Two Fast and Efficient Approaches to Rainfall Nowcasting with Recurrent Convolutional Neural Networks

Bertrand Delorme
Department of Earth System Science
Stanford University
bdelorme@stanford.edu

Vincent Dufour-Décieux
Department of Materials Science and Engineering
Stanford University
vdufourd@stanford.edu

Abstract

Deep learning methods have recently been used to perform rainfall nowcasting, which consists in predicting precipitations at very short term. In this paper, we present two novel architectures to tackle this issue based on Recurrent Convolutional Neural Networks: the DDnet and the ConvDLRM. After developing those architectures on the MovingMNIST dataset, we used them to perform rainfall nowcasting with a one-hour time horizon. The architectures that we developed have the advantage to be flexible and computationally light, allowing us to add other features than rainfall radar images. In this paper, we show that adding new features such as pressure improved the performance of the models.

1. Introduction

Rainfall nowcasting consists in forecasting precipitation with a time horizon less than 2 hours. It has crucial implications for extreme event anticipation, tourism and agriculture. For years, it has remained a challenge in meteorology despite the availability of techniques such as numerical weather prediction based on physical models or simple extrapolation of radar images. A major drawbacks of these numerical weather model is their computational cost, which is exponentially proportional to their spatial and temporal resolution. In addition, they need to be continuously ran to provide near-term forecasts. Therefore, people have used optical flow based methods to perform extrapolation of radar maps [11, 7, 20, 4, 6]. Recently, the deep learning community has also put effort into this problem and developed new solutions that outperform these traditional methods [13, 14, 9, 17, 12, 1, 2, 10, 3, 15].

Based on this recent progress, Meteo France (the French National Weather Agency) decided to publicly release a new dataset [8], and challenged the data science community to obtain accurate weather forecasts using data-driven methods. The Meteo France dataset has the advantage of containing information about rainfall based on radar images but also other variables coming from traditional weather forecast models or ground truth stations.

While most of the previous work on rainfall nowcasting uses radar images only, the availability of other features such as wind speed, pressure or temperature could significantly improve the forecast through the dependence between these different atmospheric variables. Indeed, a major limitation of previous works is that they learn how objects move through a sequence of images, but are unable to represent the dynamics of cloud initiation or decay that have tremendous importance for rainfall nowcasting. However, including more features that correspond to two-dimensional maps inherently adds computational constrains that most of the architectures proposed in previous works cannot handle efficiently.

In this work, our goal is to use Meteo France dataset to design and train an accurate and efficient prediction model for rainfall nowcasting. We propose two new architectures to tackle this problem that have the capacity to use other features in addition to radar images. These architectures have been designed to remain computationally light for both training and testing, so that they could be use in an operational way for real-time precipitation even in the absence of powerful computational resources. Both of our model uses a convLSTM cell [13] to model the temporal dependence of a sequence of input radar images, and a downsampler-upsampler scheme based on convolutional layers to encode-decode the images and handle the memory requirements at train-time. In both cases, we are able to output a new sequence of radar images corresponding to the rainfall rate in the future. The code that we developed for this paper in Tensorflow (v2.2.0) is available at https://github.com/bdelorme/RainfallNowcasting.git.

The remainder of this paper is organized as follows. Section 2 presents previous work that have been done on the same topic. Section 3 introduces our models and explain the choices we made to develop them. introduces the numerical

model and describes the simulation setup. Section 4 introduces the datasets that we used to train and test our models. Section 5 shows our results, and Section 6 concludes by discussing the strengths and limitations of our work as well as potential steps to go further.

2. Related work

In the deep learning community, people have used two different approaches to tackle the problem of precipitation nowcasting based on radar images.

The first approach consists in using a Convolutional Neural Network (CNN) to transform the sequence of past input images into a single future image. [2] used a simple "all convolutional neural nets" architecture and had performance comparable with state-of-the-art optical flow algorithms. Further studies managed to outperform these optical flow methods by using the ubiquitous U-Net architecture [10, 1, 3]. However, these architectures fail at predicting the rainfall rate for more than a few minutes in the future since an error early in the prediction will tend to be amplified for the later times.

The second approach explicitly models time by using a Recurrent Neural Network (RNN) at its core. This line of research is based on pioneer work by [13], which proposed to use convLSTM cells to tackle the rainfall nowcasting problem. In a convLSTM cell, the fully-connected linear product of an LSTM is replaced with a convolution product at the entrance of the cell. The convLSTM developed by [13] has since then been widely adopted in the broader community of video forecasting thanks to its natural approach to model spatio-temporal dynamics. With a model based on convLSTM cells, [13] introduced the first deep-learning method that could outperform traditional optical flow based methods for precipitation nowcasting. This has motivated a nice line of research that tried to find the best neural network architectures that could make use of the convLSTM idea [9, 14, 17, 12]. A successful recent example is MetNet, a neural network capable of predicting rainfall over the United States for up to 8 hours in the future developed by [15]. MetNet is the first deep learning model capable of predictions at these scales and shows the tremendous potential that deep learning has to improve weather forecasting. However, MetNet requires a lots of engineering and an important amount of ressources to acquire and process the data, and then train the model.

While the first approach has the advantage of being computationally less expensive, the second approach takes advantage of the temporal dependence of the input sequence of radar images. In this paper, we propose two new architectures that combine the philosophy of both approaches through a U-net recurrent architecture where images are first encoded and downsampled through multiple convolutional layers before being fed into a convLSTM cell. Our architectures are relatively simple, easy to build and computationnaly-light and show good performance.

3. Methods

We designed two neural network architectures to tackle the challenge of rainfall nowcasting in a light and accurate way: the Dual-Direction Network (DDnet) and the Convolutional Decoder with Light Recurrence Mesh (convDLRM). Both architectures are explained in more detail in this section.

3.1. DDnet

Our DDnet has been influenced by the Motion-Content Network (MCNet) from [18], which has been originally designed for video predictions. Its architecture offers an intuitive separation of the input data that fits our problem. To the best of our knowledge, this is the first time that such an architecture is applied to the rainfall nowcasting problem. The original MCnet separates the time frames into two inputs: the motion input and the content input. The motion input is the difference between two consecutive timeframes, representing the movement of the images. This input goes through a convolutional neural network and a convLSTM. The content input is the last frame of the input data, which contains the information of the current state of the data. This input goes through another convolutional neural network. The outputs of the two networks are then concatenated and go through a combination layer and a decoder layer to output the prediction of the next time frame.

We modified the MCNet to fit our problem best. Rather than using the difference between two consecutive timeframes for the motion part of the network, we used each timeframe since the "motion" is inherently encapsulated within the time dependence within each images. This has been shown to provide better performance in our case. In addition, instead of predicting a single image at a time, we feed the last output of the decoder part into a convLSTM that predict the next timeframes. For all layers, we use the ReLU activation function with He initialization for each convolutional layers (except for the last one that uses a sigmoid activation to get the prediction between 0 and 1). In addition, we use batch-normalization on the outputs of each layer when it is a single image and layer-normalization when it is a sequence of images to accelerate the convergence of our model.

The detailed structure of the DDnet is shown in Figure 7 in the Appendix.

3.2. convDLRM

The convDLRM network follows the philosophy of the encoder-decoder based on convLSTM introduced by [13]. It is a sequence-to-sequence learning framework that encodes information from a set of several time steps, and then

makes predictions for the following set of time steps. The encoding is done in a first convLSTM and its last hidden state is fed to a second convLSTM that takes in charge the decoding part. In addition to what [13] has done, we used the last output of the encoding convLSTM as first input of the decoding convLSTM.

A major drawback of [13] architecture is its computational complexity which makes it hard to train as the number of features increase. To avoid that, we have incorporated three major modifications to their architecture. First, we have used layer-normalization after each layer, which has shown to reduce significantly the training time. Second, we have added a "downsampler" part before the encoder-decoder and an "upsampler" part after. The downsampler-upsampler uses maxpooling to decrease the size of the input before feeding it into the convLSTM cells. The max-pool layer retains only the maximal value over a 2x2 window. This approach demonstrates very nice performance for our problem and allowed us to reduce the encoder-decoder depth to one layer (instead of 3 in [13]). Since the convLSTM cells are the most computationnaly-expensive, having less recurrent layers and smaller image sizes improved consequently the training time, reducing it by a factor of 10 relative to [13]. Last, we use the SeLU activation function with Lecun initialization instead of the tanh activation function for each convolutional layers (except for the last one that uses a sigmoid activation to get the prediction between 0 and 1). The SeLU activation is significantly less costly than the tanh activation function used by [13] and avoid the gradient vanishing problem of the ReLU activation function.

The detailed structure of the convDLRM is shown in Figure 6 in the Appendix.

3.3. Inputs and outputs

The inputs of our models are M timeframes of 64x64 images that can have several channels, each corresponding to one type of features (e.g., rainfall from radar, temperature from weather model, etc.). We normalize each feature through min-max scaling to have its values ranging between 0 and 1. The outputs are the next N timeframes of 64x64 images that follow the inputs.

3.4. Overfitting

To prevent overfitting, we use Dropout with a ratio of 0.2 for the DDnet and 0.7 for the convDLRM in front of all the convolutional layers. Adding dropout allowed us to reduce significantly the overfitting while regularizing the weights did not bring significant improvements.

3.5. Last layer initialization

A major problem of our dataset is its imbalance: there are much more dry grid points than wet grid points. Because of that, a model can achieve a relatively good score by just predicting dry cells everywhere. This is particularly true in the first steps of the training stage. At the beginning, the weights are initialized according to a centered normal distribution. Because the last activation of our networks is a sigmoid, the distribution of the output initially will be a normal distribution centered in 0.5. Because of the data imbalance, the minor positive samples are less important in the initial training stage and the model will tend to "dry" too much. Our networks have shown to be very sensitive to that and the initial training period where the models are completely dry can last relatively long. To avoid that, we initialized the bias of the last layer to a value of $b = -log(99)$. This forces the initial distribution to be centered around $1/(1 + exp(-b) = 0.01$. Hence, positive samples are now all extremely incorrect in the forward propagation, thus receiving high weights during the backpropagation. This trick has shown to work very well with our architecture.

3.6. Metrics for assessment of performance

Because of the inherently imbalanced nature of rainfall where most places are dry, common metric scores such as accuracy are not very useful since one can reach a very good score by giving only zeros. In addition, we want to be able to assess both our capacity to predict whether or not it is going to rain over a grid point as well as how much it is going to rain. To do that, we construct a new metric that we called R1 (for Rain-score) and defined as:

$$R1 = \frac{correlation + precision + recall}{3} \tag{1}$$

Here, $precision$ and $recall$ are defined based on a threshold value of 2.5×10^{-4}mm/5min. When the rainfall intensity is below this value, we assume it is a dry cell and when it is above, it is a wet cell. $correlation$ is the spatial correlation of the ground truth image with the predicted image (defined as the correlation of the flatten image). This metric takes into account both the location of the rain and its intensity and complement well the others two. Our experience showed that models that perform the best visually are the ones with the highest R1 score.

4. Dataset and features

4.1. Moving MNIST

Before training our models on the Meteo France dataset, we followed [13] approach and designed the models on the Moving MNIST dataset [16]. The reason we did that is because the Moving MNIST dataset has already been preprocessed and its features are biased towards 0 similarly to the rainfall dataset of Meteo France.

The Moving MNIST dataset of handwritten numbers contains 10,000 sequences each of length 20 show-

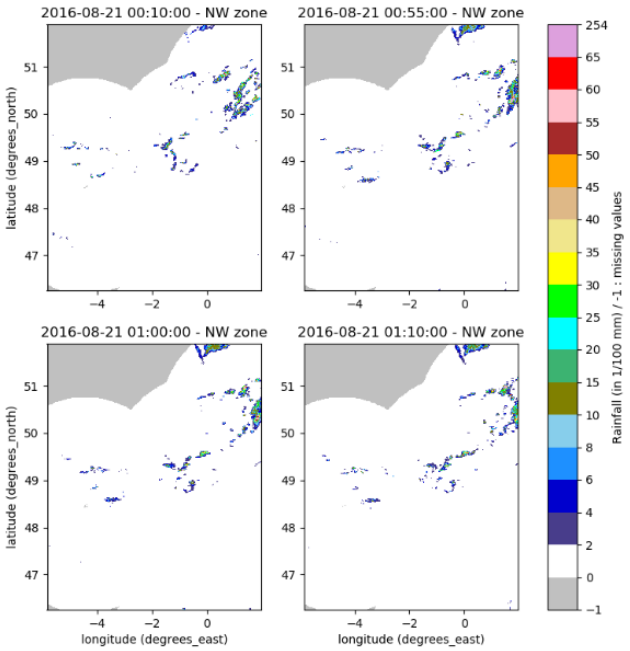

Figure 1. Four rainfall radar images coming from the Meteo France dataset.

ing 2 digits moving in a 64 x 64 frame. The digits bounce off the edges and move at a constant velocity. The dataset has been randomly generated and we downloaded it from http://www.cs.toronto.edu/ nitish/unsupervised_video/mnist_test_seq.npy. See [16] for more detail on how this dataset has been generated.

4.2. Meteo France dataset

4.2.1 Rainfall data

The MeteoNet-France dataset consists of radar images that show precipitations over a 550km x 550km region in the north-west of France. An example of such images is shown in Figure 1. The dataset spans 3 years, from 2016 to 2018, with radar images available every 5 minutes, which gives 315,600 time frames of data. Each image is a 565x784 matrix of values between 0 and 75 representing the number of 1/100mm of rain measured over the 5 minute period. In addition, the dataset is biased toward "dry" values since 93% of the available data have values between 0 and 2. Part of the radar images are corrupted because of a dysfunctionment of the instrument or because no images are available over a certain area at a certain time. In this case, we masked these values when calculating the loss to make sure that they do not influence our learning process. The data with values above 40 were floored to 40, and all the values were divided by this number, normalizing all the values between 0 and 1.

4.2.2 Additional features

In addition to the rainfall data, several other inputs are available in this dataset. The first one is the reflectivity product: the precipitation level is not directly measured from radar images, but the reflectivity product is. From those values the rainfall can be calculated using a simple equation $R = (\frac{Z}{200})^{\frac{1}{1.6}}$, where R is the rainfall rate and Z is the reflectivity product. Because of the simple proportional relationship between reflectivity and rainfall, we decided to not use this product for our analysis by worry of redundancy. Another product available that we did not use consists in the quality of the radar image taken. While this quantity can make the network understand which measurements are reliable or not, we have no way to test its performance since the original rainfall data are affected by this quality metric. In addition, two masks are shared with the dataset: the first one is a land/sea mask and the other one is the surface elevation. Finally, outputs from two numerical weather models (the AROME and ARPEGE models) are included: they give the hourly forecast for the temperature, dew temperature, relative humidity, wind speed, wind components, mean-sea level pressure, and total precipitation. Those inputs can be used either as a baseline to compare against our results, or as additional features in our models. The size of these features was changed using bilinear extrapolation in order to fit the size of the rainfall data.

4.2.3 Data processing

As mentioned above, the rainfall data were biased towards dry values. In order to deal with this issue and with the size of the images that was too big to run reasonable trainings, we followed the following procedure: after resizing the initial images to 192x256, and taking the N input timeframes, we calculated the total rainfall over subregions of size 64x64 through the N timesteps and kept only the regions that had an average precipitation superior to a chosen threshold. This technique has the advantage of both reducing the size of the input and having more "wet" samples. Another solution that we tried was to resize all the images to 64x64 but the loss of resolution made many precipitations regions to be resolved by a few gridpoints only, making it hard to predict. Finally, for computational purposes, we trained the data over the months of May, June, July and August 2017 only and tested them on the month of March 2018. The training months were chosen because they showed many days with a lot of rain, and the tested month was chosen because it had both rainy days and dry days to check if our model is able to predict correctly those extreme types of precipitation. We are aware of the slight bias that this training/testing procedure can induce in our model.

Model	Cor	Rec	Prec	R1
Shi et al. [13]	0.721	0.773	0.643	0.712
DDnet	**0.837**	0.790	0.819	0.815
ConvDLRM	0.836	**0.793**	**0.827**	**0.818**

Table 1. Selected scores on the Moving MNIST Data for testing set: correlation, precision, recall and R1 for 3 different models ([13]), DDnet and ConvDLRM.

5. Results

5.1. Moving MNIST

As stated above, we designed our architectures by testing them first on the Moving MNIST dataset, following the approach of many video-prediction papers [13, 19, 5]. We split the dataset between 9,000 training images, 900 validation images and 100 test images. We trained the models for a maximum of 100 epochs, using early stopping on the validation loss. We used the Adam optimizer with a learning rate of 0.0001 and a batch size of 15. These parameters have been chosen based on a random grid search procedure that we performed beforehand.

For our regression problem, we first tried the Mean Squared Error (MSE) and the Mean Absolute Error (MAE). Interestingly, the logcosh loss performed better than both of them on the Moving Mnist dataset. $log(cosh(x))$ is approximately equal to $x^2/2$ for small x and to $abs(x) - log(2)$ for large x. Therefore, logcosh works mostly like the MSE for small values and MAE for large values, taking the best of the two worlds.

An example of result on the test set is shown in Figure 2. While our objective for this first step was to design and understand the behavior of our two models, we were happy to see that we clearly outperformed [13] models and reached similar performance to the state of the art models as [19] and [5], specially with the convDLRM model. In addition, the convDLRM training time is an order of magnitude less than [13], showing how our architecture reaches performance and efficiency at the same time.

In order to provide a benchmark for future work, the scores of our model are shown in Table 1. As stated above, the R1 score is the mean of the three other scores and gives a nice unique metric to assess the model performance. For information, we provide also the scores of the architecture in [13] that has influenced the design of the convDLRM. We can see how much improvement we have done on this architecture.

As shown in Figure 3, the training has stopped for both model just before 50 epochs, time where the validation loss starts to increase. In this figure, we can also see that the training behavior of both models is similar even if they give visually different results.

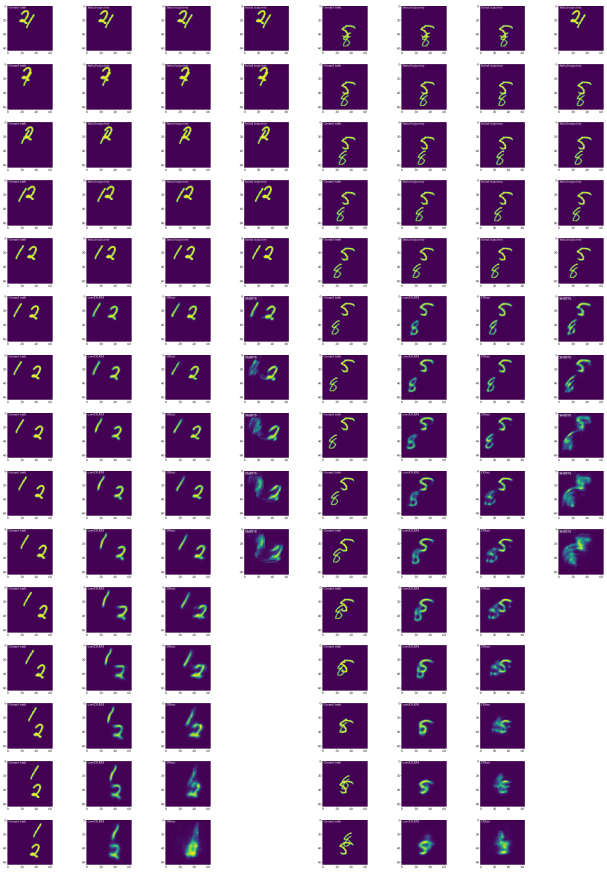

Figure 2. From left to right: Ground truth images, predictions from the convDLRM, DDnet and from Shi et al. [13]. The 4 first rows correspond to the end of the initial trajectory. We show here two examples: the one on the left is "simple" (i.e. no crossing and no digits with loops), while the one on the right is more complicated to predict.

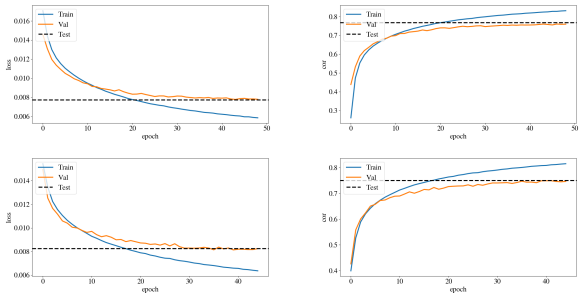

Figure 3. Loss function (left) and spatial correlation (right) for the convDLRM (top) and DDnet (bottom).

5.2. Meteo France dataset

Once the architectures were optimized on the MovingMNIST dataset, we applied the models on the rainfall dataset

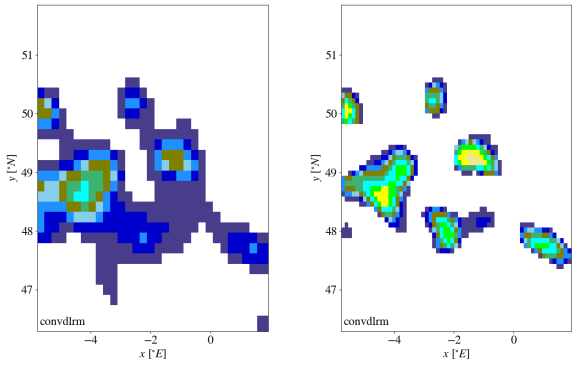

Figure 4. An example of a prediction performed with the convDLRM using the logcosh loss (left) and the MAE loss (right) on radar images.

of MeteoNet. As stated above, we worked on a dataset of 1,912 data points of 64x64 images across 12 timesteps, corresponding to one hour of data. These images were chosen to surpass a certain amount of rain. We then predicted 12 timesteps. Starting with the DDnet and the ConvDLRM, we fine-tuned some hyperparameters specifically for this new problem. For example for both architectures, a SeLU activation function showed better results than a ReLU and the Mean Absolute Error (MAE) showed better performance than the previously used logcosh loss. This observation may come from the fact that the logcosh loss do not penalize small errors as much as the MAE does but does penalize the large errors more. Therefore the logcosh tended to give less extreme values in its predictions but more wet cells (Figure 4), whereas the MAE loss was more inclined to extreme values and has a more realistic number of wet cells (Figure 4), which gave better performance.

Once the architectures were in their final forms, we added the different features mentioned above as channels. In the case of the convDLRM, the dew temperature and the pressure improved significantly the R1 score. On the other hand, for the DDnet, only the pressure improved the model. The final results are shown in Table 2 and the images predictions are shown in Figure . We did not include Shi et al. [13] results because the values are hard to compare. Indeed, the resolution, the dataset, the duration between time frames are different, making a comparison irrelevant here.

From this table we can see that the model with the best R1 score is the DDnet with the pressure included. However, all the models give results that are relatively close to one another. It can also be noted that the ConvDLRM trains around twice faster than the DDnet, which can be a considerable advantage if we want to train on more data.

Figure 5 shows an example of predictions made with the different models. As we can see, the predictions get worst with time in terms of amplitude but the wet grid cells are nicely predicted by both networks.

Model	Cor	Rec	Prec	R1
DDnet	0.630	0.731	0.792	0.717
ConvDLRM	0.608	0.753	0.755	0.705
DDnet-Pressure	**0.636**	0.724	**0.810**	**0.723**
ConvDLRM-Dew-Pres	0.630	**0.765**	0.762	0.719

Table 2. Selected scores on the MeteoNet Rainfall Data for testing set. Correlation, precision, recall and R1 for 4 different models: DDnet with no features and with pressure as feature and ConvDLRM with no features and with dew temperature and pressure as features.

6. Conclusion

In this paper, we presented two new models designed to perform short-term precipitation forecasts. Our models are light and flexible, which allow them to incorporate more features than rainfall radar images only. We have shown that taking more features into consideration allows us to improve the forecast, suggesting that our models learn some physical relationship between different weather variables.

For the purpose of this project, we have trained our network only a few sample of the data available but we already managed to reach similar performance to state-of-the-art methods (and even outperform some in the Moving Mnist case). To go further, we plan to continue to fine-tune our model on the Meteo France dataset and find the combination of features that gives the best performance.

Contributions and Acknowledgments

Both authors provided equal investment for this project and did equal work on the theoretical thinking of the project, on the gathering and processing of the data, on the design and implementation of the neural networks architecture, on the set-up of the computational resource used, on the training and testing the models and on the writing of the report.

We wish to thank Meteo France for making their data freely available. We would also like to thank the Stanford Research Computing Center for operating the Sherlock High-Performance Computing cluster that we have used for running some of our tests. Finally, we would like to thank the CS231n teaching team for an amazing class.

References

[1] Shreya Agrawal, Luke Barrington, Carla Bromberg, Jason Hickey, and John Burge. Machine Learning for Precipitation Nowcasting from Radar Images. (NeurIPS):1–6, 2019.

[2] G Ayzel, M Heistermann, A Sorokin, O Nikitin, O Lukyanova, A Sorokin, O Nikitin, and O Lukyanova. Sci-

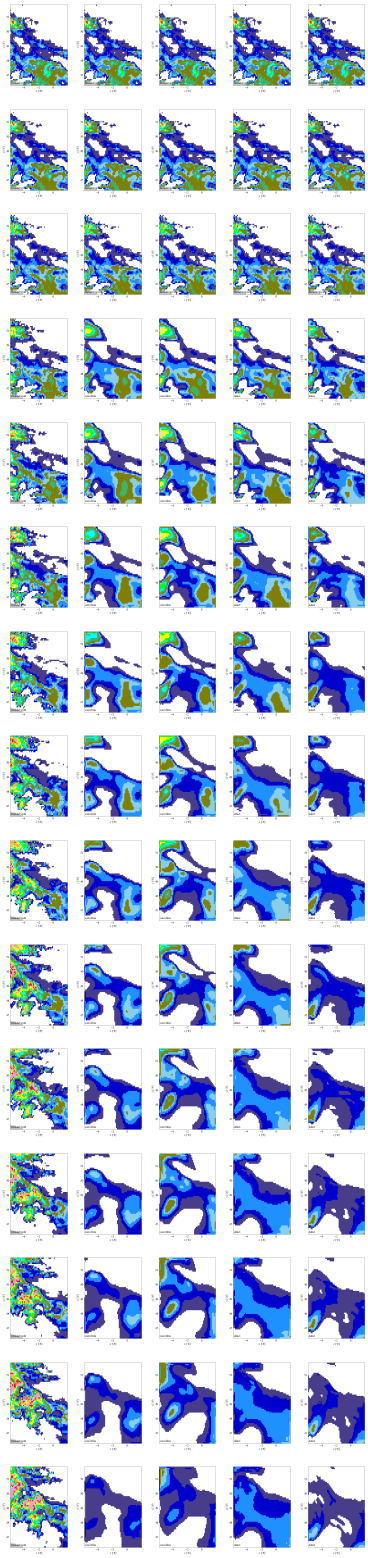

Figure 5. From left to right: Ground truth images, predictions from the convDLRM, convDLRM-Dew-Pressure, DDnet and DDnet-Pressure. The 3 first rows correspond to the end of the initial trajectory.

enceDirect All convolutional neural networks for radar-based precipitation All convolutional neural networks for radar-based precipitation nowcasting nowcasting. *Procedia Comput. Sci.*, 150:186–192, 2019.

[3] Georgy Ayzel, Tobias Scheffer, and Maik Heistermann. RainNet v1 . 0 : a convolutional neural network for radar-based precipitation nowcasting. *Geosci. Model Dev.*, (March):1–20, 2020.

[4] Neill E H Bowler, Clive E Pierce, and Alan Seed. Development of a precipitation nowcasting algorithm based upon optical flow techniques. *J. Hydrol.*, 288:74–91, 2004.

[5] Bert De Brabandere, Xu Jia, Luc Van Gool, and Tinne Tuytelaars. Dynamic Filter Networks. *NeurIPS*, pages 1–14.

[6] P Cheung, H Y Yeung, P Cheung, and H Y Yeung. Reprint 1025 Application of optical-flow technique to significant convection nowcast for terminal areas in Hong Kong The 3rd WMO International Symposium on Nowcasting and Very Short-Range Forecasting (WSN12), Rio de Janeiro , Brazil 6-10 August 2012. *3rd WMO Int. Symp. Nowcasting Very Short-Range Forecast.*, (August), 2012.

[7] Urs Germann and Isztar Zawadzki. Scale-Dependence of the Predictability of Precipitation from Continental Radar Images. Part I: Description of the Methodology. *Mon. Weather Rev.*, 130(4):2859–2873, 2002.

[8] Vincent Chabot Brice Le Pape Bruno Pradel Lior Perez Gwennaëlle Larvor, Léa Berthomier. MeteoNet, an open reference weather dataset by Meteo-France. 2020.

[9] Alexander Heye, Karthik Venkatesan, and Jericho Cain. Precipitation Nowcasting : Leveraging Deep Recurrent Convolutional Neural Networks. 2017.

[10] Vadim Lebedev, Vladimir Ivashkin, Irina Rudenko, Alexander Ganshin, Alexander Molchanov, Sergey Ovcharenko, Ruslan Grokhovetskiy, Ivan Bushmarinov, and Dmitry Solomentsev. Precipitation Nowcasting with Satellite Imagery. 2019.

[11] Hidetomo Sakaino. Spatio-Temporal Image Pattern Prediction Method Based on a Physical Model With Time-Varying Optical Flow. 51(5):3023–3036, 2013.

[12] Ryoma Sato, Hisashi Kashima, and Takehiro Yamamoto. *Short-Term Precipitation Prediction with Skip-Connected PredNet*. Springer International Publishing, 2018.

[13] Xingjian Shi, Zhourong Chen, and Hao Wang. Convolutional LSTM Network : A Machine Learning Approach for Precipitation Nowcasting. *NeurIPS*, pages 1–9, 2015.

[14] Xingjian Shi, Zhihan Gao, Leonard Lausen, Hao Wang, and Dit-yan Yeung. Deep Learning for Precipitation Nowcasting : A Benchmark and A New Model. (Nips):1–11, 2017.

[15] Casper Kaae Sønderby, Jonathan Heek, Avital Oliver, and Jason Hickey. MetNet: A Neural Weather Model for Precipitation Forecasting. pages 1–17, 2020.

[16] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015.

[17] Quang-Khai Tran and Sa-kwang Song. atmosphere Computer Vision in Precipitation Nowcasting : Applying Image Quality Assessment Metrics for Training Deep Neural Networks. *MDPI*, pages 1–20, 2019.

[18] Ruben Villegas, Jimei Yang, Seunghoon Hong, Xunyu Lin, and Honglak Lee. DECOMPOSING MOTION AND CONTENT FOR NATURAL VIDEO SEQUENCE PREDICTION. *ICLR*, pages 1–22, 2017.

[19] Yunbo Wang, Jianmin Wang, and Philip S Yu. PredRNN : Recurrent Neural Networks for Predictive Learning using Spatiotemporal LSTMs. (Nips), 2017.

[20] Wang-chun Woo and Wai-kin Wong. Operational Application of Optical Flow Techniques to Radar-Based Rainfall Nowcasting. 2017.

Appendix

Models architecture

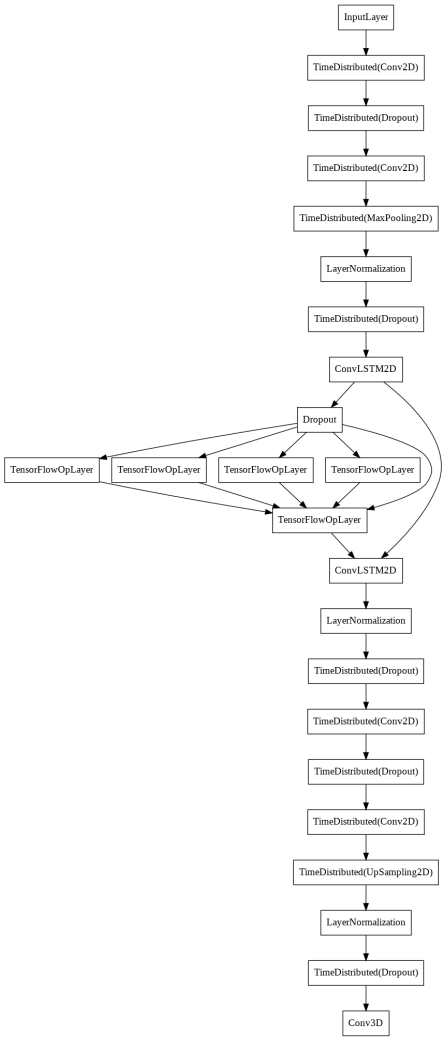

Figure 6. convDLRM

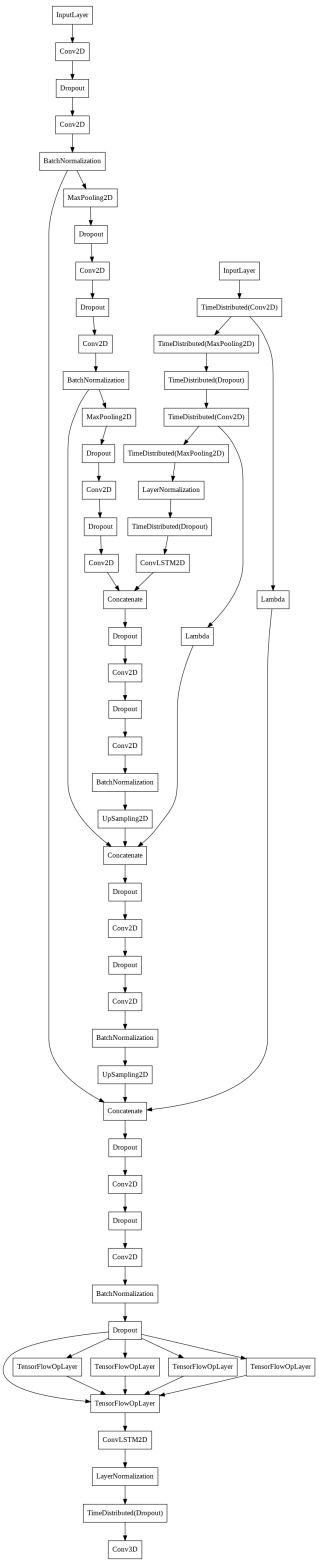

Figure 7. DDnet