A Dynamic Convolutional Layer for Short Range Weather Prediction

Benjamin Klein, Lior Wolf and Yehuda Afek
The Blavatnik School of Computer Science
Tel Aviv University
beni.klein@gmail.com, wolf@cs.tau.ac.il, afek@post.tau.ac.il

Abstract

We present a new deep network layer called "Dynamic Convolutional Layer" which is a generalization of the convolutional layer. The conventional convolutional layer uses filters that are learned during training and are held constant during testing. In contrast, the dynamic convolutional layer uses filters that will vary from input to input during testing. This is achieved by learning a function that maps the input to the filters. We apply the dynamic convolutional layer to the application of short range weather prediction and show performance improvements compared to other baselines.

1. Introduction

Deep learning and specifically Convolutional Neural Networks (CNNs) [16] are becoming increasingly popular in solving various computer vision applications. What sets CNNs apart from other neural networks is the use of the convolutional layer. This layer computes the output feature maps by convolving the feature maps of the previous layer with a set of filters. These filters are the only parameters of the convolutional layer and are learned during training, typically by using the back-propagation algorithm.

In recent years, CNNs have achieved state-of-the-art results on a variety of challenging problems such as: object recognition, objection localization, cancer detection, face recognition and scene labelling. [15, 5, 24, 25, 26, 11]. Motivated by the success of CNNs, we decided to examine their performance in the task of short range weather prediction. In this task, one receives a sequence of rain radar images (Figure 1), one image taken every 10 minutes, and the goal is to predict the next image in the sequence. While we understand the critical role of the convolutional layer in recognition problems, we think that a different network architecture could be more suitable for the task of short range weather prediction. Specifically, observing that a radar image in the sequence can be usually approximated as a translation of the previous image in the sequence, suggested the need for a new layer that will translate the last image in the sequence according to the motion behavior of the all sequence.

Motivated by this observation, we present a new deep network layer called the "Dynamic Convolutional Layer", which generalizes the conventional convolutional layer. Similar to the convolutional layer, the dynamic convolutional layer takes the feature maps from the previous layer and convolves them with filters. The novelty lies in that the filters of the dynamic convolutional layer are not the parameters of the layer, rather they are obtained as the output of a subnetwork of arbitrary depth that maps the input to a set of filters (Figure 3).

In this work, we present the dynamic convolutional layer and use it for the task of short range weather prediction. We compare the results to other baselines, including Convolutional Neural Network that does not use the dynamic convolutional layer. We show that by using the new layer, we gain improvement in performance compared to the other baselines, including the conventional CNN.

2. Related Work

In recent years, a number of papers suggested adaptations to the convolutional layer. In [17], Lin et al. suggested a novel deep network structure called "Network In Network" (NIN). This structure replaces the linear filters in the conventional convolutional layer with a multilayer perceptron. Using this novel structure, they achieved state-of-the-art performance on CIFAR-10 and CIFAR-100. This work differs from ours in two major ways: first, it replaces the convolutions with a multilayer perceptron. Second, the new blocks are entered sequentially to replace the convolutional layers, i.e., the new block has the same input layer and the same output layer as the convolutional layer it replaces. In our case, we adhere to the convolution operator, which is suitable to the image synthesis tasks that we focus on (NIN is employed for classification). The dynamic convolutions are computed using separate subnetworks that can take the output of the arbitrary previous layer as input.

In parallel to our work, Cohen et al. [6] present a generalization of CNNs. This generalization adds masking weights

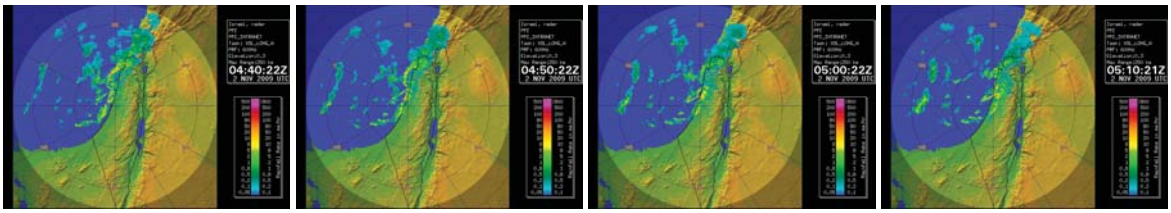

Figure 1. A sequence of 4 radar images from the Tel Aviv Dataset before the preprocessing. Each two consecutive images were taken 10 minutes apart.

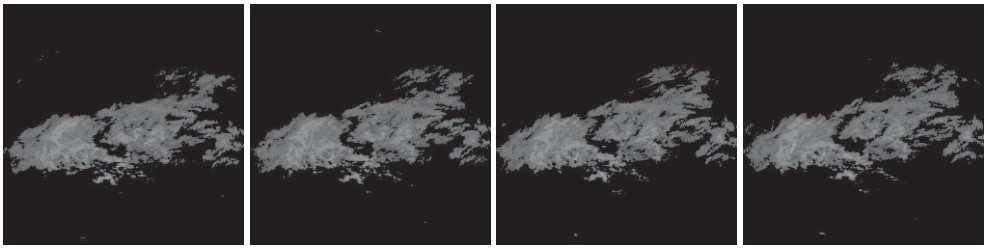

Figure 2. A sequence of 4 radar images from the Davenport Dataset after the preprocessing. In the preprocessing, a constant color transformation function was applied on every image, in order to transform the cloud intensity coded by a color map to image intensities.

to the similarity layers that generalize the conventional convolutional layer. The weights are learned during optimization and not computed while evaluating new samples, in contrast to our work. In addition, whenever the similarity layer computes linear correlations and not functions of distances, the weights can be simply absorbed into the filters. In our case, the dynamic nature of the filters cannot be expressed by conventional networks.

In each dynamic convolutional layer, the interaction between the input from the previous layer and the filter obtained from another layer is of a multiplicative nature. In the literature, multiplicative interactions appear when computing similarities between two images [1, 9] employing autoencoders and for learning interactions of two consecutive frames for action recognition [27] employing restricted Boltzmann machines. Multiplicative interactions were also used to link hidden variables with the network's input [20, 19], where the hidden variables are eliminated using marginalization. In our networks, both paths leading to the multiplicative interaction are obtained from the network's input in a feedforward manner. The input is not split into two parts that correspond to two images or two frames, each propagated separately.

Multiplicative interactions also appear as part of the Sigma-Pi units [21], in which the inputs of the previous layers are multiplied to create non-linear interactions. In contrast to our work, this very early contribution considers direct interactions of different elements from the same input layer and is explored in the context of simple artificial neural networks. More references to previous work on multiplicative interactions can be found in [18].

Weather prediction As an application, we employ weather surveillance radar images in order to make short term forecasts of future location and intensity of rain and snow. Such predictions supports both casual usages, such as helping users decide whether to ride their bike to school, and for society-level emergency alerts such as predicting flash floods. In between, the output predictions can be used routinely to predict traffic and be integrated into airport weather systems.

The literature on the subject seems to be much more focused on the physical properties of weather radars [28], on models for calibrating their output with land measurements [3], and on integrating the obtained data, using commercial products, with other weather modalities [7]. While it is not clear how commercial products work, it is apparent that the usage of recent advances in computer vision to the task of rain weather prediction is still largely lacking.

3. Dynamic Convolution Layer

In this section, we describe the dynamic convolution layer, which is a generalization of the convolution layer. The conventional convolution layer is explained is 3.1 and the dynamic convolution layer is explained in 3.2. The discussion explores how the two layers differ from one another in terms of input, output, forward pass and backward pass.

3.1. The Conventional Convolution Layer

The convolution layer receives a single input - the feature maps from the previous layer. The layer computes feature maps as its output by convolving filters across the feature

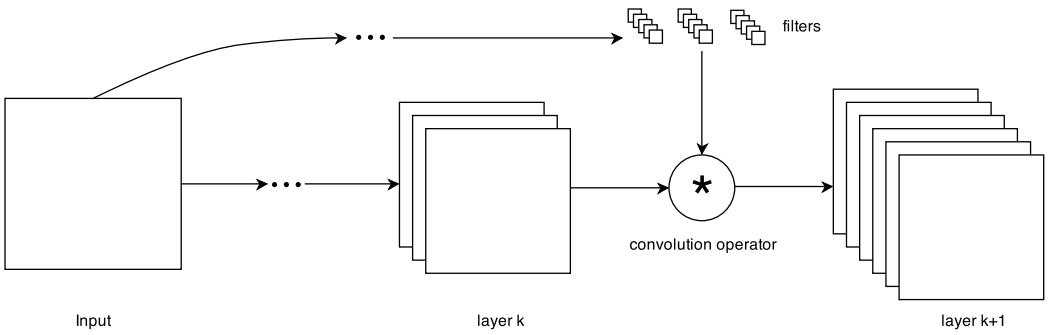

Figure 3. The dynamic convolutional layer. Layer $k+1$ is computed by convolving the filters across the feature maps in the k layer. The filters themselves are the result of applying a convolutional network on the input.

maps from the previous layer. These filters are the parameters of the convolution layer and are learned during training by using back-propagation [22]. During testing, they are held fixed and do not change from one sample to another.

Forward Pass The learning is usually done in batches of T samples. We shall denote by x_i^t, the i-th input feature map of sample t and by y_j^t the j-th output feature map of sample t. The filters would be denoted by k_{ij}. In the forward pass of the convolutional layer, the output feature maps are calculated using the convolution operator (denoted by *):

$$y_j^t = \sum_i k_{ij} * x_i^t \tag{1}$$

Backward Pass In the backward pass, the convolution layer computes the gradient of the network's loss function l with respect to x_i^t:

$$\frac{\partial l}{\partial x_i^t} = \sum_j \left(\frac{\partial l}{\partial y_j^t} \right) \underline{*} (k_{ij}) \tag{2}$$

Where $\underline{*}$ is the convolution with zero padding. As part of the back-propagation algorithm, the values of the gradient $\frac{\partial l}{\partial x_i^t}$ are then passed to the previous layer, the one that computed x_i^t. Additionally, the gradient of the loss function with respect to k_{ij} is computed:

$$\frac{\partial l}{\partial k_{ij}} = \frac{1}{T} \sum_t \left(\frac{\partial l}{\partial y_j^t} \right) * (\widetilde{x_i^t}) \tag{3}$$

Where $\widetilde{x_i^t}$ is the row/column flipped version of x_i^t. After computing $\frac{\partial l}{\partial k_{ij}}$, the parameters k_{ij} of the layer are updated by using gradient descent:

$$k_{ij} = k_{ij} - \alpha \cdot \frac{\partial l}{\partial k_{ij}} \quad , \tag{4}$$

where α is the learning rate.

3.2. The Dynamic Convolution Layer

In contrast to the convolution layer, the dynamic convolution layer receives two inputs. The first input is the features maps from the previous layer and the second is the filters. The feature maps are obtained from the input by following a sub-network A. The filters are the result of applying a separate convolutional sub-network B on the input. The output of the layer is computed by convolving the filters across the features maps from the previous layer in the same way as in the convolution layer but here the filters are a function of the input and therefore vary from one sample to another during test time. The whole system is a directed acyclic graph of layers and therefore the training is done by using the back-propagation algorithm.

Forward Pass In the forward pass, network A computes the feature maps that will be given to the dynamic convolution layer as the first input and the separated sub convolution network B computes the filters that will be given to the dynamic convolution network as the second input (Figure 3). Let x_i^t be the i-th input feature map of sample t, let k_{ij}^t be the ij input kernel of sample t and let y_j^t be the j-th output feature map of sample t, then in the forward pass of the dynamic convolution network, the output feature maps are calculated as follows:

$$y_j^t = \sum_i k_{ij}^t * x_i^t \tag{5}$$

Notice that in contrast to the conventional convolution layer, in the dynamic convolution layer every sample has a different kernel k_{ij}^t.

Backward Pass In the backward pass, the dynamic convolution layer computes the gradient of the loss function l with respect to x_i^t similarly to before:

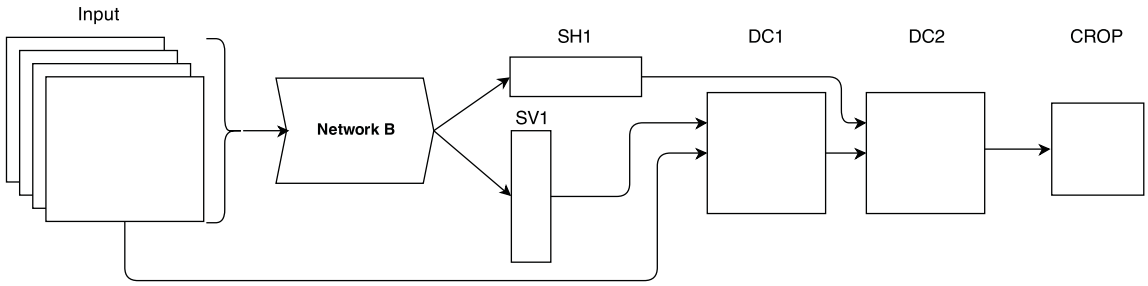

Figure 4. The architecture of the network. Network B is a sub-network which computes the filters (H1 and V1) used by the dynamic convolution layers. SH1 is the result of applying a softmax function on H1 and SV1 is the result of applying a softmax function on V1. DC1 is a dynamic convolution layer that takes the last image in the sequence and convolves it with SV1. DC2 is a dynamic convolution layer that is takes DC1 and convolves it with SH1.

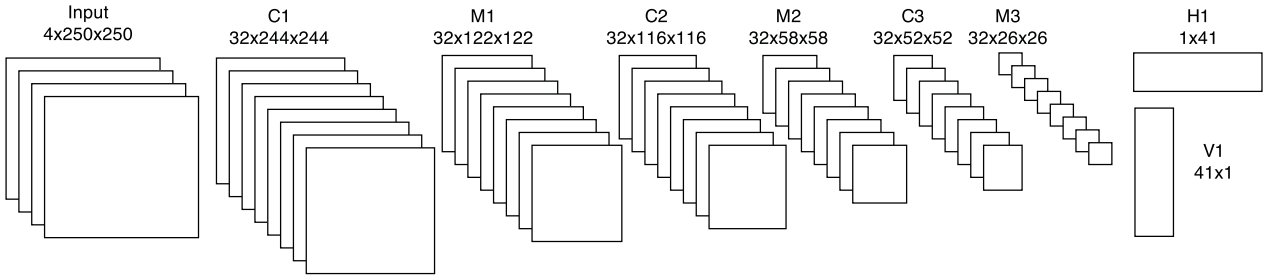

Figure 5. The architecture of the network B for the Whole Image Synthesis. C1, C2 and C3 are conventional convolution layers and M1, M2 and M3 are max-pooling layers. After each max-pooling layer, we apply $tanh$ nonlinearity activation function. H1 and V1 are each connected to M3 by a fully connected layer.

$$\frac{\partial l}{\partial x_i^t} = \sum_j \left(\frac{\partial l}{\partial y_j^t} \right) \underline{*} (k_{ij}^t) \tag{6}$$

The values of the gradient $\frac{\partial l}{\partial x_i^t}$ are passed to the layer in network A that produced x_i^t. Additionally, and similarly to the conventional convolutional layer, the gradient of the loss function with respect to k_{ij}^t is computed:

$$\frac{\partial l}{\partial k_{ij}^t} = \left(\frac{\partial l}{\partial y_j^t} \right) * (\widetilde{x_i^t}) \tag{7}$$

In contrast to the convolution layer, k_{ij}^t are not parameters of the layer - they are a function of the input t that are passed from a previous layer in network B. Therefore, the values of the gradient $\frac{\partial l}{\partial k_{ij}^t}$ are passed to the layer that computed k_{ij}^t as part of the back-propagation algorithm.

4. Whole Image Synthesis

The whole image synthesis method receives the four 250×250 radar images as an input and outputs the estimated next image in the sequence. The estimation is performed using a DNN that contains the dynamic convolution layer. In practice, the prediction contains the 200×200 center patch of the next image in sequence, since the boundaries are significantly affected by clouds that are not seen in the previous radar images.

Network Architecture The overall architecture is shown in Figure 4. The four radar images of size 250×250 are given as a four channel input to a conventional convolutional layer $(C1)$ with 32 filters of size $7\times7\times4$. The resulting 32 feature maps are then passed to a max-pooling layer $(M1)$ which takes the max over 2×2 spatial blocks with a stride of 2. This is followed by another conventional convolutional layer $(C2)$ with 32 filters of size $7\times7\times32$. The resulting 32 feature maps are then passed to another max-pooling layer $(M2)$ which takes the max over 2×2 spatial blocks with a stride of 2. This is followed by another conventional convolutional layer $(C3)$ with 32 filters of size $7\times7\times32$. The resulting 32 feature maps are then passed to another max-pooling layer $(M3)$ which takes the max over 2×2 spatial blocks with a stride of 2. A fully connected layer transforms the values in $(M3)$ into a $1D$ horizontal vector $H1$ of size 1×41 and a second fully con-

nected layer transforms the values in $(M3)$ into $1D$ vertical vector $V1$ of size 41×1. Softmax operation is applied on $V1$ and on $H1$ resulting in the vectors $SV1$ and $SH1$. The dynamic convolution layer is now being used for the first time. The last radar image in the sequence (last in time manners) is taken and convolved with the filter $SV1$ by using the dynamic convolution layer $(DC1)$. The resulting feature map is now introduced as an input to a second dynamic convolution layer $(DC2)$ and is convolved with the filter $SH1$. The 200×200 center patch of $DC2$ is taken using a crop layer $CROP1$ and is passed during training, to an Euclidean loss layer with the true 200×200 center patch of the next sequence. The $tanh$ nonlinearity is used after each max-pooling layer.

Intuition The network architecture gives meaning to the vectors $SV1$ and $SH1$. Each one of these two vectors can be seen as a vector of probabilities - they are positive and sum to 1 (the result of applying a softmax function). Since $SV1$ is being used as a vertical filter that the last image in the sequence is being convolved with, it preforms a translation in the vertical axis. Similarly, $SH1$ is a horizontal filter that the resulting feature map is being convolved with, thereby performing translation in the horizontal axis. Therefore, the network learns to map the input sequence of 4 frames and produce probability vectors that predict the proper translation in the horizontal and vertical axes that would transform the last image in the sequence to the next, unseen one.

5. Patch by Patch Synthesis

The whole image synthesis method works well in cases where all the objects in the sequence are translated by the same amount. In order to deal with more complicated scene motions, a patch by patch synthesis is presented next. In this method, a DNN is trained on patches (small image regions) of the radar sequence. The input of this network is a sequence of four radar images of size 70×70 and its output is the 10×10 center patch of those patches for the next image in the sequence. In order to compute the next image in the sequence, the DNN is applied in a sliding window fashion. This sliding window approach has been applied successfully in the past, such as in the application of detecting mitosis in breast histology images [5].

Network Architecture The network architecture is very similar to the network architecture of whole image synthesis and therefore we only described the difference between the two. Due to the smaller size of the input, we remove the last convolution layer $(C3)$ and the last max-pooling layer $(M3)$. Additionally, the crop layer $(CROP1)$ takes the 10×10 center patch of $(DC2)$.

6. Results

The dynamical convolution layer was developed, tested and compared to alternatives on three very large datasets for short range weather prediction. We describe the data sets in Section 6.1 and the preprocessing steps in Section 6.2. Training and implementation details are described in 6.3. The methods that we compared ourselves to are described in 6.4. Finally, a comparison of the accuracy of all the methods on the three datesets is done in 6.5.

6.1. Datasets

We are using three data sets. The first dataset contains radar images that were taken in Tel Aviv, Israel, the second dataset contains radar images that were taken in Davenport, Iowa and the third dataset contains radar images that were taken in Kansas City, Missouri. Each dataset was split into train, validation and test sets such that each set contains sequences that were taken on a different range of years - in order to prevent contamination. On many days, many sequences are empty or nearly empty. These sequences are easy to predict and are therefore filtered out from the benchmark.

All three datasets were subsampled to create manageable experiments. The three training sets contain $32,000$ sequences each, the validation sets contains $4,800$ sequences per radar dataset and each test set contains $3,200$ sequences.

6.2. Preprocessing

The radar images of the Tel Aviv dataset contain a map as a background image (Figure 1). In order to build meaningful models, background subtraction is first applied to every image in this dataset.

Next, for all three datasets, we apply a constant color transformation function on every image, in order to transform the color maps depicting the cloud intensity to image intensities (Figure 2). Note that simply applying a gray scale transformation to the image would not produce the desirable results since the colorbar of the radar images is not monotonic under the gray scale transformation. We then crop and scale the image to 250×250 pixels.

6.3. Implementation Details

The Dynamic Convolutional Layer was implemented using the open source Caffe library [14]. All the weights in the DNN were initialized according to the Xavier initialization [13]. The training was done using stochastic gradient descent (SGD) with a learning rate of 0.001 and a momentum of 0.9.

6.4. Baseline Methods

The dynamic convolutional layer was compared to several alternative techniques that are described next. Notice

Method	Tel Aviv Dataset	Davenport Dataset	Kansas City Dataset
Last Frame	20.059±0.536	258.818±2.552	241.392±2.975
Global Motion Estimator	16.837±0.496	173.402±1.547	179.953±2.065
Patch Based Linear Regression	13.002±0.435	164.854±1.377	160.489±1.682
Patch Based CNN	11.480±0.431	105.242±0.839	101.880±1.042
Whole Image Dynamic Convolution Network	12.340±0.461	117.316±0.929	118.402±1.174
Patch Based Dynamic Convolution Network	11.114±0.412	101.983±0.802	98.790±0.995

Table 1. Results. Comparison of our models with competitive methods on three datasets.

that linear regression and the CNN models were not used for the entire image, only for patches. This is due to the larger output size of the entire image approach that would have dramatically increased the number of parameters in the model, making these models less feasible.

Last Frame The prediction and the input sequence have strong correlations and therefore one can simply take the last frame as the prediction. Since the last frame is expected to be the closest to the predicted frame, this baseline already performs rather well.

Global Motion Estimator A global motion vector (dx, dy) is inferred using a black-box motion estimator, namely, opencv's Keypoint Based Motion Estimation function [4]. The last image is then translated according to this vector and the result is the prediction. Notice that this method sometimes fails when the motion is not estimated successfully for various reasons.

Patch Based Linear Regression A linear regression model that transforms a sequence of four radar image patches of 70×70 into the 10×10 center patch of the next image patch in the sequence is learned on the training set. The 200×200 central regions of the next image in the sequence is then computed by applying the linear regression model in a sliding window fashion.

Patch Based Convolutional Neural Network A CNN model that transforms a sequence of four radar image patches of 70×70 into the 10×10 center patch of the next image patch in the sequence is learned on the training set. The next image in the sequence is computed by repeatedly applying the network in a sliding window fashion. The architecture of the CNN is as follows: The four radar image patches of size 70×70 are given to a conventional convolutional layer $(C1)$ with 32 filters of size $7\times7\times4$. The resulting 32 feature maps are then passed to a max-pooling layer $(M1)$ which takes the max over 2×2 spatial blocks with a stride of 2. This is followed by another conventional convolutional layer $(C2)$ with 32 filters of size $5\times5\times32$. The resulting 32 feature maps are then passed to another max-pooling layer $(M2)$ which takes the max over 2×2 spatial blocks with stride of 2. This is followed by another conventional convolutional layer $(C3)$ with 32 filters of size $3\times3\times32$. The resulting 32 feature maps are then passed to a fully connected layer $F1$ that produces a vector with 1600 entries. Finally, the 1600 entries are passed to a fully connected layer $F2$ that produces a vector with 100 entries, which is the network's prediction.

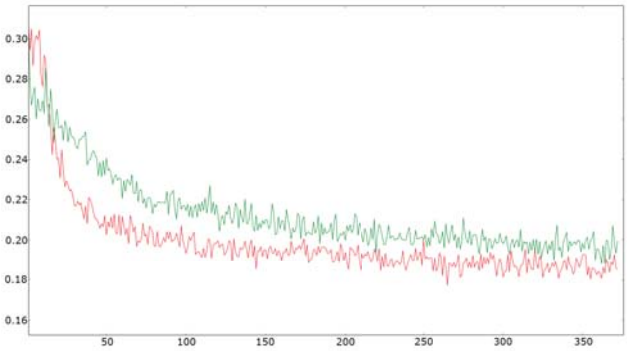

Figure 6. The validation error rate on the Kansas City validation set as a function of the epoch. The x-axis is the epoch and the y-axis is the Euclidean loss on the validation set. Shown are the error of the Patch Based CNN (green) and the error of the Patch Based Dynamic Convolution Network (Red).

The specific network architecture was chosen in a trial and error fashion to obtain the best performance on the validation set. We examined around 20 architectures both smaller and bigger and chose the best performing network for this comparison.

6.5. Accuracy Comparison

For each method and for each each sample t, the (squared) Euclidean loss is computed:

$$\sum_{i=1}^{200}\sum_{j=1}^{200}(\widehat{y_{ij}^t} - y_{ij}^t)^2 \quad , \tag{8}$$

where $\widehat{y^t}$ denotes the predicted image. The mean results, as well as the standard errors (SE) are presented in Table

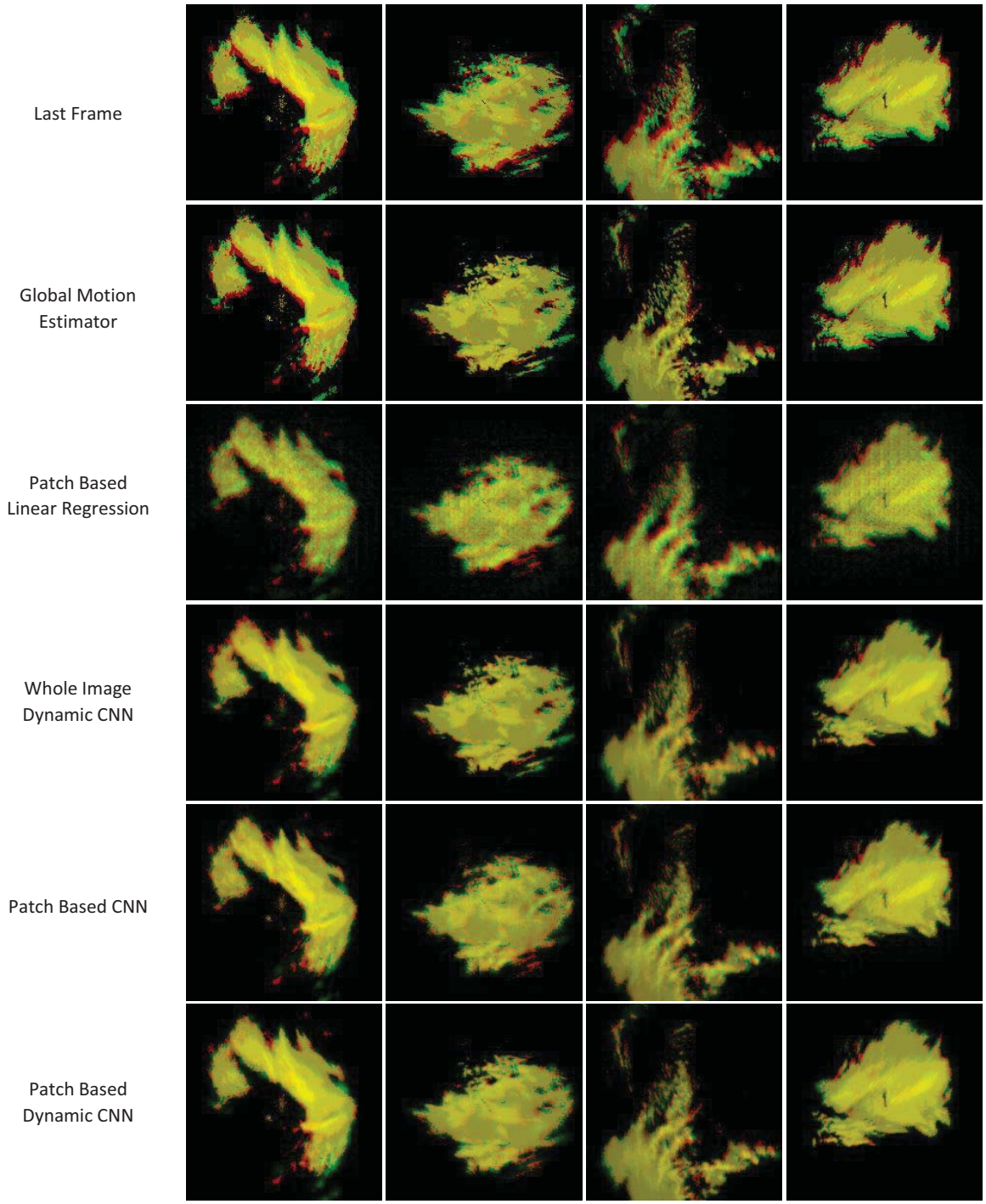

Figure 7. Each row represents a model and each column represents a sequence. In each image, the red color represents the ground truth - the real next radar image in the sequence, the green color represents the prediction according to the specific model and the yellow color represents the overlapping between the prediction and the ground truth.

1. All differences seen in the Table were verified to be statistically significant at a significance level of 0.01 using a paired t-test, except for the difference between Patch Based CNN and Patch Based Dynamic Convolution Network on the Tel Aviv dataset. As can be seen, the Tel-Aviv dataset has the smallest error. This is due to the scarcity and sparseness of rain clouds over Israel in comparison to the two US locations. Among the methods, the patch based dynamic CNN provides the lowest error rates in all three datasets. The next best performing method is the patch based conventional CNN and the following best performing method is the whole image dynamic CNN. The linear regression and the global motion estimators do not perform as well as the CNN based methods.

Additionally, due to the fact that the Tel-Aviv dataset lacks rain clouds , we compute the performance of the Tel-Aviv dataset Patch Based CNN and Patch Based Dynamic CNN models that were learned on the US dataset. As a result, the Euclidean loss of the Patch Based Dynamic CNN improved to 10.766 ± 0.414 and the Euclidean loss of Patch Based CNN degraded to 11.708 ± 0.536.

The convergence of the dynamic CNN method follows a typical pattern. The error rate of the network starts higher than the analogous conventional CNN, but after a few epochs the error rate drops sharply and remains lower than its counterpart. A typical run is depicted in Figure 6. Overall, the convergence of the patch based dynamic CNN is much faster than the patch based conventional CNN. In order to supply a fair comparison of the convergence, the running time of a single epoch was computed for the Patch Based Dynamic CNN and Patch Based CNN by taking the mean of 31 epochs. The running time of a single epoch in the Patch Based CNN is 189.935 ± 3.950 and in the Patch Based Dynamic CNN is 233.677 ± 10.331. Therefore, a single epoch in the Patch Based Dynamic CNN is a bit slower than in the Patch Based CNN, but the overall convergence of the Patch Based Dynamic CNN is faster than the Patch Based CNN.

Examples of radar image sequences and next frame prediction for every model are shown in Figure 7.

7. Discussion

The current emphasis in the recent deep learning literature is on muticlass classification. However, deep learning networks can be used for super resolution [10], blind-image deconvolution [23], noise removal, and other image processing tasks for which the output is an image. Such domains can benefit from incorporating dynamic filters that are constructed based on the input image.

In addition, deep learning systems that combine recognition with detection and segmentation, such as the semantic segmentation line of work [2, 8] could also benefit from dynamic filters. Such filters would allow the system to adapt to multiple scenarios and handle a wider variety of scenes.

While not directly related to the dynamic convolutional layer itself, the error signal which we propagate stems from the Euclidean error. This might be suboptimal for image synthesis applications since it does not take into account the structure of images. For example, blurry images, which are not visually appealing, sometimes get lower error rates than more naturally looking but shifted images. A possible way to address this is by incorporating graphical models on top of the learned convolutional neural networks, e.g., [12].

Lastly, we show that a network that contains a dynamic convolution layer outperforms a network which does not. Since the dynamic layer incorporates a subnetwork for computing the filters, the overall network becomes more complex. To be fair, we tried to make the baseline network deeper than the proposed one and have tested multiple alternatives, learning rates, and regularization schemes in order to obtain the the best performing baseline network. The proposed network which is the one incorporating the dynamic filter, was much easier to obtain. It is possible that the use of more complex network components, such as the dynamic convolutional layer, will ultimately lead to straightforward network architectures.

Acknowledgments

This research is partly supported by a Microsoft Azure Research Award. The Tel Aviv dataset was obtained from the Israel Meteorological Service.

References

[1] D. Alain and S. Olivier. Gated autoencoders with tied input weights. In S. Dasgupta and D. Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 154–162. JMLR Workshop and Conference Proceedings, May 2013.

[2] J. M. Alvarez, Y. LeCun, T. Gevers, and A. M. Lopez. Semantic road segmentation via multi-scale ensembles of learned features. In *Proceedings of the 12th International Conference on Computer Vision - Volume 2*, ECCV'12, pages 586–595, Berlin, Heidelberg, 2012. Springer-Verlag.

[3] L. Besson, C. Boudjabi, O. Caumont, and J. Parent du Chatelet. Links between weather phenomena and characteristics of refractivity measured by precipitation radar. *Boundary-Layer Meteorology*, 143(1):77–95, 2012.

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] D. C. Cireşan, A. Giusti, L. M. Gambardella, and J. Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2013*, pages 411–418. Springer, 2013.

[6] N. Cohen and A. Shashua. Simnets: A generalization of convolutional networks. *arXiv preprint arXiv:1410.0781*, 2014.

[7] C. G. Collier. Flash flood forecasting: What are the limits of predictability? *Quarterly Journal of the Royal Meteorological Society*, 133(622):3–23, 2007.

[8] C. Couprie, C. Farabet, L. Najman, and Y. LeCun. Toward real-time indoor semantic segmentation using depth information. *JMLR*, 2014.

[9] A. Dehghan, E. Ortiz, R. Villegas, and M. Shah. Who do i look like? determining parent-offspring resemblance via gated autoencoders. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1757–1764, June 2014.

[10] C. Dong, C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision ECCV 2014*, volume 8692 of *Lecture Notes in Computer Science*, pages 184–199. Springer International Publishing, 2014.

[11] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1915–1929, 2013.

[12] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, August 2013.

[13] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[17] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[18] R. Memisevic. Learning to relate images: Mapping units, complex cells and simultaneous eigenspaces. *CoRR*, pages –1–1, 2011.

[19] R. Memisevic and G. Hinton. Unsupervised learning of image transformations. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007.

[20] R. Memisevic, C. Zach, G. Hinton, and M. Pollefeys. Gated softmax classification. *Advances in Neural Information Processing Systems*, 23:1–9, 2010.

[21] D. E. Rumelhart, G. E. Hinton, and J. L. McClelland. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter A General Framework for Parallel Distributed Processing, pages 45–76. MIT Press, Cambridge, MA, USA, 1986.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Wilson. Learning representations by back-propagating errors. pages 533–536, 1986.

[23] C. J. Schuler, M. Hirsch, S. Harmeling, and B. Schölkopf. Learning to deblur. *arXiv preprint arXiv:1406.7444*, 2014.

[24] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[26] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1701–1708. IEEE, 2014.

[27] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. In *Proceedings of the 11th European Conference on Computer Vision: Part VI*, ECCV'10, pages 140–153, Berlin, Heidelberg, 2010. Springer-Verlag.

[28] T. Weckwerth, C. Pettet, F. Fabry, S. Park, M. LeMone, and J. Wilson. Radar refractivity retrieval: Validation and application to short-term forecasting. *Journal of Applied Meteorology*, 44(3), 2005.