Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische
Friedrich-Wilhelms-Universität Bonn

Master thesis

Video Prediction Based on Temporal Hierarchies and Recurrent Neural Networks

Author:
Niloofar Azizi

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Prof. Dr. Christian Bauckhage

Advisor:
Hafez Farazi
Seongyong Koo

Submitted: June 8, 2018

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

Robots in the environments interacting with humans need to be able to predict future changes of the scene to make a proper decision and act on it. Within the literature, the task of generating future images based on a sequence of past frames is called video prediction. Video prediction requires analyzing the video taken from environment temporally and spatially and then constructing a model of how the environment evolves over time. Different approaches have been proposed to predict future frames, but the most successful ones are based on artificial neural network architectures. For example, video pixel network(VPN) by encoding the time, space and color structure of video tensors as a dependency chain, reported state of the art result over the Moving MNSIT dataset. Video ladder network(VLN), a neural encoder-decoder model, augmented at all layers by both recurrent and feedforward lateral connections, yields a competitive result to state of the art with simpler architecture. However, applying deep learning techniques still involves various difficulties such as inherent blurriness and long-term coherency. Especially, long-term coherency cannot be considered in VPN and VLN due to the memory-intensive backpropagating errors over many times steps. To address this problem, we propose a new method to learn temporal features by encoding the temporal dependencies with different timescales for the video prediction task. By applying both spatial and temporal pooling, we can load more over time to memory and therefore we can model long-range dependencies both within and across video frames. Furthermore, the hierarchy allows us to extract features which change slowly over time, which helps us gain a better understanding of the video over long time horizons.

We also employed semantic segmentation based on the idea of multitask learning. The advantage of multitask learning in convolutional neural networks is that multiple types of supervision lead to better performance with the same input [**stitch**]. In addition to an improved performance and the extraction of useful high-level understanding of the video in shape of semantic segmentations, we tested whether semantic segmentations can help alleviate the inherently blurry prediction problem in video prediction[57].

Furthermore, we worked on the predictive gating pyramid(PGP) architecture[43], which its primary goal is to learn and predict transformation between frames. It uses a bilinear model to extract features temporally. We proposed a fully convolutional PGP(Conv-PGP) which not only reduces the number of parameters significantly but also extracts more spatial features.

Contents

List of Figures

1 Introduction

1.1 Motivation

One of the main difficulties with the autonomous robots, especially the ones operating in the human environment, is anticipation of the scene. Also, in self-driving vehicles, machines need to reason beyond the present and predict the future frames to react properly in the real world [76].

In these applications, we need to analyze the video received from the environment and train the agent to understand the scene. Recent experiments demonstrate that the task of video prediction helps to extract more useful features in comparison to the classical way of analyzation and reconstruction of the frames in a video[57]. The learning of internal video representations during the task of video prediction support collaborative robots in their action decision process in accordance with human actions on the shared environment.

Deep learning approaches made significant improvement in different fields of computer graphics including the task of video prediction. Although the task of video prediction has been studied by many researchers, still learning to predict future frames poses many challenges. One of the main difficulties is blurriness of the frames due to the inherent stochasticity in the highly dynamical environments. It is not only inherently challenging to train neural networks over many time steps due to high resource demands, but also subtle errors can easily add stepwise and quickly lead to divergence. Moreover, recurrent networks tend to forget the recent past quickly.

Furthermore, current methods predict in the pixel-level using approaches like GANs[40], VAE[98] or PixelCNN[92]. Pixel-level prediction is semantically uninterpretable. To utilize video forecasting for representation learning, higher level of abstraction is required[108].

This thesis strives to fix the issues addressed above. The main problem is outlined in the upcoming section.

1.2 Problem Definition

Video prediction is the task of generating future frames based on a sequence of past frames. By predicting the future frames of a video sequence, the internal representation, which models the image evolution, is constructed through learning the content and the dynamics of the video frames[57].

As mentioned previously, uncertainty is the nature of the future. Thus, to perform video prediction we need to deal with the inherent stochasticity of the video which leads to blurriness. To overcome this issue in pixel-level prediction, different approaches have been proposed. Use of adversarial loss and also use of generative models are among the promising ones[108]. Adversarial loss is important because theoretically solves the blurriness problem addressed in L2 and L1 norm losses [57]. Most recent experiments which predict pixels directly utilize recurrent neural network, especially LSTMs to extract features temporally[108]. Despite of the attempts, still the problem of blurriness in the task of video prediction is unsolved especially for non-synthetic datasets.

Moreover, the second main difficulty in video prediction is that it has semantically uninterpretable outputs as they are pixel-level prediction.

1.3 Approach

To do the task of video prediction we need to extract features both temporally and spatially. To this end, we improved and compared two existing algorithms: Video Ladder Network(VLN) and Predictive Gating Pyramid(PGP).

VLN, a neural encoder-decoder model, augmented at all layers by both recurrent and feedforward lateral connections, yields a competitive result to state of the art with simpler architecture. To extract more temporal features in VLN, we proposed the use of temporal hierarchies in the recurrent neural network to ease training over many time steps [77]. In addition, the predicted frames in VLN are semantically uninterpretable. To deal with this issue, we proposed applying the task of segmentation with the VLN architecture.

Furthermore, we proposed the fully convolutional PGP(Conv-PGP) [43]. The PGP architecture, which its primary goal is to predict transformation between frames, uses a bilinear model to extract features temporally. This modification not only reduced the number of parameters significantly but also extracted .

1.4 Structure of Thesis

The structure of the thesis will be as follows

- Chapter 2 reviews the related work on the field of video prediction, multiscale recurrent neural network, multitask learning and video segmentation
- Chapter 3 describes prerequisites for designing neural network architecture for the task of video prediction.
- Chapter 4 presents the detailed view of Video Ladder Network(VLN) and Predictive Gating Pyramid(PGP) architecture and discusses the advantage and disadvantage of both methods, which are the baseline of this thesis.
- Chapter 5 outlines the proposed modifications to PGP and VLN. First, the fully connected architecture of PGP is changed to fully convolutional architecture. Second, Time pooling, segmentation of frames, position and velocity of objects in frame is explored with VLN baseline architecture.
- Chapter 6 specifies the datasets and the environments we evaluate our experiments. It also explicates the experiments followed by their results and comparisons. It also does the comparison with results of state-of-the-art approaches.
- Chapter 7 encloses the thesis by representing the conclusion with subsequent limitations and future work.

2 Related Work

2.1 Feature Extraction

Prior works on learning to predict videos have investigated different learning methods. Early researches concentrated on learning specific features. Generally, all feature learning models can be represented as bipartite network, which maps a set of latent variables to a set of observable variables[36]. First attempts in the task of video prediction consisted of detecting Gabor-like feature representations but it could not tolerate changes in motion, lighting, and other aspects. To achieve invariant representation hierarchical feature extraction were introduced but they could not extract transformation features. To be able to extract transformation features, multiple independent groups of hidden variables were represented. One of the approaches was bilinear models which contained two groups of hidden variables. It preserved transformation and invariance information. Grimes and Rao[18], based on the earlier work of Freeman and Tenenbaum [10], proposed an unsupervised algorithm for learning localized features and their transformations simultaneously from images. In this algorithm, multiplicative interactions between pixels were used to represent correlation patterns across multiple images. The drawback of this architecture was that it could not learn the temporal features but it learns just shifted set of features over time. Olshausen et al. [23] introduced a completely unsupervised bilinear model to extract both invariance and transformation features in natural images temporally. They addressed interpolating among the feature descriptors coefficients via phase shifting and by conditioning transformed images on untransformed ones.

Memisevic and Hinton [26] introduced a conditional factorized model with a single group of hidden variable, where the feature extractors behaved as deterministic functions. It was based on the restricted Boltzmann machine[22], which addressed modeling the transformation between two successive images based on the three-way multiplicative interactions which raised a cubic computational order. To reduce the computational order Memisevic and Hinton [26], introduced sum of factors, each factor could be represented as image filter which could extract specific transformation features.

Michalski et al.[43] designed a recurrent network based on gated autoencoder, a bilinear transformation model, to learn transformations specifically rotation angles between pairs of consecutive images ([36], [26]). To predict next future frames by learning latent variables in cascading manner, they proposed a bi-linear model of transformations which were applied to pairs of frames ($\text{frame}_{t-1}, \text{frame}_t$) to predict frame_{t+1}. Particularly, by stacking multiple layers of the transformation model, the PGP architecture learned transforms between transformations[43].

2.2 Architecture Design

The task of video prediction, in case of enough training data and also a reasonable architecture, can be solved. Therefore researchers instead of dealing with selecting features, started shifting the paradigm to design the architecture. To this end, we need to extract features both spatially and temporally and merge them in three dimension properly.

2.2.1 State of the Art

Different approaches have been explored. Ranzato et al. proposed a baseline for video prediction inspired by language models [44]. Walker et al. [98] proposed using variational autoencoder to encode any necessary information that is not available in the image. Bhattacharyya et al. [75] predicted image boundary extrapolation of future frames to improve the sharpness of the predictions.

Mathieu et al. [57] proposed gradient-based loss function, as well as a loss function based on adversarial training [40] for sharper frame predictions. In generative adversarial training(GAN) [40], using a discriminative network to distinguish between a sample from a dataset and the result of a generative model, force the generative model to generate frames as much as close to samples of the dataset. This method makes the discriminator not be able to discriminate between frames better than 50% chance. Though the combination of regression and adversarial losses causes improvement in comparison to regression models which leads to the mean, still the problem of sticking to one mode of the distribution and disregarding the other modes exist [66].

Fragkiadaki et al.[102] propose an architecture based on the variational autoencoder to learn samples of the model which corresponds to future motion trajectories of the objects in frame. This approach is beneficial for the models with uncertainty beyond motion, especially when the number of conditioning frames is little.

Kalchbrenner et al. [87] address the issue of stochasticity, based on PixelCNN architecture [91]. PixelCNN computes the discrete probability of the raw RGB pixel values based on the computed probability of its left and up pixels followed by a softmax layer. In VPN architecture same as PixelCNN, the authors propose a probabilistic video model that estimates the discrete joint distribution of the raw pixel values in a video. The model and the neural architecture reflect the time, space, and color structure of video tensors and encode it as a four-dimensional dependency chain [87]. VPN largely solves the blurriness problem by predicting multimodal distributions over pixel intensities and by sampling from these distributions conditioned on samples for the neighboring pixels. Downsides of VPN are that it needs a lot of computational resources, especially at inference time, and it neither solves the problem of long term predictions nor does it provide an interpretable intermediate representation of the moving objects.

Finn, goodfellow, and Levine [81] predict object motions instead of predicting pixels directly. The geometric transformations of objects in the scene are predicted by using Dynamic Neural Advections (DNAs), and the video pixel value is then computed under the motion estimates. However, the method is limited to the applications which require following conditions: relatively simple environment, static background, and slow moving objects. Using masks for changing image parts both reduces blurriness and provides a largely explicit representation of the predicted changes in the shape of geometric transformations which is potentially useful for other tasks such as object classification, tracking, and localization. However, the method is limited only to affine transformations and flow fields for constructing the next frame, thus, for example, complex deformations, illumination changes and objects moving into view cannot be predicted accurately or not at all.

Villegas et al.[105] decompose motion and content for video sequence prediction. These two architectures are trained simultaneously with one teacher and in an unsupervised way. The motion prediction architecture contains three convolutional LSTMs to identify local dynamics of frames rather than complicated global feature tensor encoding motion. For the content prediction they simply propose convolutional LSTM for the current frame. This architecture achieves state of the art in real videos. Watters et al. [109] extend objects of frames and maps them to the list of vectors. By analyzing the list of vectors, motion features are extracted and then based on that the next frame is predicted.

Cricri et al.[79] proposed *video ladder network* (VLN) by adding recurrent connections to *ladder network* [62]. Ladder network extracts latent hidden variables by lateral connections using semi-supervised architecture. Similar to ladder network, VLN employs shortcut connections from the input to the output whereby it relieves the deeper layers from modeling details such as textures, as well as static parts of the scene. Instead, the deeper layers can expand more model capacity for modeling the semantic contents of the video. VLN is a fully convolutional encoder-decoder network, where both the encoder and decoder blocks are feed-forward neural networks. To reduce computational efforts, the resolution is reduced when going up in the model hierarchy. The idea of hierarchy is not new and before was proposed by Behnke [15] to represent images at multiple abstraction levels. In [15] the image was passed as input to the architecture and in each level the spatial resolution was decreased which helped to increase the feature diversity and invariance. It could help in case of noise representation, low image contrast, partially occluded objects, and complicated interpretations.

VLN uses residual connections between convolutional layers to improve the results of predicted frames. Residual connections[85] addresses the problem of degradation in deep neural networks. Degradation problem means though by adding more layers we expect to get at least as good result as before, the results become worse. This is because of learning $f(x) = 0$ is easier for the network than learning $f(x) = x$. By representing few stacked layer by $H(x)$ and also hypothesizing that multiple nonlinear layers can asymptotically solve complicated functions then we can conclude that it can solve $H(x) - x$ where x denotes the input to the first layer. If we suppose that added layers play role of identity mapping then residual learning can be adopted to every few stacked layers.

$$F(x) = H(x) - x \tag{2.1}$$

Which is equivalent to

$$H(x) = F(x) + x \tag{2.2}$$

Wagner et al. [107] proposed a similar idea to VLN architecture and changed a feed-forward network to a predictive model using *teacher-student*[48] approach. Teacher-student [48] is similar to Buciluă et al.[19] approach but uses *distilling* method for compression. As mentioned in 3.3 , ensembling methods are set of classifiers that learns classifying over the dataset independently and then the final result in the test dataset is the average over the independent weighted classifiers prediction[14]. This approach though it is very simple, it is computationally quite expensive and also memory consumptive. Distillation model compress a Deep CNN in a much smaller network with a reasonable approximation, which results as same as the original one. The compressed model (student) in teacher-student approach instead of being trained on the raw data directly, is trained to mimic the output of the teacher network. Wagner et al.[107] similar to teacher-student approach trains the network at each layer independently and with different losses to optimize the weights of the model. Their results represent that the architecture can learn dynamics of the model.

2.2.2 Multiscale Recurrent Neural Network

First attempts in multiscale recurrent neural network done by Schmidhuber[5] and El Hihi and Bengio[9] to deal with the issue of computational and learning efficiency. They proposed stacking multiple RNNs with lower updating frequency in upper layers. Schmidhuber introduced a self organizing hierarchical multiscale structure while El Hihi and Bengio proposed layer-wise updating with a fixed value but a different rate for different layers[77].

LSTMs[11], most popular RNNs, applies multiscale recurrent neural network implicitly based on the fact that each hidden layer contains its own forget and update gate, but this architecture is not hierarchically organized. The clockwork RNN(CW-RNN) [42] similar to El Hihi and Bengio[9] propose using hard timescales to resolve this issue in LSTMs. Clockwork architecture [42] deals with the vanishing problem in ConvLSTM. It proposes partitioning the hidden layer into separate modules, each processing its input temporally independent. As shown in figure 2.1, the architecture contains input, output, and hidden layers. Hidden layers are broken into k independent modules and each of them works with its own temporal clock T_k. The input layer is connected to all hidden layers and the connection between hidden layers are from the ones with lower clock rate to the ones with higher clock rate. module i has a clock period of $T_i = 2^{i-1}$.

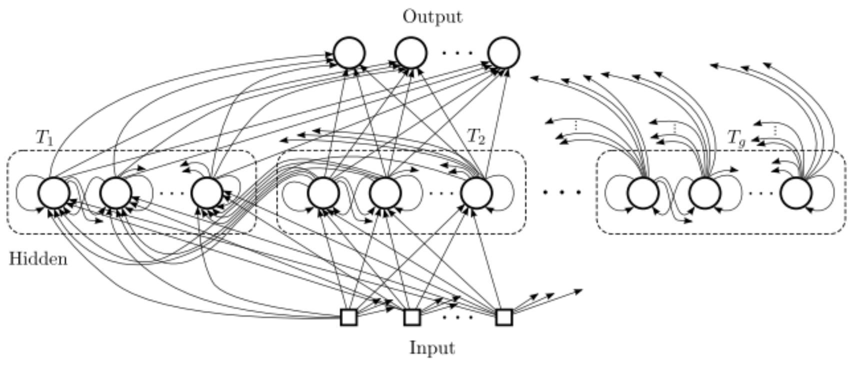


Figure 2.1: Clockwork-RNN Architecture, Source:[42]

This architecture performs better than LSTM but still finding a proper clock rate is an issue which recently considered by Chung et al. [77]. They propose using another gate which adaptively learns layer-wise timescale.

2.2.3 Multitask Learning

The advantage of multitask learning in convolutional neural networks is that multiple tasks lead to better performance with the same input [90]. In general, the goal of multitask learning(MTL) is to learn related tasks at the same time, hoping the auxiliary tasks benefit each other and improves their results[24]. In MTL architectures, the auxiliary tasks play the role of inductive bias for each other. In inductive bias, we want to make the hypotheses space small enough to have reliable generalization and large enough to contain the solution the network is trained for [13]. MTL parameters shared in hidden layers are either soft parameter sharing or hard parameter sharing. In hard parameter sharing[6], the hidden layers are shared between all tasks, while keeping several task-specific output layers. It helps to reduce the risk of overfitting. In soft parameter sharing each task has its own hidden layers but the distance between parameters of layers are regularized in order to make tasks similar.

Doing the multitask learning implicitly is equivalent to augment more data to the model, as it learns more generalized features in comparison to just having one task. In other words, tasks play the role of regularization for each other. In case of limited data for one task, doing MTL helps the task to learn useful features based on the other task and helps to distinguish between the noise and features for the task[103]. The other benefit gained in doing MTL is learning features which are difficult for one task to be learned but it is easy for the other one to learn it.

One of the recent successful MTL[55] approaches proposes sharing the convolutional layers which forces the model to learn the relation between tasks. The main problem though with this architecture is that it needs predefined sharing structure which is not adequate for all computer vision tasks. The other MTL approach introduced by Lu et al. starts with thin neural network architecture and dynamically increases it during training. The main issue of this architecture is that it does not return the optimum architecture.

The main issue with MTL is finding good tasks which can be learned together. The research done by Alonso and Plank concludes auxiliary tasks with compact and uniform label distributions returns better results for sequence tagging problems in Natural language processing(NLP), which has similar results with experiments done by Ruder et al. Limited scope research also represents that main tasks that quickly plateau with non-plateauing auxiliary tasks return promising results [101].

The proposed approaches in [69], [72], [58], and [54] reach state of the art in semantic segmentation by providing pixel-wise labels. The problem with using these approaches which returns nearly optimal results for the task of semantic segmentation is that it does not use the embedded information over time, which is needed for the task of video segmentation. To gain information temporally, semi-supervised approaches which propagate labels of annotated frames to the entire video are proposed [25], [61], [38]. Pavel et al.[60] to do object class segmentation suggested a recurrent neural network architecture. However, training their proposed architecture was difficult due to vanishing gradient. Furthermore, it could not deal with large images due to the significant increase of number of parameters. Siam et al.[96] utilized a fully convolutional network through convolutional gated recurrent networks to segment frame at time t based on the information in its preceding frames.

3 Theoretical Background

3.1 Neural Network

The formal definition of artificial neural network(ANN) is a "directed graph with the following properties: a state variable n_i is associated with each node i, a real valued weight w_{ik} is associated with each link (ik) between two nodes i and k, a real valued bias v_i is associated with each node i, and a transfer function $f_i[n_k, w_{ik}, v_i, (i \neq k)]$ is defined for each node i, which determines the state of the node as a function composed of its bias, the weights of incoming links, and the states of nodes connected to it."[8]. The network which contains hidden layer(s) in addition to the input and output layer and bounded and non-constant activation function is universal approximator function [4]. In this chapter, the principals of artificial neural network is defined.

3.2 Learning and Backpropagation

Learning in ANNs for a specific task looks for minimum of an objective function[47]. It sets the neural network's parameters θ based on the training dataset, which minimize a cost function $J(\theta)$. The cost function measures the performance which is evaluated on training set and additional regularization terms[83]. The test and validation dataset are used to make sure that the network is generalized. They should have the same probability distribution as training dataset.

One of the most used approaches for computing the train loss is *binary cross entropy*[57].

$$L_{bce}(Y, \hat{Y}) = -\sum_i \hat{Y}_i log(Y_i) + (1 - \hat{Y}_i) log(1 - Y_i) \tag{3.1}$$

where Y_i takes its values in $\{0, 1\}$ and $\hat{Y}$ is in range of $[0, 1]$.

Softmax cross entropy is usually used to compute segmentation loss. To compute cross entropy for softmax function with C classes from $c = \{1 \ldots C\}$ and the correct label of $t = c$ for the given input z, the multi-class categorical output probability distribution by the softmax function ζ is defined as

$$y_c = \zeta(z)_c = \frac{e^{(z_c)}}{\sum_{d=1}^{C} e^{(z_d)}} for \quad c = \{1 \dots C\} \tag{3.2}$$

which can be reformulated to

$$y = \begin{bmatrix} P(t=1|z) \\ \vdots \\ P(t=C|z) \end{bmatrix} = \frac{1}{\sum_{d=1}^{C} e^{(z_d)}} \begin{bmatrix} e^{(z_1)} \\ \vdots \\ e^{(z_C)} \end{bmatrix} \tag{3.3}$$

Computing the cost function for the softmax function for the parameters θ based on likelihood function $\mathcal{L}$ is:

$$argmax_{\theta} \mathcal{L}(\theta|t,z). \tag{3.4}$$

where the likelihood $\mathcal{L}(\theta|t,z)$ is equal to compute joint probability of $P(t,z|\theta)$

$$P(t,z|\theta) = P(t|z,\theta)P(z|\theta) \tag{3.5}$$

If we suppose that θ is fixed, then:

$$P(t|z) = \prod_{i=c}^{C} y_c^{t_c}. \tag{3.6}$$

As maximizing likelihood is equivalent to minimizing log-likelihood, we have:

$$-log \prod_{i=c}^{C} y_c^{t_c} = -\sum_{i=c}^{C} t_c log(y_c) \tag{3.7}$$

which is the cross entropy function.
Backpropagation is a method which is used to compute gradient based on the loss function through recursive application of chain rule with respect to all training variables of the model. For two inputs x_1 and x_2 and generating activation functions y_1, y_2 and y_3, as represented in Figure 3.1, the error E is the difference between the ground-truth and output of the network y_3. It is backpropagated through the network using the concept of chain-rule.

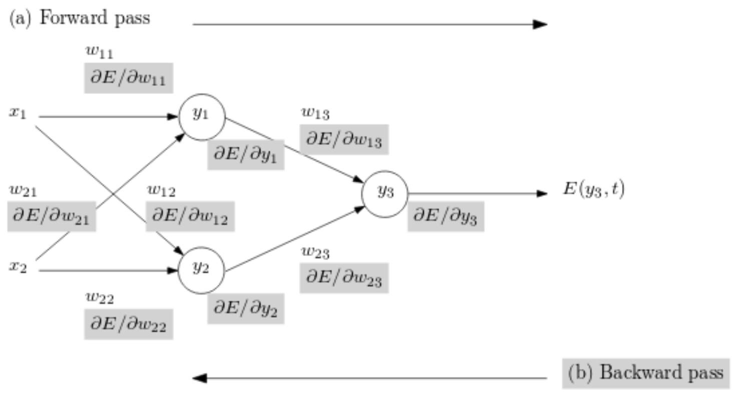

(a) Forward pass
w_{11}
$\partial E/\partial w_{11}$
x_1
y_1
w_{13}
$\partial E/\partial w_{13}$
$\partial E/\partial y_1$
w_{21}
$\partial E/\partial w_{21}$
w_{12}
$\partial E/\partial w_{12}$
y_3
$E(y_3, t)$
$\partial E/\partial y_3$
x_2
y_2
w_{23}
$\partial E/\partial w_{23}$
w_{22}
$\partial E/\partial w_{22}$
$\partial E/\partial y_2$
(b) Backward pass

Figure 3.1: Backpropagation, Source:[47]

Once the backpropagation is applied and the gradients are computed, a gradient based optimization method is used to update the weights. The most common optimizers are the ones with adaptive learning rates. The learning rate is a scalar which sets the size of the step. RMSProp[33] and Adam[41] are among the most effective and practical ones. After computing the *backward pass* we can compute through the network during *forward pass* [47].

Learning Paradigms

We have three major learning paradigms, supervised, unsupervised, and semi-supervised. In *supervised learning*, the dataset contains features and the preferred label for each example[83]. The algorithm analyses the training data and learns a generalized function which infers the correct label for the unseen input data. On the other hand, in *unsupervised learning* task, it extracts features and there is no supervision signal [83]. *Semi-supervised learning* is supervised learning task which takes advantage of hidden features extracted with unsupervised learning methods.

Activation Function

A neuron's function $f(x)$ is the summation of its weighted input functions $g_i(x)$ and the bias b; $f(x) = A(\sum_i(w_i g_i(x)) + b)$ where A is the activation function. Each neurons activation function represents the presence of a specific feature[83]. The most used activation functions are sigmoid, hyperbolic tangent, ReLU, and LReLU. If we suppose x as input then

$$ReLU : f(x) = max(0, x) \tag{3.8}$$

$$LReLU : \begin{cases} \text{f(x)} & \text{if x ¿ 0} \\ \text{ax} & \text{otherwise; where } 0 \leq a \leq 1 \end{cases} \tag{3.9}$$

$$Sigmoid(x) : \frac{1}{1+e^{-x}} \tag{3.10}$$

$$tanh(x) : \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{3.11}$$

LReLU and ReLU are the most preferred activation functions.

3.3 Regularization

The goal of the neural network is to solve the problem on not previously seen data, based on the dataset which is trained on[83], so the network should be able to generalize well. *Regularization* techniques are used to avoid overfitting during training. One of the most used approaches of regularization is the dropout [46], in which, a specific percentage of the activation functions are set to zero during training. Dropout is similar to bagging which has k different models and k different dataset and each model is trained over one specific dataset [83]. Dropout makes the network learn relevant computational information in multiple activation functions by setting a set of random nonlinearity functions to zero which causes a more robust computational graph.

In some cases, batch normalization can also be used as a regularizer, which obviates applying dropout[49]. Formally, batch normalization for the layers $l = \{1 \dots L\}$ is formulated by

$$z^{(l)} = N_B(W^{(l)}h^{(l-1)}) \tag{3.12}$$

where N_B is a component-wise batch normalization

$$N_B(x_i) = (x_i - \hat{\mu}_{x_i})/\hat{\sigma_{x_i}} \tag{3.13}$$

and $h^{(l)}$

$$h^{(l)} = \phi(\gamma^{(l)}(z^{(l)} + \beta^{(l)})) \tag{3.14}$$

where $\beta^{(l)}$ and $\gamma^{(l)}$ are trainable variables and ϕ is the activation function [67].

$$L_{reg}(x, x') = L(x, x') + \lambda_1 w_1 + \lambda_2 w_2^2 \tag{3.15}$$

where λ_1 and λ_2 control the effect of regularizers in the total loss.

Variants of the artificial neural network include *Convolutional neural network* (CNN) and *Recurrent neural network*(RNN).

3.4 CNN

Neurons in a *fully-connected layer* have full pair-wise connections in adjacent layers but contain no connection within a layer. Fully-connected layer causes an increasing number of parameters, which is wasteful.

CNN architecture, unlike fully connected layers, have a $3D$ volume of neurons: width, height, and depth. This $3D$ volume takes advantage of being connected to small regions of its previous layer, which is called *receptive field*, but still fully connected in depth. CNN has three specific hyperparameters; depth, stride and padding. *Depth* represents the number of filters which are used to extract different features in the input. *Stride* specifies the sliding factor of CNN and *padding* is responsible for padding the input volume with zeros around the border, which gives the advantage of controlling the size of the output volumes. In CNN the *parameter sharing* scheme is used to reduce the number of parameters based on the assumption that if one feature is useful at position (x_1, y_1), it is also useful at position (x_2, y_2). Considering that if all neurons in one depth using same weights then the forward pass can be interpreted as a convolution of the neuron's weights with the input volume[83]. *Dilated Convolution*[71] is a specific type of convolution which has one extra hyperparameter named *dilation*. Dilation represents space between each cell within a filter. Formally, if we suppose a discrete function $F : Z^2 \rightarrow R$ and $k : \Omega_r \rightarrow R$, where $\Omega_r = [-lr, lr]^2 \cap Z^2$, is a discrete filter of size $(lr + 1)^2$, then the discrete convolution $*_l$ is defined as

$$(F *_l k)(p) = \sum_{s+lt=p} F(s)k(t) \tag{3.16}$$

In this thesis, we also used another specific type of convolution named 1×1 *Convolution* proposed by Lin et al. [35]. In CNN as we use $3D$ volumes, such a filter extend through the full depth of the input volume. The method can be applied to change the number of channels to any arbitrary number of channels.

3.5 RNN

In many cases, we have to deal with *sequential data*. The RNNs[2] are designed to extract features temporally. In the RNNs, the parameters are shared [83]. Due to feedback connections temporal features can be stored in the form of activation functions. A simple RNN is represented in Figure 3.2.

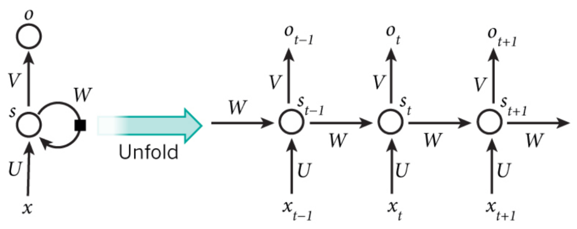
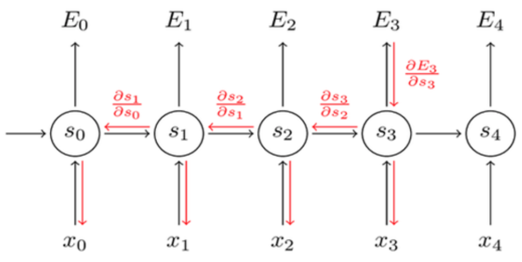


Figure 3.2: RNN Structure, Source:Nature

Where s_t contains the summary of past sequence of inputs, up to time t. U, V, and W are weights between input and the state, state and the output, and two hidden states respectively.

To backpropagate in RNN, the algorithm of backpropagation through time(BPTT) is applied, which computes the derivatives with respect to weights of the network overtime.



E_0
E_1
E_2
E_3
E_4
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4

Figure 3.3: BPTT Structure, Source: [110]

The main problem with basic RNNs is that the weights either vanish or explode very fast[11]. In case of explosion, weights oscillate and in case of vanishing, convergence either happens very slowly or never happens. To deal with these issues different approaches were proposed. Echo state networks[21] propose setting the recurrent and input weights W and U slightly close to 1 such that the recurrent hidden units capture the history of the past inputs and just learn the output weights V. The other proposed method is *long delays* which does BPTT every $d > 1$ steps which delays the problem for dT times. The most common method which is used recently is based on the *gating* mechanism. *long short term memory (LSTM)* is one of the most important gating RNNs.

3.5.1 Fully Connected LSTM

In fully connected LSTM(FC-LSTM)[34], not only the hidden state was used but also the cell state was proposed, which helps the network to deal with vanishing or exploding problem.

In the first step, the forget gate decides how much of the data should be eliminated from the cell state based on the results of sigmoid activation function over the input x_t and history h_t. It outputs a number between 0 and 1. The more the value is close to 1 the more information in the cell state is kept.

In the second step, the FC-LSTM decides about the new information which should be augmented to the cell state. This phase follows two parts in parallel. The first part contains a sigmoid layer called *input gate layer* which decides about the values we update and the second part is hyperbolic tangent layer which creates new candidate values that can be augmented to the state. Finally the two steps are merged based on *Hadamard product.*

Finally, the LSTM needs to decide which parts of cell state should be returned via the output gate which is the sigmoid activation function applied over input x_t and hidden state h_{t-1}. Also, hyperbolic tangent is applied over the cell state to push the cell state in the range of -1 to 1. Then the output of hidden state will be the Hadamard product of the output gate and the output of hyperbolic tangent layer. The formulation of FC-LSTM is:

$$i_t = \sigma(x_t W_{xi} + h_{t-1} W_{hi} + b_i), \tag{3.17}$$

$$f_t = \sigma(x_t W_{xf} + h_{t-1} W_{hf} + b_f), \tag{3.18}$$

$$\tilde{c}_t = \tanh(x_t W_{x\tilde{c}} + h_{t-1} W_{h\tilde{c}} + b_{\tilde{c}}), \tag{3.19}$$

$$c_t = f_t c_{t-1} + i_t \tilde{c}_t, \tag{3.20}$$

$$o_t = \sigma(x_t W_{xo} + h_{t-1} W_{ho} + W_{co} c_t + b_o), \tag{3.21}$$

$$h_t = o_t \tanh c_t \tag{3.22}$$

i_t, f_t, and o_t are the input gate, forget gate and the output gate at time t respectively. The hidden state and the cell state are represented as c_t and h_t respectively. h_{t-1} is the previous hidden state at time $t-1$. W_{xi}, W_{xf}, W_{xo}, W_{hi}, W_{hf}, W_{ho}, and $W_{x\tilde{c}}$ are convolutional kernel tensors. b_i, b_f, b_o, and $b_{\tilde{c}}$ are bias terms. σ and *tanh* are sigmoid and tangent hyperbolic activation functions.

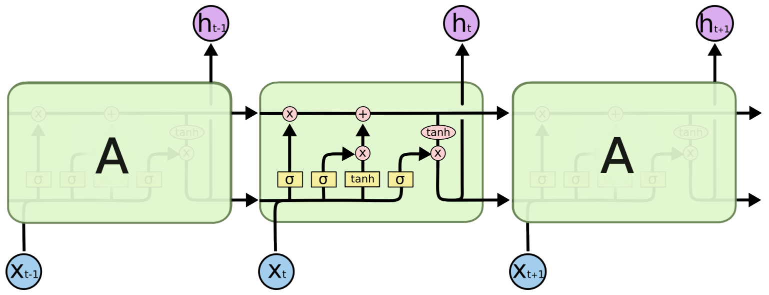


Figure 3.4: LSTM Architecture, Source:[111]

3.5.2 Convolutional LSTM

Convolutional LSTM [70] proposed by Xingjian et al., has same structure as FC-LSTM [34] except that the main problem of not encoding information over space is solved by replacing inputs in three dimensions where the second and third dimensions are rows and columns instead of full connections between input-to-state and state-to-state transitions. Therefore, the formulation becomes as

$$i_t = \sigma(x_t * W_{xi} + h_{t-1} * W_{hi} + b_i), \tag{3.23}$$
$$f_t = \sigma(x_t * W_{xf} + h_{t-1} * W_{hf} + b_f), \tag{3.24}$$
$$o_t = \sigma(x_t * W_{xo} + h_{t-1} * W_{ho} + b_o), \tag{3.25}$$
$$\tilde{c}_t = \tanh(x_t * W_{x\tilde{c}} + h_{t-1} * W_{h\tilde{c}} + b_{\tilde{c}}), \tag{3.26}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{3.27}$$
$$h_t = o_t \odot \tanh(c_t) \tag{3.28}$$

which allows the network to preserve the spatial information. $\odot$ is the element-wise multiplication and $*$ is the convolution operator.

3.6 Autoencoder

An autoencoder is a kind of ANN which learns to copy its input to its output. It can be interpreted as two networks; encoder network and the decoder network. Formally, the deterministic autoencoder is

$$h = f(x) \tag{3.29}$$
$$r = g(h) \tag{3.30}$$

where the hidden layer h represents the coded input[83] and r represents the reconstruction. The method can be expanded to stochastic mappings for the encoder and the decoder part $p_{encoder}(h|x)$ and $p_{decoder}(x|h)$. The loss function, L, is defined as

$$L(x, g(f(x))) \tag{3.31}$$

Variants of autoencoders include denoising autoencoder and variational autoencoder.

The input to denoising autoencoder is corrupted by some noise and represented by $\tilde{x}$. Therefore the loss function is defined as

$$L(x, g(f(\tilde{x}))) \tag{3.32}$$

Adding the noise makes f and g to learn the structure of $p_{data}(x)$[83].

Variational autoencoder(VAE) is another type of autoencoder which is used to regularize autoencoder by imposing an arbitrary prior on the latent representation of the autoencoder [56]. Basically VAE wants to find a solution for $p(z|x)$. To this end, the objective function can be represented as minimizing Kullback–Leibler divergence between $q(z)$ and $p(z|x)$.

4 Approach

We have chosen the VLN architecture as a starting point for our research. VLN has a competitive result with respect to VPN which is the state of the art architecture for the Moving MNIST dataset but with reduced a number of parameters. VLN learns to predict frames by extracting both spatial and temporal features.

We also investigated PGP architecture. PGP relies on a bi-linear model which learns explicit transformations. The advantage of PGP in comparison to other spatiotemporal prediction models within the field of deep learning is that hidden states can be monitored[43].

In the first two subsections, the complete PGP and VLN models are presented respectively. In the next two subsections the Conv-PGP and VLN improvements are described.

4.1 PGP

4.1.1 PGP Architecture

PGP is designed based on the fact that two temporally consecutive frames can be interpreted as a linear transformation of each other. The linear transformation, L, between two frames x_t and x_{t+1} can be represented as:

$$x_{t+1} = Lx_t \tag{4.1}$$

By using gated autoencoder(GAE) as a bi-linear model, the layer of mapping units m, which encodes transformation can be described as:

$$m = \sigma(W(Ux_t \cdot Vx_{t+1})) \tag{4.2}$$

where W, U, and V are weight matrices. U and V weight matrices contain the invariant subspace transformation class while W matrix makes it independent of the absolute angles in the subspaces. The sigmoid function σ, is used as non-linearity function. Based on this setup the frame x_{t+1} can be reconstructed using the m latent variable and frame x_t

$$\hat{x}_{t+1} = V^T(Ux_t \cdot W^T m) \tag{4.3}$$

The PGP model is expanded to learn higher order relational features by computing the layered order mappings in a pyramidal manner, as represented in Figure 4.1. For instance, to infer second-order relational features after computing first-order mappings, we use the following equations:

$$m_1^{(t-2:t-1)} = \sigma(W_1(U_1 x^{(t-1)} \cdot V_1 x^{(t-2)})) \tag{4.4}$$
$$m_1^{(t-1:t)} = \sigma(W_1(U_1 x^{(t)} \cdot V_1 x^{(t-1)})) \tag{4.5}$$
$$m_2^{(t-2:t)} = \sigma(W_2(U_2 m_1^{(t-2:t-1)} \cdot V_2 m_1^{(t-1:t)})) \tag{4.6}$$

Where W_i, U_i, and V_i are weight matrices of ith layer, $m_2^{(t-2:t)}$ is the mapping unit between $m_1^{(t-1:t)}$ and $m_1^{(t-2:t-1)}$, $m_1^{(t-1:t)}$ is the mapping unit between $frame_{t-1}$ and $frame_t$, and $m_1^{(t-2:t-1)}$ is the mapping unit between $frame_{t-2}$ and $frame_{t-1}$. This expansion relaxes the assumption of constancy of the transformation between frames.

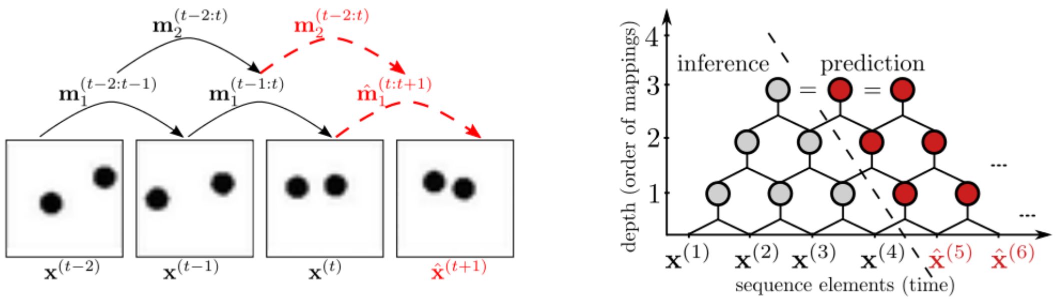


Figure 4.1: PGP architecture, Source: [43]

After extracting features in each layer based on the architecture of relational autoencoder(RAE)[36], which is explained in detail afterward, we use the extracted features to predict the future frames. The inference-prediction equations are:

$$\hat{m}_l^{(t:t+1)} = \sigma(W_l(U_l x^{(t)} \cdot V_l x^{(t-1)})) \tag{4.7}$$
$$\hat{x}^{(t+1)} = V_l^T(U_l x^{(t)} \cdot W^T m_l^{(t,t+1)}) \tag{4.8}$$

where l represents the layer.

Relational Autoencoder

RAE[36] learns transformation between temporally consequent frames. The idea of RAE is based on multiplicative interactions. Multiplicative interactions learn relation between frames independent of the frame content. For two one-dimensional binary images x and y, every component of the outer product between them represents one type of transformation and act like AND-gates. In multiplicative interaction any of the components can be computed using the other two components. In the following, we show that latent variables in RAE learn subspace-rotations after being trained over temporally consecutive frames.

In gated autoencoders[], latent variable z learns to transform two inputs x and y into the other by,

$$z_k = \sum_{ji} w_{ijk} x_i y_j \tag{4.9}$$

$$y_j = \sum_{ki} w_{ijk} x_i z_k \tag{4.10}$$

The above equation can be reformulated to

$$y = L(z)x \tag{4.11}$$

$$L_{ij}(z) = \sum_{k} w_{ijk} z_k \tag{4.12}$$

If L is orthogonal, the eigen-decomposition of L becomes complex where the eigenvalues are 1. Due to the shared variables, which means shared eigenspaces, the only way that two transformations differ is the rotation angles between eigenvalues in shared eigenspaces. Thus, with a single set of features, multiple transformations are represented. For each eigenvector v and rotation angle θ, v_θ is:

$$v_\theta = e^{i\theta} v \tag{4.13}$$

By applying weighted sum over rotation detectors to represent a single transformation and breaking down the imaginary and real part for each eigenvector and represent them by U and V respectively, the representation of the transformation for images x and y becomes:

$$t = W^T(U^T x) \cdot (V^T y) \tag{4.14}$$

One of the main problems of RAE is that the number of parameters in w_{ijk} is in cubic order ($O(n_x \times n_y \times n_z)$ where n_i is number of pixels in input i), leading

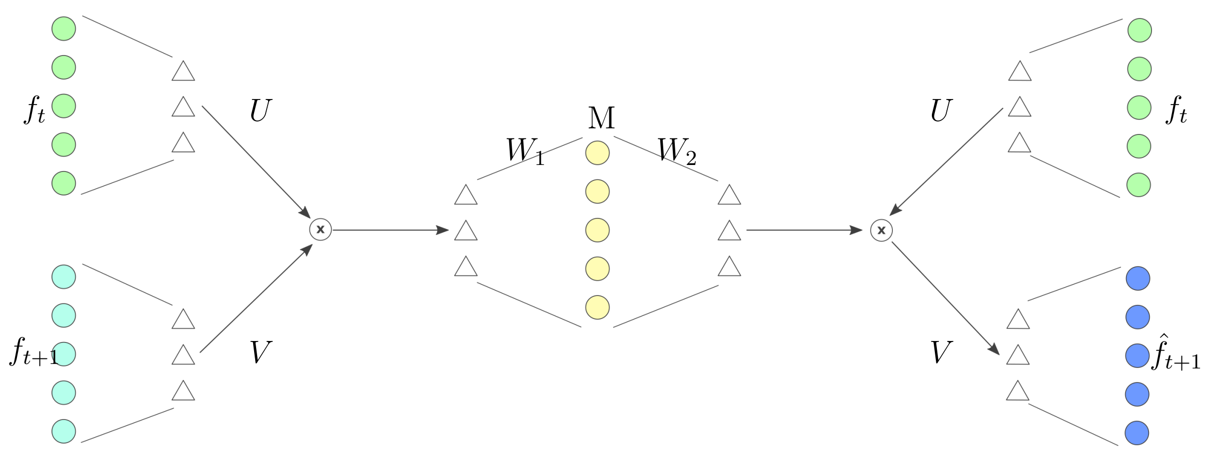


Figure 4.2: RAE architecture

to large memory consumption for input images. To overcome this drawback the parameter tensor is factorized to inner multiplication of three components w^x_{if}, w^y_{jf}, w^z_{kf} from matrices $w^x_{I\times F}$, $w^y_{J\times F}$, and $w^z_{K\times F}$ respectively, where $I = J$ is the input size and K and F is the number of latent variable and number of filters, respectively.

$$w_{ijk} = \sum_{f=1}^{F} w^x_{if} w^y_{jf} w^z_{kf} \tag{4.15}$$

The model uses the symmetric reconstruction loss to train in an unsupervised fashion. Given two input images x and y, the loss L is:

$$L = (x - \hat{x})^2 + (y - \hat{y})^2 \tag{4.16}$$

4.1.2 Convolutional PGP

The fully connected PGP architecture contains a significant number of parameters. It also abstracts away the spatial features[]. To deal with these issues, we modified the fully connected PGP to the fully convolutional PGP architecture similar to [50]. In convolutional PGP, the same setup as its equivalent fully connected version is used except that instead of multiplication, convolution is applied. The fully convolutional PGP architecture is represented in Figure 4.3.

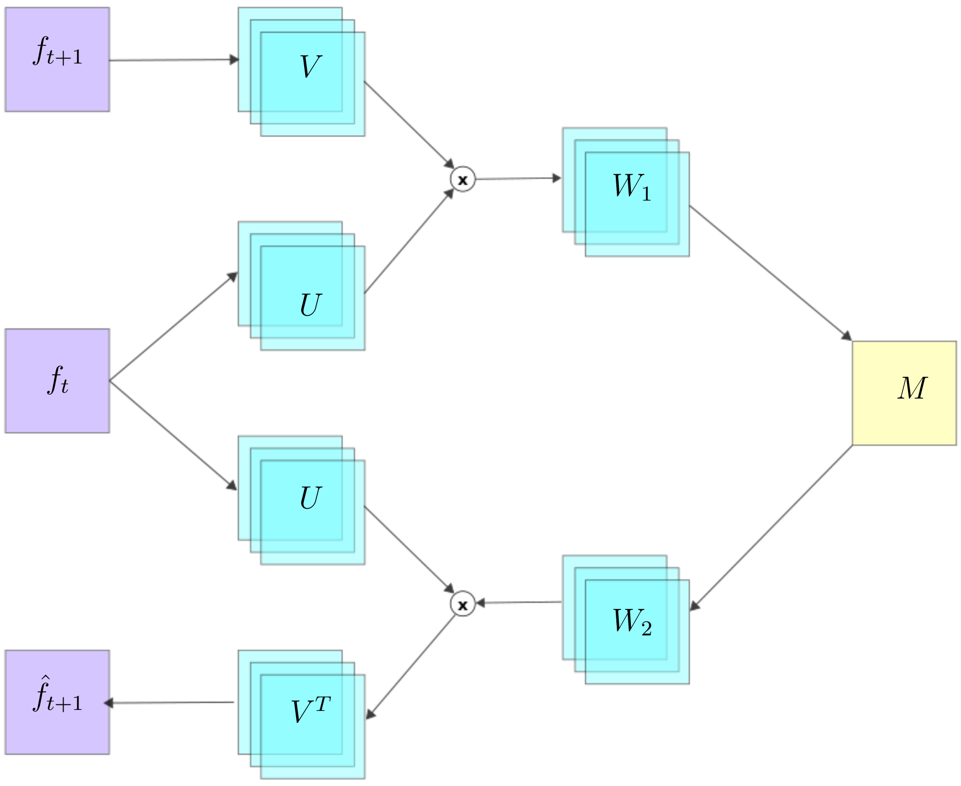


Figure 4.3: Conv-PGP Architecture

4.1.3 Implementation

We implemented Conv-PGP and PGP architectures in Python both with Tensorflow and Pytorch libraries. We ran the model over four NVIDIA TITAN X graphics card in parallel. We used multi-threading approach for the implementation. It took two days to train the network.

To initialize weights in the fully connected PGP and Conv-PGP, pretraining was required in the reconstruction phase. Furthermore, to stabilize learning, instead of predicting ten frames simultaneously we iteratively increased the number of predicted frames. The input data was a 5D tensor of size: $(batch_{Size} \times sequence_{Length} \times frame_{Width} \times frame_{Height} \times frame_{Channel})$. The batchsize of length 20 was passed as an input to the network architecture. We used 15 frames in each sequence, and each frame was $64 \times 64 \times 1$. The first corrupted five frames with Gaussian noise in each sequence were used as seed in the reconstruction phase. Similar to the original PGP architecture sigmoid function was used for the non-linearity after mapping unit. To reduce blurriness in frames, we also used sigmoid function before outputting the reconstructed and predicted frames. We utilized the MSE loss during the training phase in the reconstruction and prediction frames with respect to their equivalent ground-truth frames. For the Moving MNIST dataset we utilized sigmoid cross entropy in test mode to make our results comparable with VLN and its improvements.

In PGP architecture as well as Conv-PGP architecture the first five frames were reconstructed using RAE and Conv-RAE respectively. By using the learned variables in each layer during the reconstruction phase and copying the topmost mapping unit variables, as shown in figure 4.1, the predicted frames were reconstructed by predicting from the topmost layer and continuing it to the bottommost layer.

As explained in section 3.2, in our experiments we used RMSProp[33] optimizer with learning rate of 10^{-4} both for reconstruction and predictive training instead of using Stochastic Gradient Descent (SGD), which is used in the original PGP architecture[43].

The parameters of PGP and Conv-PGP architectures are represented in table 4.1 and 4.2 respectively.

Parameters	Layer1	layer2	layer3	layer4
m size	1000	500	200	100
U & V size	500	250	100	50

Table 4.1: PGP Parameters

Parameters	Layer1	layer2	layer3
number of features	20	10	5
kernel size	11	5	3

Table 4.2: Conv-PGP Parameters

4.2 VLN

4.2.1 VLN Architecture

VLN[79] is proposed for the task of video prediction based on the architecture of ladder network.

Ladder Network: Ladder network[67] is a deep unsupervised architecture that contains lateral shortcut connections. The lateral shortcut connections which connect encoder part to decoder part relieve the upper layers from learning details in lower layers, therefore the upper layers will be able to learn more abstract features. In this architecture, the cost function is the aggregation of independent cost functions applied to each layer. The loss in each layer is the difference between the clean path of the encoder and its correspondent decoder.

The main goal of ladder network is to design an unsupervised architecture to take advantage of the large unlabeled data that is available. To make the ladder network semi-supervised, the unsupervised architecture should be augmented to supervised one in the fashion that it extracts features related to supervised architecture and discards the non-related features.

In general, many unsupervised learning methods can be reformulated as reconstructing input $x(t)$ by utilizing latent variables $s(t)$.

$$x(t) = g(s(t); \xi) + n(t) \tag{4.17}$$

where ξ is the parameters of mapping function g and $n(t)$ is the noise. The equation can be reformulated to

$$p_x(x(t)|s(t), \xi) = p_n(x(t) - g(s(t); \xi)) \tag{4.18}$$

where p_n represents the probability density function of the noise $n(t)$. p_x represents the probability density function of being $x(t)$ with respect to $s(t)$ and ξ. The main problem with this setup is that it cannot discard the information due to the increase of reconstruction error. To deal with this problem, a hierarchical latent variable was proposed [67]. The usage of the hierarchical latent variable introduced two new problems which are not easy to deal with. We not only need to deal with probabilistic methods but also need to do layer-wise training which leads to slow training.

Concerning the discussed issues, ladder network architecture is proposed. The method is similar to autoencoders except that it utilizes lateral connections. Lateral connections make the reconstruction part of the architecture not only to be function of the top-down flow of information but also a function of encoder part flow of information.

Augmenting lateral connections require a new loss function and the difference between the reconstructed $\hat{x}$ and input x cannot be used. The routine loss, forces the lateral connection in the first layer to learn directly from the input and does not let the error goes through all the layers which leads to a worse situation. This problem does not exist in simple autoencoders as the error goes through all the layers in a hierarchical way though it makes the training very slow. To solve this issue, the method of distributed learning rather than learning from a single error term was proposed.

Rasmus et al.[62] extends the ladder network architecture to semi-supervised architecture by augmenting supervision to the model. The loss is the summation of supervised and unsupervised cost functions which is minimized by the back-propagation algorithm.(Figure 4.4).

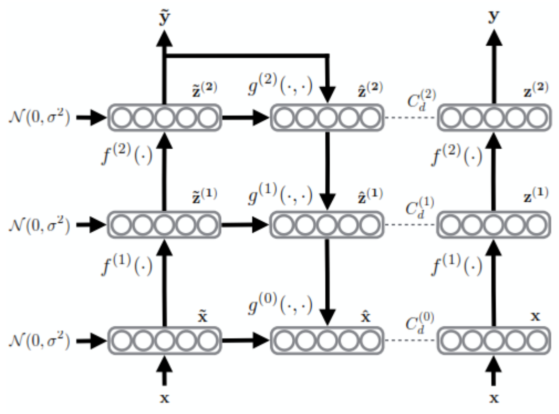


Figure 4.4: Ladder Network architecture, Source:[**ln**]

The supervised cost function is

$$C_c = -1/N \sum_{(n=1)}^{N} logP(\hat{y} = t(n)|x(n)) \tag{4.19}$$

where $t(n)$ is the target, $\hat{y}$ is the noisy output, and $x(n)$ is the input. The cost can be interpreted as the average negative log probability of $\hat{y}$ and $t(n)$ due to given input x.

The unsupervised cost in the decoder part of the architecture is to denoise the latent variable $\tilde{z} = z + n$. z and n are the clean latent variables with the variance of Gaussian distribution σ_z^2 and σ_n^2 respectively. Notifying that latent variables z need normalization as otherwise, the network has an obvious solution of $z = \hat{z} = constant$. Then the total cost function becomes:

$$C_{total} = \sum_{l=0}^{L} \lambda_l ||z^{(l)} - \hat{z}_B^{(l)} N||^2 + C_c. \tag{4.20}$$

In VLN architecture three independent blocks are used; encoder block, decoder block, and ConvLSTM block.

The encoder block contains two dilated convolution layers and one strided convolution layer, followed by LReLU activation function and batch normalization. Dilated convolution as explained in chapter 3, is a way of increasing receptive view of the network exponentially without loss of resolution or coverage[71]. The batch normalization is needed for two reasons; the first reason is the speed up in the convergence rate [49] and the second reason is to discourage the network from the trivial solution where the encoder outputs content value as this is the easiest one to denoise.

The batch normalizations in encoder blocks are followed by LReLU activation function and strided convolution is followed by ReLU activation function.

In video ladder network this residual connection helps removing loss computing in each layer like ladder network architecture.

The decoder block contains one upsampling layer followed by two convolution layers. To make the training fast, two batch normalization layers are added. Same as encoder block, the LReLU activation function is used after batch normalization. Same as encoder blocks, all decoder blocks have residual connections to avoid degradation problem.

The third block proposed in this article is convolutional LSTM.

Finally in the decoder part of the architecture the output of the blocks h_t^l, z_t^l and the upper decoder layer $\tilde{z}_{t+1}^{l+1}$, are summed up together as

$$\tilde{z}_{t+1}^{l} = LReLU((LReLU((\tilde{z}_{t+1}^{l+1}) * W_h^l), z_t^l) * W_z^l) \tag{4.21}$$

Where $(.,.)$ denotes channel-wise concatenation(Figure 4.5).

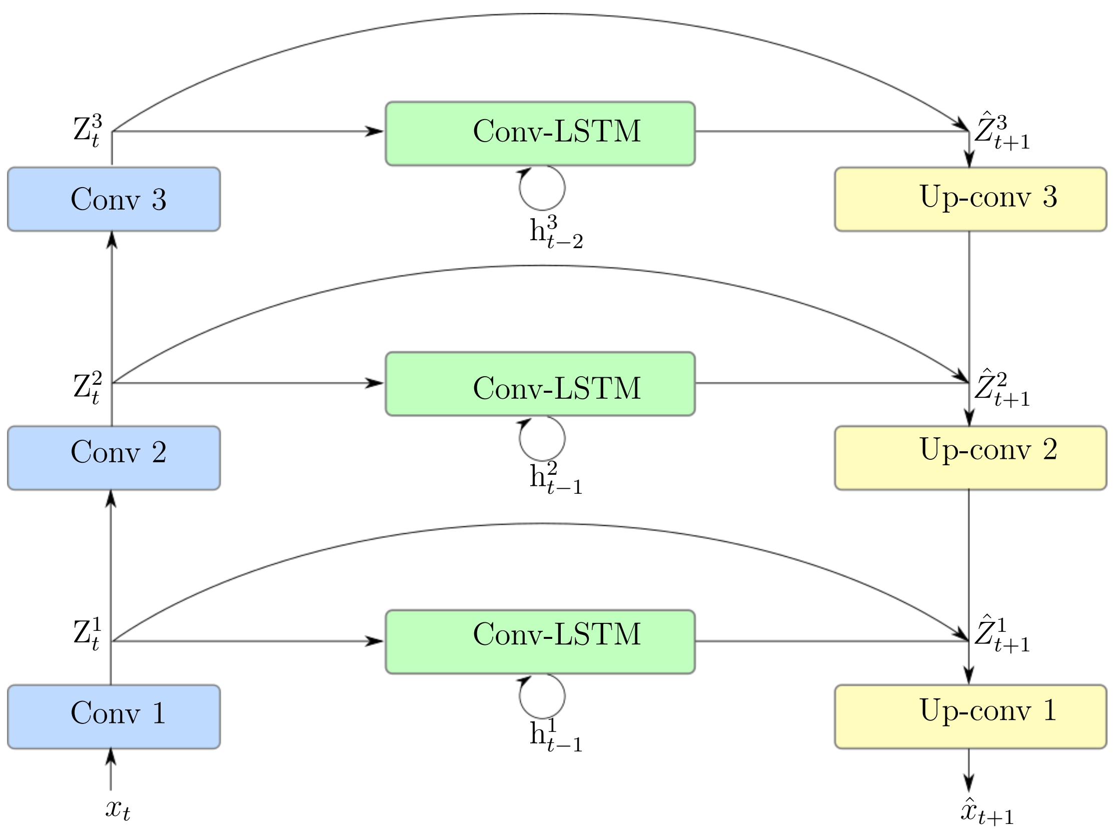


Figure 4.5: VLN architecture

The loss function used to measure the error in this architecture is sigmoid cross entropy [3.2].

4.2.2 VLN Improvements

To improve the results of the predicted frames in VLN architecture we explored the semi-supervised VLN by augmenting the velocity and position of the two digits as well as making the predicted frames object level by applying the semantic segmentation. Furthermore, we made use of temporal hierarchy and explored the effect of the backpropagation with coarser time steps.

The VLN architecture includes three Conv-LSTM blocks. The upper blocks of Conv-LSTMs extract more abstract features than the lower blocks. This leads to a slower rate of evolution between consecutive frames in the topmost Conv-LSTM. Due to the similarity of the consecutive frames in the topmost Conv-LSTM, the temporal backpropagation with coarser time steps can be applied. The temporal pooling helps in extracting more features over time which leads to

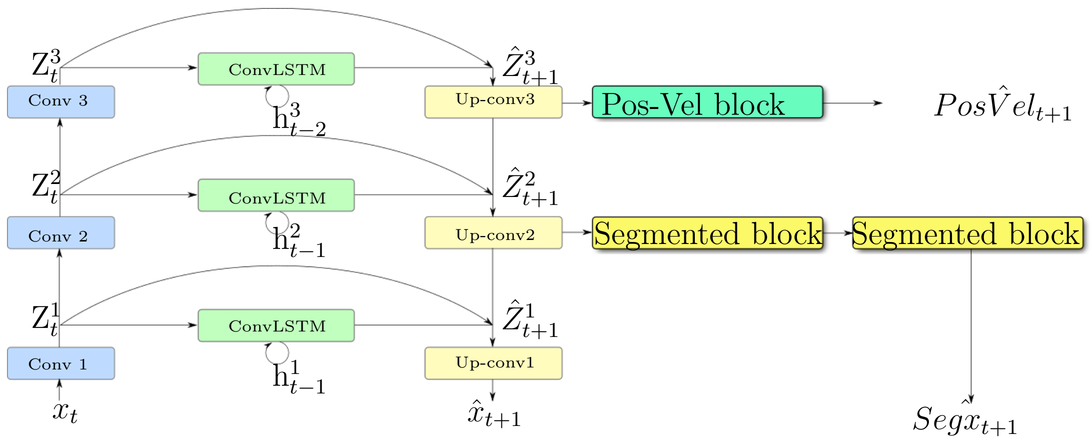


Figure 4.6: Architecture of improved VLN

a better comprehension of the video. The more comprehensive information over the video leads to a better predicted frames.

The main problem with the current methods of video prediction is that they are semantically uninterpretable. To interpret the predicted frames we utilized the task of semantic segmentation. Both prediction and segmentation can be regarded as auxiliary tasks to each other. We applied the segmentation after the second decoder block extended with two more decoder modules. The augmented decoder modules are similar to the decoder block with a small change that just the first module does the upsampling. The softmax cross entropy3.2 loss was applied to compute segmentation.

Furthermore, To explore whether the network can extract the velocity and position of the objects in the frame, we made the fully unsupervised architecture of VLN supervised. We trained the VLN architecture by explicitly applying the velocity and position of the two digits after the first decoder block. To this end, we increased the number of input channels to three by supplying mesh grid to the input. The MSE loss was applied. The architecture is represented in Figure 4.6.

In each iteration, we minimized the summation of the three losses.

4.2.3 Implementation

We used Python and specifically Tensorflow library to implement the VLN architecture and all its extensions. The models were run over four NVIDIA TITAN X graphic cards. It took two days to train the network.

The VLN architecture contains three encoder and three decoder blocks with 28, 58, and 90 channels respectively. The Conv-LSTMs also contain the same number of channels 28, 58, and 90 respectively. Each encoder block contains two dilated convolutional layers with the kernel size of $(3, 3)$ and dilation rates of $1, 2, 2, 4, 4, 8$. The decoder blocks also contain two convolutional layers with no dilation. All encoder blocks contain element-wise sum of skip connections followed by strided convolution and ReLU activation function. In each decoder block we have an up-sampling layer followed by batch normalization and LReLU activation function. The architecture is represented in Figure 4.7.

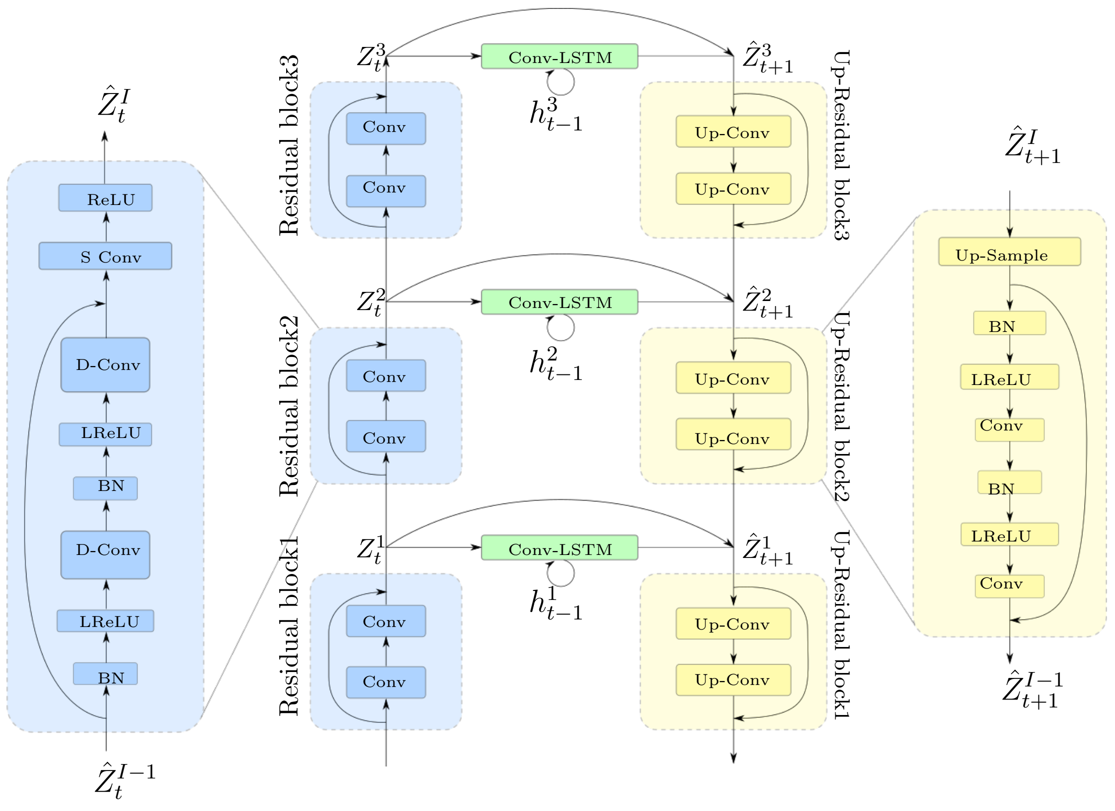


Figure 4.7: The detailed VLN architecture

To apply the idea of time pooling in the topmost Conv-LSTM, we updated the previous hidden state values in the case when the frame number is a multiple of the time-pooling step and kept its previous hidden state otherwise.

For the task of segmentation, two segmentation blocks with 58 channels each, were inserted into the architecture. The segmentation blocks similar to the decoder blocks contain two convolutional layers with no dilation. In the first segmentation block, we have an up-sampling layer while in the other block it is just a convolution layer. Each convolution is followed by batch normalization and LReLU activation function.

To apply the position and the velocity of the digits after the first decoder block, we extended the input dimension by augmenting Numpy[99] mesh grid to the input. A convolutional layer after the first decoder block with stride $(2, 2)$ was applied.

5 Evaluation and Results

5.1 Dataset

We test our architectures on moving MNIST dataset[64]. We also did the sanity check of the PGP architecture using bouncing ball dataset[43].

5.1.1 Moving MNIST dataset

Each video in moving MNIST dataset contains 20 frames with two digits moving inside a 64×64 patch. Digits chosen randomly from the training set and placed initially at random locations inside the patch with random velocity. The digits bounced-off the edges of the 64×64 frame and overlapped if they were in the same location (Figure 5.1).

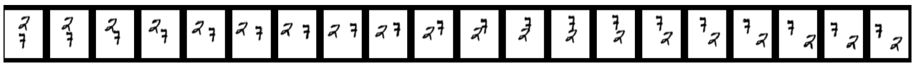


Figure 5.1: Moving MNIST dataset

5.1.2 Bouncing Ball dataset

The bouncing ball dataset contains sequences generated in real time. Each sequence contains 20 frames, each containing 2 balls interacting in a box of size $64 \times 64 \times 1$ pixels. The radius for the balls is $r = 1$. When the balls bounce the wall or bounce each other their velocity mirrors. One sequence is represented in Figure 5.2.

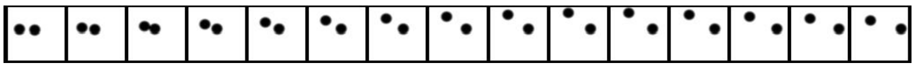


Figure 5.2: Bouncing ball dataset

5.2 PGP Results

In primary tests, PGP and Conv-PGP architectures have been applied to bouncing ball dataset and their results were compared After reproducing the result of the article for bouncing ball dataset, the convolutional version of PGP architecture was evaluated on Moving MNIST dataset to compare its results with the VLN architecture and its improvements.

5.2.1 Original PGP for Bouncing Ball Dataset

The results of the original PGP for bouncing ball dataset were reproduced with five seed frames and ten predicted frames. The entire 64×64 frame is passed to the fully connected architecture, leading to significant number of parameters of $6M$. The results are represented in Figure 5.6a. The model generating the sequences uses the architecture 4.1 and parameters in table 4.1. The input data during both the reconstruction and prediction phase was corrupted with Gaussian noise before being passed to the network architecture.

To eliminate the blurriness of the frames, we applied nonlinearity function before outputting both the reconstructed and the predicted frames. We conducted the experiment with the sigmoid nonlinearity function. In the original architecture [43] sigmoid nonlinearity was also applied after the mapping unit m. We replaced the sigmoid function with ReLU and LReLU nonlinearities to test if sparse activations would improve the results. Neither of the nonlinearity functions could improve the results. It could be because of firing the mapping units leads to more information loss in decoder part of the architecture, while in the sigmoid version of the architecture fewer mapping units are fired and thus the decoder can access and extract more information about transformation.

The value of MSE loss for the bouncing ball test dataset was 0.001. Pretraining model with the MSE loss on reconstruction phase during the training steps improved the performance of the fully-connected model as well as increasing the number of the predicted frames instead of predicting all ten frames at once. Applying noise to the input during reconstruction and predictive phase during training helped as well.

5.2.2 Conv-PGP for Bouncing Ball Dataset

The output results of Conv-PGP for bouncing ball dataset is represented in Figure 5.6b. The results are competitive with respect to the original PGP but with 0.005% of the total number of parameters. The filter size of W in Conv-PGP architecture is 1×1 with 20 number of channels. The weight parameters U and V have the kernel sizes of 11, 5, and 3 with 20, 15, and 10 number of channels from the bottommost layer to the topmost one. Similar to the original PGP, it consists of pretraining in reconstruction phase as well as inferring phase. Furthermore, same as its equivalent PGP architecture number of the predicted frames increased iteratively and the next ten frames are predicted.

The output result of each layer is represented in Figures 5.3 to 5.5 respectively. The first experiments were conducted with a one-layer Conv-PGP. The model consisted of 20 filters of size 11×11 and 20 mapping units of size 7×7. With three layers of Conv-PGP architecture the network could learn the transformation as well as the boarder and the balls did not disappear. The sequence in figure 5.6b shows a predicted series of a network with parameters from Table 4.2. The network was trained on 10-step prediction and seeded with the first 5 frames from a previously unseen test dataset.

The experiments conducted with datasets containing balls of different sizes. The large balls with radius two although could learn the transformation from the seed frames, could not predict the frames. Reducing the size of the radius to one let the network learns the transformation and generalize it to predicted frames. The model was trained to minimize the predictive error for the next ten predicted frames as well as the five seed reconstructed frames utilizing the MSE loss.

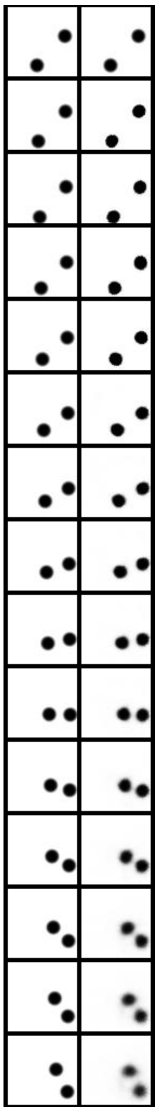
Figure 5.3: First layer output of Bouncing Ball

Figure 5.4: Second layer output of Bouncing Ball

Figure 5.5: Third layer output of Bouncing Ball

5.2.3 Conv-PGP with Moving MNIST dataset

To test functionality of Conv-PGP with respect to VLN and its improvements, we produced the results for Moving MNIST dataset. Furthermore, to make the results comparable BCE test loss was applied instead of the MSE test loss. The hyperparameters of Conv-PGP are represented in table 4.2. Conv-PGP could not return competitive results with respect to VLN and its improvements over Moving MNIST dataset. It is due to the fact that PGP and Conv-PGP architectures are


(a) Original PGP

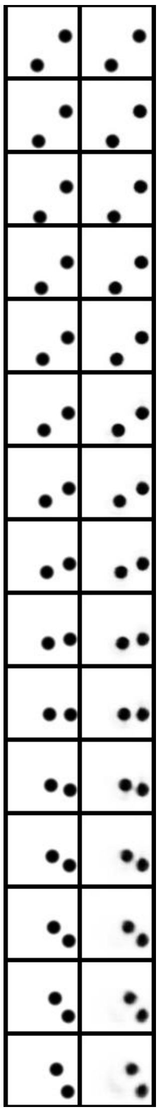

(b) Conv-PGP

Figure 5.6: The first five frames are reconstructed frames followed by ten predicted frames on the bouncing ball test dataset

designed to learn transformation and they cannot extract spatial features as good as VLN and its extensions. The results of Conv-PGP are represented in Figure 5.7. The first five frames are reconstructed frames and the next ten frames are the predicted ones. As it is represented in Figure 5.7, Conv-PGP can completely learn the transformation but it cannot completely extract the spatial information.

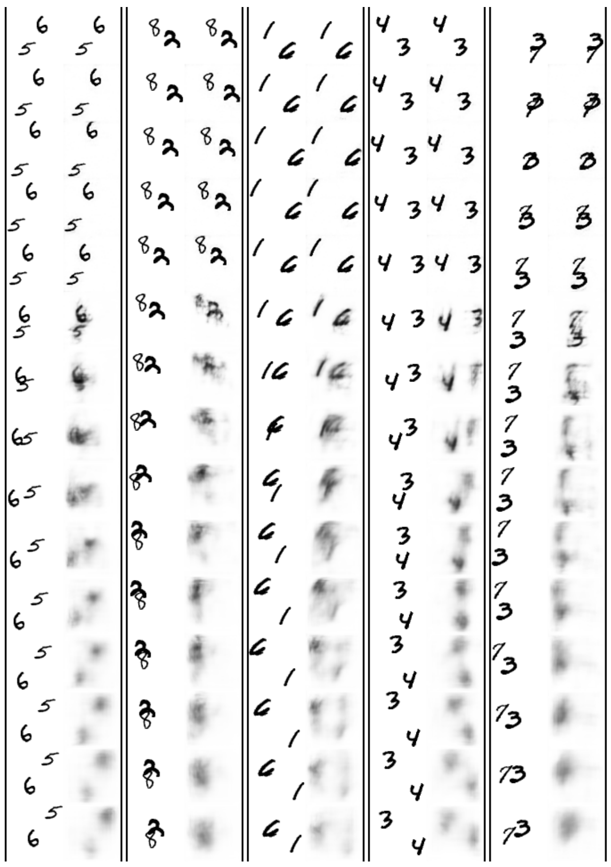


Figure 5.7: Conv-PGP Moving MNIST Results; The first five frames are the reconstructed frames and the next ten frames are the predicted ones

5.2.4 Analysis

The PGP architecture solves the vanishing/exploding problem during backpropagation due to orthogonality of the weights to each other[43].

Furthermore, in contrast to other spatiotemporal architectures, the output features of each layer can be visualized.

Moreover, the higher order relational features in this architecture can be interpreted as velocity for the first layer, accelaration for the second layer, etc. This interpretation arises when one thinks of the first layer of the architecture as first-order Taylor approximation of the input sequence. Thus, the second layer becomes the second-order Taylor approximation of the input sequence and so on. Therefore, the hidden representations model the first, second, and third derivitives of the input with respect to time [43].

One major drawback of the PGP architecture is that it cannot be extended to large frames.This architecture is fully connected and therefore the number of parameters grows significantly. For our experiments with five seed frames and ten predicted frames over bouncing ball dataset with four layers of backpropagation, the number of parameters were $6M$. To deal with this issue, we modified the original PGP to the fully convolutional PGP architecture. The number of parameters in Conv-PGP architecture is $60K$, which is reduced by a factor of 0.01 while competitive MSE loss results with respect to its equivalent fully connected version is returned.

The PGP architecture does not extract features spatially and abstracts away the image content. Although conv-PGP solves this issue to some extent still neither of them can produce competitive results with respect to VLN and its extensions.

Also, PGP and Conv-PGP architectures neither can be generalized to generic transformations [97], nor they can work well with novel environments [39].

5.3 VLN Results

5.3.1 Original VLN

We implemented and reproduced the results of VLN. It backpropagates through time for ten backpropagating time steps. The results are represented in Figure 5.12.

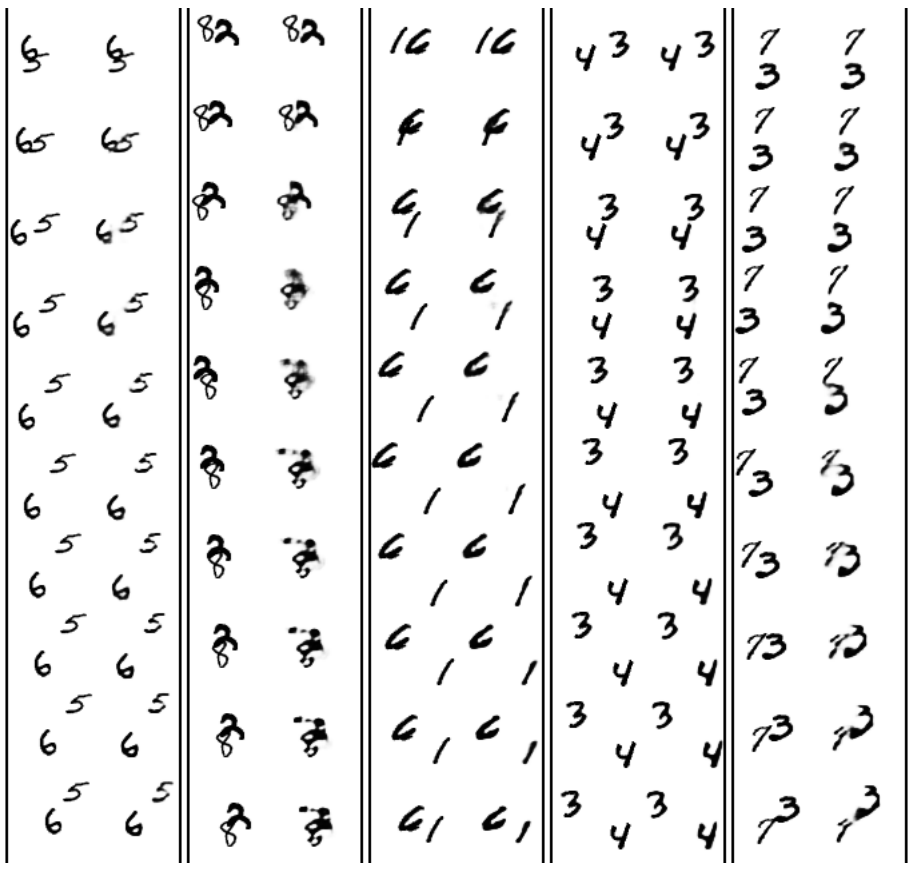


Figure 5.8: VLN Results on the Moving MNIST test dataset

5.3.2 VLN Result with coarser timesteps

In the topmost ConvLSTM, we changed the time pooling to backpropagate in coarser steps of 2 and 3 with 5 backpropagation time steps. In neither of them it could improve the results. Modifying the backpropagation time step to ten and the time pooling step to two improved the results. To make the results comparable, we conducted the experiments with the backpropagation time step of ten and the time pooling step of one. The reasons of the achieved results are discussed in section 5.3.5. Figure 5.12 represents the results of the backpropagation time step of ten and the time pooling step of two.

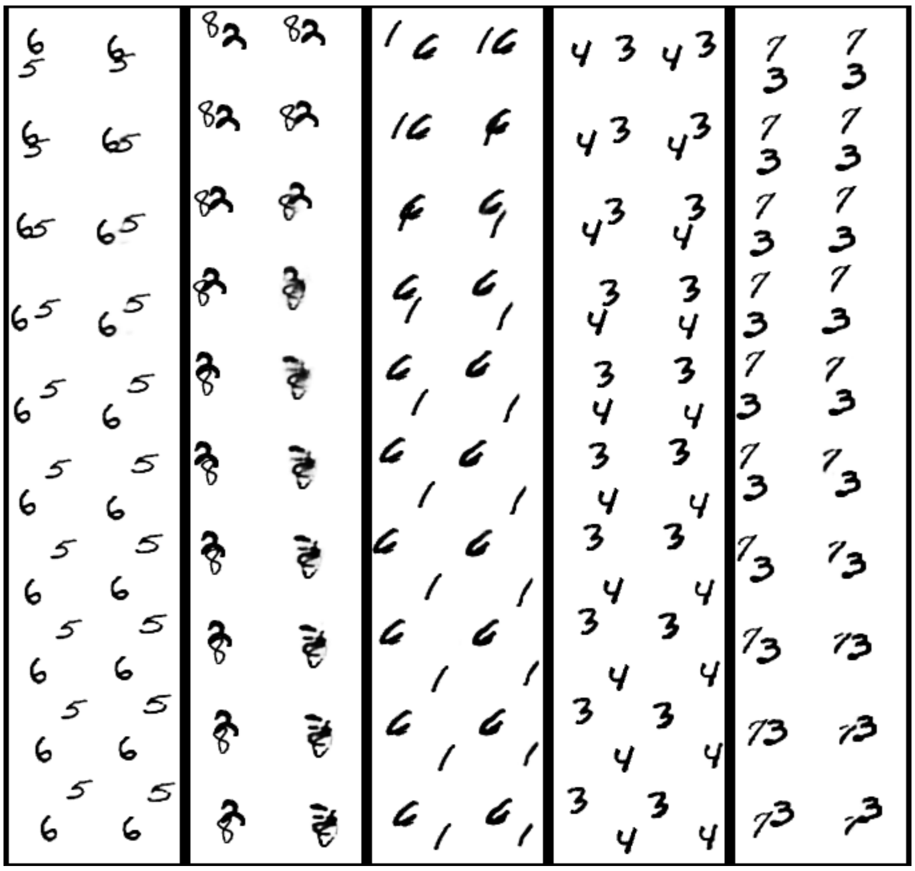


Figure 5.9: Coarser VLN Results on the Moving MNIST test dataset

5.3.3 VLN with the task of Segmentation

The segmentation of each frame is applied after the second decoder block of the VLN architecture. The predicted frames and the segmented frames are represented in Figures 5.10 and 5.11 respectively. The predicted frames are competitive to the original VLN-ResNet.

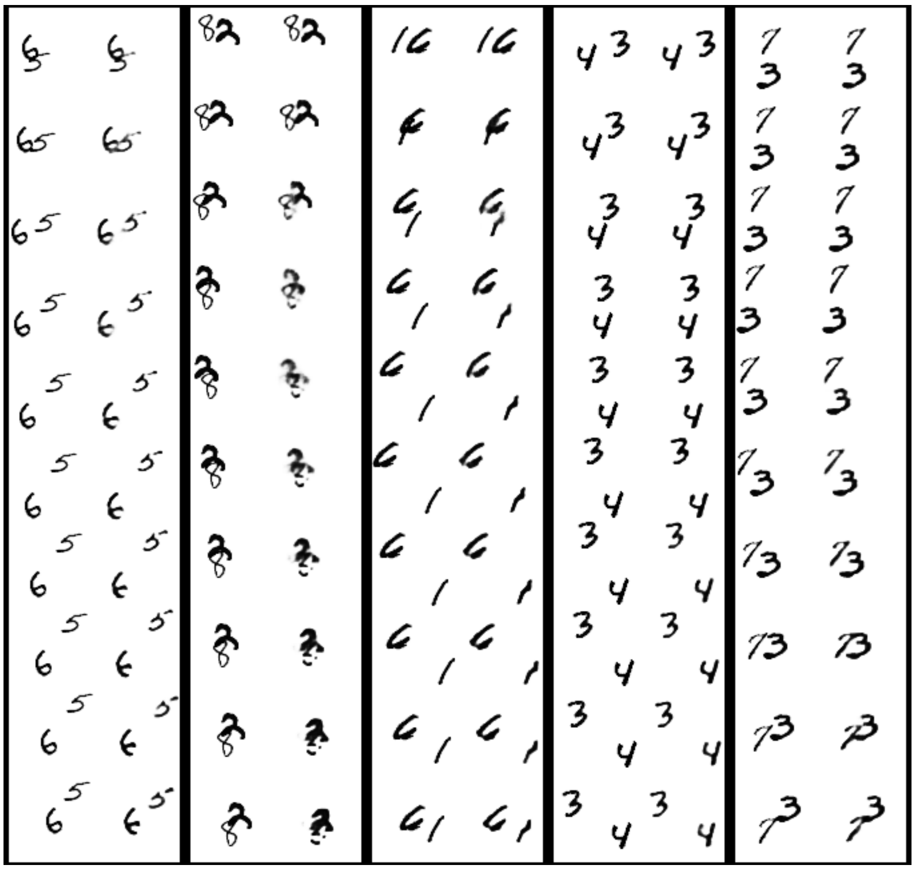
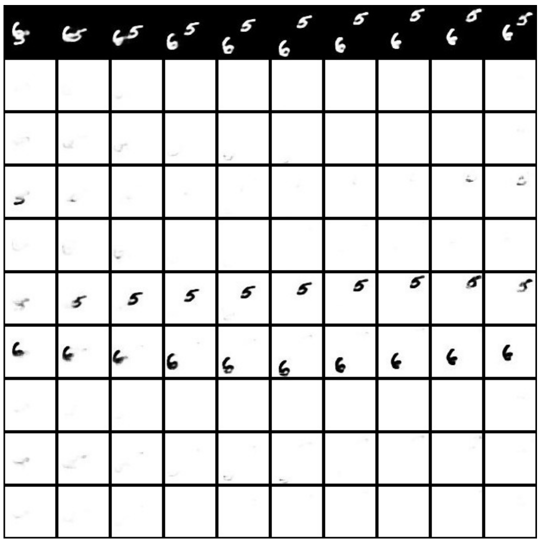
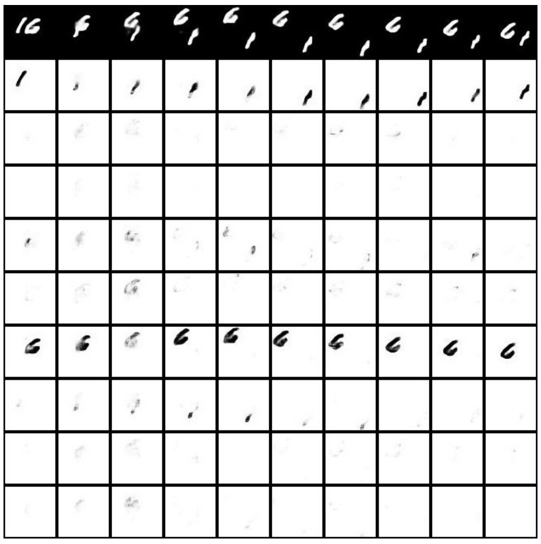
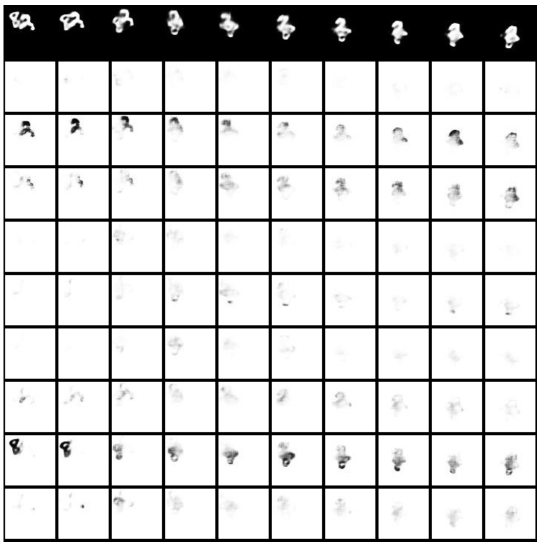
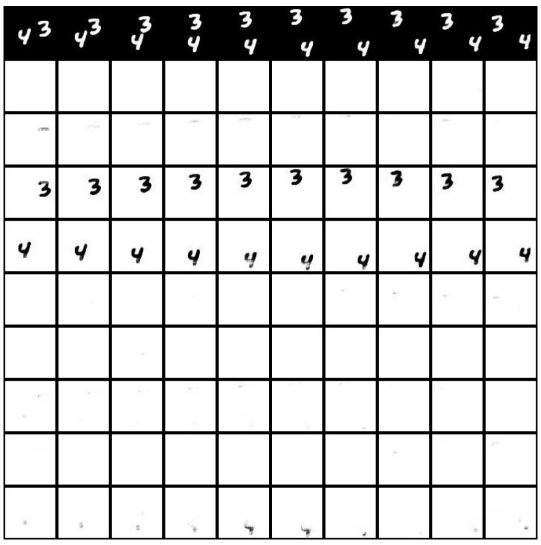


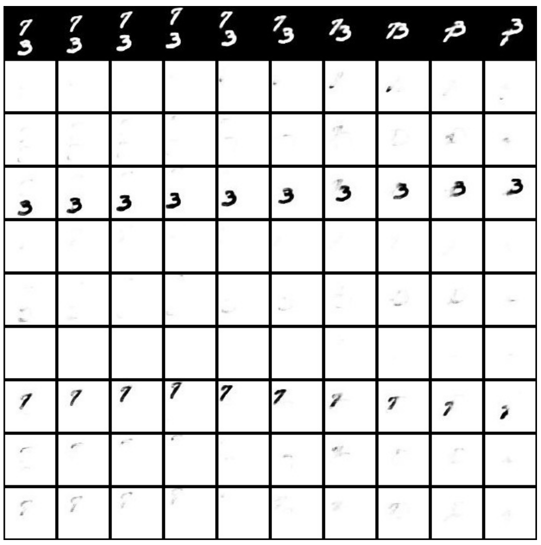
Figure 5.10: VLN-Seg Results on the Moving MNIST test dataset


(a) First sequence


(b) Second sequence


(c) Third sequence


(d) Fourth sequence



(e) Fifth sequence

Figure 5.11: Ten predicted segmentation frames on the Moving MNIST test dataset; Frame ten is the most left in each sequence.

5.3.4 VLN, Segmentation with Velocity and Position of both digits

By applying the velocity and positions of the digits after the first decoder, the results slightly improved with respect to VLN-Seg. Figure 5.12 and 5.13 represent the results of prediction and segmentation respectively.

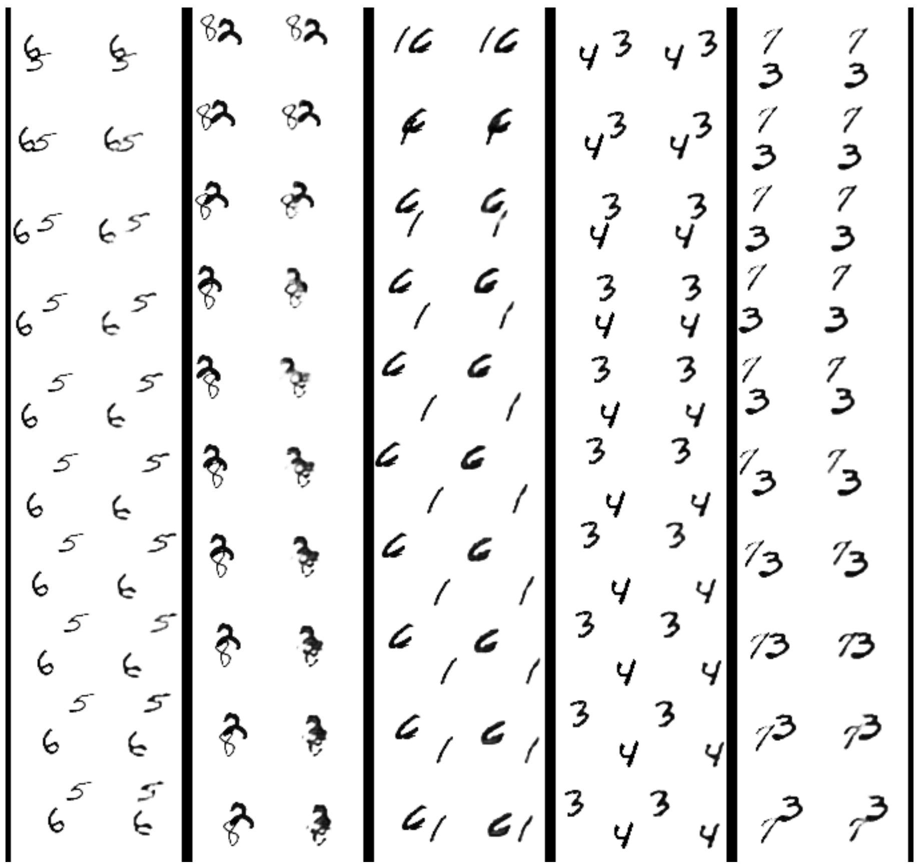
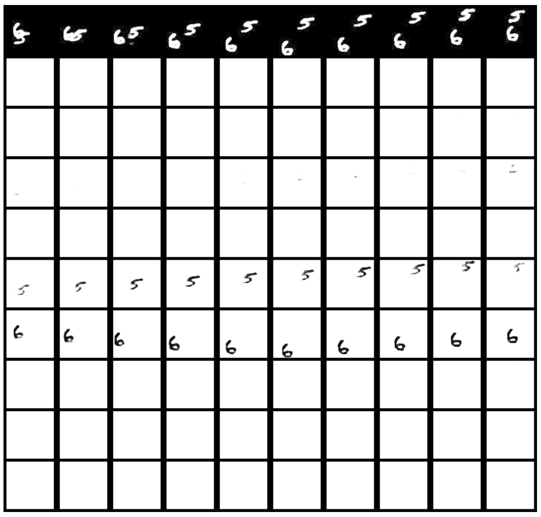
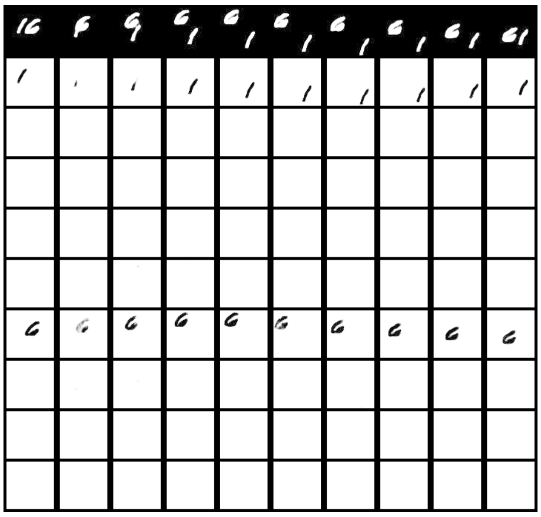
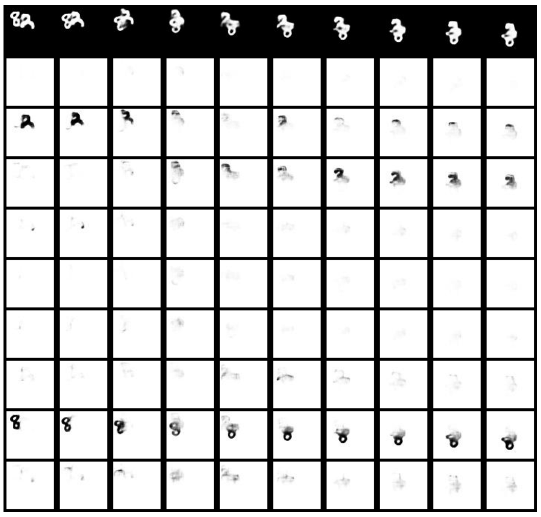
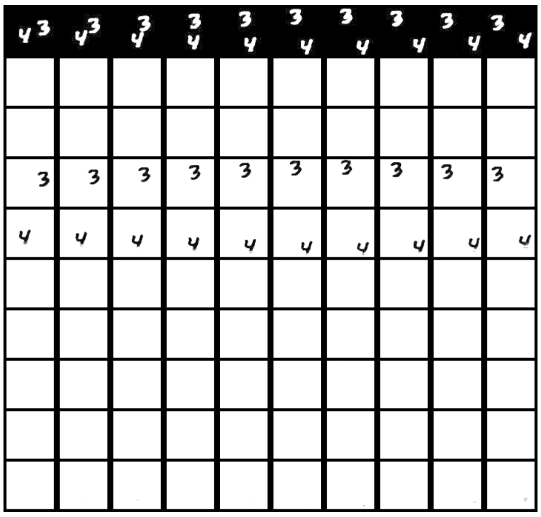


Figure 5.12: VLN-Seg with velocity and position of the digits on the Moving MNIST test dataset


(a) First sequence


(b) Second sequence


(c) Third sequence


(d) Fourth sequence

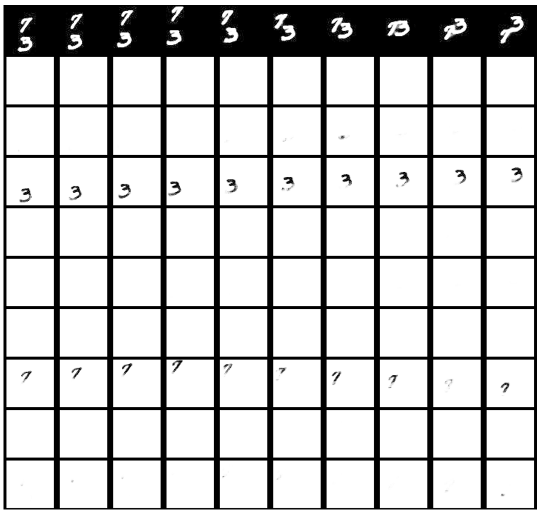

(e) Fifth sequence

Figure 5.13: Ten predicted segmentation frames on the Moving MNIST test dataset; Frame ten is the most right in each sequence.

5.3.5 Analysis

The VLN architecture returns competitive results with respect to VPN with $1.3M$ parameters which is 3% of the total number of parameters in VPN architecture. As mentioned before, VPN returns a near-optimal result for the Moving MNIST dataset using binary cross entropy loss. The VLN utilizes both the skip connections as well as the lateral connections. The lateral connections in the VLN architecture allows the upper layers of the VLN architecture to extract more abstract features resulting in better prediction frames.

Increasing the number of backpropagation steps from five-time steps to ten-time steps during training in Conv-LSTMs of the VLN architecture leads to a better result due to more temporal feature extractions. Furthermore, changing the pooling time step in the top-most Conv-LSTM improved the results with respect to specific hyper parameters. The evolution between consecutive frames in the topmost Conv-LSTM is slow and thus time pooling can be applied. The time-pooling method leads to more backpropagation time steps and therefore more temporal information extraction and consequently better predicted frames.

We conducted the experiment with various hyper-parameters. The primary tests were done with pooling time steps of two and three by backpropagating through time using five-time steps. In neither of them the results were improved. One possible explanation could be that adapting the architecture to learn different coarse time in different iterations would be difficult for the network. To deal with this problem we modified the backpropagation timestep and pooling timestep for the topmost Conv-LSTM to ten and two respectively. The reason for selecting these hyper parameters was that ten is divisible by two which relieves the network to adapt itself with different pooling timesteps. The results were improved with respect to VLN with backpropagation timestep and pooling timestep of ten and one respectively. The positive aspect of this experiment is that it improved the result without increasing the number of parameters. We also did the experiment with pooling timestep not only for the topmost Conv-LSTM but also for the middle one. We selected four and two as the hyper parameters for the topmost one and the middle one respectively. We set the backpropagation time step to twenty. These sets of hyper parameters made the results worse. The reason could be that the Conv-LSTMs skip too much information temporally. One training criteria with potential improvements for dealing with this issue is to connect the Conv-LSTMs from the bottommost Conv-LSTM to the topmost one. It makes the upper Conv-LSTMs have temporal information of the skipped previous frames while extracting more temporal features. Though, the downside of this experiment could be the growth of the number of parameters.

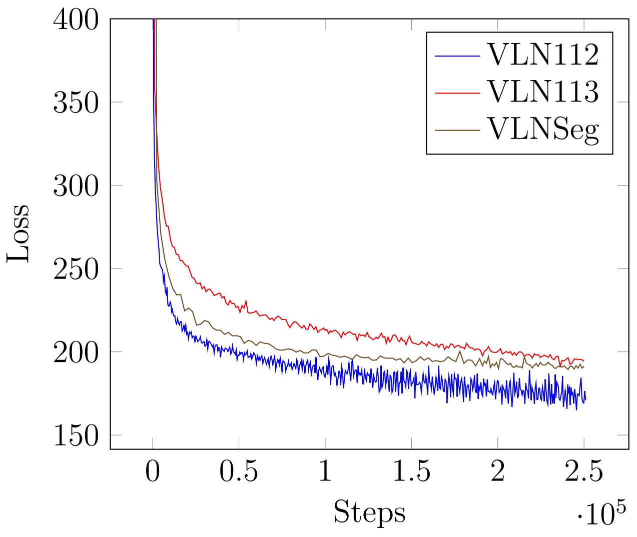


Figure 5.14: Test-set loss of VLN improvements

To make the results of the predicted frames semantically object level interpretable, we applied the task of semantic segmentation after the second decoder block. We utilized the softmax cross entropy to train the network. The primary tests were conducted with one segmentation block after the second decoder block. Though it could return a reasonable results just for the task of segmentation, the two tasks of segmentation and prediction could not collaborate with each other. We did several experiments to deal with this issue. The first experiment was to double the number of channels in the second Conv-LSTM block but it could not improve the results reasonably. One possible explanation could be that the common part between the two architectures extracted the features which could be shared among the two tasks. In the second experiment we augmented one more segmented block after the second decoder block. This experiment could return reasonable results for both architectures. The auxiliary task of semantic segmentation could work with the VLN architecture. The predicted frames were competitive with respect to its equivalent VLN architecture.

Furthermore, deploying position and velocity of the digits to the VLN-Seg architecture could help to improve the result with respect to VLN-Seg architecture. This represents that the network could not extract these features by itself. So explicitly feeding this information to the architecture improved the results of the predicted frames.

5.4 Conclusion

Theoretically, the minimum possible achievable result for predicting the next ten frames on the Moving MNIST dataset using the metric of sigmoid cross entropy is 86.5 [87]. The results of the conducted experiments are presented and compared in table 5.1.

Model	Prediction Test loss	#Parameters
conv-LSTM (Shi et al.)	367.1	7.6M
VPN-BL (Kalchbrenner et al.)	110.1	30M
VPN (Kalchbrenner et al.)	87.6	30M
VLN-ResNet-BL (Cricri et al.)	187.7	1.3M
VLN-ResNet-bptt5-1,1,3	194.8	1.3M
VLN-ResNet-bptt5-1,1,2	191.3	1.3M
VLN-ResNet-bptt10-1,1,1	184	1.3M
VLN-ResNet-bptt10-1,1,2	173.1	1.3M
VLN-ResNet-bptt10-1,1,3	191.2	1.3M
VLN-ResNet-bptt10-1,1,2-Seg.	194.3	1.5M
VLN-ResNet-bptt10-1,1,2-Seg.-Vel.&Pos.	191.15	1.5M
Conv-PGP	409.3	35K

Table 5.1: Results on Moving MNIST test dataset

The segmentation results on test dataset in the last two experiments in table 5.1 using softmax cross entropy are 0.09 and 0.08 respectively.

6 Conclusion

To conclude the results of our work, first we summarise the contributions of this thesis, then we discuss known limitations, and, finally, we present possibilities for the future work.

6.1 Summary

In this thesis the prediction of the next frames over the synthetic datasets has been investigated. The fully convolutional version of the PGP architecture and the temporal hierarchy and the semantically interpretable version of the VLN are proposed. The proposed methods have been evaluated during series of experiments and their performance were compared against several video prediction architectures.

In order to extract more features temporally we proposed time pooling in the topmost Conv-LSTM. Also to semantically interpret the predicted frames, we applied multitask learning to the VLN architecture with the task of semantic segmentation.

Furthermore, we explored the PGP architecture. This architecture extracts transformation features temporally but in contrast to RNNs the extracted features are transparent. We proposed its equivalent fully convolutional architecture which has 1% of the total number of parameters of the original PGP.

6.2 Limitations and Future Work

We explored the task of video prediction for the application of human robot anticipation. We utilized the VLN and PGP architectures as baseline. The VLN architecture uses Conv-LSTMs to extract features temporally. The extracted features of Conv-LSTMs are not transparent. On the other hand, the architecture of PGP is designed to extract transformation features but not the content features. However, modeling both structure and dynamics of the scene in one architecture leads to an uninterpretable results[108]. For the application of human robot anticipation higher level of abstraction is required.

Given the limitation of the two architectures, as a future work, we propose breaking down the process of video prediction into two parts by combining the two architectures of PGP to extract transformation features and VLN to extract pixel-level spatio-temporal features. In the VLN architecture after reducing the resolution of the frame in each step, the PGP is applied and its output is multiplied pixel-wise by the output of its relevant decoder block. This proposed architecture makes the network learn transformation separately in hierarchical manner, which leads to achieve the higher level of abstraction.

Bibliography

[1] Geoffrey F Hinton. "A parallel computation that assigns canonical object-based frames of reference". In: *Proceedings of the 7th international joint conference on Artificial intelligence-Volume 2*. Morgan Kaufmann Publishers Inc. 1981, pp. 683–685.

[2] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[3] Paul Smolensky. *Information processing in dynamical systems: Foundations of harmony theory*. Tech. rep. COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE, 1986.

[4] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural networks* 4.2 (1991), pp. 251–257.

[5] Jiurgen Schmidhuber. "LEARNING COMPLEX, EXTENDED SEQUENCES USING THE PRINCIPLE OF HISTORY COMPRESSION". In: *Neural Computation* 4.2 (1992), pp. 234–242.

[6] Richard Caruana. "Multitask Learning: A Knowledge-Based Source of Inductive Bias". In: *Proceedings of the Tenth International Conference on Machine Learning*. Morgan Kaufmann, 1993, pp. 41–48.

[7] Gilbert Strang, Gilbert Strang, Gilbert Strang, and Gilbert Strang. *Introduction to linear algebra*. Vol. 3. Wellesley-Cambridge Press Wellesley, MA, 1993.

[8] Berndt Müller, Joachim Reinhardt, and Michael T Strickland. "Neural Networks Introduced". In: *Neural Networks* (1995), pp. 13–23.

[9] Salah El Hihi and Yoshua Bengio. "Hierarchical recurrent neural networks for long-term dependencies". In: *Advances in neural information processing systems*. 1996, pp. 493–499.

[10] William T Freeman and Joshua B Tenenbaum. “Learning bilinear models for two-factor problems in vision”. In: *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*. IEEE. 1997, pp. 554–560.

[11] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[12] David G Lowe. “Object recognition from local scale-invariant features”. In: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[13] Jonathan Baxter. “A model of inductive bias learning”. In: *Journal of artificial intelligence research* 12 (2000), pp. 149–198.

[14] Thomas G Dietterich et al. “Ensemble methods in machine learning”. In: *Multiple classifier systems* 1857 (2000), pp. 1–15.

[15] Sven Behnke. *Hierarchical neural networks for image interpretation*. Vol. 2766. Springer Science & Business Media, 2003.

[16] Joao Ascenso, Catarina Brites, and Fernando Pereira. “Improving frame interpolation with spatial motion smoothing for pixel domain distributed video coding”. In: *5th EURASIP Conference on Speech and Image Processing, Multimedia Communications and Services*. Smolenice, Slovak Republic. 2005, pp. 1–6.

[17] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, pp. 886–893.

[18] David B Grimes and Rajesh PN Rao. “Bilinear sparse coding for invariant vision”. In: *Neural computation* 17.1 (2005), pp. 47–73.

[19] Cristian Buciluă, Rich Caruana, Alexandru Niculescu-Mizil, and Alexandru Niculescu-Mizil. “Model compression”. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2006, pp. 535–541.

[20] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. “Sequence Labelling in Structured Domains with Hierarchical Recurrent Neural Networks.” In: *IJCAI*. 2007, pp. 774–779.

[21] Herbert Jaeger. “Echo state network”. In: *Scholarpedia* 2.9 (2007), p. 2330.

[22] Roland Memisevic and Geoffrey Hinton. “Unsupervised learning of image transformations”. In: *Computer Vision and Pattern Recognition, 2007. CVPR’07. IEEE Conference on*. IEEE. 2007, pp. 1–8.

[23] Bruno A Olshausen, Charles Cadieu, Jack Culpepper, and David K Warland. “Bilinear models of natural images”. In: *Human Vision and Electronic Imaging XII*. Vol. 6492. International Society for Optics and Photonics. 2007, p. 649206.

[24] Laurent Jacob, Jean-philippe Vert, and Francis R. Bach. “Clustered Multi-Task Learning: A Convex Formulation”. In: *Advances in Neural Information Processing Systems 21*. Ed. by D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou. Curran Associates, Inc., 2009, pp. 745–752. URL: `http://papers.nips.cc/paper/3499-clustered-multi-task-learning-a-convex-formulation.pdf`.

[25] Vijay Badrinarayanan, Fabio Galasso, and Roberto Cipolla. “Label propagation in video sequences”. In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE. 2010, pp. 3265–3272.

[26] Roland Memisevic and Geoffrey E Hinton. “Learning to represent spatial transformations with factored higher-order boltzmann machines”. In: *Neural computation* 22.6 (2010), pp. 1473–1492.

[27] Jenny Yuen and Antonio Torralba. “A data-driven approach for event prediction”. In: *Computer Vision–ECCV 2010* (2010), pp. 707–720.

[28] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. “Algorithms for hyper-parameter optimization”. In: *Advances in Neural Information Processing Systems*. 2011, pp. 2546–2554.

[29] Kumar Abhisheka, M.P. Singha, Saswata Ghoshb, and Abhishek Anandc. “Weather Forecasting Model using Artificial Neural Network”. In: 2012.

[30] James Bergstra and Yoshua Bengio. “Random search for hyper-parameter optimization”. In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305.

[31] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97.

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

[33] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[34] Alex Graves. "Generating sequences with recurrent neural networks". In: *arXiv preprint arXiv:1308.0850* (2013).

[35] Min Lin, Qiang Chen, Shuicheng Yan, and Shuicheng Yan. "Network in network". In: *arXiv preprint arXiv:1312.4400* (2013).

[36] Roland Memisevic. "Learning to relate images". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1829–1846.

[37] Hongyi Zhang, Andreas Geiger, and Raquel Urtasun. "Understanding high-level semantics by modeling traffic patterns". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 3056–3063.

[38] S Avinash Ramakanth and R Venkatesh Babu. "Seamseg: Video object segmentation using patch seams". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 376–383.

[39] Byron Boots, Arunkumar Byravan, and Dieter Fox. "Learning predictive models of a depth camera & manipulator from raw execution traces". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 4021–4028.

[40] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[41] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[42] Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. "A clockwork rnn". In: *International Conference on Machine Learning*. 2014, pp. 1863–1871.

[43] Vincent Michalski, Roland Memisevic, Kishore Konda, and Kishore Konda. "Modeling deep temporal dependencies with recurrent grammar cells""". In: *Advances in neural information processing systems*. 2014, pp. 1925–1933.

[44] MarcAurelio Ranzato, Arthur Szlam, Joan Bruna, Michael Mathieu, Ronan Collobert, and Sumit Chopra. “Video (language) modeling: a baseline for generative models of natural videos”. In: *arXiv preprint arXiv:1412.6604* (2014).

[45] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[46] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting.” In: *Journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[47] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. “Automatic differentiation in machine learning: a survey”. In: *arXiv preprint arXiv:1502.05767* (2015).

[48] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, and Jeff Dean. “Distilling the knowledge in a neural network”. In: *arXiv preprint arXiv:1503.02531* (2015).

[49] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International Conference on Machine Learning.* 2015, pp. 448–456.

[50] Kishore Reddy Konda and Roland Memisevic. “Learning Visual Odometry with a Convolutional Network.” In: *VISAPP (1).* 2015, pp. 486–490.

[51] Lingpeng Kong, Chris Dyer, and Noah A Smith. “Segmental recurrent neural networks”. In: *arXiv preprint arXiv:1511.06018* (2015).

[52] Y. LeCun, Y. Bengio, and Hinton G. “Deep learning. Nature, 521(7553):436–444.” In: 2015.

[53] Wang Ling, Isabel Trancoso, Chris Dyer, and Alan W Black. “Character-based neural machine translation”. In: *arXiv preprint arXiv:1511.04586* (2015).

[54] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2015, pp. 3431–3440.

[55] Mingsheng Long and Jianmin Wang. “Learning multiple tasks with deep relationship networks”. In: *arXiv preprint arXiv:1506.02117* (2015).

[56] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. “Adversarial autoencoders”. In: *arXiv preprint arXiv:1511.05644* (2015).

[57] Michael Mathieu, Camille Couprie, Yann LeCun, and Yann LeCun. “Deep multi-scale video prediction beyond mean square error”. In: *arXiv preprint arXiv:1511.05440* (2015).

[58] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. “Learning deconvolution network for semantic segmentation”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1520–1528.

[59] Ankit B Patel, Tan Nguyen, and Richard G Baraniuk. “A probabilistic theory of deep learning”. In: *arXiv preprint arXiv:1504.00641* (2015).

[60] Mircea Serban Pavel, Hannes Schulz, Sven Behnke, and Sven Behnke. “Recurrent convolutional neural networks for object-class segmentation of RGB-D video”. In: *Neural Networks (IJCNN), 2015 International Joint Conference on*. IEEE. 2015, pp. 1–8.

[61] Federico Perazzi, Oliver Wang, Markus Gross, and Alexander Sorkine-Hornung. “Fully connected object proposals for video segmentation”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 3227–3234.

[62] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. “Semi-supervised learning with ladder networks”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 3546–3554.

[63] J. Schmidhuber. “Deep learning in neural networks: An overview. Neural Networks, 61:85–117.” In: 2015.

[64] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. “Unsupervised learning of video representations using lstms”. In: *International Conference on Machine Learning*. 2015, pp. 843–852.

[65] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[66] Lucas Theis, Aäron van den Oord, and Matthias Bethge. “A note on the evaluation of generative models”. In: *arXiv preprint arXiv:1511.01844* (2015).

[67] Harri Valpola. “From neural PCA to deep unsupervised learning”. In: *Advances in Independent Component Analysis and Learning Machines* (2015), pp. 143–171.

[68] Vibhav Vineet, Ondrej Miksik, Morten Lidegaard, Matthias Nießner, Stuart Golodetz, Victor A Prisacariu, Olaf Kähler, David W Murray, Shahram Izadi, Patrick Pérez, et al. "Incremental dense semantic stereo fusion for large-scale semantic scene reconstruction". In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 75–82.

[69] Francesco Visin, Kyle Kastner, Aaron C Courville, Yoshua Bengio, Matteo Matteucci, and Kyunghyun Cho. "Reseg: A recurrent neural network for object segmentation". In: *CoRR, abs/1511.07053* 2 (2015).

[70] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". In: *Advances in neural information processing systems*. 2015, pp. 802–810.

[71] Fisher Yu and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions". In: *arXiv preprint arXiv:1511.07122* (2015).

[72] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. "Conditional random fields as recurrent neural networks". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1529–1537.

[73] Héctor Martınez Alonso and Barbara Plank. "Multitask learning for semantic sequence prediction under varying data conditions". In: *arXiv preprint arXiv:1612.02251* (2016).

[74] Plamen Angelov and Alessandro Sperduti. "Challenges in deep learning". In: *Proceedings of the 24th European symposium on artificial neural networks (ESANN)*. 2016, pp. 489–495.

[75] Apratim Bhattacharyya, Mateusz Malinowski, Bernt Schiele, and Mario Fritz. "Long-Term Image Boundary Extrapolation". In: *arXiv preprint arXiv:1611.08841* (2016).

[76] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving". In: 2016.

[77] Junyoung Chung, Sungjin Ahn, Yoshua Bengio, and Yoshua Bengio. "Hierarchical multiscale recurrent neural networks". In: *arXiv preprint arXiv:1609.01704* (2016).

[78] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. “The cityscapes dataset for semantic urban scene understanding”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, pp. 3213–3223.

[79] Francesco Cricri, Mikko Honkala, Xingyang Ni, Emre Aksu, and Moncef Gabbouj. “Video Ladder Networks”. In: *arXiv preprint arXiv:1612.01756* (2016).

[80] Alexey Dosovitskiy and Thomas Brox. “Generating images with perceptual similarity metrics based on deep networks”. In: *Advances in Neural Information Processing Systems.* 2016, pp. 658–666.

[81] Chelsea Finn, Ian goodfellow, and Sergey Levine. “Unsupervised Learning for Physical Interaction through Video Prediction”. In: 2016.

[82] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. “DeepStereo: Learning to predict new views from the world’s imagery”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, pp. 5515–5524.

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

[84] Klaus Greff, Antti Rasmus, Mathias Berglund, Tele Hotloo Hao, Jurgen Schmidheber, and Harri Valpola. “Tagger: Deep Unsupervised Perceptual Grouping”. In: 2016.

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

[86] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Identity mappings in deep residual networks”. In: *European Conference on Computer Vision.* Springer. 2016, pp. 630–645.

[87] Nal Kalchbrenner, Aaron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuogluk. “Video Pixel Networks”. In: vol. 42. 2016, pp. 23–42.

[88] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron Courville, et al. “Zoneout: Regularizing rnns by randomly preserving hidden activations”. In: *arXiv preprint arXiv:1606.01305* (2016).

[89] Yongxi Lu, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, Tara Javidi, and Rogerio Feris. “Fully-adaptive Feature Sharing in Multi-Task Networks with Applications in Person Attribute Classification”. In: *arXiv preprint arXiv:1611.05377* (2016).

[90] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. “Cross-stitch networks for multi-task learning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, pp. 3994–4003.

[91] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. “Pixel recurrent neural networks”. In: *arXiv preprint arXiv:1601.06759* (2016).

[92] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. “Conditional image generation with pixelcnn decoders”. In: *Advances in Neural Information Processing Systems.* 2016, pp. 4790–4798.

[93] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. “The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016, pp. 3234–3243.

[94] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. “Meta-learning with memory-augmented neural networks”. In: *International conference on machine learning.* 2016, pp. 1842–1850.

[95] Shreyas Saxena and Jakob Verbeek. “Convolutional neural fabrics”. In: *Advances in Neural Information Processing Systems.* 2016, pp. 4053–4061.

[96] Mennatullah Siam, Sepehr Valipour, Martin Jagersand, and Nilanjan Ray. “Convolutional Gated Recurrent Networks for Video Segmentation”. In: *arXiv preprint arXiv:1611.05435* (2016).

[97] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. “Multi-view 3d models from single images with a convolutional network”. In: *European Conference on Computer Vision.* Springer. 2016, pp. 322–337.

[98] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. “An uncertain future: Forecasting from static images using variational autoencoders”. In: *European Conference on Computer Vision.* Springer. 2016, pp. 835–851.

[99] Alejandro Weinstein, Wael El-Deredy, Stéren Chabert, and Myriam Fuentes. “Fitting Human Decision Making Models using Python”. In: *Proceedings of the 15th Python in Science Conference*. Ed. by Sebastian Benthall and Scott Rostrup. 2016, pp. 1–6.

[100] Barret Zoph and Quoc V Le. “Neural architecture search with reinforcement learning”. In: *arXiv preprint arXiv:1611.01578* (2016).

[101] Joachim Bingel and Anders Søgaard. “Identifying beneficial task relations for multi-task learning in deep neural networks”. In: *arXiv preprint arXiv:1702.08303* (2017).

[102] Katerina Fragkiadaki, Jonathan Huang, Alex Alemi, Sudheendra Vijayanarasimhan, Susanna Ricco, and Rahul Sukthankar. “Motion Prediction Under Multimodality with Conditional Stochastic Networks”. In: *arXiv preprint arXiv:1705.02082* (2017).

[103] Sebastian Ruder. “An overview of multi-task learning in deep neural networks”. In: *arXiv preprint arXiv:1706.05098* (2017).

[104] Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. “Sluice networks: Learning what to share between loosely related tasks”. In: *arXiv preprint arXiv:1705.08142* (2017).

[105] Ruben Villegas, Jimei Yang, Seunghoon Hong, Xunyu Lin, and Honglak Lee. “Decomposing motion and content for natural video sequence prediction”. In: *ICLR* 1.2 (2017), p. 7.

[106] Ruben Villegas, Jimei Yang, Yuliang Zou, Sungryull Sohn, Xunyu Lin, and Honglak Lee. “Learning to Generate Long-term Future via Hierarchical Prediction”. In: *arXiv preprint arXiv:1704.05831* (2017).

[107] Jörg Wagner, Volker Fischer, Michael Herman, and Sven Behnke. “Learning Semantic Prediction using Pretrained Deep Feedforward Networks”. In: *Proceedings of 25th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)* (2017).

[108] Jacob Walker, Kenneth Marino, Abhinav Gupta, and Martial Hebert. “The Pose Knows: Video Forecasting by Generating Pose Futures”. In: *arXiv preprint arXiv:1705.00053* (2017).

[109] Nicholas Watters, Andrea Tacchetti, Theophane Weber, and Theophane Weber. “Visual Interaction Networks”. In: 2017.

[110] Denny Britz. URL: `http://www.wildml.com`.

[111] Christopher Olah. URL: `http://colah.github.io`.