Frame-like prediction for weather forecasting: novel approach using self-supervised deep learning models.

Daniel Hernández Portugues, Lara García Martinez, Victor Machado Perez, Jesus Peña and Emigdio Chavez-Angel

Abstract

Great advances in supervised and unsupervised (or self-supervised) deep learning (DL) models have encouraged the use of these protocols in a wide scientific and non-scientific research fields. Furthermore, a specific attention has been paid to unsupervised learning due to the chance of working without the use of tagged data. In this sense, this approach has attracted even further the scientific attention due to the possibility of having unlimited data. For the case of computer vision, a number of algorithm has been created using models that can learn the representation of abstract features of images and videos without the use of tagged data. DL models such as: generative adversarial networks (GANs), autoencoders, frame interpolation and prediction to name a few are particular examples of development tagged-free approaches.

In this work, we present a set of unsupervised models for weather forecasting. Inspired by the frame prediction algorithms from video data analysis, we have proposed three different models based on: (*i*) two-dimensional convolutional layers with long-short-term-memory (ConvLSTM2D) and a three-dimensional convolutional layer (Conv3D), (*ii*) Conv3D with encoder-decoder architecture and (*iii*) Conv3D with U-net like architecture.

For training and validation, we have used a set global images containing information about clouds distribution, pressure fields and temperature distribution. This dataset was concatenated by time, in a similar manner than a frames in a videos. Then the first n frames were used as inputs while the $n + 1$, $n + 2$ and so on were used as outputs for each model. The different proposed models were analysed and modified during the training process. Finally, the models were compared between them to the best performance of each architecture.

Keywords: Weather forecasting, Frame prediction, U-net, ConvLSTM2D, Conv3D

Introduction

The atmospheric science and its subtopic weather forecasting focuses their efforts mainly in to create numerical models based on dynamics of fluids and thermodynamics to predict the atmospheric state at a given location and time. Weather forecast are carried out by using a large collection of experimental observations (e.g., temperature, pressure) applied to a physical model. However, the physical laws that govern the dynamics of the atmosphere are based in partial differential equations that include many non-linear terms. Therefore, these simulations are chaotic in nature very susceptible to boundary and initial state conditions. Furthermore, the lack of understanding of all physical process behind, makes the weather forecasting in a very challenging task. Ferranti et al.[1] showed that the weather predictability lays in a range of 3-14 days.

In general, the experimental observations includes: atmospheric pressure, local temperature, wind velocity, humidity, precipitation, etc., which are recollected as a function of time in a number of meteorological stations, weather satellites and xxx. [ref.] Once all the experimental data is recollected and treated, a complicated set equation (known as primitive equations) are solved numerically to predict a future time step of the atmosphere. This procedure is known as numerical weather predict (NWP) models. These models have been used for several years, with an increment in their complexity as the computational power has allowed.

In this context, the irruption of the artificial neural network (ANN) in meteorology and climate has have a remarkable impact. For example, Hewitson and Crane[2] found empirical relationship between meteorological variables, Pasini et al.[3] applied to climate attribution studies and Hsieh et al.[4] and Tangang et al.[5] applied successfully to "El Niño" forecasting. In parallel, the use of ANN together with support vector machines have been applied to downscale complicated global climate models.[6] Other application includes and are not limited to: cloud classification, tornado forecasting, forecasting uncertainty to name a few.[7]

In this work, we present three closely related methods based on two-dimensional convolutional layers with long-short-term-memory (ConvLSTM2D) and a three-dimensional convolutional layer (Conv3D) to link clouds, temperature and pressure fields for weather forecasting. The different models are trained with a set of global images-like dataset ordered to mimic the frames in a video. Then, inspired by the frame prediction algorithms, we attempt to predict the next frame.

Data

For the training, validation and testing ERA5 dataset have been used.[8] It consist in a set of meteorological data taken from 2000 to 2010. Depending of the dataset, it can consist in image-

[1] L. Ferranti, et al., Quarterly Journal of the Royal Meteorological Society, **141** (2015), 916–924.
[2] B.C. Hewitson and R.G. Crane. Geophysical Research Letters, 19 (1992), 1835–1838.
[3] A. Pasini, et al., Scientific Reports, **7** (2017), 17681.
[4] W.W. Hsieh et al., Bulletin of the American Meteorological Society, **79** (1998), 1855–1870.
[5] F.T Tangang et al. Journal of Climate, **11** (1998), 29–41.
[6] Y.B. Dibike, et al. Neural Networks, **19** (2016), 135–144.
[7] G. Shrivastava et al., International Journal of Computer Applications **51** (2012), 17-29
[8] ERA5 dataset available at: https://www.ecmwf.int/en/forecasts/datasets/reanalysis-datasets/era5 (accessed on 01 July 2019)

like taken every one or six hours. The data cover a number of atmospheric, land and oceanic variables. The dataset covers the planet on a 30x30 km^2 and resolve the atmosphere using 137 levels from the surface up to a height of 80 km. **Figure 1** shows an image example taken from this dataset.

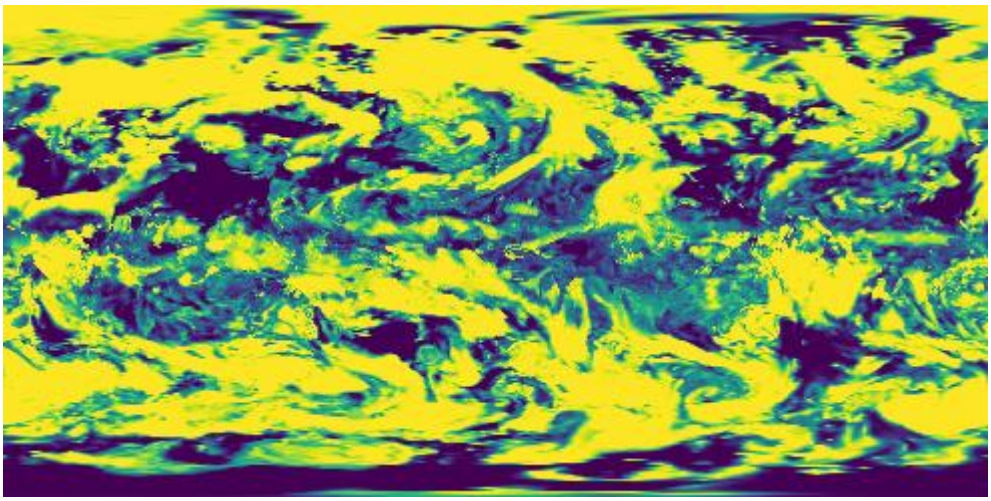

Figure 1 Global cloud distribution

Model 1: Two-dimensional convolutional layers with long-short-term-memory (ConvLSTM2D)

In order to predict a frame in a sequence of images two type of intelligence are needed. The first one is in charge of interpreting the image, while the second part is dedicated to understand the sequence of frames. To solve this task we have used a model inspired by the work of Finn et al.[9], it consists in a stack of ConvLSTM2D follow by batch normalization layers. The ConvLSTM2D is similar to the classic LSTM except that the input and recurrent transformations are both two-dimensional convolutional transformations. These layers allow to the model interpret the visual information (e.g. cloud distribution) and at the same time understand time sequences (sequential data).

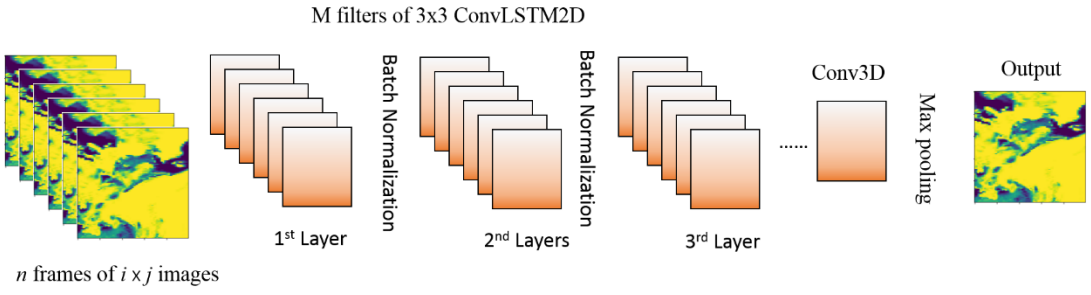

Figure 2 schematic representation of the ConvLSTM2D architecture used in section.
After each convolutional layer a batch normalization layer was added. This layers were used to change the inputs for the next ConvLSTM2D layers. The introduction of this normalization

[9] C Finn et al. 30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain. Available at: https://arxiv.org/pdf/1605.07157.pdf

can boost the speed of the training part and let each separated layer learn more independently of the others. Finally for the last layer a three-dimensional convolution layer was added (Conv3D). The main idea of this layer was to predict more than one frame, however, due to it low performance during the training process, we have tried several other configuration based on three-dimensional maxpooling, ConvLSTM2D, Conv2D, 2D maxpooling or 2D average pooling layers.

Results

The following calculations were carried out by using 10-years (10Y) and 1 year (1Y) global cloud dataset. 10Y dataset was generated by taking global cloud images (satellite images) every six hours, i.e., 4 images per day, 1460 images per year and a total of 14600 images. While the 1Y dataset was generated by taking image every hour, i.e., 24 images per day with a total of 8760 images. Both datasets were separated in 80% for training and 5 % validation, the rest of images were reserved for testing purposes.
Finally, the images were cropped to 40x40 pixels, representing a small area in the middle of Atlantic Ocean. This subset data was used to test the different architecture configurations.

Approach 1: n-ConvLSTM2D + n-batch normalization +1-Conv3D

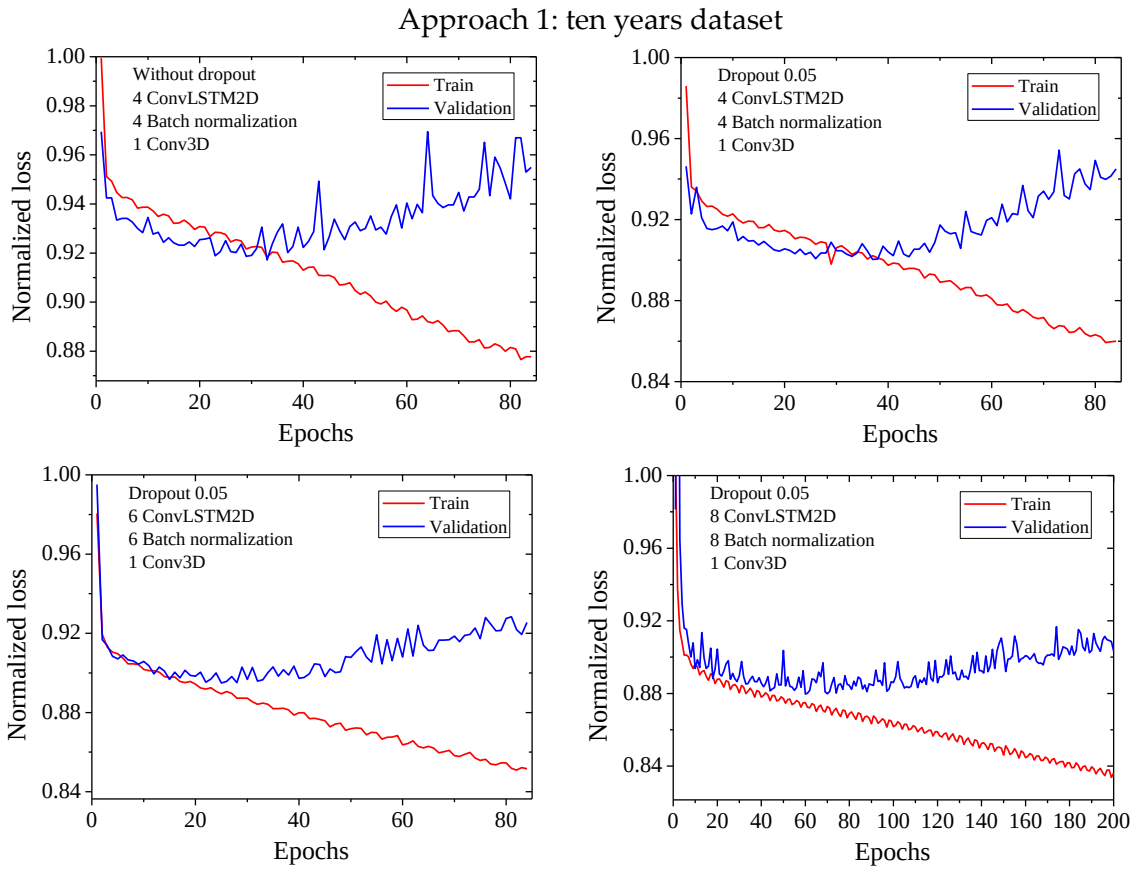

Figure 3 Training and validation curve loss as function of number of epochs for the models described in Figure 2 with different amount of ConvLSMT2D layers and small dropout in each layer.

The first approached used here consisted in a concatenation of ConvLSTM2D layers con 40 filters each followed by batch normalization layers. The last layer used was a Conv3D, to predict more than one frame for the output. As it is displayed in the **Figure 3**, a clear overfitting curve is obtained using this architecture.

Approach 2: n-ConvLSTM2D + n-batch normalization +1-Conv3D + 3D Maxpooling

This model is very similar than the previous one, but here we have used a 3D maxpooling. The introduction of 3D maxpooling was thought to eliminate the time dependence and create a single image from the array of frames coming from Conv3D layer. **Figure 4** and **Figure 5** and shows a similar trend of overfitting for 10Y and 1Y dataset. In this architecture the amount of filters was fixed to 40.

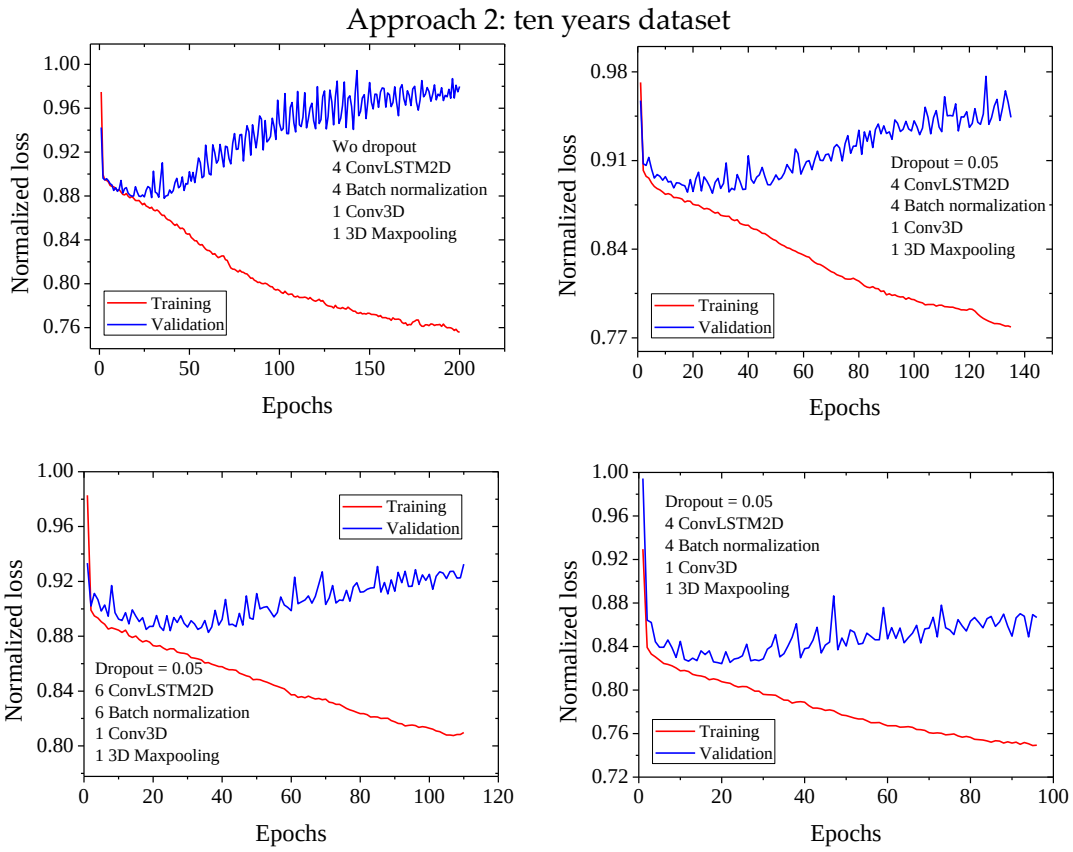

Figure 4 Training and validation curve for 10 years dataset using a 3D maxpooling at the end of the architecture

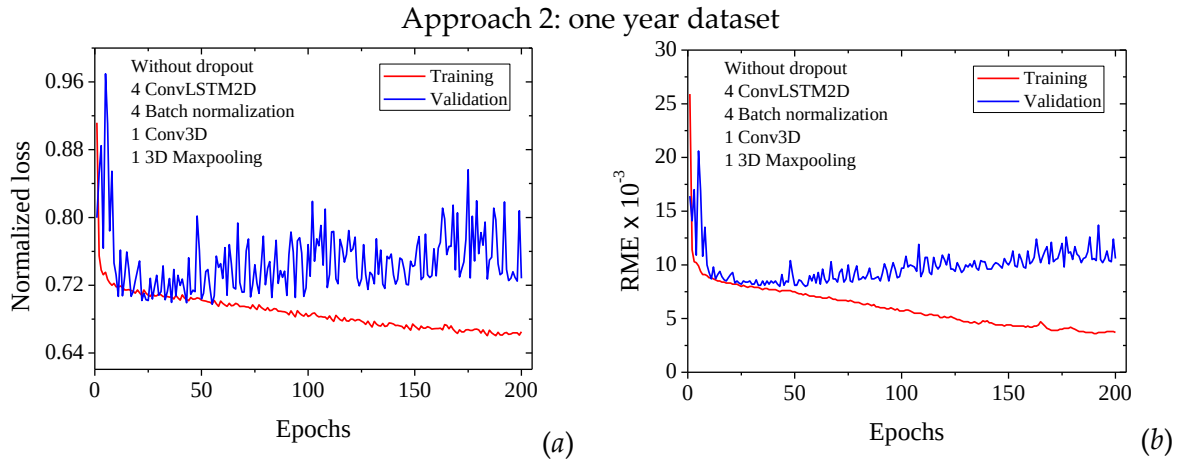

Figure 5 Training and validation loss (a) and mean square error (b) curves as function of number of epochs.

Approach 3: 5-ConvLSTM2D + 4-batch normalization+ 2DMaxpooling

In this architecture we have exchanged the 3DConv layers by 2D maxpooling layer at the end of the architecture. As is displayed in **Figure 6** and **Figure 7**, the training curves for this model enhance significantly the results. It seems that the overfitting observed previously have disappeared. However, many spikes can be observed in training and validations curves. These spikes are typically associated to LSTM layers, and/or when the dataset is indivisible by batchsize. Despite of that, this architecture seems to work well for this 10-year (**Figure 6**) and 1-year (**Figure 7**) dataset.

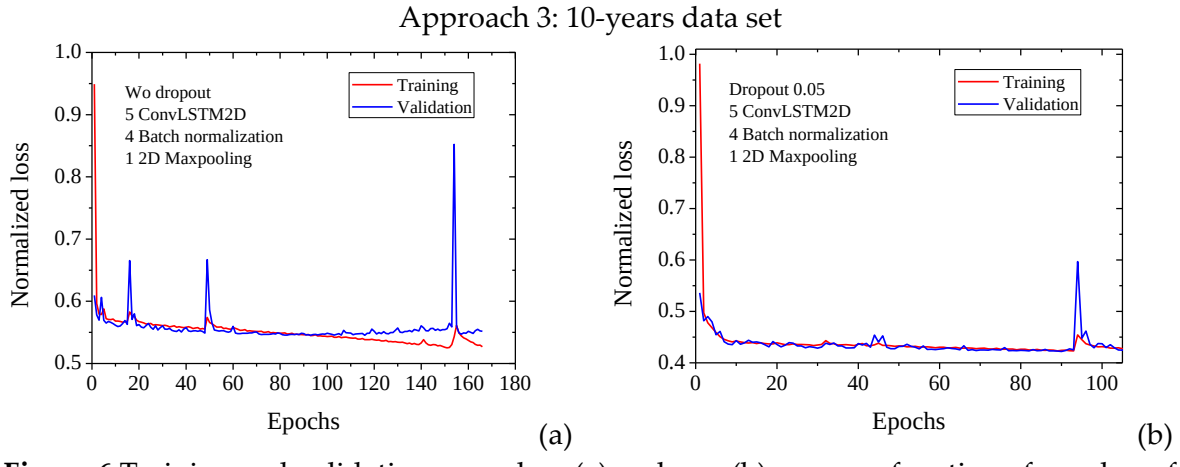

Figure 6 Training and validation curve loss (a) and rms (b) curve as function of number of epochs.

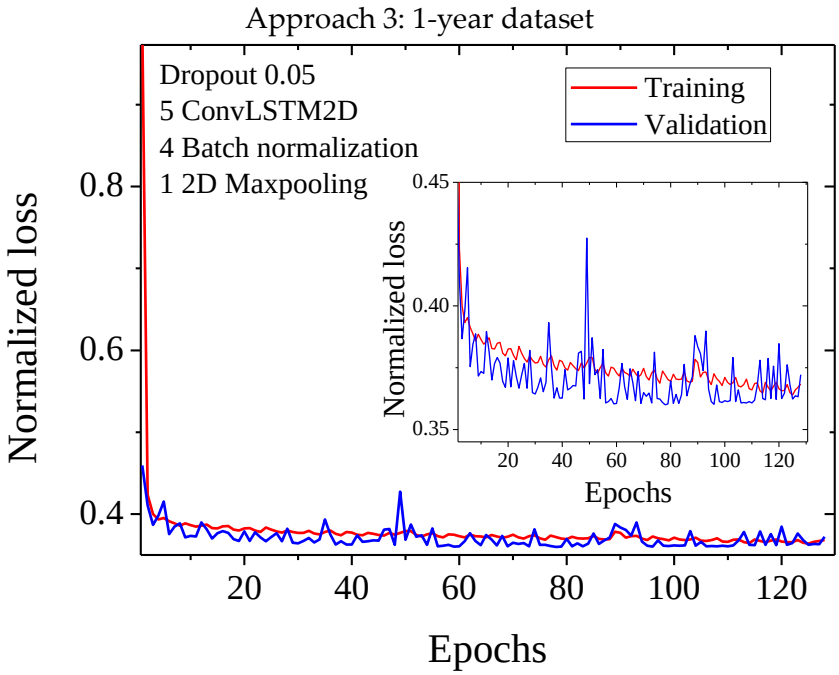

Figure 7 Training and validation curve as function of number of epochs. Inset: small zoom out to show the convergence of the model.

Approach 4: 4-ConvLSTM2D + 4-batch normalization+ 2-Conv2D and 2 2D-Average pooling

This was the last model used in this section, the model was similar to the previous one but with small change in the last layers. Here, we have used an average pooling after the last ConvLSTM2D layer followed by a Conv2D and finalizing with other 2D-average pooling. The number of filters were changed in an increasing-decreasing manner, i.e., 4-32-8-1.

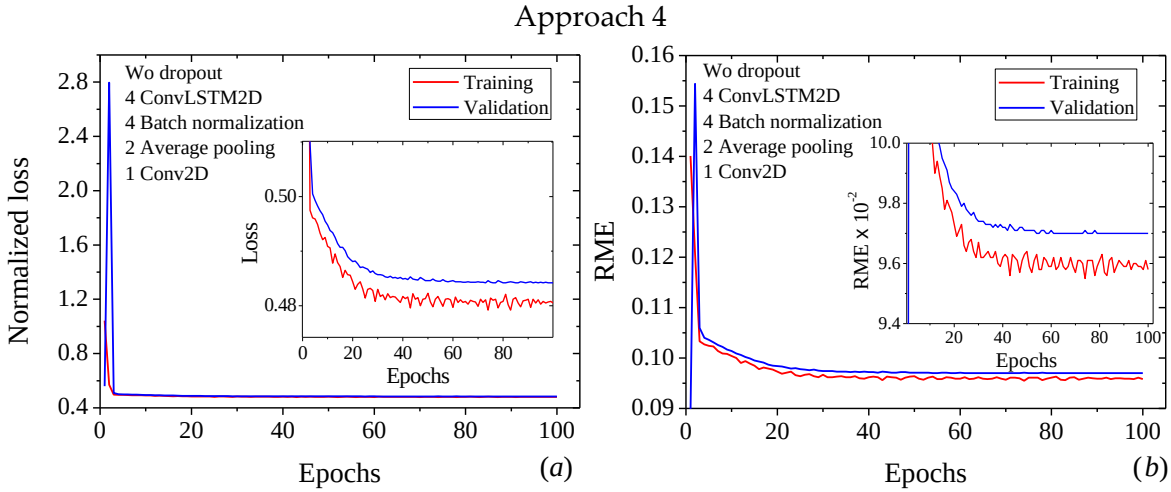

Figure 8 Training and validation loss (a) and mean square error (b) curves as function of number of epochs.

Model 2: Conv2D Encoder + LSTM + Conv2D Decoder

Following the same idea as the previous Encoder-Decoder model, we have tested an architecture Encoder – Decoder adding an LSTM layer between in order to make profit of the temporal information.

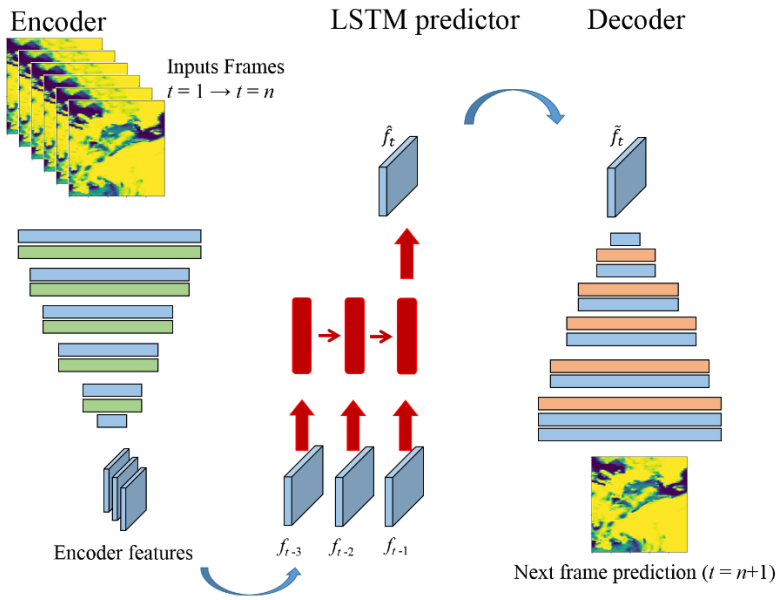

Figure 9 Schematic representation of the Conv2D Encoder + LSTM + Conv2D Decoder architecture

Model structure

Our network architecture is as follows, previous layers to LSTM are placed within a TimeDistributed layer in order to conserve temporal dimensionality and information:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed_1 (TimeDist (None, None, 128, 128, 4) 40
_________________________________________________________________
batch_normalization_1 (Batch (None, None, 128, 128, 4) 16
_________________________________________________________________
time_distributed_2 (TimeDist (None, None, 64, 64, 4)   0
_________________________________________________________________
time_distributed_3 (TimeDist (None, None, 64, 64, 8)   296
_________________________________________________________________
batch_normalization_2 (Batch (None, None, 64, 64, 8)   32
_________________________________________________________________
time_distributed_4 (TimeDist (None, None, 32, 32, 8)   0
_________________________________________________________________
time_distributed_5 (TimeDist (None, None, 32, 32, 16)  1168
_________________________________________________________________
batch_normalization_3 (Batch (None, None, 32, 32, 16)  64
_________________________________________________________________
time_distributed_6 (TimeDist (None, None, 16384)       0
_________________________________________________________________
lstm_1 (LSTM)                (None, 2048)              151003136
_________________________________________________________________
batch_normalization_4 (Batch (None, 2048)              8192
_________________________________________________________________
reshape_1 (Reshape)          (None, 8, 8, 32)          0
_________________________________________________________________
```

```
conv2d_transpose_1 (Conv2DTr (None, 8, 8, 16)        4624
_________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 16)        64
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 16, 16, 16)      0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 16, 16, 8)       1160
_________________________________________________________________
batch_normalization_6 (Batch (None, 16, 16, 8)       32
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 32, 32, 8)          0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 32, 32, 4)       292
_________________________________________________________________
batch_normalization_7 (Batch (None, 32, 32, 4)       16
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 64, 64, 4)       0
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 64, 64, 2)       74
_________________________________________________________________
batch_normalization_8 (Batch (None, 64, 64, 2)       8
_________________________________________________________________
up_sampling2d_4 (UpSampling2 (None, 128, 128, 2)     0
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 128, 128, 1)     19
_________________________________________________________________
batch_normalization_9 (Batch (None, 128, 128, 1)     4
=================================================================
Total params: 151,019,237
Trainable params: 151,015,023
Non-trainable params: 4,214
```

Input Format

Our weather original images are of size 256x512x1, only one channel (cloud distribution) is being used, although we are using a 128x128x1 crop, and we are using 4 consecutive / time ordered images. So, our input for this architecture is: [batch_size, 4, 128, 128, 1] where 4 is the number of frames per sequence.

Output Format

Our network returns one single image of size 128x128 with one channel, representing the next time frame.

Optimizations

- Batch normalization à Performance improvement.

Results

A summary of the results using this architecture is displayed in **Figure 10** and **Figure 11**. The only difference between these architecture is the use of batch normalization layers.

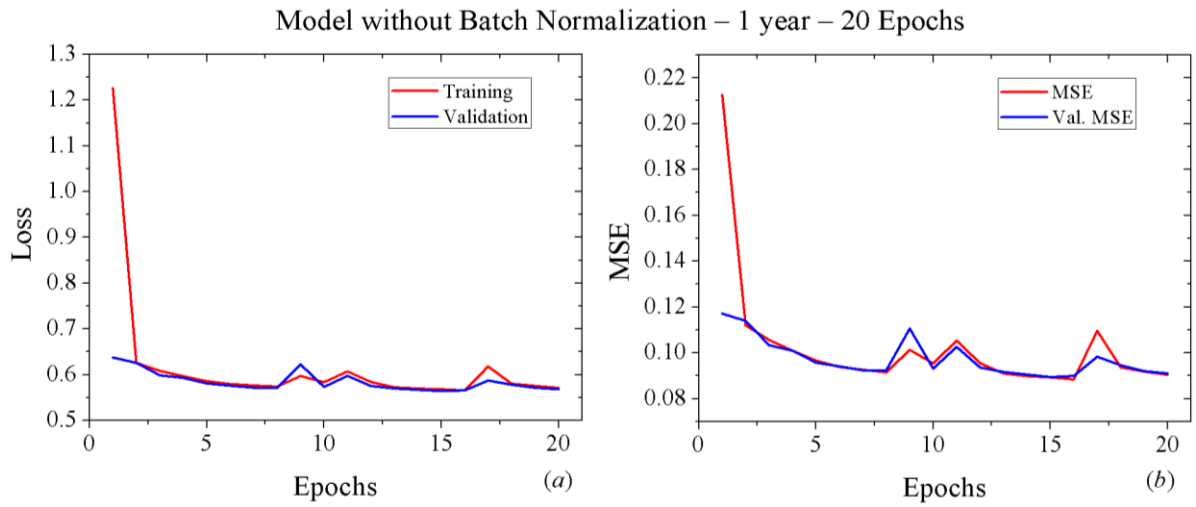

Figure 10 (a)Training and validation and (b) mean square error curves as function of number of epochs. Both images were estimated by using the architecture described above without a batch normalization layer

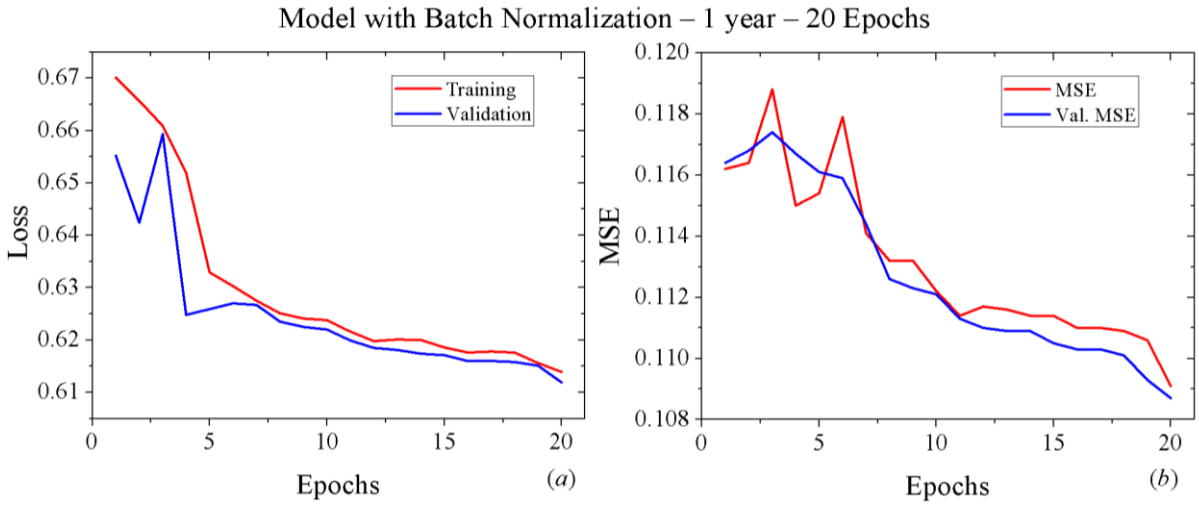

Figure 11 (a) Training and validation and (b) mean square error curves as function of number of epochs. In this case,

Model 3: Conv2D U-Net

Following the same idea as the previous Encoder-Decoder model, we have tested an architecture based on U-Net[10] to produce a more detailed result thanks to the skip connections at different layers of the network. This way we recover early features at late stages of the decoder. **Figure 4** shows a schematic representation of the original U-Net network

[10] O. Ronneberger et al. In: Navab N., Hornegger J., Wells W., Frangi A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI 2015. Lecture Notes in Computer Science, vol 9351. Springer, Cham

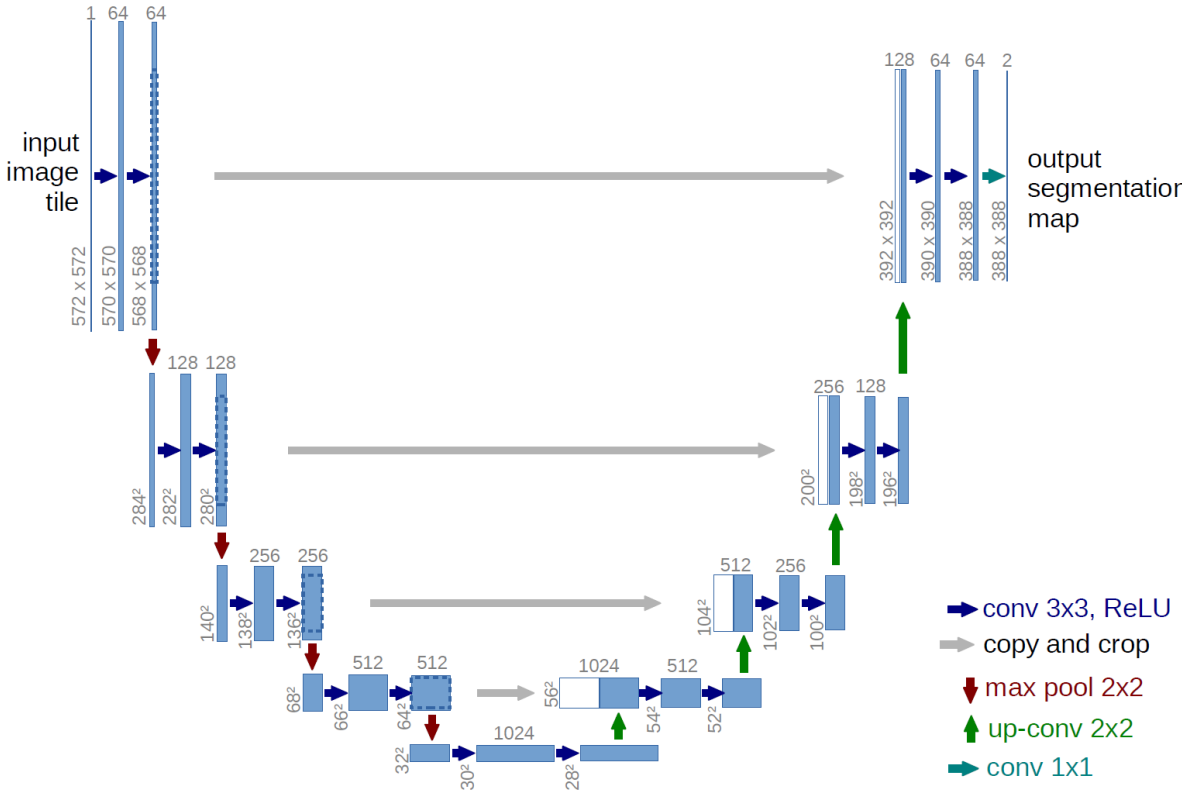

Figure 12 Schematic representation of the U-net Adapted from reference [10]

Our network does not preserve the same layers and feature sizes as the original posted above, this is the current architecture:

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 128, 128, 10)	0	
conv2d_20 (Conv2D)	(None, 128, 128, 32)	2912	input_2[0][0]
batch_normalization_1 (BatchNor	(None, 128, 128, 32)	128	conv2d_20[0][0]
conv2d_21 (Conv2D)	(None, 128, 128, 32)	9248	batch_normalization_1[0][0]
batch_normalization_2 (BatchNor	(None, 128, 128, 32)	128	conv2d_21[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 64, 64, 32)	0	batch_normalization_2[0][0]
conv2d_22 (Conv2D)	(None, 64, 64, 64)	18496	max_pooling2d_5[0][0]
batch_normalization_3 (BatchNor	(None, 64, 64, 64)	256	conv2d_22[0][0]
conv2d_23 (Conv2D)	(None, 64, 64, 64)	36928	batch_normalization_3[0][0]
batch_normalization_4 (BatchNor	(None, 64, 64, 64)	256	conv2d_23[0][0]
max_pooling2d_6 (MaxPooling2D)	(None, 32, 32, 64)	0	batch_normalization_4[0][0]
conv2d_24 (Conv2D)	(None, 32, 32, 128)	73856	max_pooling2d_6[0][0]
batch_normalization_5 (BatchNor	(None, 32, 32, 128)	512	conv2d_24[0][0]
conv2d_25 (Conv2D)	(None, 32, 32, 128)	147584	batch_normalization_5[0][0]
batch_normalization_6 (BatchNor	(None, 32, 32, 128)	512	conv2d_25[0][0]

max_pooling2d_7 (MaxPooling2D)	(None, 16, 16, 128)	0	batch_normalization_6[0][0]
conv2d_26 (Conv2D)	(None, 16, 16, 256)	295168	max_pooling2d_7[0][0]
batch_normalization_7 (BatchNor	(None, 16, 16, 256)	1024	conv2d_26[0][0]
conv2d_27 (Conv2D)	(None, 16, 16, 256)	590080	batch_normalization_7[0][0]
batch_normalization_8 (BatchNor	(None, 16, 16, 256)	1024	conv2d_27[0][0]
max_pooling2d_8 (MaxPooling2D)	(None, 8, 8, 256)	0	batch_normalization_8[0][0]
conv2d_28 (Conv2D)	(None, 8, 8, 512)	1180160	max_pooling2d_8[0][0]
batch_normalization_9 (BatchNor	(None, 8, 8, 512)	2048	conv2d_28[0][0]
conv2d_29 (Conv2D)	(None, 8, 8, 512)	2359808	batch_normalization_9[0][0]
batch_normalization_10 (BatchNo	(None, 8, 8, 512)	2048	conv2d_29[0][0]
max_pooling2d_9 (MaxPooling2D)	(None, 4, 4, 512)	0	batch_normalization_10[0][0]
conv2d_30 (Conv2D)	(None, 4, 4, 512)	2359808	max_pooling2d_9[0][0]
batch_normalization_11 (BatchNo	(None, 4, 4, 512)	2048	conv2d_30[0][0]
conv2d_31 (Conv2D)	(None, 4, 4, 512)	2359808	batch_normalization_11[0][0]
batch_normalization_12 (BatchNo	(None, 4, 4, 512)	2048	conv2d_31[0][0]
up_sampling2d_5 (UpSampling2D)	(None, 8, 8, 512)	0	batch_normalization_12[0][0]
concatenate_5 (Concatenate)	(None, 8, 8, 1024)	0	up_sampling2d_5[0][0] batch_normalization_10[0][0]
conv2d_32 (Conv2D)	(None, 8, 8, 512)	4719104	concatenate_5[0][0]
batch_normalization_13 (BatchNo	(None, 8, 8, 512)	2048	conv2d_32[0][0]
conv2d_33 (Conv2D)	(None, 8, 8, 512)	2359808	batch_normalization_13[0][0]
batch_normalization_14 (BatchNo	(None, 8, 8, 512)	2048	conv2d_33[0][0]
up_sampling2d_6 (UpSampling2D)	(None, 16, 16, 512)	0	batch_normalization_14[0][0]
concatenate_6 (Concatenate)	(None, 16, 16, 768)	0	up_sampling2d_6[0][0] batch_normalization_8[0][0]
conv2d_34 (Conv2D)	(None, 16, 16, 256)	1769728	concatenate_6[0][0]
batch_normalization_15 (BatchNo	(None, 16, 16, 256)	1024	conv2d_34[0][0]
conv2d_35 (Conv2D)	(None, 16, 16, 256)	590080	batch_normalization_15[0][0]
batch_normalization_16 (BatchNo	(None, 16, 16, 256)	1024	conv2d_35[0][0]
up_sampling2d_7 (UpSampling2D)	(None, 32, 32, 256)	0	batch_normalization_16[0][0]
concatenate_7 (Concatenate)	(None, 32, 32, 384)	0	up_sampling2d_7[0][0] batch_normalization_6[0][0]
conv2d_36 (Conv2D)	(None, 32, 32, 128)	442496	concatenate_7[0][0]
batch_normalization_17 (BatchNo	(None, 32, 32, 128)	512	conv2d_36[0][0]
conv2d_37 (Conv2D)	(None, 32, 32, 128)	147584	batch_normalization_17[0][0]
batch_normalization_18 (BatchNo	(None, 32, 32, 128)	512	conv2d_37[0][0]
up_sampling2d_8 (UpSampling2D)	(None, 64, 64, 128)	0	batch_normalization_18[0][0]

```
__________________________________________________________________________________________________
concatenate_8 (Concatenate)     (None, 64, 64, 192)  0           up_sampling2d_8[0][0]
                                                                 batch_normalization_4[0][0]
__________________________________________________________________________________________________
conv2d_38 (Conv2D)              (None, 64, 64, 64)   110656      concatenate_8[0][0]
__________________________________________________________________________________________________
batch_normalization_19 (BatchNo (None, 64, 64, 64)   256         conv2d_38[0][0]
__________________________________________________________________________________________________
conv2d_39 (Conv2D)              (None, 64, 64, 64)   36928       batch_normalization_19[0][0]
__________________________________________________________________________________________________
batch_normalization_20 (BatchNo (None, 64, 64, 64)   256         conv2d_39[0][0]
__________________________________________________________________________________________________
up_sampling2d_9 (UpSampling2D)  (None, 128, 128, 64) 0           batch_normalization_20[0][0]
__________________________________________________________________________________________________
concatenate_9 (Concatenate)     (None, 128, 128, 96) 0           up_sampling2d_9[0][0]
                                                                 batch_normalization_2[0][0]
__________________________________________________________________________________________________
conv2d_40 (Conv2D)              (None, 128, 128, 32) 27680       concatenate_9[0][0]
__________________________________________________________________________________________________
batch_normalization_21 (BatchNo (None, 128, 128, 32) 128         conv2d_40[0][0]
__________________________________________________________________________________________________
conv2d_41 (Conv2D)              (None, 128, 128, 32) 9248        batch_normalization_21[0][0]
__________________________________________________________________________________________________
batch_normalization_22 (BatchNo (None, 128, 128, 32) 128         conv2d_41[0][0]
__________________________________________________________________________________________________
conv2d_42 (Conv2D)              (None, 128, 128, 1)  33          batch_normalization_22[0][0]
==================================================================================================
Total params: 19,667,169
Trainable params: 19,657,185
Non-trainable params: 9,984
```

Input Format

Our weather images are of size 256x512 and we are using 10 consecutive / time ordered images as channels.

Output Format

Our network returns one single image of size 256x512 with one channel, representing the next time frame.

Results

A summary of the results using this architecture is displayed in **Figure 11** and **Figure 12**. The only difference between these architecture is the use of batch normalization layers.

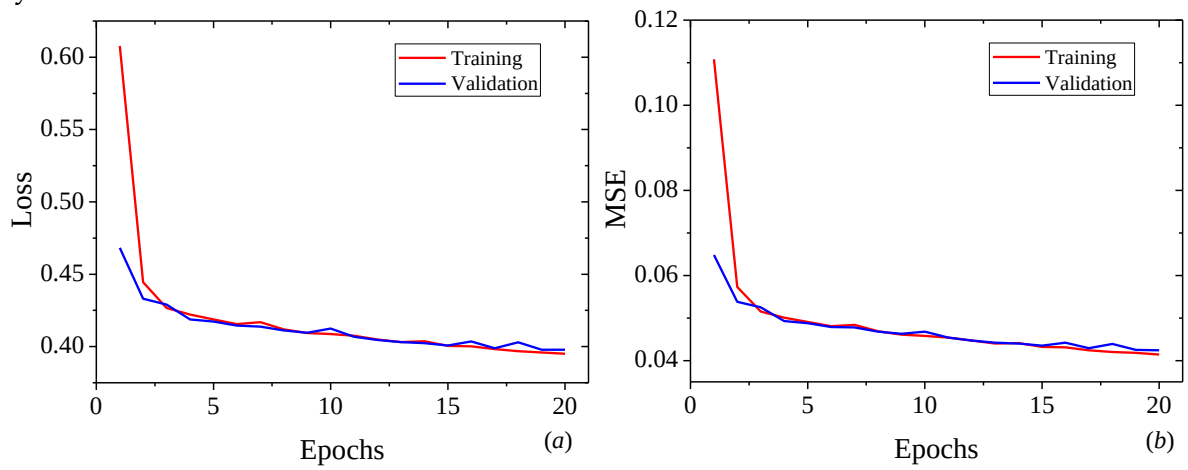

Figure 13 Training and validation loss (a) and mean square error (b) curves as function of number of epochs.