include clustering and dimensionality-reduction, *self-supervised learning* aims to make predictions based on the data, for instance in the scope of classification or regression tasks. The video prediction task [77] investigated in this work also fits in the *self-supervised learning* paradigm.

2.1.3 Training of Neural Networks

Neural networks are trained to minimize a so called cost (or loss) function [28], which has the following general form:

$$\mathcal{J}(\Theta) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(f(x^{(i)}; \Theta), y^{(i)}), \tag{2.4}$$

where $\mathcal{L}$ denotes the loss defined for a single data point $(x^{(i)}, y^{(i)})$ with input $x^{(i)}$ and target value $y^{(i)}$, and the final objective function averages $\mathcal{L}$ over the given m training examples. The goal is to find some parameter values Θ^* for the network, so that the cost function $\mathcal{J}(\Theta)$ is minimized, i.e. $\Theta^* = \arg\min \mathcal{J}(\Theta)$.

Stochastic gradient descent [28] (SGD) and its variants are the most popular optimization algorithms used for training of the modern deep neural networks. Similar to the standard Gradient descent (Cauchy [9]), SGD starts from a random initialization of the network parameters, and iteratively updates them according to the gradient estimate of the cost function at the current state. The key difference is that in classical Gradient descent the update step uses the average of the gradients over all training samples (*batch*), whereas SGD uses a random subset of the training data (*minibatch*) at each iteration for gradient estimation.

The update operations of SGD at iteration k are depicted in Figure 2.3. At each step, the gradient of the cost function is estimated by averaging per-sample gradients over m random training samples, and the network parameters are updated along the opposite direction of the estimated gradient vector. The learning rate ε_k, usually decreased over time, is an important hyperparameter controlling size of the parameter updates.

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
end while

Figure 2.3: Stochastic gradient descent update step at iteration k. Figure extracted from Goodfellow et al. [28].

During training, the cost function's value is produced by propagating the input values across the network (*forward propagation*). Afterwards, the gradient of the cost function with respect to the network parameters is computed through a so called *back-propagation* [85] algorithm. It starts from the last layer of the network and recursively applies the chain rule of calculus (2.5), to compute the partial derivatives for all the nodes and weights of the network in backward order [28].

$$z = f(y) = f(g(x)), \text{ then } \frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x} \tag{2.5}$$

The calculated gradients are then used by the optimization algorithm (e.g. SGD) to adjust the weights of the network. The most widely used optimization algorithms, such as RMSProp [35] and Adam [48], employ adaptive learning rate methods.

2.1.4 Regularization

One of the problems with training of deep neural networks is concerned with overfitting on the training data, resulting in poor generalization performance. Different regularization methods [28] have been designed that control the *representational* capacity of the network in order to avoid overfitting. Among the most effective regularization techniques for deep neural networks is the *dropout* (Srivastava et al. [94]). As a rule, dropout is applied to entire layers of the network. At each training iteration, it randomly removes some of that layer's units by setting their output values to zero with some probability. Thus, dropout has the effect of reducing the network capacity during training, moreover, it helps the model to learn several representations of the same input, stopping each of the network units from specializing only on a specific aspect of the input.

Batch normalization (Ioffe et al. [38]) can also be considered as a regularization technique. It standardizes the output of any input or hidden layer as follows. Given a minibatch of data, let matrix $\mathcal{H}$ denote the output activations of the given layer, with each row corresponding to a single data sample. Then batch normalization [28] transforms the activations $\mathcal{H}$ to:

$$\mathcal{H}' = \frac{\mathcal{H} - \mu}{\sigma}, \tag{2.6}$$

where μ and σ vectors are composed of the mean and standard deviation values for each unit's activations over the given minibatch. In practice, during training the model tracks average estimates for the values of μ and σ, and freezes these values at inference time.

Other methods indirectly constrain the *effective* capacity of the network. In particular, in order to avoid overfitting due to a very large number of training iterations, it is common to separate a so called validation set from the training dataset, and stop the training once the loss value on the validation set stops decreasing. This technique is known as *"early stopping"* [28].

2.1.5 Popular Loss Functions

The most widely used loss functions for regression problems include mean-squared error (MSE) and mean-absolute error (MAE) [28]. MSE and MAE measure the mean-squared and mean-absolute deviation, respectively, between the ground-truth $y^{(i)}$ and predicted $\hat{y}^{(i)}$ values over a minibatch of m data points:

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2, \tag{2.7}$$

$$\text{MAE} = \frac{1}{m}\sum_{i=1}^{m}\|\hat{y}^{(i)} - y^{(i)}\|. \tag{2.8}$$

Both functions can be generalized to multidimensional inputs, such as images. For instance, pixel-wise MSE computed on two images produces the average of the squared differences between corresponding pixel values.

For classification problems, it is common to use the cross-entropy [5] (CE) function. Consider the classification problem for K-classes. Given two probability vectors of length K, namely the predicted class-labels $\hat{y}$ and the target label y (normally a one-hot vector), for a specific data sample, the cross-entropy loss is computed as:

$$\text{CE} = -\sum_{k=1}^{K} y_k \cdot \log(\hat{y}_k). \tag{2.9}$$

In case of "hard-labeling" [28] of the samples, i.e. when the ground-truth vectors y are represented as one-hot vectors, the cross-entropy reduces to negative log-likelihood $-\log(\hat{y}_c)$ for the target class c.

Similar to MSE, cross-entropy can also be generalized to high-dimensional inputs. For instance, pixel-wise cross-entropy [11] is a common loss function in semantic segmentation tasks. In such cases, the model normally predicts a probability vector of length K (number of classes) at each pixel location of the image. Pixel-wise cross-entropy computes the standard cross-entropy averaged across all pixels.

2.2 Convolutional Neural Networks

Traditional fully-connected networks densely connect neurons between layers and operate on flat vectors neglecting any spatial information. Hence, they are not effective for tasks such as image recognition. In contrast, convolutional layers are able to extract feature representations that reflect the spatial structure of the input.

The mathematical operation of convolution [28] for an input I and a kernel K (2d arrays) is defined as follows:

$$(\mathrm{I} \star \mathrm{K})(i,j) = \sum_m \sum_n \mathrm{I}(m,n) \cdot \mathrm{K}(i-m, j-n). \tag{2.10}$$

Convolutional layers typically convolve the input with multiple learnable kernels (filters), and are often followed by a non-linear activation function (ReLU) and a pooling operation. The *convolution stride* specifies the number of pixels by which the kernel is shifted at each step when sliding over the image. The pooling [28] operation summarizes the input by sliding a rectangular window over it and extracting a single statistic for each region. The popular pooling functions include max-pooling, average-pooling and L_2-norm. It is common to set the *pooling stride* equal to the size of the pooling window, so that the spatial size of the input is decreased.

In general, convolutional layers have fewer parameters than fully-connected ones, and therefore are more memory efficient and faster to compute [28]. The key advantages of convolutional layers are their sparse and local connectivity, as well as parameter sharing properties [28]. Choosing a kernel size much smaller than the input size provides the sparse connectivity, since each unit of a convolutional layer is computed based on a local region of its input corresponding to the size of the kernel. Moreover, the kernel weights (parameters) are shared across the input,

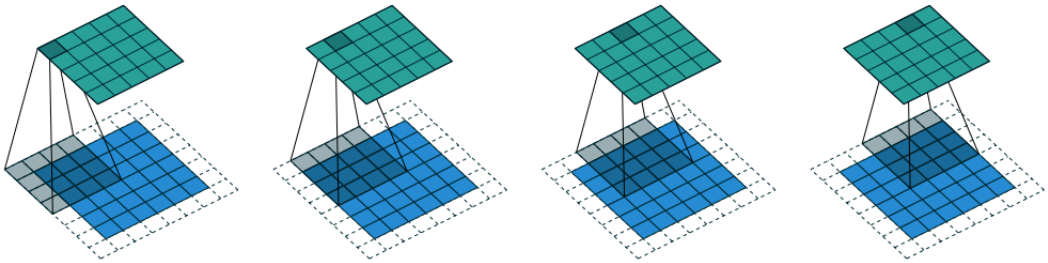


Figure 2.4: Transposed convolution of a 4×4 kernel with a 5×5 input. In contrast to convolution layer, here the padding is applied to the output instead of the input. Figure extracted from Dumoulin et al. [16].

since a single kernel extracts the same type of features from all input locations. This greatly reduces the number of learnable parameters in the layer.

Convolutional neural networks (CNN) usually stack several convolutional and pooling layers, thereby extracting hierarchical features from the data, which is followed by a few fully-connected layers at the end. It is also common to include batch normalization layers [38] between convolutional layers and ReLU activations. CNNs are widely used as feature extractors, e.g. in encoder-decoder architectures.

A CNN typically reduces the spatial size of the input. However, sometimes one needs to increase the size of the constructed feature maps, such as in the decoder of a convolutional autoencoder (Section 2.4). One way is to apply a standard 2d interpolation method, such as bilinear interpolation [80]. A more effective way is to use a learnable layer with *transposed convolution* [62, 16] operation. As mentioned by Dumoulin et al. [16], transposed convolution corresponds to the gradient of simple convolution operation, and vice versa. Figure 2.4 illustrates a few steps of an example transposed convolution operation. At each step, the kernel is scaled by the input value at the current location, and the result is accumulated to the output.

2.3 Recurrent Neural Networks

While CNNs have become very popular in applications including spatial data processing (images and videos), recurrent neural networks [17] (RNN) have shown success in processing of sequential data [28]. RNNs model functions involving a recurrence relation. This is implemented by maintaining a "memory" called the *hidden-state* of the network, which represents an accumulated history of inputs from all the previous time-steps. The most general form of recurrence relation is (2.11), when the hidden-state $h^{(t)}$ at each time-step gets updated based on the

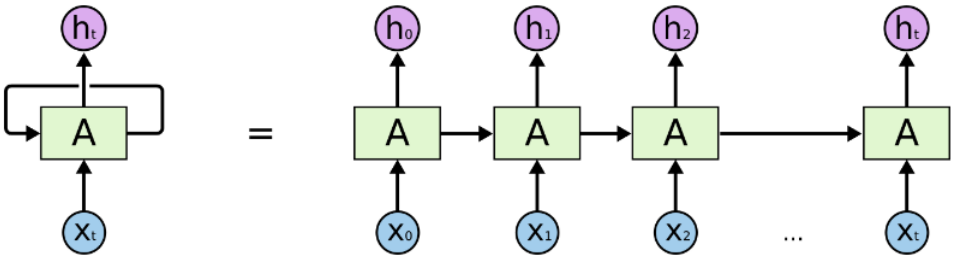


Figure 2.5: Computational graph (left) of an RNN and the visualization of corresponding unrolled graph (right), showing the replicated network across $t + 1$ time-steps. Figure extracted from Olah [75].

previous hidden-state $h^{(t-1)}$ and the current input $x^{(t)}$ [28].

$$h^{(t)} = \mathrm{f}(h^{(t-1)}, x^{(t)}; \Theta) \tag{2.11}$$

The output of the network is then computed based on the current hidden-state and the current input.

RNNs can be categorized into three main classes [28], corresponding to different task scenarios:

- *One-to-many*: The RNN takes a single value as input and maps it to a sequence of output values (e.g. image captioning).
- *Many-to-one*: A sequence of input values is mapped to a single value by the RNN (e.g. sentiment classification, action recognition).
- *Many-to-many*: A sequence is mapped to another sequence by the RNN (e.g. machine translation, video prediction).

The recurrence relation (2.11) allows RNN to operate on a sequence of any length, and to capture the history of inputs in its hidden-state. Moreover, the flexibility of RNN also resides in the fact that a single transition function f is learned for all time steps.

The standard optimization techniques can be applied for training RNNs, with a difference that the backpropagation algorithm, for computing the gradients of the loss function with respect to the network parameters, is applied to the unrolled computational graph. This procedure is called backpropagation through time [28] (BPTT). Since the network parameters are shared across T time-steps of the unrolled computational graph (Figure 2.5), T gradients are computed for each parameter (weight). Then the optimization algorithm updates the model weights with the sum of the changes contributed by individual gradients.

If there is a recurrent connection from the output of the RNN to its hidden-state, then it is common to employ a so called *teacher-forcing* [28] method for training. Instead of feeding the previously predicted outputs back into the network, teacher-forcing uses the target output of the current time-step as the next input. This technique can ease the training process, and hence is widely used in various scenarios, particularly for training of video prediction models. However, in case of open-loop inference mode [28] (current prediction serving as the next input), teacher-forcing is prone to causing performance discrepancy between training and inference time, due to varying input distribution fed to the model. This is the reason why we avoid adopting teacher-forcing in this thesis.

RNNs trying to capture long-term dependencies from the data, inevitably construct computational graphs that become very deep when unrolled. This makes the gradient backpropagation quite challenging, since it includes repeated matrix multiplications (with the same matrix), which is prone to producing *vanishing* or *exploding* gradients [97, 28]. Vanishing gradients cause slower (or no) training convergence, whereas exploding gradients make the learning very unstable. This makes a motivation for this work to design a model that has smaller temporal frequencies (larger RNN processing periods) in upper hierarchy-levels, and is able to make long-term predictions by forecasting for only a few time-steps in different temporal resolutions.

Several techniques have been introduced to mitigate these issues, such as *gradient clipping* [28] (specifying the maximum possible gradient value) to address exploding gradients, as well as various ways of proper weight initialization to avoid vanishing gradients problem. However, the most effective models achieve long-term dependencies by employing special types of recurrent networks, e.g. a long short-term memory (LSTM).

2.3.1 LSTM

Long short-term memory [36, 30, 75] (LSTM) cells are specialized in learning long-term dependencies from data. In addition to maintaining a hidden-state, LSTM has a so called *cell-state*, which interacts with the rest of the cell units through only a few linear operations. This creates a path for smoother gradient flow through the network, alleviating the problems related to gradient backpropagation. The building blocks of the LSTM cell are called the *gates*, which control the information flow of the cell-state. Each gate is composed of a linear (fully-connected) layer followed by a sigmoid activation. Figure 2.6 illustrates the structure of the LSTM cell.

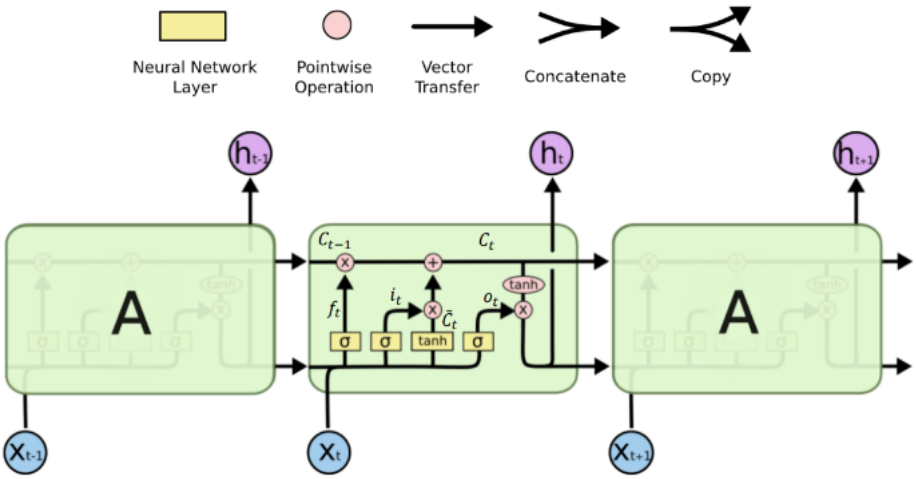


Figure 2.6: Long short-term memory cell structure. The three linear layers with *sigmoid* activation (forget, input and output gates), as well as the linear layer with *tanh* (for creating the candidate values of the next cell-state) are plotted using yellow boxes. The pointwise operations (sum and multiplication) of vectors are shown in pink circles. Figure extracted from Olah [75].

First, the *forget gate* determines the amount of information that should be thrown away from the previous cell-state.

$$f_t = \sigma(x_t \cdot W_{xf} + h_{t-1} \cdot W_{hf} + b_f) \tag{2.12}$$

The new information is then extracted in two stages. The *input gate* decides the amount of new information to be added to the current cell-state (2.13). And a vector of candidate values for the new cell-state is created through another linear layer followed by a *tanh* activation (2.14). Afterwards, the cell-state is updated according to the previously computed information (2.15).

$$i_t = \sigma(x_t \cdot W_{xi} + h_{t-1} \cdot W_{hi} + b_i) \tag{2.13}$$

$$\tilde{C}_t = \tanh(x_t \cdot W_x\tilde{c} + h_{t-1} \cdot W_{h\tilde{c}} + b_{\tilde{c}}) \tag{2.14}$$

$$C_t = f_t \otimes C_{t-1} + i_t \otimes \tilde{C}_t \tag{2.15}$$

Finally, the *output gate* filters the information from the cell-state and determines the output of LSTM cell at the current time-step which in turn serves as the next

hidden-state.

$$o_t = \sigma(x_t \cdot W_{xo} + h_{t-1} \cdot W_{ho} + b_o) \tag{2.16}$$

$$h_t = o_t \otimes \tanh(C_t) \tag{2.17}$$

In the above formulas, f_t, i_t and o_t denote the outputs of the forget, input and output gates respectively. C_t and h_t correspond to the cell-state and the hidden-state of the LSTM. The learnable weights and biases of the cell are denoted by W and b symbols. Finally, the arithmetical operations applied in the above formulas are the matrix multiplication ($\cdot$), pointwise multiplication ($\otimes$) and pointwise sum ($+$).

2.3.2 Convolutional LSTM

The standard LSTM is composed of fully-connected layers, hence it is not beneficial for processing grid-like data, such as images. In contrast, convolutional LSTM [91] (ConvLSTM) replaces the linear layers of the LSTM with convolutional layers. The only change to the formulas (2.12) to (2.17) is that the weights W are exchanged with learnable kernels, and the matrix multiplication ($\cdot$) is replaced with convolution operation ($\star$).

2.4 Autoencoders

Autoencoder [28, 84] (AE) is a special type of neural network that is trained to replicate its input to the output. Autoencoders have many applications, such as dimensionality-reduction and representation-learning among others. The main goal of an AE is to capture an efficient low-dimensional hidden representation that can aid in learning other tasks. Autoencoders are composed of two main building blocks, namely an encoder and a decoder. In a standard *deterministic* AE, the encoder (E) maps the input to a *code* (embedding) h, and the decoder (D) attempts to reconstruct the original input x from h. The loss $\mathcal{L}$ is then minimized so that the reconstruction $\hat{x}$ matches the input x as close as possible:

$$\mathcal{L}(x, \hat{x}) = \mathcal{L}(x, D(E(x))). \tag{2.18}$$

An *undercomplete* AE [28] is a very common type of autoencoder, where the code h has a much smaller dimensionality than the input x. This encourages the AE to extract the most essential features from the data for a particular task. There are several techniques of regularizing an AE, in order to force it to learn a more robust

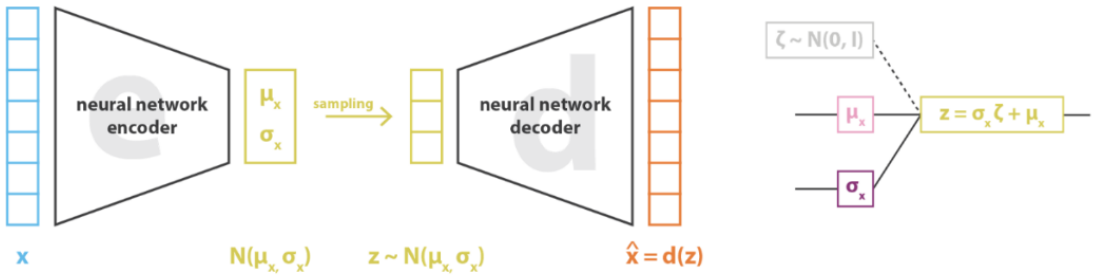


Figure 2.7: Illustration of the structure of a simple Variational Autoencoder (left), and the reparametrization trick used for training of VAEs. Figures extracted from Rocca [84].

hidden representation h. In particular, a *denoising* AE [28] tries to reconstruct the initial input x from its corrupted version $\tilde{x}$, thus achieving robustness to noisy data.

In image processing tasks, the encoder of an AE is typically implemented as a convolutional network, and the decoder is a mirrored version of the encoder with simple upsampling [80] or transposed convolutional [16] layers.

VAE

Typically, the latent space of the standard AE is not organised in a way that enables the decoder to serve as a data *generator*, since there is no guarantee that most of the points in the latent space correspond to meaningful reconstructions. Variational Autoencoder (VAE) [50, 83, 84, 28] resolves this problem by employing explicit regularization during training, to force the construction of a latent space suitable for the generative process.

Instead of computing a single encoding for the given input, VAE estimates a probability distribution $q(z \mid x)$ over the latent space, parameterized as the mean and standard deviation of a Gaussian distribution $\mathcal{N}(\mu_x, \sigma_x)$. During the forward pass through the network, a single sample z is drawn from the predicted distribution and passed through the decoder to compute the final output. Figure 2.7 (left) depicts the structure of a simple VAE.

VAE includes a regularization to enforce the distributions produced by the encoder to be close to a standard normal distribution (*prior*). The final objective function is composed of reconstruction and regularization terms. The regularization term is realized through a Kullback–Leibler divergence [53] (KL-divergence) loss between the predicted $q(z \mid x)$ and the prior $p(z)$ distributions. Thus, the

combined loss function has the following general form:

$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\| + \mathrm{D}_{\mathrm{KL}}(q(z \mid x) \parallel p(z)), \tag{2.19}$$

where $\|\cdot\|$ denotes the reconstruction error.

For the given discrete probability distributions Q and P over the space $\mathcal{X}$, KL-divergence [53] is defined as:

$$\mathrm{D}_{\mathrm{KL}}(Q \parallel P) = -\sum_{x \in \mathcal{X}} Q(x) \cdot \log \frac{P(z)}{Q(x)}. \tag{2.20}$$

An important advantage of predicting Gaussian distributions in VAEs is that the KL-divergence [53] between two Gaussians can be implemented as a closed-form expression [84] based on the means and covariance matrices of the two distributions.

In order to enable the backpropagation algorithm on a VAE, a so called *re-parameterization trick* [49] is used (Figure 2.7, right). The sampling of a random vector z from the produced distribution $\mathcal{N}(\mu_x, \sigma_x)$ presents a stochastic operation (node) in the computational graph of the network. Re-parameterization trick moves this stochastic node out of the path of gradient flow, by first sampling a random vector ξ from standard normal distribution, then calculating the vector z based on ξ and the estimated mean and standard deviation values:

$$\xi \sim \mathcal{N}(0, \mathbf{I}), \quad \text{and} \quad z = \sigma_x \cdot \xi + \mu_x. \tag{2.21}$$

3 Related Work

Video prediction has been extensively studied as a promising avenue for intelligent decision-making systems, due to its representation-learning power, since rich internal representations are required for making coherent predictions. To a certain extent, the interest behind future frame prediction task is attributable to the plethora of unlabeled video data available for learning, which makes it a suitable feature-learning framework for different downstream tasks, in the hope of generalizing to various real-life scenarios. The applications include anomaly detection [61], weather forecasting [91, 92], autonomous driving [63, 102] and semantic video segmentation [43, 73] among others.

3.1 State of the Art

The field of future frame prediction has seen much progress in the last ten years. Many deep-learning based approaches have been designed and investigated to solve this problem. In this section, we review the most relevant prior works. For an extensive review and categorization of deep-learning-based video prediction methods we refer the interested reader to [77].

In 2014, Ranzato et al. [82] introduced a baseline for video prediction inspired by natural language processing models at that time. It applies vector quantization technique and makes patch-level short-term predictions. Srivastava et al. [95] applied a long short-term memory (LSTM) [36] network for predicting future frames based on extracted low dimensional feature representations, which are in turn used for improving classification of human actions through transfer learning. A few other methods, such as PixelRNN/PixelCNN [76] and Video Pixel Network [45], estimate the conditional probability distribution over values of each pixel location in video frames and generate the next frames in a raster scanning manner. Inspired by the biological function of human brain to repeatedly predict and improve its virtual model of the environment based on observations, Lotter et al. [63] designed the Predictive Coding Network (PredNet).

To take into account the spatial similarity between consecutive frames, some of the early approaches re-parameterize the problem to explicitly learn local trans-

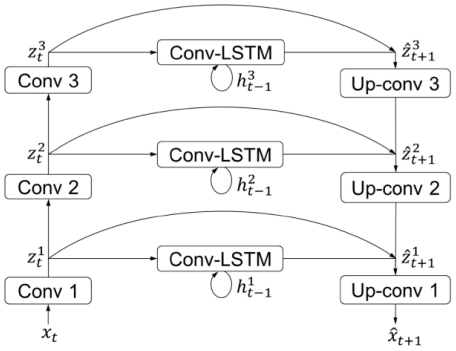


Figure 3.1: Video Ladder Network (VLN) architecture. ConvLSTM modules, processing feature maps from different encoder layers, provide the decoder with a hierarchy of predicted features. Residual connections, descending from different levels of the encoder, are meant to aid in reconstructing the static parts of the scene. Figure extracted from [13].

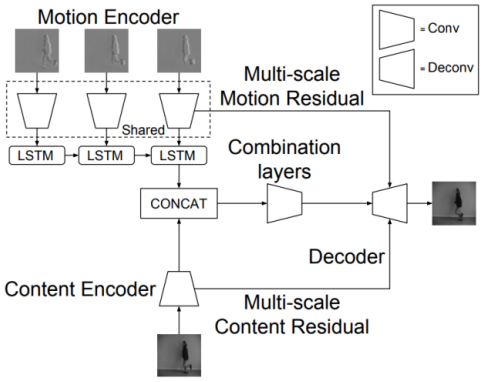


Figure 3.2: MCnet [101] model architecture. The *Content Encoder* creates relevant spatial features from last input frame, whereas the *Motion Encoder* extracts motion component from the sequence of frame differences. The next frame is then predicted by the *Decoder* which receives inputs from both encoders (directly and through skip connections). Figure extracted from [101].

formations rather than directly predicting raw images [40, 78, 25, 65]. Jaderberg et al. [40] design a Spatial Transformer (ST) module to enhance the existing convolutional networks with an estimator of a set of parameters for several affine transformations applied on the input feature maps. Finn et al. [25] propose the Spatial Transformer Predictor (STP) network constructing 2D affine transformations for the next frame computation. A number of works simplify the video prediction problem, and rely on forecasting optical flow fields to capture the changes in the environment [112, 70, 99, 66]. Optical fields present per-pixel 2D displacement vectors that, when applied to an image ("warping"), result in a slightly modified arrangement of the scene [26]. Terwilliger et al. [99] make use of FlowNet [26] to predict semantic segmentation of future frames based on segmented current frame and predicted optical flow for the next frames. Z. Luo et al. [70] develop an encoder-decoder-RNN network to predict optical flow sequence conditioned on RGB-D inputs. More recent works by Farazi et al. [21, 22] perform video prediction by estimating transformations and forecasting future frames in the frequency domain. To achieve long-term stable predictions, Oh et al. [74] and Chiappa et al. [10] assume control actions to be given at each time-step, considering synthetic environments, such as video games. Chiappa et al. [10] develop an

action-conditioned LSTM, which fuses the input action with the current hidden state when processing visual inputs at each time-step.

An extensive line of work in video prediction is based on recurrent networks in combination with convolutional autoencoders, that extract features from the context frames and make predictions in feature space, employing various extensions and modifications of the standard LSTM [36] cell. Our proposed MSPred model can be categorized into this group. The most famous among them are ConvLSTM [91], VLN [13], MCnet [101], PredRNN [107], PredRNN++ [105], TrajGRU [92], SVG [15], HRPAE [19] and PhyDNet [54], several of which are chosen as baselines for the evaluation of our approach presented in Chapter 5. A pioneer work in this line by X. Shi et al. [91] proposes ConvLSTM model for precipitation forecasting, which proved to outperform the existing methods based on optical-flow prediction. Another work by X. Shi et al. [92] presented Trajectory GRU (TrajGRU), which employs location-dependent feature extraction to address precipitation nowcasting. Video Ladder Network (VLN) by Cricri et al. [13] is one of the first successful encoder-decoder networks for video prediction, consisting of a few stacked convolutional layers followed by upsampling layers, along with convolutional long short-term memory [91] (ConvLSTM) blocks serving as lateral connections between different encoder-decoder levels (Figure 3.1). PredRNN by Y. Wang et al. [106] introduces a spatio-temporal recurrent module (ST-LSTM) composed of ConvLSTM cells [91], with a zigzag memory flow delivering information in both horizontal and vertical directions (Figure 3.3). PredRNN++ [105] extends PredRNN, replacing the ConvLSTMs with so called Causal LSTM units, comprising two cascaded memory cells, which increases the expressiveness of the model. Furthermore, the authors enhance the model with so called Gradient Highway Unit (GHU) cells in order to achieve a better gradient flow. Recently, a new version of PredRNN has been published by Y. Wang et al. [107], where they introduce a novel curriculum learning method (reverse scheduled sampling), which encourages the model to capture long-term dependencies by processing a random subsequence of the input at each iteration, and ignoring fewer inputs as training progresses. The Motion-Content network (MCnet) by Villegas et al. [101], illustrated in Figure 3.2, is a pioneer work in making future frame prediction by disentangling visual content from scene dynamics. A more successful recent model PhyDNet by Le Guen et al. [54] makes video prediction by capturing object movement and appearance details separately conditioned on the same feature representations, prior to combining them into the actual future frames through a convolutional decoder.

Video prediction, as a suitable representation-learning framework, can have a variety of useful applications for downstream tasks. A remarkable work by

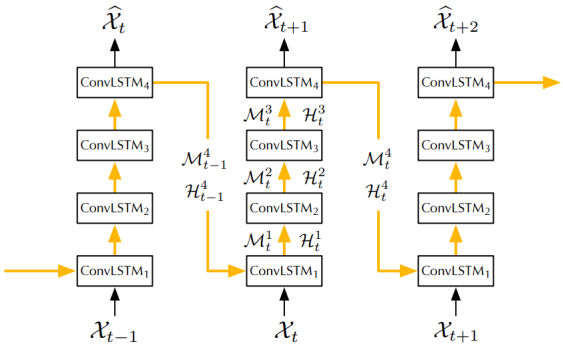


Figure 3.3: Memory flow of Spatiotemporal-LSTM (ST-LSTM) unit. Figure extracted from [107].

Jin et al. [43] leverages the features learned by video prediction, introducing "Parsing with prEdictive feAtuRe Learning" (PEARL) for improved segmentation of video frames. To achieve better temporally coherent semantic segmentation of video frame sequences, the authors [43] fuse the predicted segmentation map, based on the frames up to the current time-step, with pre-segmentation output of the sole current frame. A concurrent work by Nilsson et al. [73] employs a spatial transformer module [40] in a convolutional RNN network [17], and improves video segmentation by assembling features both from previous and future frames.

3.2 Stochastic Video Prediction

Deterministic models dealing with real-life datasets are prone to blurry and low-quality generations due to the inherent uncertainty of dynamic environments. Several approaches have been proposed that integrate variational inference into the recurrent networks in order to model the underlying uncertainty of data.

Babaeizadeh et al. [2] introduce an inference network to approximate the true latent distribution, from which a *time-invariant* latent vector is sampled, trying to model the stochastic properties of the data. Denton et al. [15] adopt a more flexible latent variable network, that produces a different distribution at each time-step, hence learning a more expressive stochastic model. To address the problem of unrealistic generations of existing stochastic models, A. X. Lee et al. [57] combine latent variable networks with adversarial-training, achieving both diverse and more realistic predictions.

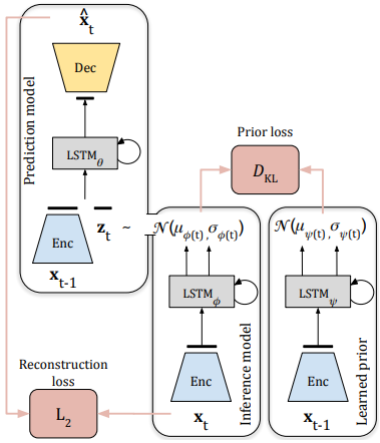


Figure 3.4: SVG-LP [15] model components used during training. The red boxes show the individual loss terms. At inference time, the *Inference* model is used only while processing the context frames, whereas the *Learned-prior* network is applied during the actual prediction stage. Figure extracted from [15].

SVG

The design choices for some components of our MSPred model are highly inspired by the famous Stochastic Video Generation (SVG) architecture, introduced by Denton et al. [15]. Being the first efficient method for stochastic video prediction, SVG serves as a quite simple and strong baseline, as well as a core component for several extensions [60, 102, 108]. The authors propose two models for stochastic video generation, namely SVG-FP (fixed-prior) and SVG-LP (learned-prior, Figure 3.4) variants. The main components of SVG are the convolutional encoder for extracting a feature representation from an input frame, a recurrent predictor and latent variable networks operating on the feature space, and a convolutional decoder for generating the next frame based on the predicted feature vector. All recurrent modules are implemented using vanilla LSTM blocks [36]. SVG represents a probabilistic framework, where the next frame x_t is generated based on the current frame x_{t-1} and a latent sample z_t drawn from a latent distribution. Due to the recurrence of predictor P_Θ, the next frame depends on all previous inputs and latent samples: $\hat{x}_t = P_\Theta(\hat{x}_{1:t-1}, z_{1:t})$.

The design of SVG model adopts some key paradigms used for training of variational autoencoders [50, 83] (VAE, Section 2.4). In order to approximate the real posterior distribution over the latent space, a so called *posterior* (inference) network $q_\Phi(z_t \mid x_{1:t})$, conditioned on the target frames up to time-step t, is employed while training. To prevent the posterior network from learning to simply copy the target frame x_t, without encoding any useful information into z_t, the learned distri-

bution is forced to be close to a *prior* distribution $p(z)$. The outputs of both prior and posterior distributions are parameterized as the mean and standard deviation of a Gaussian distribution $\mathcal{N}(\mu, \sigma)$.

In the SVG-FP model, the prior is a fixed standard normal distribution $\mathcal{N}(0, \mathbf{I})$. Whereas, SVG-LP employs a learned-prior network $p_\Psi(z_t \mid \hat{x}_{1:t-1})$ conditioned on the previous predicted frames (Figure 3.4). Similar to the regularization method of variational autoencoders, the posterior network of SVG is trained to produce distributions close to the prior distributions, through the minimization of a KL-divergence term [53, 15] added to the reconstruction loss (MSE). The combined loss for single data sample has the following form:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left[\|\mathbf{I}_i - \hat{\mathbf{I}}_i\|^2 + \beta \cdot D_{KL}\left(\mathcal{N}(\mu_\Phi^i, \sigma_\Phi^i) \parallel \mathcal{N}(\mu_\Psi^i, \sigma_\Psi^i)\right)\right], \tag{3.1}$$

where the sum is taken over N predicted time-steps. The $\mathbf{I}_i$ and $\hat{\mathbf{I}}_i$ denote the ground-truth and predicted frames, respectively, at prediction time-step i. And the $(\mu_\Phi^i, \sigma_\Phi^i)$ and $(\mu_\Psi^i, \sigma_\Psi^i)$ correspond to the Gaussian distribution parameters produced by the posterior and prior networks respectively. The β coefficient of KL-divergence term in the loss is an important hyperparameter controlling the trade-off between minimizing reconstruction error of the predicted frame and fitting the prior. On the one hand, if β is too small, the posterior network may acquire large enough capacity to copy the target frames resulting in undesirable discrepancy between training and testing performances. On the other hand, in case of very large β values, the latent space might collapse, resulting in a deterministic model.

The SVG model is autoregressive in image space, i.e. the currently predicted frame becomes the input for the next time-step. Moreover, the authors use teacher-forcing [28] method to train the model. Similar to VAE [50, 83], each training step (backpropagation) of SVG is based on a single latent sample z_t. However, SVG applies the prior model for drawing latent samples at inference time (during the actual prediction stage).

3.3 High-Level Structured Prediction

Most of the successful approaches towards long-term video prediction reduce the problem to forecasting in a high-level and more tractable space, such as the structured space of human poses [103, 104, 71, 27] or semantic segmentation maps [68, 96, 67, 58], prior to the actual future frame generation.

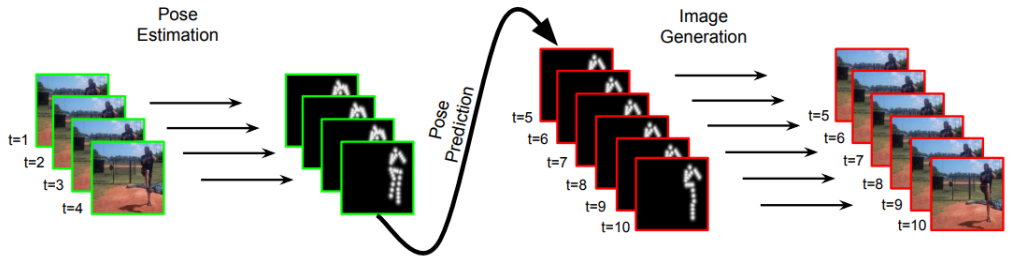


Figure 3.5: Video prediction model by Villegas et al. [103]. Figure extracted from [103].

A series of such methods [103, 104, 27, 58] perform future frame prediction by concatenating multiple models, namely a model for mapping the context frames into some intermediate space, another one for making predictions in this space, and finally the actual future frame generator. However, most of these models are not trained in an end-to-end manner, and are often limited to specific domains. A second line of methods [68, 67, 71, 96] focus on making predictions directly into a high-level space from raw image frames. Nevertheless, these models are constrained to making predictions of specific structure, such as segmentation maps, which can result in a loss of relevant information for other tasks.

One of the first effective methods for long-term future prediction in this line of work has been proposed by Villegas et al. [103], depicted in Figure 3.5. It stacks three models to first make human pose estimation from the context frames, then predict future states of the extracted poses, and finally generate the actual future frames based on the predicted poses and the last observed frame. A similar approach by Fushishita et al. [27] adopts generative adversarial learning (GAN [29]) adding stochasticity to human pose prediction. The model predicts multiple possible sequences of poses, conditioned on not only input poses and a latent noise vector z, but also two additional latent codes reflecting different actions and trajectories, encouraging diversity of the generated poses (Figure 3.6). Another GAN-based method by W. Lee et al. [58] applies a segmentation generator to autoregressively forecast a sequence of future semantic label maps, followed by an image generator that estimates the next frame combining the predicted segmentation map and the previously predicted frame.

A few recent works adopt multitask-learning to simultaneously predict multiple structured outputs within the context of the same task. Jin et al. [42] make the first attempts to jointly predict motion flow together with scene parsing, in order to extract more meaningful spatio-temporal features and aid the main scene parsing task. Hu et al. [37] present a model to simultaneously predict full-frame ego-motion, static scene, and object dynamics on challenging urban driving datasets.

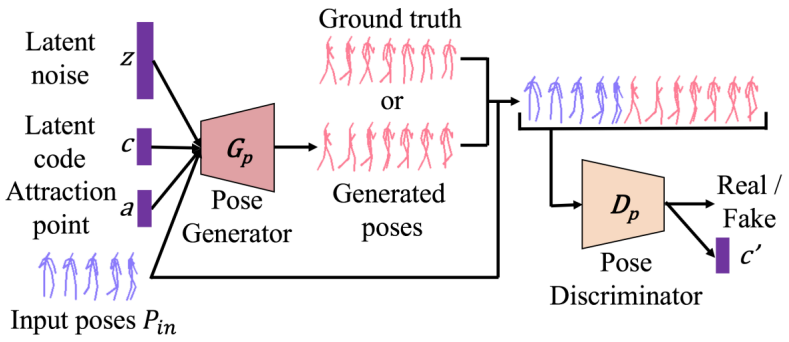


Figure 3.6: Video prediction model by Fushishita et al. [27]. Figure extracted from [27].

3.4 Multiscale/Hierarchical RNNs

Since the introduction of RNNs [17, 36], a number of extensions [34, 8, 51, 12, 47, 87, 60] have been designed that construct temporal hierarchies of recurrent modules, in order to better model the underlying dynamics and uncertainty of the data.

Hihi et al. [34] propose different architectures using several RNNs processing the input at different time-scales to capture long-term dependencies from sequential data. Clockwork RNN [51] (CW-RNN) is a modification of the simple RNN that splits the hidden layer into several sub-modules each processing its inputs at a different temporal granularity level, hence allowing the model to capture dependencies between distant inputs (Figure 3.8). In a similar spirit, Chung et al. [12] propose HM-RNN, a multi-scale recurrent network with different modules operating at distinct clock-rates. Moreover, they develop an adaptive mechanism for learning the specific clock-rate values for the given dataset. Kim et al. [47] propose Hierarchical Recurrent State Space Model (HRSSM), the first model to perform stochastic vector estimation in a hierarchical manner, assigning different "temporal abstraction factors" to each level of recurrent blocks. Inspired by these ideas, Saxena et al. [87] design Clockwork VAE (CW-VAE), a video prediction network constructing a hierarchy of recurrent latent variable models with increasingly coarser time-scales along the hierarchy levels, where each level is conditioned on all the upper ones. Moreover, the authors deal with error accumulation problem in pixel space, by making the model autoregressive in latent space. In order to leverage the full representational power of deep hierarchical latent variable models for real-world datasets, and to overcome the difficulty of training such models, Wu et al. [111] have recently proposed an efficient method for training of hierarchical autoencoders, level-by-level in a greedy fashion.

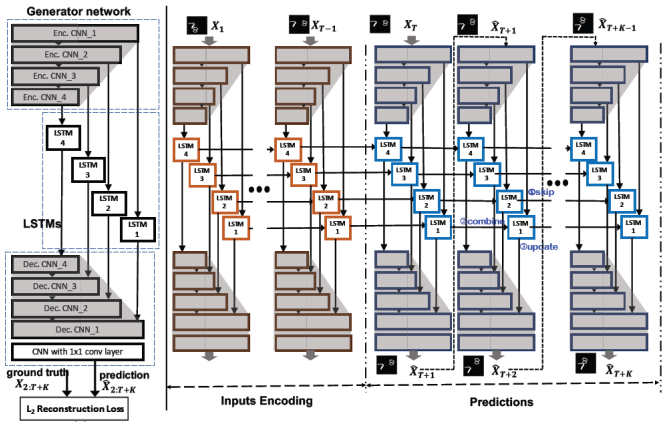


Figure 3.7: HRPAE [19] model architecture. Figure extracted from [19].

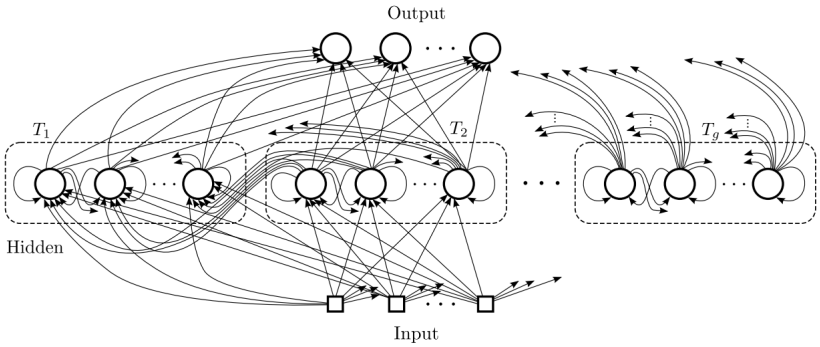


Figure 3.8: Clockwork RNN architecture. Similar to vanilla RNN, CW-RNN consists of input, hidden and output layers. Different clock rates are assigned to different hidden sub-modules, such that each module's update rate is twice as small as the previous one. The units within each block are fully connected, and units in faster modules depend on the ones in slower modules. Figure extracted from [51].

The architecture of our MSPred model mostly resembles Hierarchical Recurrent Predictive Auto-Encoder (HRPAE) proposed by Fan et al. [19], depicted in Figure 3.7. The idea is to simultaneously make predictions in different levels of the feature space, using the hierarchy of features constructed by a CNN. ConvLSTM blocks [91] are bridging between different encoder-decoder layers, providing the decoder with both low-level predicted features helping to reconstruct static scene details, and high-level predicted features capturing moving objects and identities as a whole. In addition, our model employs a hierarchy of recurrent predictor networks along with recurrent latent variable modules, operating at distinct timescales, in order to capture representations at distinct temporal resolutions, inspired by the ideas of CW-VAE [87].

4 Approach

4.1 Architecture

Video prediction [77] is formally defined as the task of generating a possible continuation $\hat{\mathcal{I}} = \hat{\mathbf{I}}_1, \hat{\mathbf{I}}_2, ..., \hat{\mathbf{I}}_N$ of a given sequence of C input (or *context*) frames $\mathcal{C} = \mathbf{C}_1, \mathbf{C}_2, ..., \mathbf{C}_C$. In this thesis, we extend the task of video prediction to forecast future frames along with higher-level representations $(\hat{\mathcal{H}}^1, \hat{\mathcal{H}}^2)$, such as human poses or object locations, conditioned on the same context frames.

In this section, we introduce *MSPred* (**M**ulti-**S**cale Hierarchical **Pred**iction) architecture, depicted in Figure 4.1. MSPred is able to simultaneously forecast future possible outcomes of different abstraction levels at distinct time resolutions, conditioned on the given video frames. More precisely, the model makes predictions at three different abstraction levels, i.e. it forecasts subsequent video frames into immediate future, as well as predicts more abstract representations longer into the future using coarser temporal resolutions.

The design of MSPred in terms of a probabilistic framework follows the famous SVG-LP [15] architecture. The key extension by MSPred resides in its hierarchical predictor module, inspired by HRPAE [19], CW-RNN [51] and CW-VAE [87], which makes predictions at distinct spatial and temporal granularity levels. During the processing stage of the context frames, a convolutional encoder network extracts a hierarchy of feature representations, which initializes the memory of the predictor network. The predicted multi-scale features are then decoded into future image frames and higher-level representations through the convolutional decoders.

4.1.1 Encoder

MSPred utilizes a 2d convolutional encoder for processing the context frames. The encoder network consists of four convolutional blocks, that extract increasingly abstract features of coarser spatial resolution. The three recurrent modules of the predictor network are descending from the second, third and fourth encoder levels, providing features of different abstraction levels to corresponding decoder layers, as shown in Figure 4.1. Low-level features of high spatial-resolution aim to prevent loss of details in the reconstruction process, whereas high-level representations of

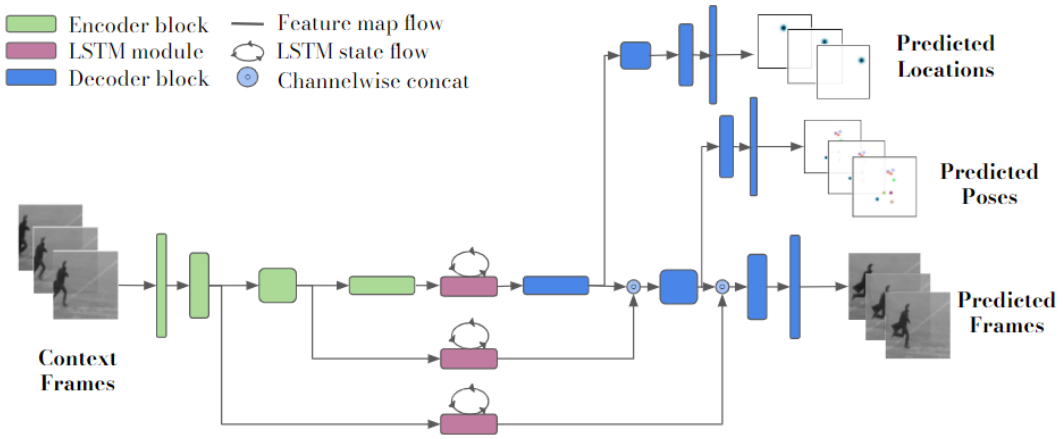


Figure 4.1: MSPred architecture. The encoder blocks are plotted in green, followed by the hierarchy of ConvLSTMs in purple, and the decoder blocks in blue. Decoder heads stemming from different blocks of the main decoder branch predict representations of distinct abstraction levels, such as object poses and locations. The skip connections between different encoder-decoder levels are not shown in order to unclutter the figure.

lower resolution provide the decoder with abstract semantic features. The combination of these features allows the model to achieve feasible future predictions for increasingly complex structures.

4.1.2 Multi-Scale Predictor

Image processing methods often leverage features of different spatial resolution and abstraction levels by extracting hierarchical representations from the input. In similar fashion, in case of video data the information flow can be modeled via a temporal hierarchy. Faster-changing details are represented by lower-level features, whereas slowly-changing information is captured by higher-level representations. MSPred constructs such a temporal hierarchy of features through a predictor network consisting of three recurrent modules (Figure 4.1) processing the input at distinct time-scales, i.e. at processing periods of T_0, T_1 and T_2, such that upper levels operate at coarser temporal resolutions ($T_0 < T_1 < T_2$). In between two processing time-steps, each RNN module simply copies forward its latest predicted output.

As depicted in Figure 4.2, the input to each recurrent module is a feature representation extracted by the corresponding encoder level. The recurrent network of the lowest-level, with fixed processing period of $T_0 = 1$, receives low-level feature representations of high spatial resolution, which correspond to fine-grained details

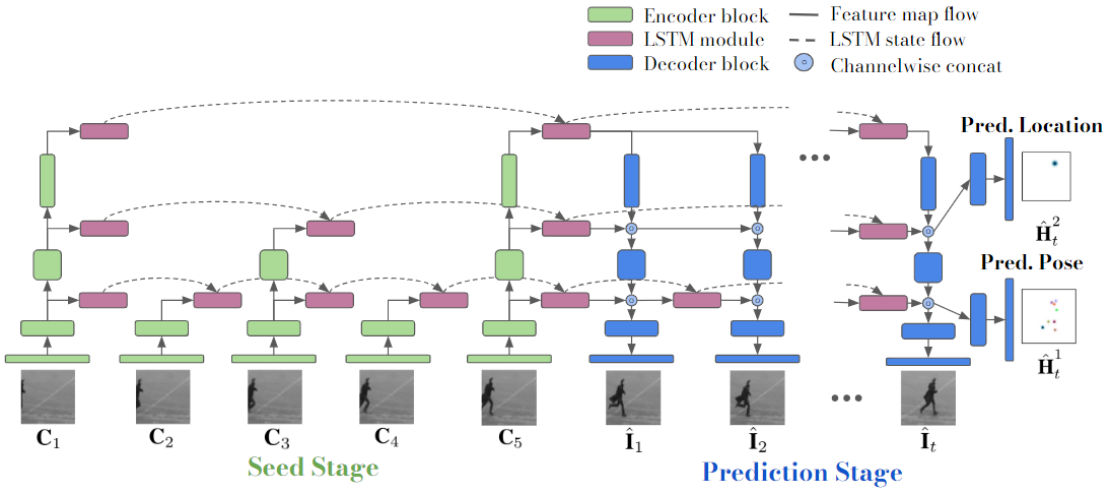


Figure 4.2: Illustration of the MSPred model flow. The *Seed Stage* shows the encoding and processing of the context frames. In this example, LSTM modules are operating at time-periods of $T_0 = 1$, $T_1 = 2$ and $T_2 = 4$. The LSTM-flow is indicated with dot-lines. The decoder blocks come into action only during the *Prediction Stage*. The middle and upper-level decoder heads are displayed only at the last time-step, as well as the skip connections are not plotted, to unclutter the visualization. Figure reused and modified from Karapetyan et al. [46].

that can differ highly across successive frames, such as texture or edges. The second level's recurrent network processes feature maps of smaller spatial resolution, that represent more abstract and slowly changing information. Thus, it is assigned a larger temporal period of T_1. Finally, the upper-most RNN receives abstract representations of even coarser spatial resolution, which contain high-level features that remain relatively constant and can be predicted over longer time intervals. Therefore, the highest level processes the input at only one out of T_2 time steps.

The design of such a hierarchy of RNNs combines ideas from several architectures, such as HRPAE [19], CW-RNN [51] and CW-VAE [87]. This hierarchical design helps our model to capture features varying at different time-scales by disentangling temporal information flow into three levels. Each level of recurrent predictors learns to make predictions at a different time-scale by processing feature encodings at its own temporal stride. Furthermore, the larger temporal abstraction factors in upper levels allow the model to predict high-level features far into the future using only a handful of RNN iterations, thus mitigating the error accumulation problem inherent to long-term prediction scenarios.

4.1.3 Decoder

Each branch of the convolutional decoder network resembles a mirrored version of the encoder, with transposed convolutional [16] layers or simple 2d interpolation methods [80] (bilinear, nearest-neighbor) for upsampling the input feature maps. At each decoder stage, the representations decoded by the previous decoder block are channel-wise concatenated with the predicted feature maps from the corresponding recurrent network. In addition, similar to SVG [15] model, MSPred includes residual connections directly connecting the encoder blocks with their corresponding decoder counterparts, and providing feature encodings of the last context frame. These feature maps get added to the predicted feature maps of respective RNN levels, in order to aid in reconstruction of the static objects and background.

As depicted in Figure 4.1, MSPred employs three different decoder heads each corresponding to one predictor level, in order to forecast representations of different abstraction levels. Each decoder head leverages the most recent predicted feature maps from the current and all higher levels. For instance, generating pixel-level detailed predictions requires both low-level information, such as object texture and shape, as well as high-level representations, which capture dynamic and semantic knowledge from video data.

As upper levels generate actual predictions less frequently, the predicted feature maps of each RNN are re-used by lower levels until the next operating time-step. The decoder of the lowest level generates an output at each time step, while the upper decoder heads clone forward the previously generated outputs according to the respective RNN ticking periods. More precisely, the main decoder branch uses the most recent predicted features of all levels of the RNN hierarchy in order to predict the next video frame. The mid-level decoder head produces abstract representations, such as object keypoints or semantic maps, at each T_1 time-step by indirectly utilizing the most recent feature maps of the middle and highest levels. Finally, the decoder head of the uppermost level generates even more abstract representations, such as object locations, every T_2 time steps, using only the predicted features from the highest RNN module.

4.1.4 Stochastic Components

To account for the stochasticity of real-world datasets and to give our model the ability to make diverse and sharper predictions, we employ similar approach to SVG-LP model [15], described in Section 3.2.

SVG-LP makes predictions based on the feature embeddings of the observed frames and additional latent vectors, sampled from a latent distribution model at each time-step. SVG-LP trains a recurrent *posterior* network q_Φ, conditioned on the previous target frames, to approximate the real posterior distribution over the latent space. Another recurrent *learned-prior* model p_Ψ is trained along with the posterior model, to regularize the training process. The learned-prior model takes as input the previously seen or predicted features and is trained to output distributions close to those output by the posterior model. The learned-prior model is then applied at inference time. The output of each of these stochastic models is parameterized as the mean and standard deviation of a Gaussian distribution $\mathcal{N}(\mu, \sigma)$, from which latent vectors are sampled. A KL-divergence [53] error term, between the produced posterior and prior distributions, is added to the reconstruction loss.

Although we adopt a very similar approach to SVG-LP, our method employs a hierarchy of latent distribution models, i.e. different recurrent modules for each level. This is motivated by the presence of different prediction levels in our model, as well as inspired by the success of several existing approaches towards more expressive latent distribution modeling [34, 8, 51, 12, 47, 87, 60]. Separate recurrent posterior q^{Φ_h} and prior p^{Ψ_h} networks are trained to capture the latent variable distribution for each abstraction level. Each pair of latent variable models takes as input the same feature map as the recurrent predictor network of respective level. Similar to SVG, the training procedure adopts the re-parameterization trick [49] used for learning of variational autoencoders [50, 83], and draws a single set of latent samples at each iteration [15].

4.2 Model Inference

The data-flow of MSPred model is illustrated in Figure 4.2. First, the context frames are processed by the CNN encoder and the embeddings are fed into the respective recurrent modules. These recurrent networks, operating at distinct time-periods, make predictions in different abstraction levels. Finally, the multi-branch convolutional decoder receives the predicted pyramid of features as input to its different stages, and makes the future frame and high-level structured predictions. While image frames are forecasted at each time-step, more abstract future representations are predicted at coarser time-scales, i.e. according to the clock-rates of the respective recurrent predictor modules.

Although the main components of MSPred model, especially the stochastic ones, follow the SVG-LP [15] architecture, the key differences reside in the hierarchical predictor structure, the auxiliary decoder heads and the autoregressive flow used

for inference. SVG is autoregressive in pixel-space, i.e. the currently predicted frame is fed back to the encoder in order to predict the next frame. Since we only aim to achieve feasible image-level predictions for a quite short time span, we employ a more flexible variant of autoregressive flow in our model, similar to [87]. During the prediction stage (Figure 4.2 right), instead of encoding the predicted image frame and using it as the next input, our model operates autoregressively in *feature-space*, i.e. it uses the outputs of the recurrent networks at each level as respective inputs in the subsequent time-step. First of all, this alleviates the fast error accumulation problem typical of pixel-autoregressive models, as well as decreases the overall computational complexity, since the predicted images do not need to be re-encoded. Moreover, it enables a hierarchical design where each level's output is independent of the generation process of lower levels. In particular, it allows MSPred to efficiently make abstract high-level predictions long into the future, with no need for referring to detailed frame predictions. Therefore, by specifying the same number of RNN iterations, i.e. actual number of outcomes for all hierarchy levels, we obtain predictions of distinct time-horizons, due to increasingly coarser time resolutions upper along the hierarchy levels.

At inference time, the *learned-prior* models at each level are used to sample the latent samples, that get concatenated to feature embeddings as additional inputs for respective predictor RNNs. Moreover, we employ ancestral-sampling method leveraged by Castrejon et al. [8], to sequentially sample the latent input of each level conditioned on not only the given feature maps, but also on all the sampled vectors from levels above. We use bilinear interpolation to upsample the latent tensors drawn from upper level modules, in order to apply channel-wise concatenation with the input feature maps.

4.3 Model Training

We trained and evaluated the model on three different datasets, each dictating its specifications reflected on the dimensionality of high-level representations, and the respective loss function terms. In all cases we use three channels for the input and output image frames and downsample the original input images if necessary. For all three datasets, we generate multichannel heatmaps for the upper-level structured predictions. The number of channels of the heatmaps vary across the datasets and specific tasks. In all experiments, spatial dimensions of the heatmaps are specified to correspond to the shape of predicted image frames, although a relatively smaller heatmap size would also be acceptable.

We evaluate our model's ability to predict object keypoints and semantic segmentation maps for different use cases. In case of predicting object keypoints, we generate tensors of shape $H \times W \times C$, using a separate channel per keypoint, and construct each heatmap ($H \times W$) by adding a 2d Gaussian kernel with fixed standard deviation (σ) around the ground-truth pixel location [110, 72]. Given a predicted heatmap for a specific joint or keypoint location, we extract the final pixel position by taking the one with maximum value on the heatmap. For predicting semantic segmentation maps, we specify one output channel per object category present in the dataset. At each location, we predict the probability that a pixel belongs to the class presented by the respective channel. For each pixel location, we then assign the class value corresponding to the highest predicted probability across all channels.

The loss function used for training our MSPred model has the following general form:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left[\mathcal{L}_0^i + \alpha_1 \cdot \mathcal{L}_1^i + \alpha_2 \cdot \mathcal{L}_2^i + \beta \cdot \mathcal{L}_{\text{KL}}^i\right], \tag{4.1}$$

where the sum is taken over the N prediction time-steps. The first three terms represent the objective function for each hierarchy-level, and are implemented as mean-squared error (MSE) loss (unless specified otherwise) between corresponding predicted and ground-truth frames or heatmaps:

$$\mathcal{L}_0^i = \|\mathbf{I}_i - \hat{\mathbf{I}}_i\|^2, \tag{4.2}$$

$$\mathcal{L}_1^i = \|\mathbf{H}_{T_1 i}^1 - \hat{\mathbf{H}}_{T_1 i}^1\|^2, \tag{4.3}$$

$$\mathcal{L}_2^i = \|\mathbf{H}_{T_2 i}^2 - \hat{\mathbf{H}}_{T_2 i}^2\|^2. \tag{4.4}$$

The last (regularization) term corresponds to the KL-divergence error between the outputs of posterior q^{Φ_h} and prior p^{Ψ_h} distributions, computed by averaging the individual KL-divergence errors across the three hierarchy-levels:

$$\mathcal{L}_{\text{KL}}^i = \frac{1}{3}\sum_{h=0}^{2} \mathrm{D}_{\text{KL}}\left(\mathcal{N}(\mu_{\Phi_h}^{T_h i}, \sigma_{\Phi_h}^{T_h i}) \parallel \mathcal{N}(\mu_{\Psi_h}^{T_h i}, \sigma_{\Psi_h}^{T_h i})\right). \tag{4.5}$$

Notably, for prediction time-step i, the KL-loss for level h is computed on the distributions produced by the latent models of the respective level, at time-step $T_h i$ (with $T_0 = 1$), similar to the corresponding reconstruction loss term.

The coefficients α_1 and α_2 of the second and third loss terms (4.1) are meant to keep the balance between optimizing predictions of different hierarchy levels. KL-divergence loss term is weighted by another hyper-parameter β, controlling the trade-off between minimizing the overall reconstruction error of predictions and fitting the prior.

Moreover, as discussed in Section 4.1.4, the posterior models are used at training time to learn the hierarchy of prior models to be applied at inference time. The posterior models take as input the feature map embeddings of target frames, whereas the prior models are conditioned on the predicted features.

4.4 Implementation Details

We implemented the proposed MSPred architecture in Python using the PyTorch deep-learning library. We run our experiments on an NVIDIA RTX A6000 GPU with 48 Gigabytes of memory.

The encoder and decoder are implemented following the SVG [15] baseline architecture to achieve a more fair comparison. For Moving-MNIST dataset, we follow the DCGAN [81] discriminator and generator architectures, whereas in case of KTH-Actions [88] and SynPickVP [46] datasets we employ VGG16-based [93] encoders and decoders. The DCGAN-based encoder is composed of four blocks, each containing a 4×4 convolutional layer with stride of two, followed by batch normalization and a Leaky ReLU activation with $\alpha = 0.2$ (Section 2.1.1). Each encoder block decreases the spatial resolution of its input by a factor of two. The corresponding decoder reverses the encoder layers, replacing each convolution operation by a 4×4 transposed convolution [16], to upsample the input feature map by a factor of two. The VGG-based encoder follows a similar architecture, however each block contains two convolutional layers with kernel size of 3×3 and stride of one, and downsamples the feature maps by a factor of two using max pooling. Moreover, the respective decoder almost replicates the encoder's structure, applying nearest-neighbor upsampling [80] by a factor of two, after each of its four blocks. Both DCGAN- and VGG-based encoders have a *tanh* activation at their last layer, whereas the decoders have a *sigmoid* activation at their output layer.

In contrast to SVG [15] baseline, we implement all recurrent blocks, i.e. the hierarchical predictor and latent variable modules, using ConvLSTM [91] cells. The main predictor blocks stack four ConvLSTMs per hierarchy level, each with 128 kernels of size 3×3, whereas the latent variable models have two ConvLSTMs, with 64 kernels of size 3×3.

For Moving-MNIST [95] and KTH-Actions [88] datasets, we set the processing periods upper-levels of our model to $T_1 = 4$ and $T_2 = 8$, whereas for the more complex SynPickVP dataset, we choose periods of $T_1 = 2$ and $T_2 = 4$ to also account for the lack of long sequences in the dataset. The number of context frames is chosen as $C = 17$ for Moving-MNIST and $C = 9$ for SynPickVP dataset, making sure that all predictor levels receive at least three context frames. However, we experimentally prove that context size of $C = 9$ is enough for KTH-Actions dataset, even though the upper-most level receives only two context frames.

We train our model using the Adam optimizer [48] with an initial learning rate of 10^{-4} (unless specified otherwise), which is decayed by a factor of 0.1 once during training. The models are trained to predict for $N = 5$ RNN iterations at each hierarchy level. Thus in case of inference with context size of $C = 9$, and periods of $T_0 = 1$, $T_1 = 4$ and $T_2 = 8$, we actually predict the image frames for time-steps $\mathrm{t} = 10, 11, ..., 14$, and high-level structured predictions for $\mathrm{t} = 13, 17, ..., 29$ and $\mathrm{t} = 17, 25, ..., 49$ for the middle and upper levels respectively.

5 Evaluation and Results

In this chapter, we discuss the experimental details, as well as the results obtained on three different datasets. Section 5.1 describes the datasets used for the evaluation, as well as the generation of high-level structured representations needed for training. Section 5.2 summarizes the evaluation metrics employed for assessing the predictions made by our model. Next, Section 5.3 illustrates some quantitative and qualitative results achieved by our model, as well as shows a comparative analysis against some existing methods. Finally, Section 5.4 concludes the chapter with an ablation study and an in-depth analysis of the architectural design choices of our model.

5.1 Datasets

5.1.1 Moving-MNIST

Moving-MNIST [95] (M-MNIST) is a popular synthetic dataset for video prediction, typically used for proof of concept (PoC) purposes. It consists of sequences depicting two moving handwritten digits from the MNIST [56] dataset, located at some random positions in a 64×64 image, moving with constant speed, and bouncing off the image boundaries in a deterministic manner, overlapping from time to time. The training sequences are generated on the fly, by randomly sampling two digit images from MNIST [56], as well as their initial locations and velocity vectors. Thus we have potentially infinite number of sequences of any length. Moreover, we evaluate the model on a pre-generated test set of $10,000$ sequences.

In our experiments, we treat M-MNIST frames as RGB images, i.e. repeating the MNIST digits for the RGB channels. For the high-level representations, we predict omnichannel heatmaps with Gaussian blobs centered at the digit locations. Due to creating the dataset on the fly, it is straightforward to generate the ground-truth heatmaps with available digit positions. We choose $\sigma = 2$ and $\sigma = 5$ for the Gaussian blobs in the mid- and high-level ground-truth heatmaps respectively, assuming that a more concentrated blob around a moving object changes faster.

5.1.2 KTH-Actions

KTH-Actions [88] (KTH) is a popular video prediction and action recognition benchmark dataset consisting of real videos of humans performing one of the six basic actions, namely *boxing*, *handclapping*, *handwaving*, *walking*, *running* and *jogging*. The dataset includes 600 videos containing over 290k frames split into 2400 sequences. There are 25 different actors performing the actions in a variety of indoor and outdoor scenes. Video sequences with 16 out of 25 actors are chosen for training and the other 9 for testing data.

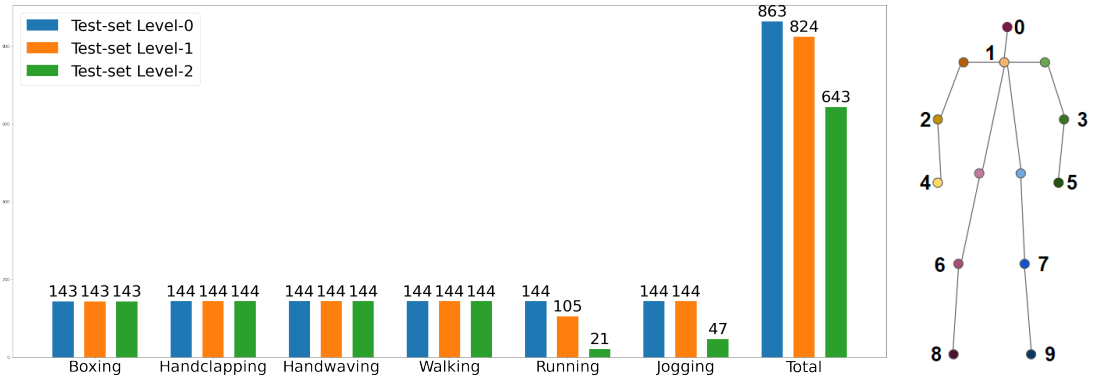
We use a resolution of 64×64 for the input frames. We train our model on a subset of the original training dataset, choosing 1466 sequences with length of at least 29 (number of frames required by the mid-level), and for sequences shorter than 49 (number of frames required by the high-level), we extend the target outputs for upper-level to repeat the representations of the last available frame (only for "moving" classes like *walking*).

In our experiments, we predict nine body-keypoints (joints 0, and 2-9 of Figure 5.1b) in the mid-level, and a single keypoint for the "center of mass" of the moving person (joint 1 of Figure 5.1b) in the upper-level. We choose the body-joints that are relatively more challenging in the context of video prediction. We generate the ground-truth keypoints using a pre-trained OpenPose [7] model for human pose estimation.

We perform the evaluation of predicted outputs of different levels using different subsets of the original KTH test-set, to account for the difference in the required sequence lengths for each of the levels and to get as large test-sets of class-balanced data as possible. Figure 5.1a shows the number of sequences of each class, present in each subset of the original test set used for evaluating the predictions of corresponding hierarchy level.

5.1.3 SynPickVP

We also evaluate our model on SynPickVP [46], a new synthetic video prediction dataset, consisting of videos of various bin-picking scenarios in a simulated environment, where a suction-cap gripper robot moves in arbitrary directions in a box of cluttered objects (21 categories). The dataset is generated by selecting sequences from the recently proposed SynPick [79] dataset, rendered with the Stillleben [89] engine. We use 1260 training and 200 evaluation sequences. This is a challenging video prediction benchmark, since the model needs to capture the motion of the robotic gripper, as well as predict the future arrangement of displaced objects, while still representing a cluttered complex background.



(a) (b)

Figure 5.1: (a) Statistics for the number of per-class sequences in KTH dataset, present in each subset of the test-set, used for evaluating the respective hierarchy level's predictions. We only use sequences with more than 14, 29 and 49 frames for the low-, mid-, and high-levels respectively. **(b)** The nine body-joints (numbered) predicted by the mid- and high-levels of MSPred on KTH dataset.

We evaluate MSPred on SynPickVP dataset, training the model to predict image frames on the low-level, semantic segmentation maps on the mid-level and a single-keypoint heatmap for the robotic gripper position on the upper-level. These annotations are inherently available due to the synthetic nature of the dataset. We downsample the resolution of the original frames to 64×112 before feeding them into the model. Moreover, we sample every second frame from the original videos for our experiments.

5.2 Metrics

We employ a number of metrics designed for different tasks in order to evaluate the predictions of distinct abstraction levels individually. In particular we compute several popular metrics for future frame prediction, such as SSIM [109] and LPIPS [115], measuring visual similarity between the predicted and ground-truth frames. Furthermore, we employ some metrics for evaluating the ability of our model to make high-level structured predictions (object pose estimation and semantic segmentation) into the future.

Since MSPred is a stochastic video prediction method, we perform the evaluation following a famous protocol [15] used for probabilistic approaches. To evaluate the performance for a given metric, we sample 20 random latent vectors (a latent tensor per level, per time-step) for each context sequence, and consider the results

for the best match with the ground-truth. For all metrics, we average the results across all $N = 5$ predicted time-steps.

5.2.1 Image Similarity Metrics

We evaluate our model in terms of future frame prediction using four popular metrics. In particular we employ mean-squared error [28] (MSE), structural similarity index measure [109] (SSIM), peak signal-to-noise ratio [23] (PSNR) and learned perceptual image patch similarity [115] (LPIPS). MSE, PSNR and SSIM measure pixel or statistical similarity between predicted and target images. Although these metrics are widely used for assessing the performance of various tasks, they match poorly with the human visual perception system. These metrics favor blurry predictions over more detailed, but imperfect, generations [115]. Moreover, these metrics are prone to misleading good performance in case of perfectly matched background pixels [77]. In contrast, the LPIPS metric makes use of deep visual representations of a pre-trained CNN for comparison and has been proven to correlate well with human perception [115].

MSE / MAE

As discussed in Section 2.1.5, mean-squared error (MSE) between the predicted $\hat{\mathbf{I}}$ and target $\mathbf{I}$ 3d tensors, such as images or heatmaps, is computed as:

$$\text{MSE} = \frac{1}{M}\sum_{i=1}^{M}(\hat{\mathbf{I}}_i - \mathbf{I}_i)^2, \tag{5.1}$$

where M denotes the number of elements in each of the tensors, and i spans all the dimensions.

Similarly, mean-absolute error (MAE) is calculated as:

$$\text{MAE} = \frac{1}{M}\sum_{i=1}^{\text{M}}\|\hat{\mathbf{I}}_i - \mathbf{I}_i\|, \tag{5.2}$$

SSIM

Structural similarity index measure [109] (SSIM) estimates the visual resemblance of two images, operating on extracted patches rather than pixels. It ranges from –1 being very dissimilar, to 1 denoting identical images. We adjust the range of computed SSIM values to $[0, 1]$ in this work. SSIM performs the comparison

between image patches by extracting three useful features, namely *luminance*, *contrast* and *structure*, and combining the corresponding similarity scores.

The SSIM score between two image patches x and y is computed by the following formula:

$$\text{SSIM} = \frac{(2\mu_x\mu_y + \text{C}_1)(2\sigma_{xy} + \text{C}_2)}{(\mu_x^2\mu_y^2 + \text{C}_1)(\sigma_x^2\sigma_y^2 + \text{C}_2))}, \tag{5.3}$$

where μ_x (μ_y) and σ_x^2 (σ_y^2) denote the mean and variance of x (y) respectively, and σ_{xy} denotes the covariance of x and y. We set the parameter values as $\text{C}_1 = 10^{-4}$ and $\text{C}_2 = 9 \cdot 10^{-4}$. Moreover, we choose a sliding window of size 11×11 and compute the average score over all local statistics.

PSNR

Peak signal-to-noise ratio [23] (PSNR) originates from signal processing area, and assesses the similarity between two signals, helping to detect distortions of the received signal from the original one. PSNR is defined by the following formula:

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{MAX^2}{\text{MSE}}\right) \tag{5.4}$$

As shown by the formula, PSNR is based on the inverse of MSE between the inputs, and is scaled by the maximum possible value (MAX) of the signal. For instance, in case of gray-scale images with values between 0 and 255, PSNR is scaled by $MAX = 255$ [23]. Thus, the larger values of PSNR metric indicate higher similarity between images.

LPIPS

Learned perceptual image patch similarity [115] (LPIPS) metric measures the distance between two images by leveraging high-level feature map embeddings of a pre-trained CNN, such as VGG [93] or AlexNet [52]. In particular, we choose AlexNet [52] trained on ImageNet [14] classification dataset as the feature extractor for computing LPIPS in our evaluation process. In contrast to SSIM and PSNR, a smaller LPIPS value indicates higher similarity between the given images. As mentioned by Oprea et al. [77], probabilistic models tend to achieve higher LPIPS metric values due to sharper and more feasible predictions, in spite of deviating from the given ground-truth.

5.2.2 Pose Estimation Metrics

In order to assess our model's performance for body-keypoint prediction on KTH dataset (mid-level), we employ mean per joint position error (MPJPE), percentage of detected joints (PDJ), and percentage of correct keypoints (PCK) metrics, widely used in the relevant literature [32, 4, 39, 7, 1, 114, 100]. All of these metrics oparate on multi-keypoint poses, assessing the similarity between two sets of matching points. Therefore, given a predicted heatmap for a specific joint location, we extract a single pixel position by taking the location with maximum value on the heatmap, provided that the maximum value exceeds a certain threshold $\min_{\text{conf}}$. We set the threshold value $\min_{\text{conf}} = 0.03$, through empirical validation.

In addition, for assessing the single keypoint prediction on both KTH and SynpickVP datasets (upper-level), we measure a so called position error (PE) metric, simply computing Euclidean distance between predicted and ground-truth 2d points.

Mean Per Joint Position Error

Mean per joint position error (MPJPE) is estimated by calculating the mean Euclidean distance between corresponding 2d keypoint locations from the extracted predicted and ground-truth heatmaps. In action prediction literature, MPJPE is computed on two poses after aligning them based on a root joint. However, we do not align any keypoints since in case of future prediction, it is crucial to not only estimate relative arrangement of the keypoints with respect to each other, but also to accurately capture the overall translation of the pose keypoints.

Percentage of Detected Joints

Percentage of detected joints (PDJ) between predicted and ground-truth poses measures the fraction of the joints in ground-truth pose that are correctly estimated. A predicted joint is marked as a correct detection if its distance from the respective target keypoint does not exceed a certain threshold. This threshold is usually chosen as a specific fraction of the human torso diameter for the given ground-truth pose [86]. However, since we only predict nine joints, we choose this threshold as a certain fraction Θ of the ground-truth person's height. In both cases, choosing a threshold proportional to the person's body size makes PDJ metric to be invariant to varying scales of people throughout the dataset. We compute PDJ for different Θ values, and denote by $\text{PDJ}_{\text{H}}@\Theta$ the corresponding metric, where Θ takes values from $0.1, 0.2, ..., 0.5$.

Percentage of Correct Keypoints

PDJ is analogous to the *Recall* measure used for evaluating classification accuracy. Therefore, we also employ the percentage of correct keypoints (PCK) metric, which plays the role of *Precision* in the same analogy. PCK measures the fraction of predicted joints that match the ground-truth. A keypoint is marked as correctly detected if the distance between corresponding predicted and ground-truth joints is within a certain threshold. Similar to PDJ, we choose a threshold equal to the person's height scaled with a certain factor Θ, and compute $\text{PCK}_\text{H}@\Theta$ for different factors Θ.

We also compute summary statistics for the PDJ and PCK metrics over a range of Θ values. We evaluate so called Average Precision ($\text{AP}_\text{H}@.1:.5$) and Average Recall ($\text{AR}_\text{H}@.1:.5$) as the mean values of PCK and PDJ metrics, respectively, computed for a set of Θ values $(0.1, 0.2, ..., 0.5)$.

5.2.3 Semantic Segmentation Metrics

We predict semantic segmentation maps at the mid-level of our model for SynpickVP dataset. We generate output tensors of shape $\text{H} \times \text{W} \times \text{C}$ with a separate channel for each object category, and compute the final segmentation map as follows. For each pixel location, we assign the class label corresponding to the highest predicted probability across all classes (channels).

For evaluation of our model, we measure semantic segmentation metrics such as pixel accuracy, mean per-class accuracy and mean per-class IoU [62, 24, 90].

Pixel Accuracy

Pixel accuracy [62] measures the fraction of pixels in the image that are correctly classified. This is a very intuitive, though a naive metric, since it yields to misleading high values if there are prevalent classes that are easily segmented. To account for the class imbalance problem, especially the prevalence of the background segments (SynPickVP dataset), we also evaluate pixel accuracy per-class and the mean pixel accuracy over all classes (*mClsAcc*).

Pixel accuracy for the given predicted $\mathcal{P}$ and target $\mathcal{T}$ segmentation maps can be defined by the following formula:

$$\text{PixelAccuracy} = \frac{\left|\left\{p \mid \mathcal{P}[p] = \mathcal{T}[p]\right\}\right|}{\text{H} \cdot \text{W}} \tag{5.5}$$

where H and W are the height and width of the maps, and $\mathcal{M}[p]$ denotes the value of pixel p on segmentation map $\mathcal{M}$.

Similarly, pixel accuracy for class $\mathcal{C}$ is defined as:

$$\text{PixelAccuracy}(\mathcal{C}) = \frac{\left|\{p \mid \mathcal{P}[p] = \mathcal{C}\} \cap \{p \mid \mathcal{T}[p] = \mathcal{C}\}\right|}{\left|\{p \mid \mathcal{T}[p] = \mathcal{C}\}\right|} \tag{5.6}$$

Jaccard Index (IoU)

A more popular metric for semantic segmentation, addressing the class imbalance problem, is the intersection-over-union [24, 90] (or Jaccard index). Intersection-over-union (IoU) for a given class is the ratio of the corresponding number of correctly estimated pixels, i.e. the area of overlap between the predicted and ground-truth segments of the given class, over the area of union of the very segments.

Given predicted $\mathcal{P}$ and target $\mathcal{T}$ segmentation maps, IoU metric for class $\mathcal{C}$ can be formally defined as:

$$\text{IoU}(\mathcal{C}) = \frac{\left|\{p \mid \mathcal{P}[p] = \mathcal{C}\} \cap \{p \mid \mathcal{T}[p] = \mathcal{C}\}\right|}{\left|\{p \mid \mathcal{P}[p] = \mathcal{C}\} \cup \{p \mid \mathcal{T}[p] = \mathcal{C}\}\right|} \tag{5.7}$$

IoU takes values from 0 to 1, higher values indicating larger overlap between the estimated and target pixels belonging to a certain class, thus better segmentation. We compute the IoU metric for each class and report the average IoU across all classes ($mClsIoU$).

5.3 Results

5.3.1 Evaluation on Moving-MNIST

First we evaluate our model on synthetic Moving-MNIST dataset [95]. We create RGB images from the generated gray-scale frames (Section 5.1.1) to feed into the model. We choose a context size of $C = 17$, and predict $N = 5$ time-steps into the future in all three levels (each with its own processing frequency). In addition, we set the LSTM time-periods to $T_0 = 1$, $T_1 = 4$ and $T_2 = 8$ for the low-, mid- and high-levels of MSPred predictor respectively.

We predict image frames at the low-level, and omnichannel heatmaps with Gaussian blobs centered on the digit locations at the upper-levels of our model. We implement all three reconstruction error terms of the loss function (Section 4.3) using pixel-wise mean-squared-error (MSE). The optimized values of the individual loss term coefficients are as follows: $\alpha_1 = \alpha_2 = 2.5$ and $\beta = 5 \cdot 10^{-4}$. We train the model for $175,000$ iterations with batch size of 16 and an initial learning rate

Table 5.1: Resulting metrics of our model evaluation on **Moving-MNIST** test-dataset. The image similarity metrics such as SSIM [109] and LPIPS [115] are computed for the frame predictions (lowest-level), whereas MSE and MAE metrics are reported for the upper-level heatmaps.

Low-level				**Mid-level**		**High-level**	
MSE↓	**SSIM↑**	**PSNR↑**	**LPIPS↓**	**MSE↓**	**MAE↓**	**MSE↓**	**MAE↓**
41.52	0.970	25.99	0.030	0.672	5.60	10.40	63.03

of $2 \cdot 10^{-4}$, which is decayed by a factor of 0.1 after $125,000$ iterations. It takes around 1.8 days to train the model on a single GPU (NVIDIA RTX A6000).

We exploit this dataset for proof of concept (PoC), in order to assess our model's potential of making predictions in different levels of abstraction and distinct time-scales simultaneously. We observe that our model generates feasible and quite accurate prediction for both the image frames of immediate future and the digit locations longer into the future. Sample predicted sequences of the three hierarchy levels, along with respective ground-truth sequences, are illustrated in Figure 5.2. Different metrics are computed for each level, and their averaged values across all $N = 5$ predictions are listed in Table 5.1.

5.3.2 Evaluation on KTH-Actions

The next dataset that we evaluate our model on is KTH-Actions [88], an action recognition dataset consisting of real videos. As in case of Moving-MNIST dataset, we predict $N = 5$ time-steps into the future for all three levels (with corresponding processing-periods of $T_0 = 1$, $T_1 = 4$ and $T_2 = 8$), however for KTH dataset we choose an even smaller context size of $C = 9$ which proves to be enough, even though the uppermost-level receives only two context frames (C_1 and C_9). Moreover, we apply random horizontal-flip augmentation to entire sequences in order to generate more data.

On the intermediate-level of our model we predict heatmaps of nine body-joints composing a sparse pose of the moving person, and at the high-level we estimate a single keypoint heatmap for the location of the person at a future time-step. When dealing with KTH dataset, we exploit the fact that there is only a single person moving in each video, thus predict a single body joint per heatmap-channel. However, the method can potentially be extended to multi-person pose prediction in analogy to the existing top-down [72, 20] or bottom-up methods [59, 7] for human pose estimation.

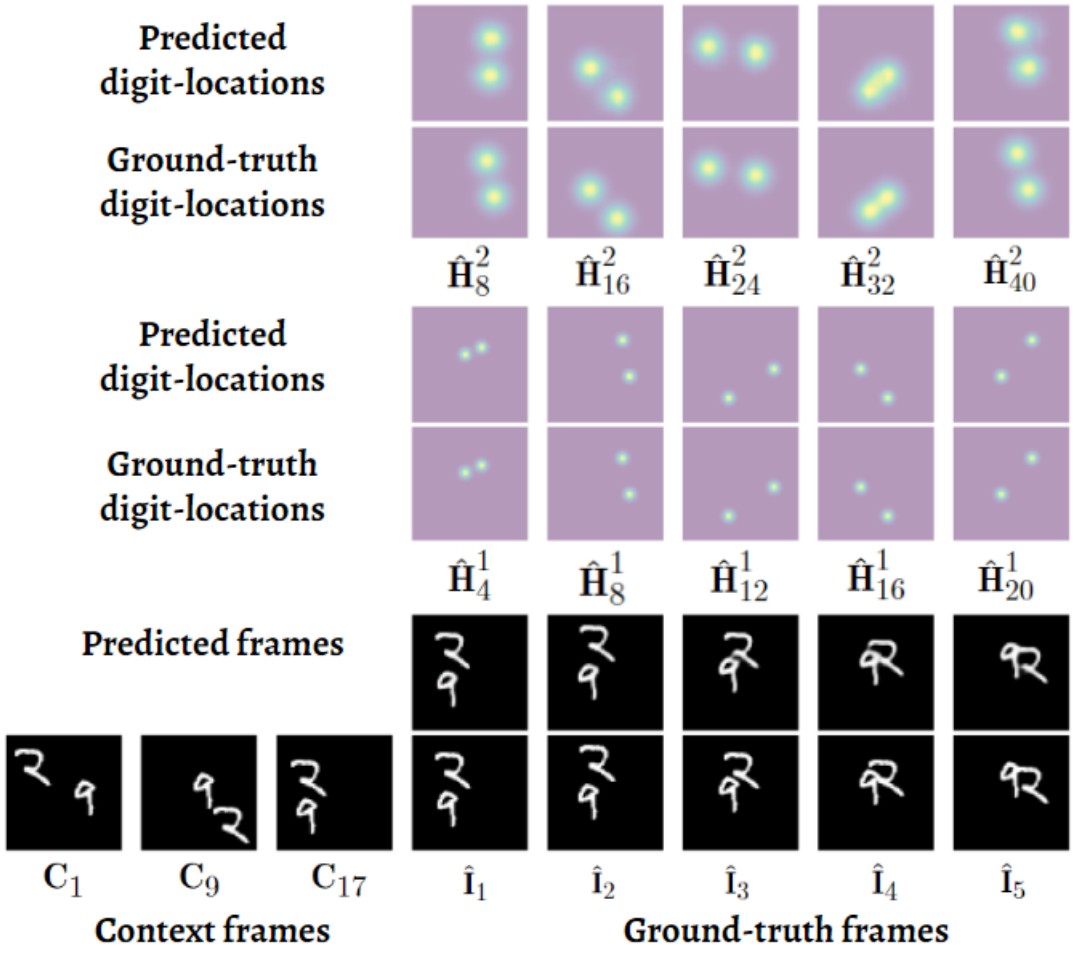


Figure 5.2: An example of hierarchical prediction made by our MSPred model on **Moving-MNIST** test-dataset. Only three of the context frames are plotted. The three pairs of rows indicate three different levels of abstraction. The upper row of each pair illustrates the predicted sequence and the lower one shows the corresponding ground-truth. Note that each level makes predictions at a different time-horizon with its own time-resolution.

Table 5.2: Resulting metrics of MSPred model evaluation on **KTH-Actions** test-set. We report image-similarity metrics for image frame predictions (Low-level), body-keypoint metrics for predicted pose (Mid-level), and Position-Error metric for the estimated person location (High-level). Note: PCK, PDJ, AP and AR stand for $PCK_H@.2$, $PDJ_H@.2$, $AP_H@.1:.5$ and $AR_H@.1:.5$ respectively.

Low-level				Mid-level					High-level
MSE↓	**SSIM↑**	**PSNR↑**	**LPIPS↓**	**MPJPE↓**	**PCK ↑**	**PDJ↑**	**AP↑**	**AR↑**	**PE↓**
33.88	0.941	26.55	0.042	6.15	0.800	0.858	0.827	0.884	4.62

We use pixel-wise mean-squared Error (MSE) for all three reconstruction loss terms in the loss function, i.e. for regression of both the predicted image frames and the multichannel keypoint-heatmaps. Moreover, we exclude the loss components on the channels with missing ground-truth keypoints when dealing with the KTH dataset, in order to avoid misleading labels while training. Although we get some missing keypoints when generating the ground-truth joint positions with pre-trained OpenPose [7] model, our model is still able to generalize well when learning from this noisy dataset.

We set the coefficients of the individual loss terms (Section 4.3) to $\alpha_1 = 1.4, \alpha_2 = 0.2$ and $\beta = 10^{-6}$. The model is trained for $150,000$ iterations with batch size of 16 and an initial learning rate of 10^{-4}, which is decayed by a factor of 0.1 after $130,000$ iterations. It takes around 1.3 days to train the model on a single GPU.

An example of hierarchical prediction, i.e. paired predicted and ground-truth sequences per hierarchy-level, is illustrated in Figure 5.3. We perform evaluation using different test-sets (subsets of the original test-dataset) for each level (Figure 5.1a), to account for distinct sequence lengths required by each level, and report some of the metric values in Table 5.2.

5.3.3 Evaluation on SynPickVP

Finally, we evaluate our model on a novel SynPickVP dataset [79, 46], depicting videos of various bin-picking scenarios performed by a robotic gripper. We choose a context size of $C = 9$ and predict $N = 5$ time-steps into the future in all three levels. However, in contrast to the previous two datasets, in this case we set the processing-periods of predictor modules to $T_0 = 1$, $T_1 = 2$ and $T_2 = 4$, to account for the lack of longer sequences, and the complexity of SynPickVP dataset.

We employ this dataset, in order to evaluate the model's ability to predict more complex structures, such as semantic maps. We train the model to predict a semantic segmentation map of 23 channels (22 object categories and the "background"

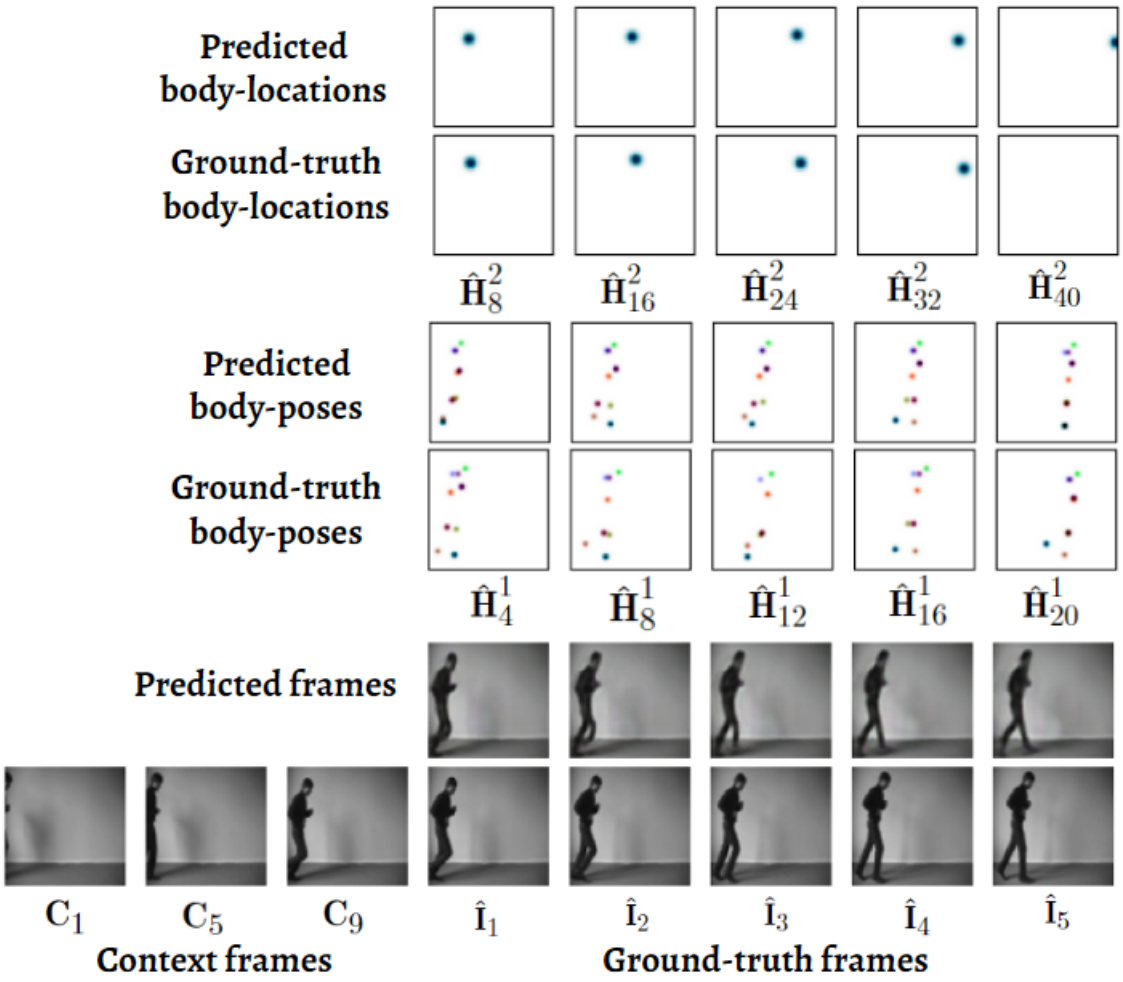


Figure 5.3: An example of hierarchical prediction made by MSPred model on **KTH-Actions** test-set. Only three of the context frames are shown. The three pairs of rows indicate three different levels of abstraction. The upper row of each pair depicts the predicted sequence, and the lower one shows the corresponding ground-truth. Note that each level makes predictions at a different time-horizon with its own time-resolution.

Table 5.3: Resulting metrics of MSPred model evaluation on **SynPickVP** test-set. We report image-similarity metrics for low-level predictions, semantic segmentation metrics for mid-level, and Position-Error of the estimated "gripper" coordinate for the upper-level. Note: Acc, mClsAcc and mClsIoU stand for the pixel accuracy, mean per-class accuracy and mean per-class IoU respectively.

Low-level				**Mid-level**			**High-level**
MSE↓	**SSIM↑**	**PSNR↑**	**LPIPS↓**	**Acc↑**	**mClsAcc↑**	**mClsIoU↑**	**PE ↓**
50.25	0.903	28.61	0.032	0.799	0.254	0.198	14.32

class), and a single keypoint for the gripper position, at its mid- and high-levels respectively. We employ pixel-wise cross-entropy [11] and mean-squared error (MSE) for the mid- and upper-level reconstruction loss terms respectively.

We set the coefficients of individual loss terms (Section 4.3) to $\alpha_1 = 2.0, \alpha_2 = 0.3$ and $\beta = 5 \cdot 10^{-5}$. Moreover, in the cross-entropy loss, we assign weights of 1.0 to all the classes other than "gripper" and "background", which get weights of 2.0 and 0.5 respectively. We slightly over-penalize the segmentation error for "gripper" class in terms of the only "moving" object, and slightly downweight the impact of "background" segmentation error on the overall loss considering the prevalence of easily distinguishable background pixels in the dataset. We train the model for $100,000$ iterations with batch size of 12 and an initial learning rate of $5 \cdot 10^{-5}$. It takes around 5 days to train the model on a single GPU.

An example of hierarchical prediction (pairs of predicted and ground-truth sequences per level) is illustrated in Figure 5.4. Moreover, different metrics are measured for different hierarchy levels, and their mean values across all $N = 5$ predictions are listed in Table 5.3. Similar to evaluation on KTH dataset, we use different test-sets (subsets of original test-set) for evaluation of each level's predictions.

5.3.4 Comparison to Other Methods

Image-Level Predictions

We compare our model's performance for the image frame predictions to several existing video prediction methods based on recurrent neural networks, namely ConvLSTM [91], TrajectoryGRU [92], PredRNN++ [105], PhyDNet [54], SVG-Det (deterministic) and SVG-LP (learned-prior) variants of SVG [15]. In addition, we include a naive baseline CopyLast that simply repeats the last context frame. For the sake of a fair and reproducible comparison, we train all methods with similar configurations using VP-Suite [46], an open-source framework

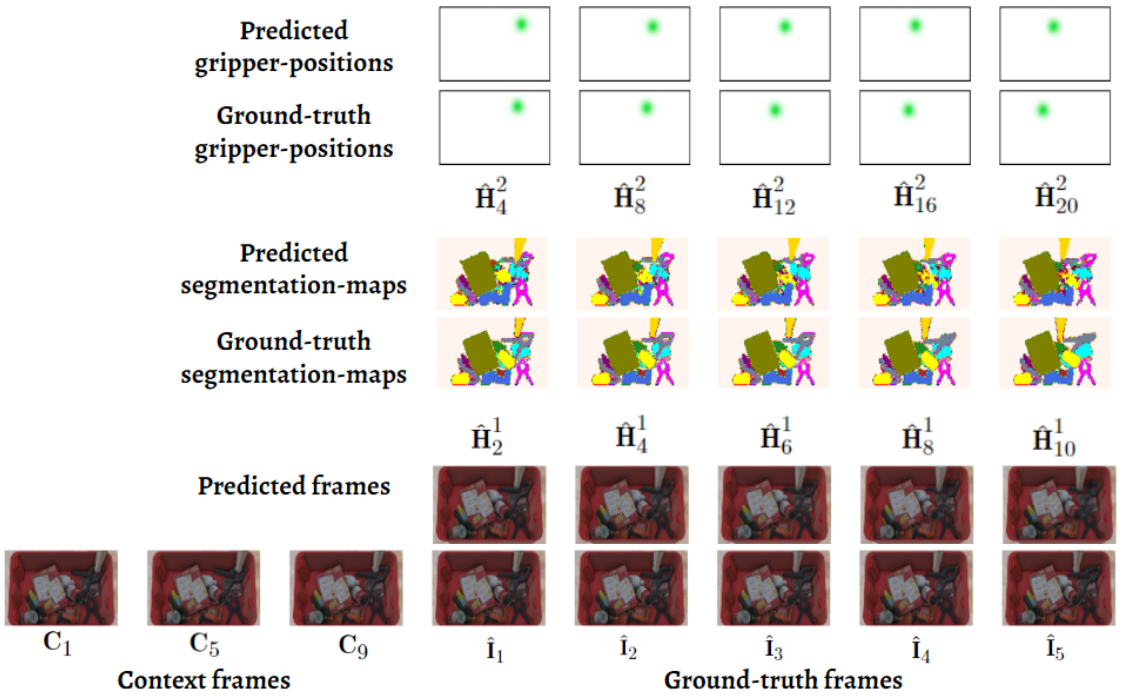


Figure 5.4: An example of hierarchical predictions made by our MSPred model on **SynPickVP** test-set. Only three of the context frames are visualized. The three pairs of rows indicate three different levels of abstraction, with predicted (upper) and ground-truth (lower) sequences in each. Note that each level makes predictions at a different time-horizon with its own time-resolution. One can observe that the future segmentation maps are predicted quite accurately, including the gripper (in gold-yellow) movement. Moreover, the overall direction of gripper movement is captured in the upper-level predictions, in spite of some deviation from the actual ground-truth locations.

Table 5.4: Quantitative comparison between different methods for future frame prediction on **Moving-MNIST** dataset. The best score per metric is highlighted in bold and the second best score is underlined. Our model (MSPred) outperforms all mentioned methods on this dataset across all four metrics.

	Future frame prediction			
	MSE↓	**SSIM↑**	**PSNR↑**	**LPIPS↓**
CopyLast	857.69	0.638	11.73	0.233
ConvLSTM [91]	271.95	0.833	17.22	0.144
TrajGRU [92]	164.75	0.895	20.02	0.075
SVG-Det [15]	134.22	0.900	20.31	0.114
SVG-LP [15]	<u>133.68</u>	0.907	20.36	0.115
PredRNN++ [105]	154.52	0.911	20.20	0.055
PhyDNet [54]	153.54	<u>0.915</u>	<u>20.43</u>	<u>0.054</u>
MSPred (ours)	**41.52**	**0.970**	**25.99**	**0.030**

for deep-learning-based video prediction. The results for Moving-MNIST, KTH-Actions and SynPickVP datasets are listed in Table 5.4, Table 5.5 and Table 5.6 respectively. The best and the second best results for each metric (column) are emphasized by bold and underlined numbers respectively. We observe that on all three datasets, our model (MSPred) achieves either the best or very close to the best results among the compared models across the computed metrics for future frame prediction.

Moving-MNIST: As listed in Table 5.4, MSPred achieves quite accurate and sharp reconstructions for all $N = 5$ predicted frames, outperforming all other models by a considerable margin. Figure 5.5 illustrates a qualitative comparison of different methods on two sample sequences from the Moving-MNIST dataset. In general, due to the synthetic nature and simplicity of the dataset, all models achieve quite accurate frame predictions. Figure 5.5b depicts a challenging sequence in which the ambiguity in digit identities rises when the digits overlap. Although the overall dynamics after the overlap are captured by most of the methods, the baseline methods output blurry predictions and are unable to recover the original digit shape, whereas MSPred successfully achieves accurate frame predictions.

KTH-Actions: As shown in Table 5.5, MSPred is beaten by the compared methods in MSE, PSNR and SSIM scores, which indicates higher pixel differences with respect to the ground-truth frames. However, MSPred produces near the best LPIPS result, exhibiting a high perceptual similarity to the target frames. For a qualitative comparison on KTH-Actions dataset, Figure 5.6 displays predictions

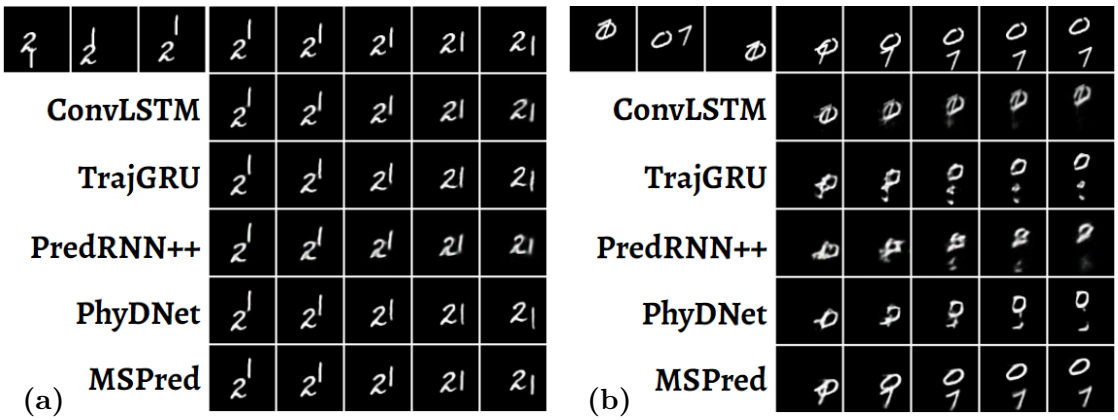


Figure 5.5: Qualitative comparison between different models on **Moving-MNIST**. The top row displays the ground-truth frames. We show three context frames followed by the five predicted frames for both sequences. In general, all compared methods predict feasible future frames. However, only MSPred accurately resolves the ambiguity caused by digits overlaps. Figure reused from Karapetyan et al. [46].

Table 5.5: Quantitative comparison between different future frame prediction models on **KTH-Actions** dataset. The best score per metric is highlighted in bold and the second best score is underlined. Our model (MSPred) achieves very close to the best results in LPIPS and SSIM metrics on this dataset.

	Future frame prediction			
	MSE↓	**SSIM↑**	**PSNR↑**	**LPIPS↓**
CopyLast	50.72	0.909	23.84	0.049
ConvLSTM [91]	<u>12.49</u>	<u>0.957</u>	<u>31.54</u>	0.048
TrajGRU [92]	**12.24**	**0.958**	**31.71**	**0.039**
SVG-Det [15]	35.74	0.927	26.64	0.068
SVG-LP [15]	26.60	0.932	27.60	0.063
PredRNN++ [105]	13.74	0.941	30.68	0.068
PhyDNet [54]	26.35	0.913	28.01	0.125
MSPred (ours)	33.88	0.941	26.55	<u>0.042</u>

on two sequences by different methods. On the one hand, baseline methods that show the best results on almost all metrics, i.e. ConvLSTM [91] and TrajGRU [92], obtain blurry predictions, such as the arm reconstructions shown in Figure 5.6. On the other hand, our model (MSPred) achieves sharper predictions, especially with respect to the human silhouette and performed action.

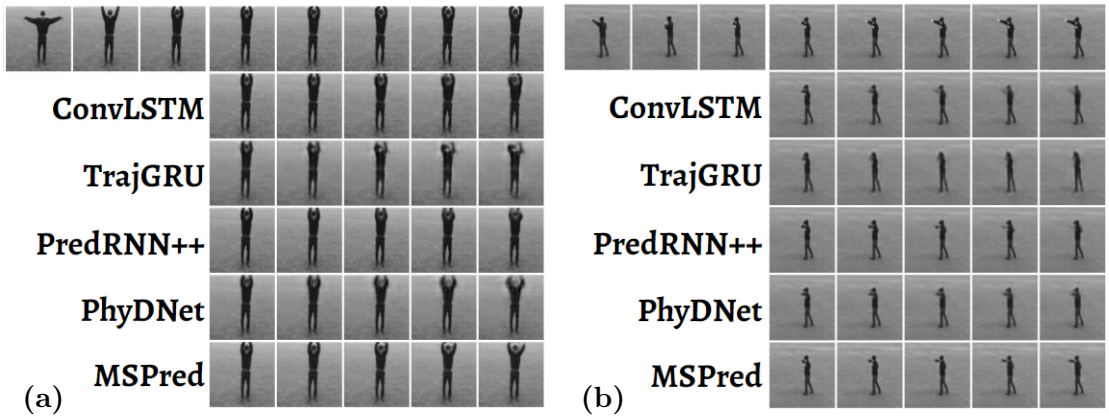


Figure 5.6: Qualitative results on the **KTH-Actions** dataset. The two plotted sequences show actions of *handwaving* and *boxing*. The top row corresponds to ground-truth frames. We display three context frames followed by the five predicted frames for both sequences. MSPred achieves the sharpest and most accurate reconstructions among the compared methods. Figure reused from Karapetyan et al. [46].

Table 5.6: Quantitative comparison between different future frame prediction models on **SynPickVP** dataset. The best score per metric is highlighted in bold and the second best score is underlined. Our model (MSPred) achieves the best PSNR result and very close to the best results for the other three metrics on this dataset.

	Future frame prediction			
	MSE↓	**SSIM↑**	**PSNR↑**	**LPIPS↓**
CopyLast	87.24	0.889	25.97	**0.028**
ConvLSTM [91]	**49.90**	<u>0.907</u>	27.98	0.059
TrajGRU [92]	51.12	**0.908**	<u>28.10</u>	0.041
SVG-Det [15]	60.60	0.879	26.92	0.068
SVG-LP [15]	51.82	0.886	27.38	0.066
PredRNN++ [105]	51.73	0.894	27.50	0.053
PhyDNet [54]	57.31	0.877	26.84	0.053
MSPred (ours)	<u>50.25</u>	0.903	**28.61**	<u>0.032</u>

SynPickVP: Table 5.6 implies that ConvLSTM and TrajGRU achieve the best MSE and SSIM results on SynPickVP dataset, as in case of KTH-Actions dataset. However, MSPred outperforms compared methods on PSNR metric and achieves near the best result on LPIPS, SSIM and MSE metrics. Notably, due to the high complexity of the SynPickVP dataset, all models tend to produce blurry predicted frames, thus the naive CopyLast baseline achieves the best LPIPS result.

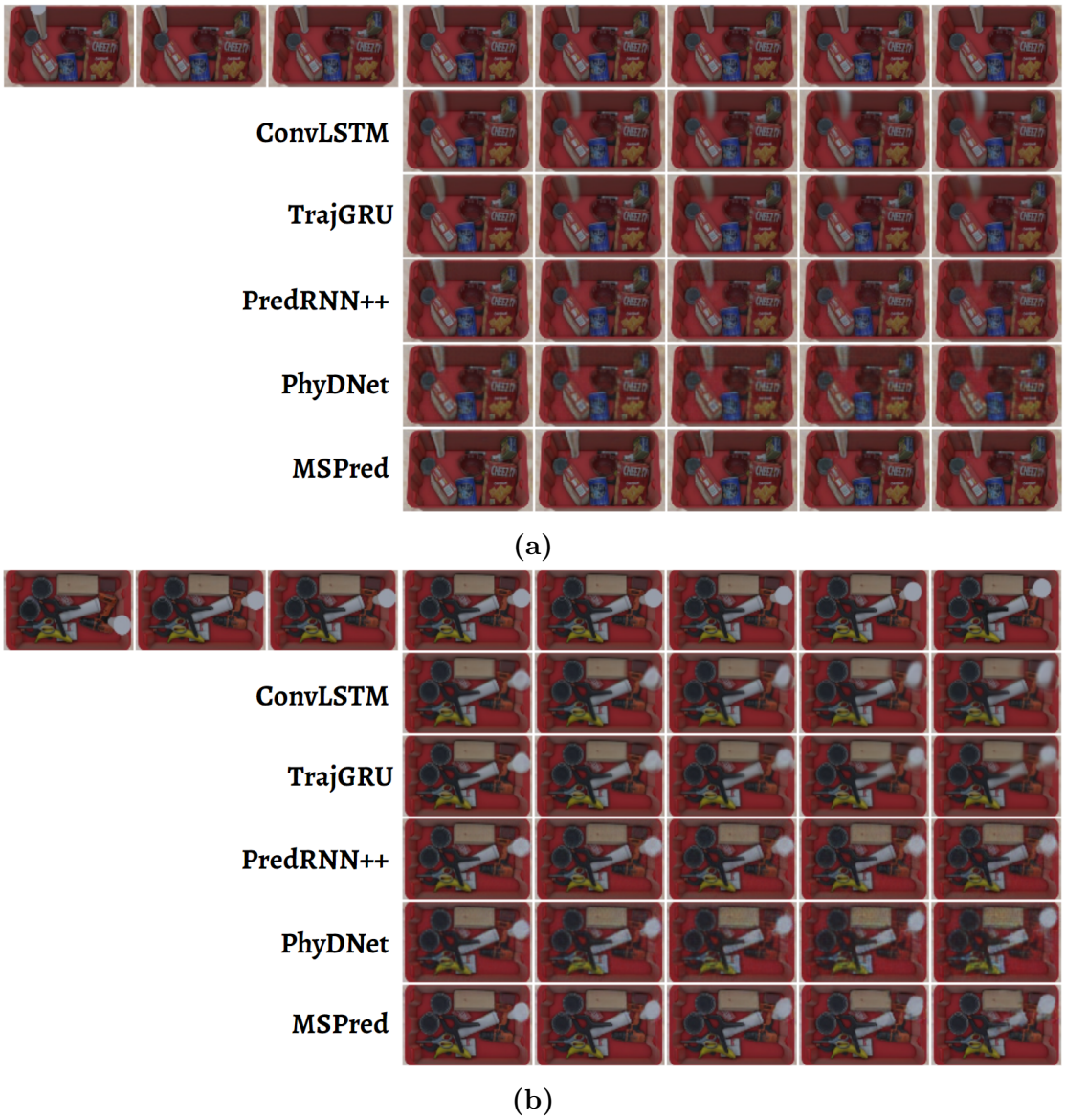


Figure 5.7: Qualitative results on **SynPickVP** dataset. The top row of each subfigure displays the respective target frame sequence. We display three context frames followed by the five predicted frames for two test-set sequences. MSPred achieves sharper reconstructions, whereas the baseline methods tend to blur the predictions. Figure reused from Karapetyan et al. [46].

A qualitative comparison on SynPickVP dataset between different video prediction models is illustrated in Figure 5.7. The baseline methods mostly tend to blur the suction-cap gripper as well as the objects moved by it, whereas MSPred achieves relatively more accurate predictions.

Table 5.7: Quantitative comparison between our model (MSPred) and two baselines for body-pose prediction on **KTH-Actions** dataset. The best score per metric is highlighted in bold and the second best score is underlined. Note: PCK, PDJ, AP and AR stand for $PCK_H@.2$, $PDJ_H@.2$, $AP_H@.1:.5$ and $AR_H@.1:.5$ respectively.

	Body-pose prediction				
	MPJPE↓	**PCK↑**	**PDJ↑**	**AP↑**	**AR↑**
CopyLast	10.50	0.550	0.558	0.620	0.628
SVG-Frame2Kpoints	<u>6.63</u>	**0.810**	**0.888**	<u>0.825</u>	**0.905**
MSPred (ours)	**6.15**	<u>0.800</u>	<u>0.858</u>	**0.827**	<u>0.884</u>

High-Level Structured Predictions

We further evaluate the performance of our model for the higher-level predictions by comparing it to some simple baselines on KTH-Actions and SynPickVP datasets.

KTH-Actions: First, for comparison on KTH dataset, we create and train a baseline model (SVG-Frame2Kpoints) based on a modified SVG-LP [15] architecture, which predicts body-keypoints conditioned on input frames. The key modifications reside in the number of output channels, since the model outputs multichannel heatmaps instead of RGB images, as well as in the recurrent blocks that are replaced by ConvLSTMs for a more fair comparison to MSPred. The SVG-Frame2Kpoints baseline takes $C = 9$ context frames as input and is trained to estimate human poses (the same nine joints as MSPred) for the next $N = 20$ time-steps. Since there are no predicted image frames to be fed back into the model at the next time-step, we design it to be autoregressive in feature space, similar to MSPred. Moreover, for comparing it to MSPred model, we pick a subsequence (at time-steps $13, 17, 21, 25$ and 29) of the predicted sequence for computing the evaluation metrics. In addition, we employ CopyLast as a simple baseline for comparison. The calculated values of some metrics for MSPred and the two baseline methods are reported in Table 5.7. We observe that MSPred achieves either the best or the second best score over all computed metrics.

Considering the distinct levels of complexity across different classes, as well as class-imbalance present in the dataset, we distinguish two categories and evaluate some metrics for samples of each category separately. "Moving" category covers sequences of *running*, *walking* and *jogging* actions, and "Standing" category comprises samples of *handwaving* and *handclapping* and *boxing* classes. Table 5.8 depicts values for different metrics per-category computed with regard to the pre-

Table 5.8: Comparison of per-category metrics between MSPred and two baseline models (for predicting body-joints and position of the person) on **KTH-Actions** dataset. The best score per metric is highlighted in bold and the second best score is underlined. Note: AP and AR stand for $AP_H@.1:.5$ and $AR_H@.1:.5$ respectively.

	Body-pose				**Body-location**	
	Moving		Standing		Moving	Standing
	AP↑	**AR↑**	**AP↑**	**AR↑**	**PE↓**	**PE↓**
CopyLast	0.306	0.322	<u>0.907</u>	0.906	<u>22.90</u>	**1.30**
SVG-Frame2Kpoints	<u>0.723</u>	**0.878**	**0.912**	**0.929**	-	-
MSPred (ours)	**0.743**	<u>0.847</u>	0.892	<u>0.914</u>	**9.89**	<u>1.72</u>

dicting body-joints and person-coordinates. On the one hand, in case of "Standing" category samples, it is relatively easier to achieve good quantitative metric values, however it is much harder to get qualitatively acceptable predictions by capturing the small hand movements performed on those videos. Notably, even the naive CopyLast method is able to outperform MSPred model for the body-location prediction on "Standing" category sequences. On the other hand, MSPred outperforms CopyLast by a large margin, as well as achieves comparable results to SVG-Frame2Kpoints baseline for videos of "Moving" category. Although MSPred does not outperform SVG-Frame2Kpoints baseline across all metrics, it owns the advantage of making predictions of different abstraction levels and temporal scales in parallel, without sacrificing the performance of individual levels.

SynPickVP: Furthermore, similar to KTH dataset, we also evaluate and compare MSPred model to two baseline methods on SynPickVP dataset, namely SVG-Frame2Seg trained to predict segmentation-maps from input frames, and the naive CopyLast baseline. In analogy to SVG-Frame2Kpoints model, the SVG-Frame2Seg baseline is also based on a modified SVG-LP [15] architecture, which uses ConvLSTMs recurrent blocks, produces multichannel segmentation map in the output, and operates in a feature-space autoregressive flow. It takes $C = 9$ context frames as input and is trained to estimate future semantic maps for the next $N = 10$ time-steps. Moreover, we evaluate the baseline on a subsequence (at time-steps $11, 13, 15, 17$ and 19) of the predicted sequence, for comparing the performance to MSPred model.

We compute the segmentation metrics per object-class (23) and average the results across different groups of classes as follows. We distinguish three categories, namely "Moving" (*gripper* object class), "Static" (all the other objects) and "Background". The computed metric values are listed in Table 5.9. On the one hand,

Table 5.9: Comparison of per-category metrics between MSPred model and two baselines, for predicting semantic segmentation maps and the position of moving robotic gripper on **SynPickVP** dataset. The best score per metric is highlighted in bold and the second best score is underlined. MSPred outperforms the other two models for "Moving" class sequences, i.e. for the gripper segmentation and coordinate prediction. However, for segmentation of the static objects and the background (potentially changed by the gripper-movement), the last context frame repeated by CopyLast better matches with the ground-truth, because no drastic overall changes are present in the scenes across frames. Note: Acc and IoU stand for the mean per-class accuracy and the mean per-class IoU respectively.

	Segmentation-map						**Gripper-position**
	Moving		Static		Background		
	Acc↑	**IoU↑**	**Acc↑**	**IoU↑**	**Acc↑**	**IoU↑**	**PE↓**
CopyLast	0.397	0.324	**0.557**	**0.455**	**0.957**	**0.919**	<u>16.49</u>
SVG-Frame2Seg	<u>0.606</u>	<u>0.443</u>	<u>0.255</u>	<u>0.197</u>	<u>0.956</u>	0.899	-
MSPred (ours)	**0.651**	**0.471**	0.202	0.151	0.952	<u>0.901</u>	**14.32**

we observe that CopyLast shows better segmentation results on static objects and the background, since the last context frame repeated by the baseline better matches with the ground-truth due to no salient overall changes present in the scenes across video frames. On the other hand, MSPred outperforms CopyLast and SVG-Frame2Seg by a considerable margin for the gripper-segmentation ("Moving" class) and coordinate prediction.

5.4 Ablation Study

To conclude our evaluation process, we perform an ablation study, investigating the importance of separate components and design choices of our proposed model.

5.4.1 Effect of Hierarchy

First of all, we investigate the relevance of the key design choices in our model, namely the spatial and temporal hierarchy, as well as the recurrent block type of the predictor component. We perform the related experiments on Moving-MNIST dataset, and report the evaluation metrics for the image frame predictions in Table 5.10.

The experiments switch between two types of RNN cells, namely the standard fully-connected LSTM and ConvLSTM cells. In the experiments where we remove temporal hierarchy (Table 5.10, rows 3, 4 and 6), we set the processing-periods

Table 5.10: Ablation study for different design choices of MSPred on **Moving-MNIST** dataset. We investigate the impact of different recurrent cells (LSTM and ConvLSTM), as well as the importance of spatio-temporal hierarchy on the image-level predictions. We highlight the best and second best results by bold and underlined numbers respectively.

	Design choices			**Results**			
	RNN Cell	Spatial	Temporal	**MSE↓**	**SSIM↑**	**PSNR↑**	**LPIPS↓**
1	ConvLSTM	✓	✓	**41.52**	**0.970**	**25.99**	**0.030**
2	ConvLSTM	-	✓	<u>73.47</u>	<u>0.950</u>	<u>22.81</u>	<u>0.057</u>
3	ConvLSTM	✓	-	92.45	0.921	20.81	0.093
4	ConvLSTM	-	-	112.18	0.912	20.97	0.097
5	LSTM	✓	✓	208.71	0.827	17.95	0.202
6	LSTM	-	-	134.22	0.900	20.31	0.114

of all levels to one, i.e. all recurrent blocks operate at every time-step. Whereas eliminating spatial hierarchy (Table 5.10, rows 2, 4 and 6) creates separate decoder branches that do not share any of their blocks. We explore how different combinations of these design choices affect the resulting metric values.

The results show that the base MSPred model (Table 5.10, row 1) with both spatio-temporal hierarchy and ConvLSTM recurrent cells outperforms the other versions by a significant margin. Moreover, elimination of either the spatial- (Table 5.10, row 2) or temporal-hierarchy (Table 5.10, row 3), or both structures (Table 5.10, row 4) from the model results in a performance loss, indicating that features of different granularity levels (temporal and spatial) are essential for making accurate and feasible predictions. Finally, we observe a significant drop in the performance when replacing the ConvLSTM cells with linear LSTMs in all recurrent blocks (Table 5.10, row 5). This leads to the worst quantitative results compared to all the other model variants. However, the resulting performance loss is relatively smaller, when using standard LSTM cells in a simpler baseline with no spatial hierarchy (Table 5.10, row 6). This is expected, since the linear LSTMs, operating on flat vector embeddings, require reshaped feature maps from the lower-levels as input, which leads to the loss of useful positional information and hence to a large drop in performance.

Next, we investigate the impact of hierarchical-supervision on the success of image frame predictions. In this case we perform the experiments on the KTH dataset. We compare the base MSPred model to another baseline trained only for future frame prediction. We create this baseline by removing the two upper-level decoder heads. Table 5.11 implies that the MSPred model trained to simultaneously make predictions of different granularity levels, shows quite comparable

Table 5.11: Ablation study for the impact of hierarchical-supervision on image frame predictions for **KTH-Actions** dataset. The first row corresponds to the model trained without hierarchical-supervision, and the second row corresponds to the base MSPred model. The best of the two scores per metric is highlighted in bold.

Hierarchical-supervision	**Results**			
	MSE↓	**SSIM↑**	**PSNR↑**	**LPIPS↓**
-	**25.88**	**0.941**	**27.69**	0.044
✓	33.88	**0.941**	26.55	**0.042**

Table 5.12: Ablation study for the effect of each RNN-level on MSPred model trained on **KTH-Actions** dataset. Inference is run for two variants of the base model, namely $\text{MSPred}_{\text{H[0,1]}}$ with eliminated high-level RNN, and $\text{MSPred}_{\text{H[0,2]}}$ with eliminated mid-level RNN. The best score per metric is highlighted in bold and the second best score is underlined. Note: AP and AR stand for $\text{AP}_{\text{H}}@.1:.5$ and $\text{AR}_{\text{H}}@.1:.5$ respectively.

	Hierarchy-level			**Results**				
				Frames		**Body-pose**		**Body-location**
	Low	Mid	High	**PSNR↑**	**LPIPS↓**	**AP↑**	**AR↑**	**PE↓**
$\text{MSPred}_{\text{H[0,1]}}$	✓	✓	-	<u>26.54</u>	<u>0.043</u>	0.610	0.649	<u>16.59</u>
$\text{MSPred}_{\text{H[0,2]}}$	✓	-	✓	26.45	**0.042**	<u>0.796</u>	<u>0.853</u>	**4.46**
MSPred	✓	✓	✓	**26.55**	**0.042**	**0.827**	**0.884**	**4.46**

results to the model trained only for image-level predictions. This indicates that hierarchical-supervision (or multi-tasking) is not a key factor for the success of our model. Moreover, MSPred sacrifices only a little of performance in favor of making long-term predictions of structured representations on its upper-levels, along with predicting image frames into immediate future.

5.4.2 Effect of Each RNN-Level

We analyze the effect of individual levels of recurrent networks in the MSPred predictor. We run the inference using a model trained on KTH dataset. In each experiment, one of the RNN networks of the trained model is disabled, by making it to simply copy forward its input. We evaluate the trained MSPred model once with eliminated mid- then high-level RNN.

The results are listed in Table 5.12. First, we observe that the high-level RNN of the trained model is greatly responsible for the accurate predictions of both

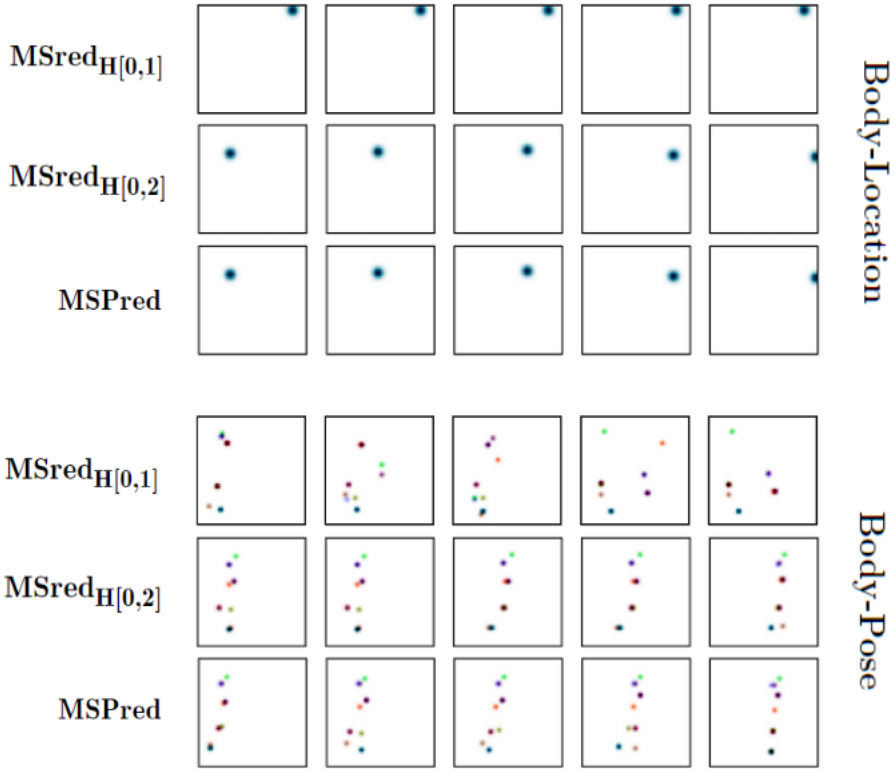


Figure 5.8: Qualitative comparison between predictions made by MSPred model and its two variants (with either disabled mid-, or high-level RNN) on **KTH-Actions**. The upper and lower parts of the figure show the predicted body-locations and body-pose respectively (in different time-resolutions). The lower, middle and upper rows of each part correspond to predictions made by the base MSPred model, and its MSPred$_{H[0,2]}$ and MSPred$_{H[0,1]}$ versions respectively.

its own level and the level below (MSPred$_{H[0,1]}$). However, we also notice that the performance loss is more moderate when eliminating the mid-level from the trained model (MSPred$_{H[0,2]}$). As expected, the image-level predictions mostly stay unaffected in both cases, since the MSPred model is trained to predict only the next $N = 5$ frames into immediate future. Furthermore, Figure 5.8 illustrates an example of predicted sequences generated by the base MSPred model, as well as the MSPred$_{H[0,1]}$ and MSPred$_{H[0,2]}$ variants. This once again shows the fact that in the trained model on this particular dataset, the upper-level RNN learns to predict the body-location of the moving person in a coarse time-resolution, and the mid-level RNN learns to correct the body-pose information on a finer time-scale, in terms of more specific body-location and detailed joint coordinates. As demonstrated in the Figure 5.8, MSPred$_{H[0,2]}$ model with eliminated mid-level,

Table 5.13: Quantitative comparison between MSPred and MSPred-Det models on **Moving-MNIST** dataset. The best of the two scores per metric is highlighted in bold. While the image-level metric values are almost the same for both models, the metrics for mid- and upper-level predictions are much worse in case of the deterministic baseline.

	Moving-MNIST					
	Frames		**Digit-locations**		**Digit-locations**	
	SSIM↑	**LPIPS↓**	**MSE↓**	**MAE↓**	**MSE↓**	**MAE↓**
MSPred-Det	**0.972**	0.032	0.971	7.36	12.79	72.86
MSPred	0.970	**0.030**	**0.672**	**5.60**	**10.40**	**63.03**

Table 5.14: Quantitative comparison between MSPred and MSPred-Det models on **KTH-Actions** dataset. The best of the two scores per metric is highlighted in bold. While the metric values for low- and high-level predictions are comparable between the two models, there is a noticeable drop in performance for the mid-level estimations in case of the deterministic baseline. The best scores for each metric are highlighted in bold. Note: AP and AR stand for $AP_H@.1:.5$ and $AR_H@.1:.5$ respectively.

	KTH-Actions					
	Frames		**Body-pose**			**Body-location**
	SSIM↑	**LPIPS↓**	**MPJPE↓**	**AP↑**	**AR↑**	**PE↓**
MSPred-Det	0.940	**0.042**	6.76	0.788	0.831	**4.38**
MSPred	**0.941**	**0.042**	**6.15**	**0.823**	**0.884**	4.46

keeps repeating the person silhouette with imperfect joint predictions until the next ticking time-step of the upper-level RNN. Moreover, the predictions made by $MSPred_{H[0,1]}$ variant are much worse, proving that the upper-level RNN has an undeniable role in the overall success of long-term abstract predictions, for this particular use-case.

5.4.3 Effect of Stochastic Prediction

The next key part of our ablation study explores the role of stochastic video prediction in the overall success of MSPred model. For this purpose, we train the deterministic version of MSPred model (MSPred-Det) on both Moving-MNIST and KTH-Actions datasets, by removing the hierarchy of latent variable networks from the model. We report the comparative evaluation results in Table 5.13 and Table 5.14 for Moving-MNIST and KTH-Actions datasets respectively. We observe

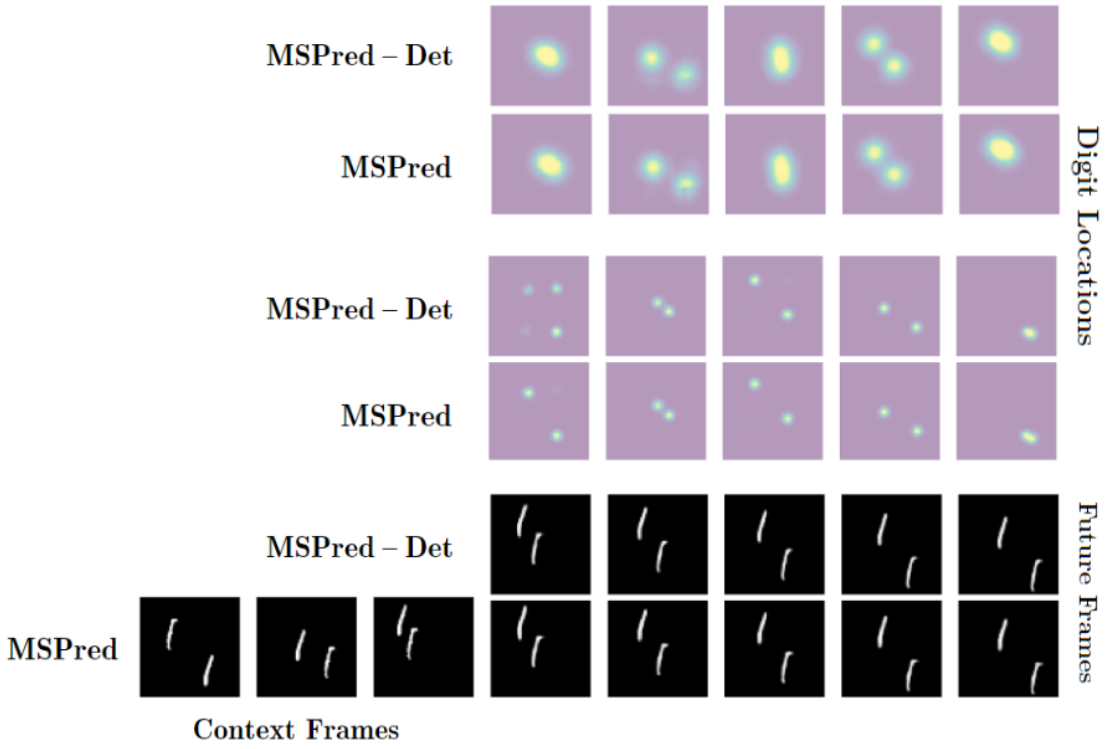


Figure 5.9: Qualitative comparison between MSPred and MSPred-Det models on **Moving-MNIST**. Three of the context frames are shows, followed by the predictions of different levels made by both models. The lower and upper rows of each pair of sequences correspond to the predictions made by MSPred and MSPred-Det model respectively.

that the base MSPred model enhanced with a hierarchy of learned-prior networks, outperforms the deterministic variant on both datasets. In case of Moving-MNIST dataset, there is a noticeable drop in performance for the upper-level predictions. As one can observe from Figure 5.9, MSPred-Det baseline is prone to producing ambiguous predictions for the digit locations, combining multiple possible outputs on a single heatmap (first heatmap of mid-level predictions). Similarly, the metrics on KTH dataset also indicate a significant performance loss for the mid-level predictions. Figure 5.10 shows an example demonstrating this fact. Although the MSPred-Det baseline is able to make some feasible predictions for the body-locations, it produces extremely poor estimations for the body-pose, as well as quite blurry generations for the future frames. Furthermore, we observe that the effect of added ancestral-sampling [8] is not noticeable enough, probably due to the characteristics of the chosen datasets.

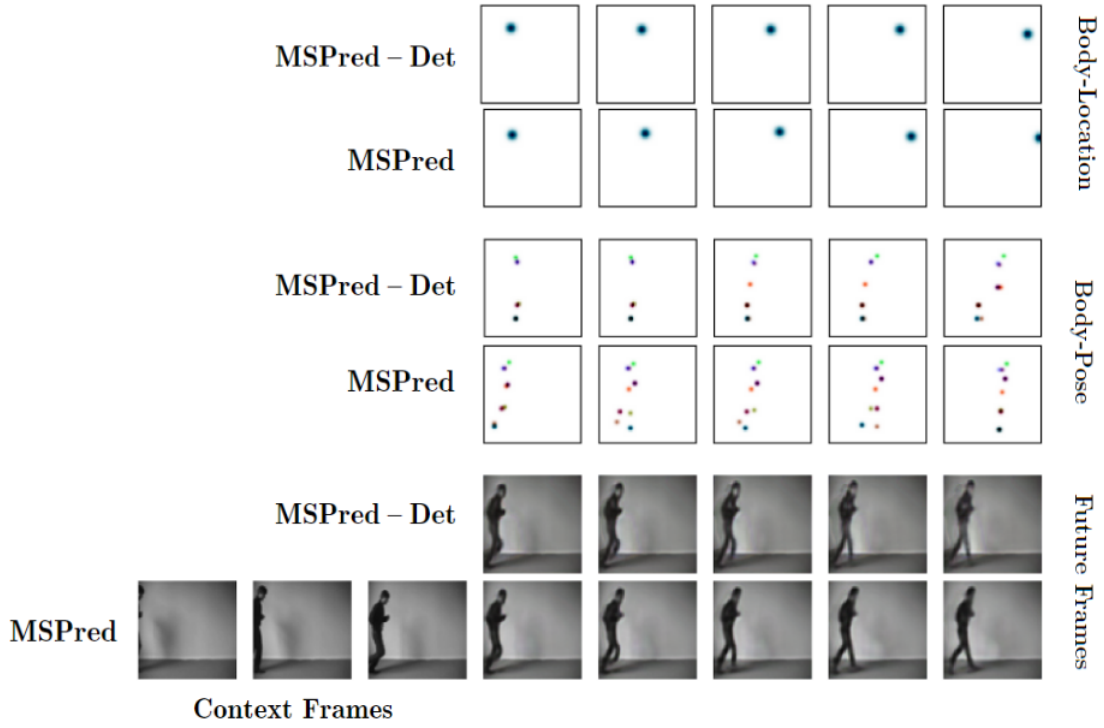


Figure 5.10: Qualitative comparison between MSPred and MSPred-Det models on **KTH-Actions** dataset. Three of the context frames are shows, followed by the predictions of different levels made by both models. The lower and upper rows of each pair of sequences correspond to the predictions made by MSPred and MSPred-Det model respectively.

Finally, we explore the diversity of predictions on KTH-Actions dataset. Figure 5.11 illustrates two distinct sequences conditioned on the same context frames, made by MSPred model using different random latent samples, generated by the learned latent distribution models and fed into the predictor module. Only the predictions for body-pose and body-locations are shown in the figure, since the immediate future frame predictions undergo inessential modifications across the latent space. We add red squares to highlight the parts of the visualizations, where there are some differences in exact coordinates of the predicted keypoint-locations (between the two depicted samples) noticeable for the eye. However, we observe that the overall diversity of the model predictions is not high enough for the specific datasets that we use for evaluation.

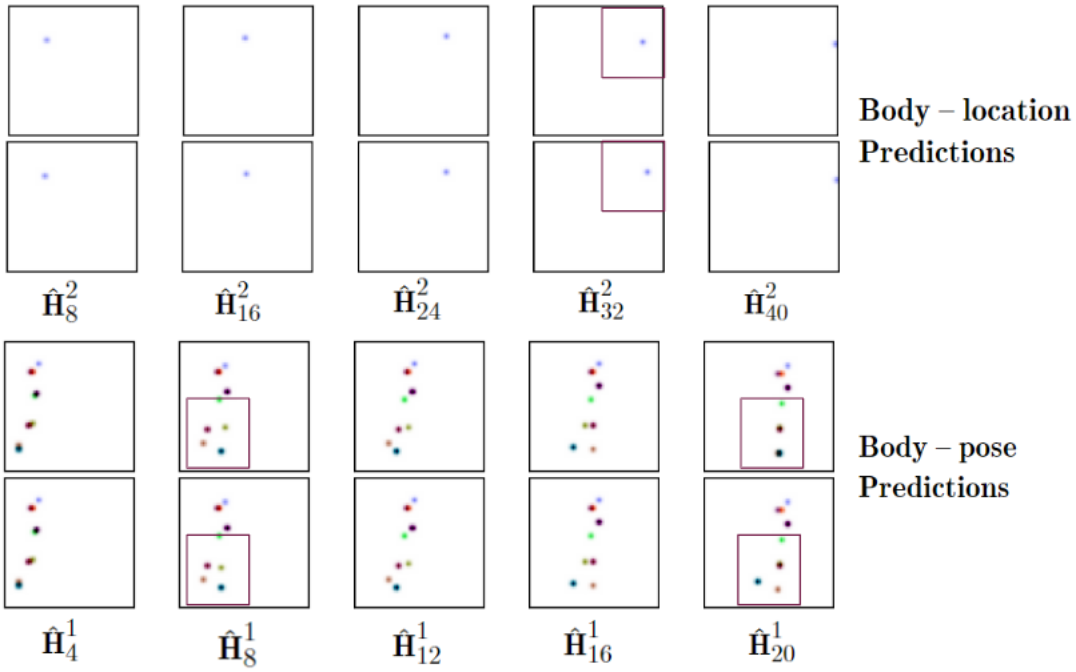


Figure 5.11: Two distinct predictions for the same input sequence (**KTH-Actions**), made by MSPred model using different random latent samples as input to the predictor. We draw the reader's attention to the parts with such differences (between the two given samples) using red squares.

6 Conclusion

In this thesis, we proposed and investigated *MSPred*, a novel video prediction model for multi-scale hierarchical prediction. Our proposed model is able to simultaneously forecast future possible outcomes of different granularity levels at distinct time resolutions, conditioned on the given video frames. More precisely, MSPred makes predictions at three different abstraction levels, i.e. it forecasts subsequent video frames into immediate future, as well as predicts more abstract representations longer into the future using coarser temporal resolutions.

The key module of MSPred, i.e. the hierarchical predictor, leverages a hierarchy of recurrent neural networks along with recurrent latent variable models, operating at distinct spatial and temporal granularity levels. This module allows the model to achieve an extended prediction time-horizon for high-level structures by using large temporal resolutions and only a handful of RNN iterations, thus mitigating the error accumulation problem inherent to long-term prediction scenarios. Moreover, the employed feature-space autoregressive flow not only decreases the overall computational complexity but also enables a hierarchical design where each level's output is independent of the generations of lower levels.

The proposed model was evaluated on three different datasets, depicting various use-cases. The model was compared to several existing video prediction methods for future frame prediction. Our model achieves the best or near the best results across all the measured metrics, especially for the perceptual similarity metric (LPIPS). Moreover, we showed by qualitative comparison, that MSPred obtains competitive results among the compared models in terms of sharper and more accurate predictions, by leveraging temporal and spatial hierarchy of features. In addition, we demonstrated our model's ability to make high-level plausible predictions, e.g. human poses and semantic segmentation maps, longer into the future. However, in certain cases we observed that MSPred is outperformed even by the naive CopyLast baseline, for instance when predicting future body-joints in "handwaving" videos of KTH-Actions dataset, or future segmentation maps of the bin objects moved by the robotic gripper in SynPickVP dataset. Nevertheless, unlike baseline video prediction methods, MSPred allows to perform predictions of different abstraction levels and temporal scales in parallel, without sacrificing much of the performance of individual levels.

Finally, we presented a comprehensive ablation study and analysis of our model, showing the importance of the spatial and temporal hierarchy employed by our model, as well as investigating the effect of individual predictor-levels for specific datasets. Moreover, we empirically showed the importance of the stochastic framework adopted by our model, for the quality and diversity of the predictions.

For the future work, an important step can be investigating methods for adaptively [12] determining optimal temporal frequencies for the different hierarchy levels of the model, given a specific dataset. Moreover, although our model successfully demonstrates the concept of predicting different representations at different time scales, we observe that the diversity of predictions is limited for the given datasets. With better fine-tuning of parameters and adjusting the capacity of the latent variable models, as well as data augmentation that inherently inserts stochasticity in the dataset, one could enable more diverse outcomes. Another step is to scale the model for more complex and diverse datasets, such as for autonomous driving scenarios. One possible direction would be extending the model for multiperson-pose prediction, by employing a more powerful encoder-decoder (e.g. ResNet-based [33]) and larger RNN networks. A further extension might include adding supervision for guiding the prediction process, such as conditioning the predictive model on some representations (e.g. pose, location, action, trajectories) and predicting realistic frames that match the given representation. This could have the side effect of encouraging the model to make more diverse predictions [27].

List of Figures

List of Tables

Bibliography

[1] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele. "2D human pose estimation: New benchmark and state of the art analysis". In: *Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 3686–3693.

[2] M. Babaeizadeh, C. Finn, D. Erhan, R. H. Campbell, and S. Levine. "Stochastic variational video prediction". In: *International Conference on Learning Representations (ICLR)*. 2018.

[3] H. Bay, T. Tuytelaars, and L. Van Gool. "Surf: Speeded up robust features". In: *European Conference on Computer Vision (ECCV)*. 2006, pp. 404–417.

[4] A. Benzine, B. Luvison, Q. C. Pham, and C. Achard. *Single-shot 3D multi-person pose estimation in complex images*. 2021. arXiv: `1911.03391 [cs.CV]`.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

[6] Y. Cao and J. Wu. *Rethinking self-supervised Learning: Small is beautiful*. 2021. arXiv: `2103.13559 [cs.CV]`.

[7] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. "Realtime multi-person 2d pose estimation using part affinity fields". In: *Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 7291–7299.

[8] L. Castrejon, N. Ballas, and A. Courville. "Improved conditional VRNNs for video prediction". In: *International Conference on Computer Vision (ICCV)*. 2019.

[9] A. Cauchy. "Méthode générale pour la résolution de systèmes d'équations simultanées". In: *In Compte rendu des séances de l'académie des sciences* (1847), pp. 536–538.

[10] S. Chiappa, D. Racaniere Sébastienand Wierstra, and S. Mohamed. "Recurrent environment simulators". In: *International Conference on Learning Representations (ICLR)*. 2017.

[11] H.-k. Chiu, E. Adeli, and J. C. Niebles. *Segmenting the future*. 2019. arXiv: `1904.10666 [cs.CV]`.

[12] J. Chung, S. Ahn, and Y. Bengio. "Hierarchical multiscale recurrent neural networks". In: *International Conference on Learning Representations (ICLR)*. 2017.

[13] F. Cricri, X. Ni, M. Honkala, E. Aksu, and M. Gabbouj. "Video ladder networks". In: *NIPS workshop on ML for Spatiotemporal Forecasting*. 2016.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition (CVPR)*. 2009, pp. 248–255.

[15] E. Denton and R. Fergus. "Stochastic video generation with a learned prior". In: *International Conference on Machine Learning (ICML)*. 2018, pp. 1174–1183.

[16] V. Dumoulin and F. Visin. *A guide to convolution arithmetic for deep learning*. 2016. arXiv: `1603.07285 [stat.ML]`.

[17] J. L. Elman. "Finding structure in time". In: *Cognitive science* 14.2 (1990), pp. 179–211.

[18] J. E. van Engelen and H. H. Hoos. "A survey on semi-supervised learning". In: *Machine Learning* 109.2 (2020), pp. 373–440.

[19] K. Fan, C. Joung, and S. Baek. "Sequence-to-sequence video prediction by learning hierarchical representations". In: *Applied Sciences* (2020).

[20] H.-S. Fang, S. Xie, Y.-W. Tai, and C. Lu. "RMPE: Regional multiperson pose estimation". In: *International Conference on Computer Vision (ICCV)*. 2017.

[21] H. Farazi and S. Behnke. "Motion segmentation using frequency domain transformer networks". In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. 2020.

[22] H. Farazi, J. Nogga, and S. Behnke. "Local frequency domain transformer networks for video prediction". In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. 2021.

[23] F. A. Fardo, V. H. Conforto, F. C. de Oliveira, and P. S. Rodrigues. *A formal evaluation of PSNR as quality measurement parameter for image segmentation algorithms*. 2016. arXiv: `1605.07116 [cs.CV]`.

[24] G. Feng, S. Wang, and T. Liu. "New benchmark for image segmentation evaluation". In: *Journal of Electronic Imaging* 16.3 (2007).

[25] C. Finn, I. Goodfellow, and S. Levine. "Unsupervised learning for physical interaction through video prediction". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[26] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. "Flownet: Learning optical flow with convolutional networks". In: *International Conference on Computer Vision (ICCV)*. 2015.

[27] N. Fushishita, A. Tejero-de Pablos, Y. Mukuta, and T. Harada. "Long-term human video generation of multiple futures using poses". In: *European Conference on Computer Vision (ECCV)*. 2020, pp. 596–612.

[28] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[29] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative adversarial nets". In: *Advances in Neural Information Processing Systems (NIPS)*. 2014, pp. 2672–2680.

[30] A. Graves. *Generating sequences with recurrent neural networks*. 2014. arXiv: `1308.0850 [cs.NE]`.

[31] N. Günnemann and J. Pfeffer. "Predicting defective engines using convolutional neural networks on temporal vibration signals". In: *First International Workshop on Learning with Imbalanced Domains: Theory and Applications*. 2017, pp. 92–102.

[32] I. Habibie, W. Xu, D. Mehta, G. Pons-Moll, and C. Theobalt. "In the wild human pose estimation using explicit 2d features and intermediate 3d representations". In: *Computer Vision and Pattern Recognition (CVPR)*. 2019.

[33] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[34] S. E. Hihi and Y. Bengio. "Hierarchical recurrent neural networks for long-term dependencies". In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 8. 1995, pp. 493–499.

[35] G. Hinton. *Neural networks for machine learning*. Coursera, video lectures. 2012.

[36] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[37] A. Hu, F. Cotter, N. Mohan, C. Gurau, and A. Kendall. "Probabilistic future prediction for video scene understanding". In: *European Conference on Computer Vision (ECCV)*. 2020.

[38] S. Ioffe and C. Szegedy. "Batch normalization: accelerating deep network training by reducing internal covariate shift". In: *International Conference on Machine Learning (ICML)*. 2015, pp. 448–456.

[39] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu. "Human3.6M: Large scale datasets and predictive methods for 3d human sensing in natural environments". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 36.7 (2014), pp. 1325–1339.

[40] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. “Spatial transformer networks”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2015.

[41] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. “What is the best multi-stage architecture for object recognition?” In: *International Conference on Computer Vision (ICCV)*. 2009, pp. 2146–2153.

[42] X. Jin, H. Xiao, X. Shen, J. Yang, Z. Lin, Y. Chen, Z. Jie, J. Feng, and S. Yan. “Predicting scene parsing and motion dynamics in the future”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2017.

[43] X. Jin, X. Li, H. Xiao, X. Shen, Z. Lin, J. Yang, Y. Chen, J. Dong, L. Liu, Z. Jie, J. Feng, and S. Yan. “Video scene parsing with predictive feature learning”. In: *International Conference on Computer Vision (ICCV)*. 2017, pp. 5581–5589.

[44] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan. “Deep learning with S-shaped rectified linear activation units”. In: *AAAI Conference on Artificial Intelligence*. AAAI Press, 2016, pp. 1737–1743.

[45] N. Kalchbrenner, A. v. d. Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu. “Video pixel networks”. In: *International Conference on Machine Learning (ICML)*. 2017, pp. 1771–1779.

[46] A. Karapetyan, A. Villar-Corrales, A. Boltres, and S. Behnke. *Video prediction at multiple scales with hierarchical recurrent networks*. 2022. arXiv: `2203.09303 [cs.CV]`.

[47] T. Kim, S. Ahn, and Y. Bengio. “Variational temporal abstraction”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2019.

[48] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *International Conference on Learning Representations (ICLR)*. 2015.

[49] D. P. Kingma and M. Welling. “Auto-encoding variational bayes”. In: *International Conference on Learning Representations (ICLR)*. 2014.

[50] D. P. Kingma. “Fast gradient-based inference with continuous latent variable models in auxiliary form”. In: *CoRR* abs/1306.0733 (2013).

[51] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber. “A clockwork RNN”. In: *International Conference on Machine Learning (ICML)*. 2014, pp. 1863–1871.

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2012.

[53] S. Kullback and R. A. Leibler. “On information and sufficiency”. In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86.

[54] V. Le Guen and N. Thome. "Disentangling physical dynamics from unknown factors for unsupervised video prediction". In: *Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 11471–11481.

[55] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning". In: *Nature* 521 (2015), pp. 436–444.

[56] Y. LeCun, C. Cortes, and C. J. Burges. "The mnist database of handwritten digits". In: *IEEE Signal Processing Magazine* 10 (1998), p. 34. URL: `http://yann.lecun.com/exdb/mnist`.

[57] A. X. Lee, R. Zhang, F. Ebert, P. Abbeel, C. Finn, and S. Levine. *Stochastic adversarial video prediction*. 2018. arXiv: `1804.01523 [cs.CV]`.

[58] W. Lee, W. Jung, H. Zhang, T. Chen, J. Y. Koh, T. Huang, H. Yoon, H. Lee, and S. H. Lee. "Revisiting hierarchical approach for persistent long-term video prediction". In: *International Conference on Learning Representations (ICLR)*. 2021.

[59] M. Li, Z. Zhou, J. Li, and X. Liu. *Bottom-up pose estimation of multiple person with bounding box constraint*. 2018. arXiv: `1807.09972 [cs.CV]`.

[60] Z. Lin, C. Yuan, and M. Li. "HAF-SVG: Hierarchical stochastic video generation with aligned features". In: *International Joint Conference on Artificial Intelligence*. 2020, pp. 991–997.

[61] W. Liu, W. Luo, D. Lian, and S. Gao. "Future frame prediction for anomaly detection - A new baseline". In: *Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 6536–6545.

[62] J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation". In: *Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3431–3440.

[63] W. Lotter, G. Kreiman, and D. Cox. "Deep predictive coding networks for video prediction and unsupervised learning". In: *ICLR (Poster)*. 2017.

[64] D. G. Lowe. "Distinctive image features from scale-invariant keypoints". In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110.

[65] C. Lu, M. Hirsch, and B. Schölkopf. "Flexible spatio-temporal networks for video prediction". In: *Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2137–2145.

[66] W. Lu, J. Cui, Y. Chang, and L. Zhang. "A video prediction method based on optical flow estimation and pixel generation". In: *IEEE Access* 9 (2021), pp. 100395–100406.

[67] P. Luc. "Self-supervised learning of predictive segmentation models from video". MA thesis. Universite Grenoble Alpes, 2019.

[68] P. Luc, N. Neverova, C. Couprie, J. Verbeek, and Y. LeCun. "Predicting deeper into the future of semantic segmentation". In: *International Conference on Computer Vision (ICCV)*. 2017.

[69] P. Luo and H. F. Li. "Research on quantum neural network and its applications based on Tanh activation function". In: *Computer and Digital Engineering* (2012), pp. 33–39.

[70] Z. Luo, B. Peng, D.-A. Huang, A. Alahi, and L. Fei-Fei. "Unsupervised learning of long-term motion dynamics for videos". In: *Computer Vision and Pattern Recognition (CVPR)*. 2017.

[71] M. Minderer, C. Sun, R. Villegas, F. Cole, K. Murphy, and H. Lee. "Unsupervised learning of object structure and dynamics from videos". In: *Advances in Neural Information Processing Systems (NIPS)*. 2019.

[72] A. Newell, K. Yang, and J. Deng. "Stacked hourglass networks for human pose estimation". In: *European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 483–499.

[73] D. Nilsson and C. Sminchisescu. *Semantic video segmentation by gated recurrent flow propagation*. 2018.

[74] J. Oh, X. Guo, H. Lee, R. Lewis, and S. Singh. "Action-conditional video prediction using deep networks in Atari games". In: *Advances in Neural Information Processing Systems (NIPS)*. 2015.

[75] C. Olah. *Understanding LSTM networks*. 2015. URL: `https://colah.github.io/posts/2015-08-Understanding-LSTMs` (visited on 05/30/2022).

[76] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. "Pixel recurrent neural networks". In: *International Conference on Machine Learning (ICML)*. 2016.

[77] S. Oprea, P. Martinez-Gonzalez, A. Garcia-Garcia, J. A. Castro-Vargas, S. Orts-Escolano, J. Garcia-Rodriguez, and A. Argyros. "A Review on deep learning techniques for video prediction". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* (2020).

[78] V. Patraucean, A. Handa, and R. Cipolla. "Spatio-temporal video autoencoder with differentiable memory". In: *ICLR (Workshop)*. 2015.

[79] A. S. Periyasamy, M. Schwarz, and S. Behnke. "Synpick: A dataset for dynamic bin picking scene understanding". In: *International Conference on Automation Science and Engineering (CASE)*. IEEE, 2021, pp. 488–493.

[80] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C: the art of scientific computing (2nd ed.)* New York, USA: Cambridge University Press, 1992, pp. 123–128.

[81] A. Radford, L. Metz, and S. Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *International Conference on Learning Representations (ICLR)*. 2016.

[82] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra. *Video (language) modeling: a baseline for generative models of natural videos*. 2014. arXiv: `1412.6604 [cs.LG]`.

[83] D. J. Rezende, S. Mohamed, and D. Wierstra. "Stochastic backpropagation and approximate inference in deep generative models". In: *International Conference on Machine Learning (ICML)*. Vol. 32. 2014.

[84] J. Rocca. *Understanding variational autoencoders (VAEs)*. 2019. URL: `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73` (visited on 05/30/2022).

[85] D. Rumelhart, G. Hinton, and R. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536.

[86] B. Sapp and B. Taskar. "MODEC: Multimodal decomposable models for human pose estimation". In: *Computer Vision and Pattern Recognition (CVPR)*. 2013, pp. 3674–3681.

[87] V. Saxena, J. Ba, and D. Hafner. "Clockwork variational autoencoders". In: *NIPS (Poster)*. 2021.

[88] C. Schuldt, I. Laptev, and B. Caputo. "Recognizing human actions: A local SVM approach". In: *International Conference on Pattern Recognition (ICPR)*. 2004.

[89] M. Schwarz and S. Behnke. "Stillleben: Realistic scene synthesis for deep learning in robotics". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 10502–10508.

[90] R. Shi, K. N. Ngan, and S. Li. "Jaccard index compensation for object segmentation evaluation". In: *International Conference on Image Processing (ICIP)*. 2014, pp. 4457–4461.

[91] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 28. 2015.

[92] X. Shi, Z. Gao, L. Lausen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo. "Deep learning for precipitation nowcasting: A benchmark and a new model". In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 30. 2017.

[93] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *International Conference on Learning Representations (ICLR)*. 2015.

[94] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *Journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[95] N. Srivastava, E. Mansimov, and R. Salakhutdinov. "Unsupervised learning of video representations using LSTMs". In: *International Conference on Machine Learning (ICML)*. 2015.

[96] J. Sun, J. Xie, J.-F. Hu, Z. Lin, J. Lai, W. Zeng, and W.-s. Zheng. "Predicting future instance segmentation with contextual pyramid convLSTMs". In: Association for Computing Machinery, 2019, pp. 2043–2051.

[97] D. Sussillo and L. F. Abbott. *Random walks: Training very deep nonlinear feed-forward networks with smart initialization*. 2015. arXiv: `1412.6558 [cs.NE]`.

[98] Z. Tang, L. Luo, H. Peng, and S. Li. "A joint residual network with paired ReLUs activation for image super-resolution". In: *Neurocomputing* (2018), pp. 37–46.

[99] A. M. Terwilliger, G. Brazil, and X. Liu. "Recurrent flow-guided semantic forecasting". In: *Workshop on Applications of Computer Vision*. 2019.

[100] A. Toshev and C. Szegedy. "DeepPose: Human pose estimation via deep neural networks". In: *Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 1653–1660.

[101] R. Villegas, J. Yang, S. Hong, X. Lin, and H. Lee. "Decomposing motion and content for natural video sequence prediction". In: *International Conference on Learning Representations (ICLR)*. 2017.

[102] R. Villegas, A. Pathak, H. Kannan, D. Erhan, Q. V. Le, and H. Lee. "High fidelity video prediction with large stochastic recurrent neural networks". In: *Advances in Neural Information Processing Systems (NIPS)*. 2019, pp. 81–91.

[103] R. Villegas, J. Yang, Y. Zou, S. Sohn, X. Lin, and H. Lee. "Learning to generate long-term future via hierarchical prediction". In: *International Conference on Machine Learning (ICML)*. 2017, pp. 3560–3569.

[104] J. Walker, K. Marino, A. Gupta, and M. Hebert. "The pose knows: Video forecasting by generating pose futures". In: *International Conference on Computer Vision (ICCV)*. 2017.

[105] Y. Wang, Z. Gao, M. Long, J. Wang, and P. S. Yu. "PredRNN++: Towards a resolution of the deep-in-time dilemma in spatiotemporal predictive learning". In: *International Conference on Machine Learning (ICML)*. 2018, pp. 5123–5132.

[106] Y. Wang, M. Mingsheng Long, J. Wang, Z. Gao, and P. S. Yu. "PredRNN: Recurrent neural networks for predictive learning Using spatiotemporal LSTMs". In: *Advances in Neural Information Processing Systems (NIPS)*. 2017, pp. 879–888.

[107] Y. Wang, H. Wu, J. Zhang, Z. Gao, J. Wang, P. S. Yu, and M. Long. "PredRNN: A recurrent neural network for spatiotemporal predictive learning". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* (2022).

[108] Y. Wang, J. Wu, M. Long, and J. B. Tenenbaum. "Probabilistic video prediction from noisy data with a posterior confidence". In: *Computer Vision and Pattern Recognition (CVPR)*. 2020.

[109] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing*. Vol. 13. 4. 2004, pp. 600–612.

[110] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. "Convolutional pose machines". In: *Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4724–4732.

[111] B. Wu, S. Nair, R. Martín-Martín, L. Fei-Fei, and C. Finn. "Greedy hierarchical variational autoencoders for large-scale video prediction". In: *Computer Vision and Pattern Recognition (CVPR)*. 2021.

[112] T. Xue, J. Wu, K. L. Bouman, and W. T. Freeman. "Visual dynamics: probabilistic future frame synthesis via cross convolutional networks". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[113] J. Yang and G. Yang. "Modified convolutional neural network based on dropout and the stochastic gradient descent optimizer". In: *Algorithms* 11.3 (2018).

[114] Y. Yang and D. Ramanan. "Articulated human detection with flexible mixtures of parts". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* (2013).

[115] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. "The unreasonable effectiveness of deep features as a perceptual metric". In: *Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 586–595.

[116] Y. N. Zhang, L. Qu, J. Chen, J. Liu, and D. Guo. "Weights and structure determination method of multiple-input Sigmoid activation function neural network". In: *Application Research of Computers* 29 (2012), pp. 4113–4116.