UNet with Axial Transformer : A Neural Weather Model for Precipitation Nowcasting

Sumit Mamtani
New York University
sm9669@nyu.edu

Maitreya Sonawane
New York University
mss9240@nyu.edu

Abstract

Making accurate weather predictions can be particularly challenging for localized storms or events that evolve on hourly timescales, such as thunderstorms. Hence, our goal for the project was to model Weather Nowcasting for making highly localized and accurate predictions that apply to the immediate future replacing the current numerical weather models and data assimilation systems with Deep Learning approaches. A significant advantage of machine learning is that inference is computationally cheap given an already-trained model, allowing forecasts that are nearly instantaneous and in the native high resolution of the input data. In this work we developed a novel method that employs Transformer-based machine learning models to forecast precipitation. This approach works by leveraging axial attention mechanisms to learn complex patterns and dynamics from time series frames. Moreover, it is a generic framework and can be applied to univariate and multivariate time series data, as well as time series embeddings data. This paper represents an initial research on the dataset used in the domain of next frame prediciton, and hence, we demonstrate state-of-the-art results in terms of metrices (PSNR = 47.67, SSIM = 0.9943) used for the given dataset using UNet with Axial Transformer. The code is available at **https://github.com/sumitmamtani/Precipitation-Nowcasting**

1. Introduction

Deep Neural Networks (DNN) [1, 12, 13, 17] have been successfully applied in many diverse domains such as image classification, video analysis, language modeling and translation, medical imaging, and weather. Recently, there has been increasing interest in using DNNs to generate and improve weather forecasting which is an unsupervised representation problem. In these kinds of problems, next-frame prediction is a new, promising direction of research in computer vision, predicting possible future images by presenting historical image information.

Weather forecasting is the prediction of future weather conditions such as precipitation, temperature, pressure, and wind and is fundamental to both science and society. Our particular interest is in the area of nowcasting, a term used to describe high resolution, short-term (e.g. 0 to 2 hours), weather forecasts of precipitation, or other meteorological quantities. The precipitation nowcasting field helps in the accurate prediction of rainfall over an area by looking at radar images. The field deals with the generation of the radar image at some points in the near future.

One of the various architectures researchers have tried to implement is Conv-LSTM [8, 14, 16] that has shown great results in dealing with time-series data due to them being pretty good at extracting patterns in input feature space, where the input data spans over long sequences. Given the gated architecture of LSTM's that has this ability to manipulate their memory state, they are ideal for such problems.

With the continuous input of data from one end, we know that the prediction needs to be swift while dealing with a huge amount of data. With the same thing in mind, some recent research introduced UNet to counter the same issue and were successful to get good results on image to image translation problems [2, 19].

Usage of conditional GANs (cGANs) [6] has became widely popular under the domain of image-to-image translation. As we require a model that would learn from previous inputs in the sequence, we tried to make use of this property where cGANs predicts N future radar frames given M past-conditional frames.

In this Paper, we also introduced encode-decoder architecture [7] with the Axial Transformer [4] in our Novel model, a simple yet effective self-attention-based [20, 21] autoregressive model for data organized as multidimensional tensors. Rather than applying attention to a flattened string of tensor elements, Axial Transformer instead applies attention along a single axis of the tensor without flattening, so this is referred to as "axial attention." Since the length of any single axis (that is, the height or width of an image) is typically much smaller than the total number of elements, an axial attention operation enjoys a significant saving in computation and memory over standard self-attention. We achieve state-of-the-art results by training the encode-decoder with Axial Transformer framework. Our contributions are summarized as follows:

- We implemented an encoder decoder architecture and passed the frames of videos as input to the architecture.
- The output of encoder decoder architecture is passed as input to the Axial Transformer network.
- With sequence frames given as input, the final output of the UNet with Axial Transformer model is a distribution, out of which we select the mean value for every pixel.

2. Related Work

Previously, the Conv-LSTM model has shown promising results on the next frame prediction problem on the Moving MNIST dataset [14]. The dataset was initially created in the context of Unsupervised Learning of Video Representations [15] and used LSTM to learn the representation of video sequence. As the radar images too are a kind of time series data and we know that clouds can not abruptly change direction or disappear, we know that there are some motion parameters associated with a continuous sequence of radar images too. Using the same idea of next frame prediction in a video sequence, ConvLSTM can be applied to a dataset of radar images too.

UNet is one of the recent models that has showcased its ability to predict the next frame of time-series data very well [2]. In the paper, as part of Traffic4cast challenge 2019, UNet was used to predict short-term traffic flow volume. The input and output of the model were the same sizes that are also relevant in our context as we need to reproduce radar images of the same location sometime in the future. Another paper that actually implemented UNet for the weather forecasting problem [19] saw a significant increase in results on the dataset consisting of precipitation maps from a region of the Netherlands and a binary image of cloud coverage of France. The size of the model here is very small compared to previous models that were used to solve the same problem, which is also a significant advantage considering the latency requirement of our problem statement.

Researchers have now tried an observations-driven approach for probabilistic nowcasting using deep generative models (DGMs). DGMs are statistical models that learn probability distributions of data and allow for easy generation of samples from their learned distributions [9, 11]. As generative models are fundamentally probabilistic, they have the ability to simulate many samples from the conditional distribution of future radar given historical radar.

One category of DGM model is GANs [3]. GANs learn a loss that tries to classify if the output image is real or fake, while simultaneously training a generative model to minimize this loss.Blurry images will not be tolerated since they look obviously fake. Because GANs learn a loss that adapts to the data, they can be applied to a multitude of tasks that traditionally would require very different kinds of loss functions.

3. Our Method

We next describe the neural network architectures used in the expirement of this project in the following section:

3.1. Conv-LSTM

This model was our baseline model which was inspired by the paper that implemented the same model on the original complete HKO-7 dataset, along with precipitation data in .csv and .pkl files [14]. The PyTorch implementation of the code is available on GitHub [1]. As the code mentioned worked on the Moving MNIST dataset, we had to integrate our dataset running for the given model. The model architecture was suitable to use in the context of this problem, as the original paper too works on the next frame prediction problem and learns the representation of video sequence. By extending the fully connected LSTM (FC-LSTM) to have convolutional structures in both the input-to-state and state-to-state transitions, the paper proposes the convolutional LSTM (ConvLSTM) and used it to build an end-to-end trainable model for the precipitation nowcasting problem. The sequence to sequence model acts as an encoder-decoder structure with *64* kernels each with size of *3* and padding of *1*, to ensure the input and output of the model stay of equal size. The activation function used was *ReLU* and optimizer being *Adam*.

Each of the 3 sequential layers used was ConvLSTM cell which consisted of a *Conv-2D* layer, input, output, and forget gate all having a *Sigmoid* activation in the LSTM cell. Each of the gates takes the hidden state and the current input as input, it concatenates the vectors and applies the sigmoid activation, now the input gate controls if the memory cell should be updated. Forget gate controls how much of the old state should be forgotten. The LSTMs are able to learn sequential data due to the above properties.

3.2. Conditional generative adversarial network (cGANs)

GANs are generative models that learn a mapping from random noise vector z to output image y, G : z → y. But conditional GANs learn a mapping from observed image x and random noise vector z, to y, G : x, z → y. The generator

[1]https://github.com/sladewinter/ConvLSTM

G is trained such that it produces outputs that cannot be distinguished from "real" images by a trained discriminator (D) which is trained to do as well as possible at detecting the generator's fakes images.
The objective of a conditional GAN can be expressed as:

$$L_{cGAN}(G, D) = E_{x,y}[log(D(x, y)] + E_{x,z}[log(1 - D(x, G(x, z))] \tag{1}$$

where G tries to minimize this objective against an adversarial D that tries to maximize it, so overall objective function is given by:-

$$G^* = \ arg\ min_G(\ arg\ max_D(L_{cGAN}(G, D))) \tag{2}$$

$L(G, D)$ is known as adversarial loss. We have also used Pixelwise L1 content loss to train generator and discriminator model. So the generator task is to not only fool the discriminator but also to be near the ground truth output in an L1 sense. So our total loss comprises of addition of both the pixelwise loss and adversarial loss. We have used L1 distance rather than L2 as L1 encourages less blurring. So L1 loss are as follows is given by :

$$L_{L1}(G) \ = \ E_{x,y}[\ ||\ y\ -\ G(x, z)\ ||_1] \tag{3}$$

So final combined loss function is given by :

$$G^* = \ arg\ min_G\ (arg\ max_D(L_{cGAN}(G, D) + \lambda L_{L1}(G))) \tag{4}$$

where λ is a hyper-parameter and in our implementation we have set its value to 100.

3.2.1 Generator

The generator consists of a series of layers that progressively downsample until a bottleneck layer is reached, after that we reverse the same process i.e we upsample. Such a network requires that all information flow passes through all the layers, including the bottleneck. So there is a low level of information shared like the prominent edge locations between input and output which produces high-quality images than simple CNN generator model.
We have added skip connections, to avoid the vanishing gradient problem and to incorporate information from earlier layers. This is the general shape of a UNet architecture. Specifically, we add skip connections between each layer i and layer $n - i$, where n is the total number of layers. Each skip connection simply concatenates all channels at layer i with those at layer $n - i$.

3.2.2 Discriminator

The discriminator is a type of Convolution Neural Network(CNN) model. The discriminator has 4 sequential blocks having 64, 128, 256 , 512 filters respectively. Each sequential block consists of Convolutional layer, Batch-normalization layer and activation layer as LeakyRelu. The generator tries to fool the discriminator by creating fake images. The discriminator is trained in such a way that it tries to distinguish real data from the data created by the generator.

3.3. UNet

UNets, first introduced in Convolutional Networks for Biomedical Image Segmentation paper [10], have been able to expand their use case from image segmentation to predicting the future sequence too. The architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization. Encoder (downsampling path) extracts a meaningful feature map from an input image. As is standard practice for a CNN, the Encoder, doubles the number of channels at every step and halves the spatial dimension. Next, the Decoder (upsampling path) actually upsamples the feature maps, where at every step, it doubles the spatial dimension and halves the number of channels (opposite to that of what an Encoder does).
The contractive path (Encoder) consists of the repeated application of two 3x3 convolutions (padding=1), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step we double the number of feature channels. As the image size we use (128 x 128) is very small to be downsampled too much, we couldnt traverse the contractive path as much as given in the reference papers, i.e. rather than having out channels as 1024 in final output of our Encoder, we had 256 as our number of out channels. Hence, the sequence of in and out channels for each block in the Encoder consisted of [2,64,128,256].
The decoder layer is explained as following: every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution (up-convolution) that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution. The *ConvTranspose2d* operation performs the up-convolution and again the same block consisting of Conv2D and ReLU in between is used to half the number of channels. The output of this Decoder is 128 x 128 image for each batch, which can be again compared to the target and initiate the learning of our model.
The usage of residual skip connections helps alleviate the vanishing gradient problem allowing for UNet models with

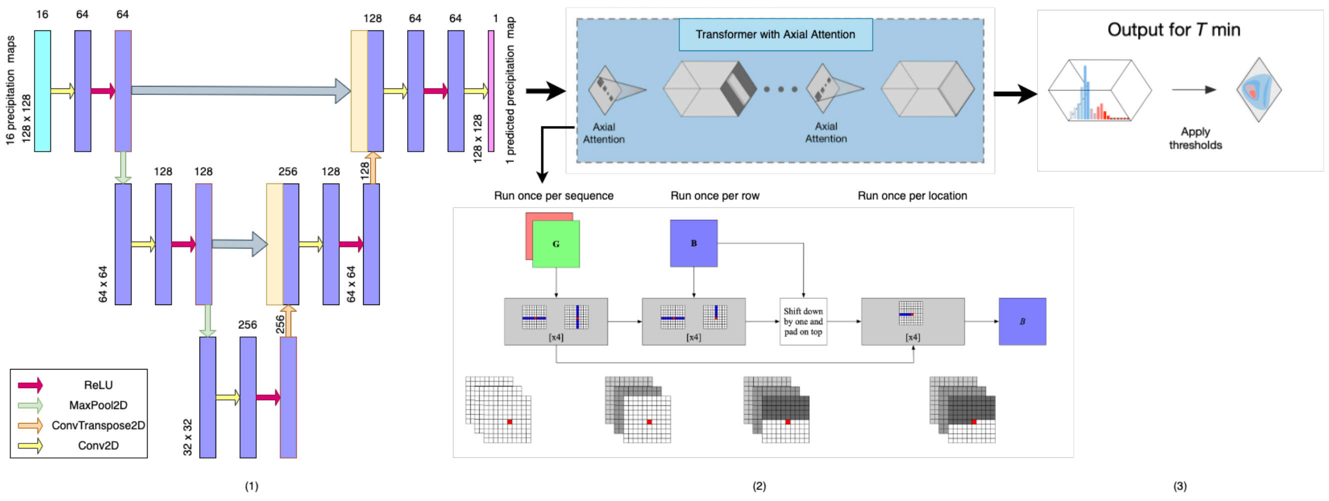


Figure 1. The figure shows the final novel UNet with Axial Transformer model. Part (1) is our UNet model that encompasses encoder and decoder using Conv2D. Part (2) is the transformer [18] that attends every row and column for each location (pixel), hence an axial attention [4]. Part (3) is the distribution we get as our final output of which we take mean to make final prediction for Tth minute

deeper neural networks to be designed. Each residual unit can be denoted by the following expressions:

$$y_l = h(x_l) \ + F(x_l, w_l)x_{l+1} = y_l \tag{5}$$

$$x_{l+1} = y_l \tag{6}$$

To handle a time series data that we have, we could have probably used Conv-3D and MaxPool-3D layers in our UNet. But as the data was preprocessed into a GRAYSCALE image, we could just substitute the number of frames as the number of input channels, an idea inspired by SmaAt-UNet. [19].

3.4. UNet with Axial Transformer

Our Novel model is based on axial attention, a simple generalization of self-attention that naturally aligns with the multiple dimensions of the tensors in both the encoding and the decoding settings. As we know Attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences. So we have used Axial Transformers, an axial-attention-based autoregressive model for images and other data organized as high dimensional tensors.

The sequence of images, which are the first 16 frames from the sequence of length 20, are fed into our Downsampler, which is same as the Encoder of our UNet model used before. This acts as an image processing pipeline and the first stage of our model. The next layer is a Decoder layer that outputs the a representation of the time series, i.e. integrating the information over-time. The Internal layers of both Encoder and Decoder are made of several layers of CNN. We produce one feature map per input image from the sequence of length 16, previously in the simple UNet model we compressed all information within a single image output from the model, but now these 16 images will act as an input to our next stage- Axial Transformer [4].

Axial attention can be used within standard Transformer layers in a straightforward manner to produce Axial Transformer layers. The basic building blocks are the same as those found in the standard Transformer architecture:

Inner Decoder: using masked row attention layers to create a “row-wise” model: L_{row} is the number of masked row attention blocks applied to h, The operation ShiftRight shifts the input right by one pixel. PositionEmbeddings is a tensor of position embeddings that inform the attention layers of the position. x is the gray scale frame.

```
h ← Embed(x)
h ← ShiftRight(h) + PositionEmbeddings
h ← MaskTransformerBlock2(h) ×$L_{row}$
```

Outer Decoder: Each pixel in the model depends on previous pixels in its own row. To capture all previous rows, insert unmasked row and masked column layers in the beginning of the model as described below:

The tensor u represents context captured above the current pixel. It is computed by unmasked row and masked column attention layers, repeated to a total of L_{upper} layers.

```
h ← Embed(x)
u ← h + PositionEmbeddings
u ← MaskTransformerBlock1(Block2(u)) × $L_{upper}/2$
h ← ShiftDown(u) + ShiftRight(h) + PositionEmbeddings
h ← MaskedTransformerBlock2(h) × $L_{row}$
```

The idea of using a transformer with a Encoder-Decoder model was inspired by a recent development in the field of Precipitation Nowcasting at Google who introduced a Neural Weather model called MetNet [18].
Now what Axial Tranformer will try to do is, encode information in these feature maps from the space around each point of interest. Hence, attending parts that are relevant to justify motion of a cloud patch in a radar image. The way Transformer works is- we have a series of feature maps that are continuous with respect to time. Each pixel from the latest frame emits a query vector and each of the pixel from the older image feature maps emits a key. And each of the pixel emitting query can look at (attend) each of the pixel in the lower layer (keys). Hence, we are able to incorporate long range dependencies aggregating information from the downstream and increasing the resolution. But in Axial Transformers we will specifically attend the pixels that are either in the same row or same column in the images produced. This saves the required memory for the computation over the layers of attention while aggregating information over the spatial dimensions. The code implementation of the Transformer was inspired by the PyTorch implementation of axial attention available on GitHub [2].
The final output of the this model is a distribution across 128 frames output from transformer head out of which we need to take the mean value as the predicted state of a pixel after the prediction. This predicted tensor, flattened, along with the target image tensor also flattened, is sent to calculate the loss and start model learning.

4. Experimental Results

In this section, we first introduce related datasets and implementation details in our experiments. Then, we report results and comparison of all the experiments conducted on each of the model used.

4.1. Datasets

The dataset used in this project is a part of dataset used in paper ConvLSTM for Precipitaion Nowcasting [14]. The dataset for the original research work consisted of HKO-7 dataset which contains radar echo data from 2009 to 2015 near Hong Kong. Along with the radar images, the dataset also compressed of proportions of rainfall events with different rain-rate threshold.

[2]https://github.com/lucidrains/axial-attention

The HKO-7 dataset used in the baseline research contains radar CAPPI reflectivity images, which have resolution of 480×480 pixels, are taken from an altitude of 2km and cover a 512km × 512km area centered in Hong Kong. The data are recorded every 6 minutes and hence there are 240 frames per day. The pixel values are clipped between 0 and 255. There are some noisy radar images in the dataset generated by factors like ground clutter, sea clutter, anomalous propagation and electromagnetic interference.
The complete dataset with all the csv and pkl files was not available publicly. But a part of it, i.e., the radar images that can be conveniently used in our problem are available publicly [5]. We have continuous radar images from 514 folders. Each folder contains variable number of radar images sampled, but all of them are continuous. As the radar echo maps arrive in a stream, nowcasting algorithms can apply online learning to adapt to the newly emerging spatiotemporal patterns.
This was the most convenient dataset we could use, because, the datasets used in previous research under this topic were too large to be handled on the GPU provided (datasets of size in TBs). Other dataset were either paid or not publicly available. Even the dataset that we have used, is a part of a original dataset. But it was enough to carry out several experimentations on various models used.

4.2. Implementation Details

After taking the dataset as our input, first we had to ensure that the number of radar images in each folder's sequence remain same. This is due to the fact that we will be sending the dataset in batches for training and testing and we need to have uniform number of images in each batch. Hence, we clipped the dataset to 240 images per folder (if the number of radar images in a folder is greater than 240, if it is less than 240 we reject that folder), i.e. as soon as the number of images in that particular folder go past 240, we stop taking that folder's images as input. We found out that dataset now had 482 sequences of length 240. While we input each image, we also convert the image into grayscale, as we need to map the cloud motion in the radar images and having RGB channels in the images will only cost computation time and power without much significant difference in the results.
At first, we also tried to send these data directly for training in batches. But again, due to computational power limitations, we could not send a batch of size 240 for training. The sequence of images again had to be broken to the length of 20, i.e. instead of 482 sequences of length 240, now we had 5784 sequences of length 20. Now sequence of this length can be comfortably loaded for training and testing.
Now that we had sequence of data, we also needed to filter out the images that are either too dark or too light. There were some faulty images in our dataset as previously talked

about because of faulty radar images. So we first stored the summation of all the pixel values in each image of our dataset in a list. This is important to normalise our dataset, as then we found the 25th and 75th percentile of the list values. These gives us the range we will use to filter images whose sum of pixel values are above or below the specified range.
Now going through each sequence, we again calculated the sum of pixel values of each image in 5784 sequences, if the sum is outside the range, we count that as a bad-frame. If the number of bad-frames in a sequence exceed 10, we discard that sequence as a whole, as we cant discard few frames from a sequence; discontinuity in frames would make them useless. Now once the data is cleaned, we are now left with 2901 sequences of length 20, each being an image of size 128 x 128. The sequences then were loaded in an .npy file to used later. As observed the usable images left at last very almost half of the dataset we created at start and this was one of the reasons why we couldnt use higher batch size while training our models.
Once the .npy file is loaded, we randomly shuffled these sequences without altering their internal continuity in frames, for uniform distribution of types of images for training, validation and testing. Out of these, 2000 sequences were used in training dataset, 500 for validation and 400 for testing purpose. Whenever the numpy array representing the images in the dataset is loaded, the batch is is normalised by dividing by 255, to bring each pixel value between 0 and 1. Shuffle is kept true to make the distribution among the types of images uniform.
The sequence length given to train Conv-LSTM was 15, so that it can count next 5 frames as target and make prediction on those. For cGANs, the input sequence length was 4, as it was an image-to-image translation technique and produced one image per input image, hence predicting 4 frames from 4 inputs. For UNet and UNet with axial attention, we had to give 16 frames as input to the model as we needed to ensure downsampling and later upsampling produce images (given in number of channels dimensions) of same number at each corresponding layer. The batch size for ConvLSTM = 2, cGANs = 4, UNet = 4, UNet with axial transformer = 1. Every model was given a batch size with limitation due to risk of GPU running out of memory or model overfitting too soon.
For a given input sequence, 1 frame was predicted as output from every model. Now the same frame was used as input while predicting the next frame and so on. Hence, by this method we were able to predict 4-5 frames in future using the original sequence of input image.
Number of epochs used to train the models is variable due to effect of overfitting for larger and deeper models. ConvLSTM: 5, cGANs: 10, UNet: 5, UNet with Axial Transformer: 15. Optimizer used is Adam, with a learning rate of: *1e-3* for ConvLSTM, *1e-4* for cGANs, *1e-3* for UNet, *1e-4* for UNet with Axial Transformer. The loss criterion for each of the model used is *MSELoss*.
Given a noise-free flattened monochrome image I and its flattened noisy approximation Image K, MSELoss between two tensors is defined as:

$$MSELoss = \frac{1}{2}\sum_{i=0}^{n-1}(\, I_i - K_i \,) \tag{7}$$

The metrices we used to compare the results of different models were: PSNR and SSIM. PSNR: Peak signal-to-noise ratio is the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation.

$$PSNR = -10\,\log_{10}(MSE) \tag{8}$$

MSE is defined as mean squared error between a noise-free flattened monochrome image and its flattened noisy approximation Image K
SSIM: Structural Similarity Index (SSIM) measures the similarity between two images by quantifying the image degradation of one image with respect to another.
The SSIM between two images x and y of common size N×N is:

$$SSIM(\,x,y) = \frac{(2u_x u_y + c_1)(2\sigma_{xy} + c_2)}{\left(u_x^2 + u_y^2 + c_1\right)\left(\sigma_x^2 + \sigma_y^2 + c_2\right)} \tag{9}$$

where μ_x is the average of x, μ_y is the average of y, σ_x^2 is the variance of x; σ_y^2 is the variance of y and c_1 and c_2 are constants.

4.3. Results and Comparisons

While training each of the model we calculated the loss for training and cross validation data, the results of which are given below:

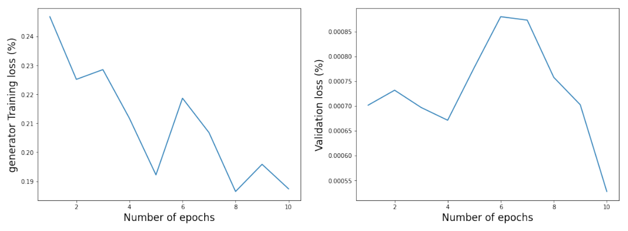
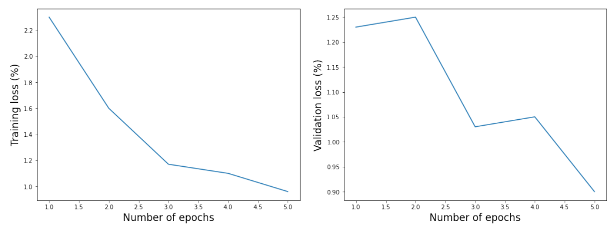


Figure 2. cGANs training



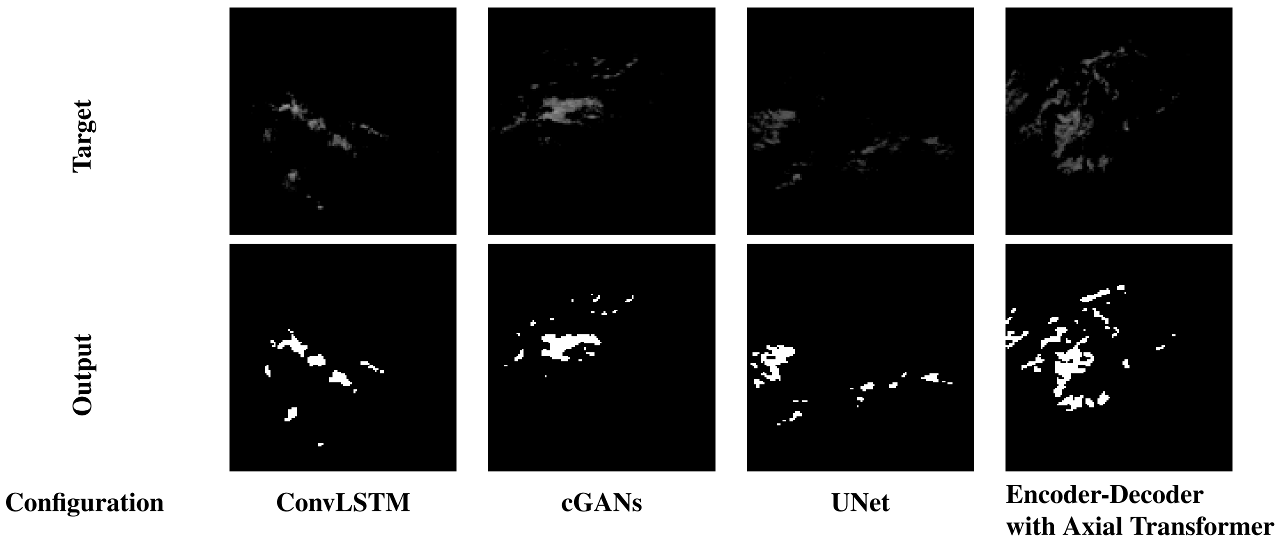


Table 1. While testing each of the model, we first input sequence (length similar to that used for training of each model) of images $(t_0, t_1, ..., t_m)$ into the model and get 1 image as output (p_1) from it. Now this predicted image p_1 is used along with input sequence $(t_1, .., t_m)$ to predict the next radar image and so on. The result of each model are given at the top and as we can see the novel model has the highest quality of prediction.

Figure 3. UNet training

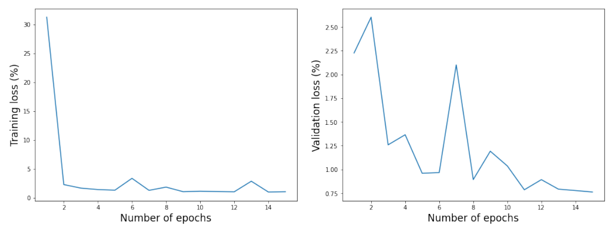


Figure 4. UNet with Axial Transformer training

We see some unusual spikes in the graph which might be due to faulty radar images and abrupt shuffling of data.
The models used in the experiments showed a gradual increase in both the metrices and hence show that our novel architecture performs better than the baseline model used.

Configuration	PSNR	SSIM
ConvLSTM	40.8852	0.9710
cGANs	41.1319	0.9826
UNet	47.2862	0.9929
Encoder-Decoder Axial Transformer	**47.6678**	**0.9943**

Table 2. The table shows comparison of results from each model on the test dataset

While the baseline ConvLSTM showed satisfactory results on the test dataset there was an improvement when cGANs were used. Using the pixel to pixel translation in radar images using generator and discriminator, the model is robust to random noise introduced.
The generator used in cGANs was UNet which was able to give much better results while it was used alone. This might be possible due to the fact that dataset used was comparatively small to be used by larger and deeper models. Hence, a simple model can perform better when there is less complexity involved in learning. Furthermore, the main task of GANs model is to generate new images with same statitics, hence with pix2pix translation we could only use a single image to generate another image which corresponds to a sequence of image to generate other half of sequence and hence almost half the dataset available for training as compared to rest of the models.
The skip connections in UNet not only helps to resolve vanishing gradient problem, but also helps in feature reusability. With a very simple architecture it is able to embed enough information of sequence to predict the next frame in the sequence. A further enhancement was use of axial transformers on top of UNet. This ensured that model is able to attend the pixels in same rows and columns making it more suitable for trajectory tracking problems like these.
While testing a model, once we have an output, we actual have its pixel value scaled between 0 and 1, due to initial normalisation applied on input data. Hence, we first see if the value of pixel is above some threshold (typically 0.2), if it is, we multiply it with 255, hence directly converting it to a white pixel. This is the reason why we have B&W image in the output.

5. Discussion

In this paper, we propose a novel framework for precipitation nowcasting by predicting 4-5 frames in future using a sequence of frames from past, each 6 minutes apart. Our models exploits spatio-temporal features of the radar images and create realistic predictions. Moving beyond traditional weather forecasting we exhibit state-of-art solution for the HKO-7 dataset used.
We experimented on several models to find the best fit for the problem and found that UNet paired with Axial Transformer is able to predict the next radar images more accurately than any of the other models used, due to the skip connections and ability to attend pixels in same row and column. We started with a baseline model and added components systematically and saw the results amplifying.
With enhanced dataset like that including actual rainfall values with images can help in building end-to-end model to predict % rainfall for a given area. The methodology used can also find its use in the areas of traffic management, bio-medical, and autonomous vehicles, due to being able to counter the fundamental problem of next frame prediction.

6. Future Work

As discussed earlier, the pixel values of radar images given as input to our model was normalised between 0 and 1. Currently the goal of the project was to predict the radar images that match the trajectory of the current motion of the clouds in the given sequence of images, hence in the output we essentially classify each pixel as cloud or not cloud. In future, we could keep the pixel values in images as it is and hence calculate the amount of precipitation under each pixel using the pixel value. Higher the pixel value, higher the intensity of precipitation.
We also converted the radar images into GRAYSCALE during the preprocessing this was important to reduce the computational cost but with little to no effect in results due to original images themselves being B&W colour. If we are able to work on dataset that consists of colourful satellite images with RGB channels, first we could do image segmentation in the radar images identifying the chunk of land and cloud as seprate and predicting the precipitation intensity corresponding to each of the location in the radar images. This location can be town, city or even as huge as states as they are mapped from actual satellite images and not radar images.

References

[1] Elizabeth A. Barnes, Kirsten Mayer, Benjamin Toms, Zane Martin, and Emily Gordon. Identifying opportunities for skillful weather prediction with interpretable neural networks, 2020.

[2] Sungbin Choi. Traffic map prediction using unet based deep convolutional neural network, 2019.

[3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[4] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers, 2019.

[5] Chen Lei. A Deep Learning Based Methodology for Precipitation1 Nowcasting with Radar, 2019.

[6] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

[7] Tong Mo and Bang Liu. Encoder-decoder neural architecture optimization for keyword spotting, 2021.

[8] Maitreya Patel, Anery Patel, and Dr. Ranendu Ghosh. Precipitation nowcasting: Leveraging bidirectional lstm and 1d cnn, 2018.

[9] Suman Ravuri, Karel Lenc, Matthew Willson, Dmitry Kangin, Remi Lam, Piotr Mirowski, Megan Fitzsimons, Maria Athanassiadou, Sheleem Kashem, Sam Madge, and et al. Skilful precipitation nowcasting using deep generative models of radar. *Nature*, 597(7878):672–677, Sep 2021.

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[11] Lars Ruthotto and Eldad Haber. An introduction to deep generative modeling, 2021.

[12] Siddharth Samsi, Christopher J. Mattioli, and Mark S. Veillette. Distributed deep learning for precipitation nowcasting. *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep 2019.

[13] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, Jan 2015.

[14] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai kin Wong, and Wang chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting, 2015.

[15] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms, 2016.

[16] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding lstm – a tutorial into long short-term memory recurrent neural networks, 2019.

[17] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel Emer. Efficient processing of deep neural networks: A tutorial and survey, 2017.

[18] Casper Kaae Sønderby, Lasse Espeholt, Jonathan Heek, Mostafa Dehghani, Avital Oliver, Tim Salimans, Shreya Agrawal, Jason Hickey, and Nal Kalchbrenner. Metnet: A neural weather model for precipitation forecasting, 2020.

[19] Kevin Trebing, Tomasz Stanczyk, and Siamak Mehrkanoon. Smaat-unet: Precipitation nowcasting using a small attention-unet architecture, 2021.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[21] Zeyu Yun, Yubei Chen, Bruno A Olshausen, and Yann LeCun. Transformer visualization via dictionary learning: contextualized embedding as a linear superposition of transformer factors, 2021.