Electronics and Communication Engineering

Academic Year 2019 - 2023

BTP Report

Rainfall Nowcasting and Classification through Deep Transfer Learning

Professor Mentor: Dr. ANISH CHAND TURLAPATHY

Code: B22ACT01
PREETHI G - S20190020241
SHRI TEJA NAIK - S20190020223

1 Abstract

All the Deep Learning based models require huge historical data for training. If the knowledge learned from one deep learning model trained for one region can be applied to other regions, it will be beneficial in many scenarios. With a goal of building accurate nowcasting model with relatively less amount of data, Deep Transfer Learning models come into picture.

2 Introduction

An application of Deep Transfer Learning in Weather Nowcasting is when a new radar station is established wherein obtaining a long-term data set for training the deep learning nowcasting model is not possible. However, if the model is not retrained to incorporate local precipitation characteristics, it may result in significant uncertainties in the radar nowcasting product. Thus, Deep Transfer Learning will result in an improved Precipitation Nowcasting model. In this project, we have built a total of 3 models: Base CNN, CNN-FT and CNN-GRU for a comparative performance analysis.

3 Description of the Data

Dataset Information:

NEXRAD Level-II (Base) Data
Level-II (L2) data are grouped into three meteorological base quantities: reflectivity, mean radial velocity and spectrum width. Additional categories include differential reflectivity, correlation coefficient, and differential phase. Data is stored in files that typically contain five or ten minutes of base data depending on the volume coverage pattern. A data file consists of a 24-byte volume scan header record followed by numerous 2,432-byte base data and message records.

Input location: Miami, Florida, USA
Output location: New Orleans, Louisiana, USA

Attribute Information:

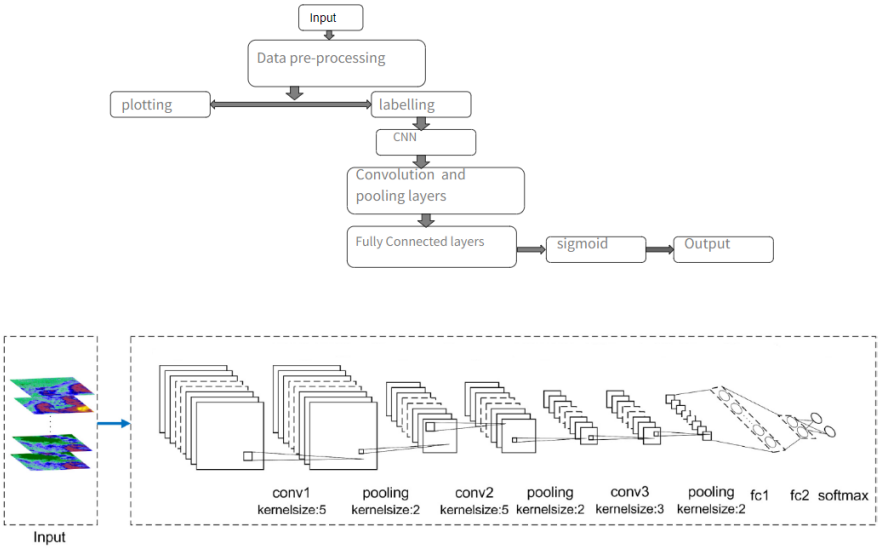
A single file contains the following:
Altitude, Azimuth Angle, Elevation, Fields such as Cross Correlation Ratio, Reflectivity, Differential phase, Differential reflectivity, Velocity, Spectrum Width, Clutter filter power removed, Fixed angle, Latitude, Longitude, etc. We only require Reflectivity, Latitude and Longitude Fields for Spatial Analysis.

We initially use encoded values of Reflectivity to detect precipitation. The encoding is such that a thresold of 35 dBz has been set and any value in the matrix containing a reflectivity value greater than 35 dBz is labelled as '1' and is considered as raining and any value less than or equal to 35 dBz is labelled as '0' and taken as not raining.

Example of a Reflectivity Matrix is given below:

```
KAMX20210601_000300_V06
[[-15.0 -4.0 -2.5 ... -- -- --]
 [-10.0 -7.0 -6.5 ... -- -- --]
 [-13.5 -10.5 -4.0 ... -- -- --]
 ...
 [-20.0 -20.0 -21.0 ... -- -- --]
 [-22.0 -11.0 -11.0 ... -- -- --]
 [-15.5 -13.5 -12.5 ... -- -- --]]
```

4 Methodology



Block Diagram of Base CNN model

4.1 Downloading the Dataset

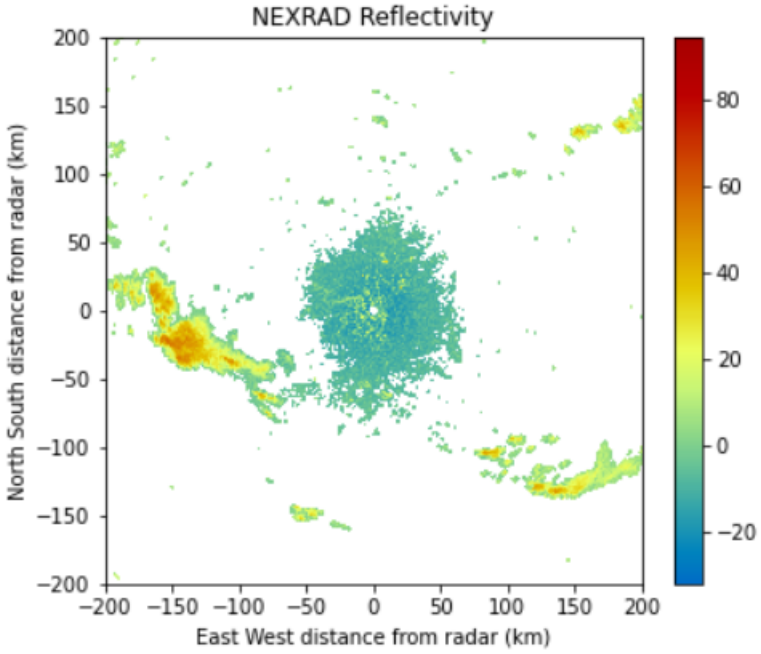
The Dataset was downloaded from the NOAA NEXRAD on AWS. The following locations were downloaded:

- Input location: Miami, Florida, USA – Code: KAMX
- Output location: New Orleans, Louisiana, USA – Code: KLIX

4.2 Pre-processing

- To read, process and visualize the dataset, we used Python ARM Radar Toolkit (Py-ART) which is an open source library to work with weather radar data. It is supported by the United States Department of Energy as a part of the Atmospheric Radiation Measurement (ARM), Climate Research Facility.

- The radar reflectivity matrix can be visualized as follows:



  The Latitude and Longitude values are taken in horizontal and vertical directions respectively, while the colour bar indicated the reflectivity values. The precipitation range starts from light orange and continues till dark red indicating the intensity of precipitation.

- Given the large amount of data contained in each file, the size of the data set is over 100 GB. To reduce the storage required, only the useful data i.e. Reflectivity, Latitude and Longitude has been stored as numpy arrays and used further.
- For classification the total rainfall area was detected and if the area is more than 400 sq. Km then it was labelled as large scale and area less than 400 sq. km was labelled as small scale.
- Dynamic plots are harder to execute compared to the static plots but they make it very easy to visualize and assess patterns. The Dynamic plots were also plotted spanning across a time period of 4 hrs.

4.3 Deep Learning Models

The following models were built:

- Base CNN (Convolutional Neural Network) Model
- CNN-FT (Convolutional Neural Network - Fine Tuned) Model
- CNN-GRU (Convolutional Neural Network - Gated Recurrent Unit) Model

The Base CNN Model is trained with Reflectivity Matrix from Miami region over the months of June-July, 2021.
The CNN-FT and CNN-GRU Models are trained with a small amount of data (1st week of July 2021) from New Orleans to implement transfer learning.

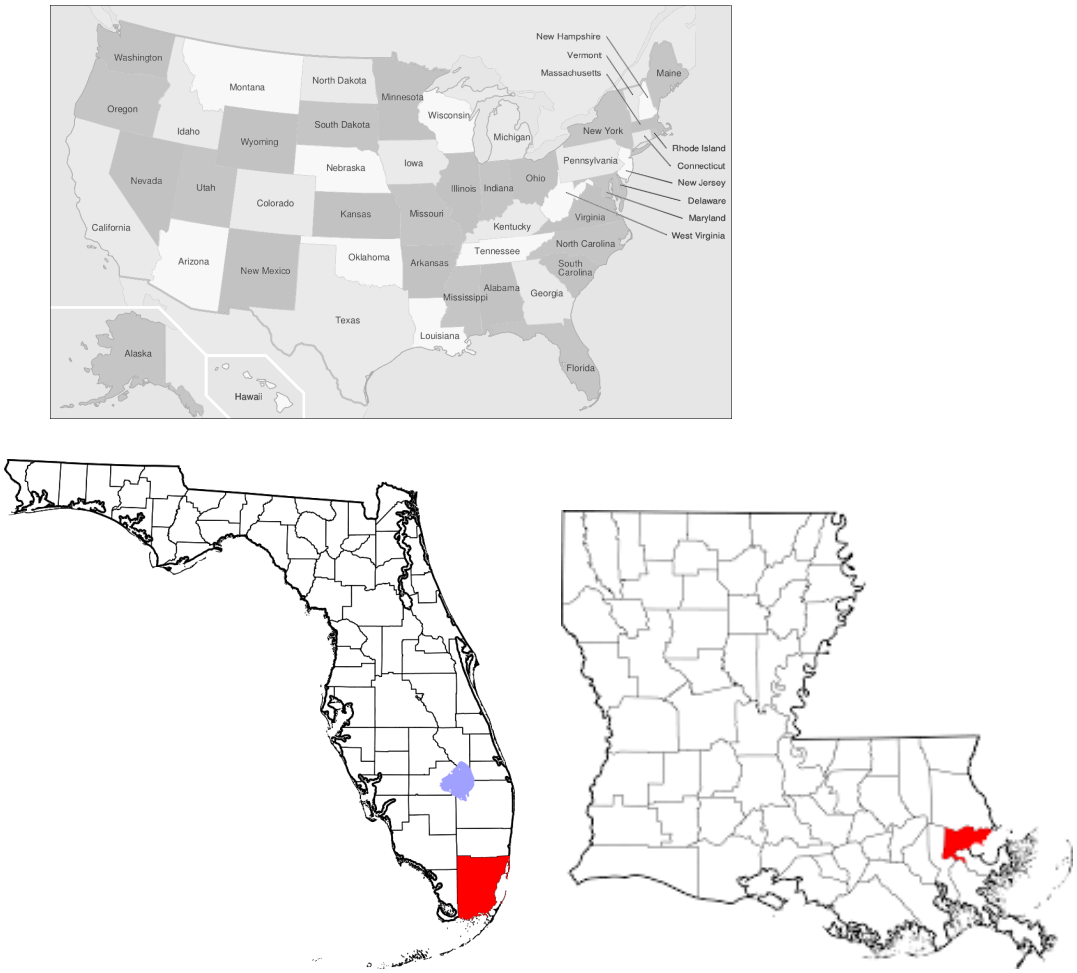
4.3.1 Base CNN Model

- The work flow starts with taking the radar reflectivity data as input and then pre-processing it using Py-ART and plotting it to get a visual representation.
- We label the data and send the labeled data as input to the CNN model which is made up of a series of Convolution and Pooling layers in the following order:
  - Convolution Layer 1 with a Kernal Size of 5
  - Pooling Layer 1 with a Kernal Size of 2
  - Convolution Layer 2 with a Kernal Size of 5
  - Pooling Layer 2 with a Kernal Size of 2
  - Convolution Layer 3 with a Kernal Size of 3
  - Pooling Layer 3 with a Kernal Size of 2

The model then goes through 2 Fully Connected layers (Dense Layers) and at last the sigmoid function normalizes the neural network to fit between 0 and 1.
The Output Shapes of Convolution and Pooling layers are as given in the below CNN Model Summary:

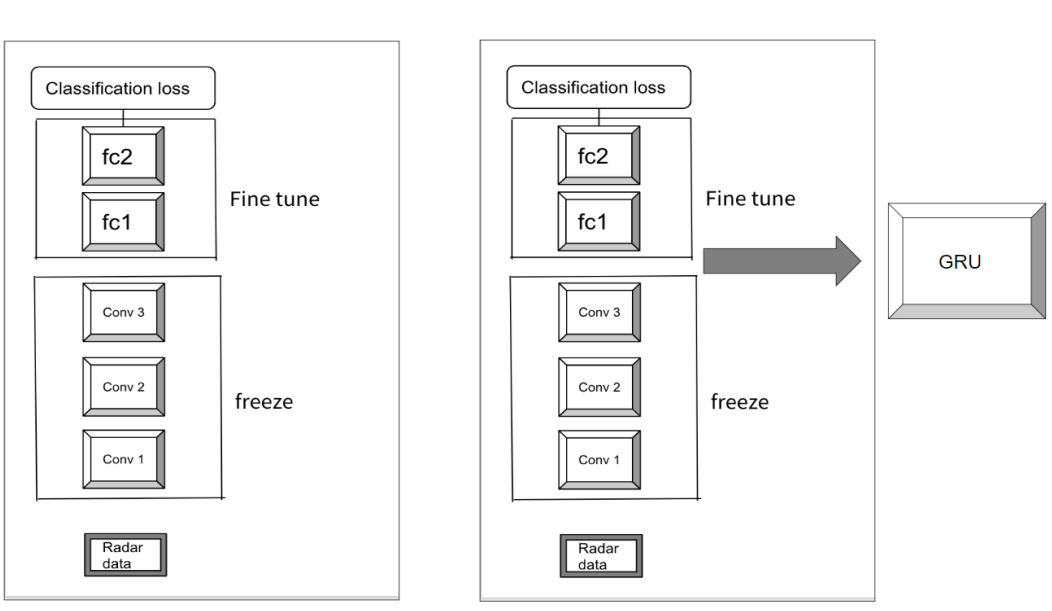
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 234, 720, 64)      2931264

 max_pooling2d (MaxPooling2D  (None, 117, 360, 64)     0
 )

 conv2d_1 (Conv2D)           (None, 117, 360, 64)      102464

 max_pooling2d_1 (MaxPooling  (None, 59, 180, 64)      0
 2D)

 conv2d_2 (Conv2D)           (None, 59, 180, 64)       36928

 max_pooling2d_2 (MaxPooling  (None, 30, 90, 64)       0
 2D)

 flatten (Flatten)           (None, 172800)            0

 dense (Dense)               (None, 50)                8640050

 dense_1 (Dense)             (None, 1319040)           67271040

=================================================================
Total params: 78,981,746
Trainable params: 78,981,746
Non-trainable params: 0
_________________________________________________________________
```

After training the model with Miami Data, the trained model parameters (weights of Neural Network) are saved for further models (CNN-FT and CNN-GRU).



Demonstration study domains: (a) map of USA, (b) source study domain (red region indicated in Florida Map) in Miami, Florida, USA and (c) target study domain (red region indicated in Louisiana Map)in New Orleans, Louisiana, USA.

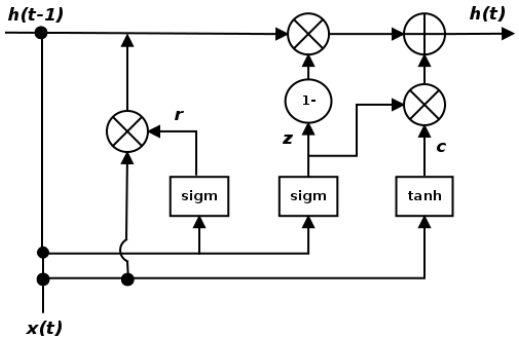
4.3.2 CNN-FT Model



The first couple of layers in the Base CNN mainly outputs only general features and the features learned by the network become more specialized as the Depth of the Neural Network increases. Features transition from general to specific mainly in the last Fully Connected (Dense) layers in a CNN model. Also, the base CNN model has to be trained from scratch with randomly assigned weights initially.

Therefore, we first train a base CNN model using a relatively large amount of data from Miami (source-domain data) and then copy its weights to New Orleans region (target CNN model) so as to take advantage of the pre-trained weights. The layers of the target network is retrained using a small amount of target-domain data, which is also called FT.

4.3.3 CNN-GRU Model



$$z = \sigma(W_z \cdot x_t + U_z \cdot h_{(t-1)} + b_z)$$
$$r = \sigma(W_r \cdot x_t + U_r \cdot h_{(t-1)} + b_r)$$
$$\tilde{h} = tanh(W_h \cdot x_t + r * U_h \cdot h_{(t-1)} + b_z)$$
$$h = z * h_{(t-1)} + (1 - z) * \tilde{h}$$

Gated Recurrent Unit is a sequence modeling technique just like RNN and LSTM. GRUs are very similar to Long Short Term Memory(LSTM). Just as in LSTM, GRU also uses gates to control the flow of information with the main difference in the number of gates being used i.e. two gates in GRU (reset and update) and three in LSTM (input, output, forget). This is the reason they offer similar results while having a simpler architecture.

The GRU Layers are added after flattening the outputs from the Convolution and Pooling Layers. The 1st GRU layer is implemented with a units count of 32, activation function as tanh (default) and recurrent activation function as sigmoid (default) followed by a 2nd GRU layer with a units count of 16, activation function as relu and recurrent activation function as sigmoid (default). The GRU output is then passed to a Fully Connected (Dense) Layer followed by another Fully Connected Layer which is reshaped to generate the forecast output.

The model weights for the convolution and pooling layers are loaded from the CNN model. Only the GRU and Dense layer parameters are trained from randomly assigned weights using New Orleans Data.

Each model has been run for a total of 50 epochs (Experimentally Hyperparameter Tuned given high computational requirements) with a batch size of 32 per epoch.

Model Summary: GRU

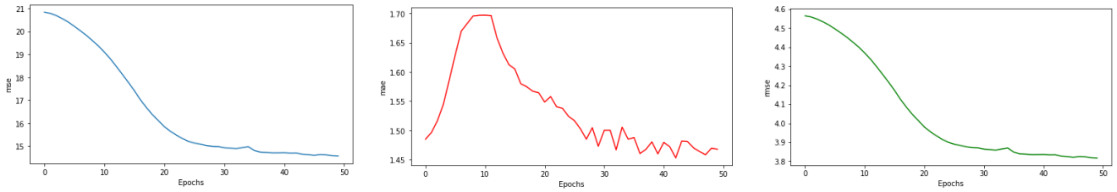
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 5, 720, 1832)]    0

conv2d (Conv2D)              (None, 4, 719, 64)        469056

max_pooling2d (MaxPooling2D  (None, 2, 360, 64)        0
)

conv2d_1 (Conv2D)            (None, 1, 359, 64)        16448

max_pooling2d_1 (MaxPooling  (None, 1, 180, 64)        0
2D)

conv2d_2 (Conv2D)            (None, 1, 180, 64)        16448

max_pooling2d_2 (MaxPooling  (None, 1, 90, 64)         0
2D)

dropout (Dropout)            (None, 1, 90, 64)         0

flatten (Flatten)            (None, 5760)              0

reshape (Reshape)            (None, 90, 64)            0

gru (GRU)                    (None, 90, 32)            9408

gru_1 (GRU)                  (None, 16)                2400

dense (Dense)                (None, 50)                850

dense_1 (Dense)              (None, 1319040)           67271040

=================================================================
Total params: 67,785,650
Trainable params: 67,785,650
Non-trainable params: 0
_________________________________________________________________
```

5 Performance Evaluation:

5.1 Base CNN

Output for 50 epoch run:

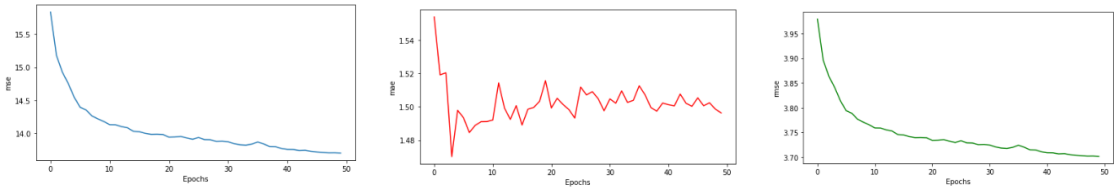
```
Epoch 1/50
6/6 [==============================] - 14s 659ms/step - loss: 20.8331 - mse: 20.8331 - mae: 1.4849 - root_mean_squared_error: 4.5643 - val_loss: 22.9014 - val_mse: 22.9014 - val_mae: 1.6036 - val_root_mean_squared_error: 4.7856
Epoch 2/50
6/6 [==============================] - 2s 299ms/step - loss: 20.7802 - mse: 20.7802 - mae: 1.4963 - root_mean_squared_error: 4.5585 - val_loss: 22.8089 - val_mse: 22.8089 - val_mae: 1.6201 - val_root_mean_squared_error: 4.7759
Epoch 3/50
6/6 [==============================] - 2s 302ms/step - loss: 20.6806 - mse: 20.6806 - mae: 1.5164 - root_mean_squared_error: 4.5476 - val_loss: 22.6855 - val_mse: 22.6855 - val_mae: 1.6442 - val_root_mean_squared_error: 4.7629
Epoch 4/50
6/6 [==============================] - 2s 298ms/step - loss: 20.5477 - mse: 20.5477 - mae: 1.5445 - root_mean_squared_error: 4.5330 - val_loss: 22.5160 - val_mse: 22.5160 - val_mae: 1.6758 - val_root_mean_squared_error: 4.7451
Epoch 5/50
6/6 [==============================] - 2s 309ms/step - loss: 20.3970 - mse: 20.3970 - mae: 1.5866 - root_mean_squared_error: 4.5163 - val_loss: 22.3075 - val_mse: 22.3075 - val_mae: 1.7185 - val_root_mean_squared_error: 4.7231
Epoch 6/50
6/6 [==============================] - 2s 297ms/step - loss: 20.2129 - mse: 20.2129 - mae: 1.6294 - root_mean_squared_error: 4.4959 - val_loss: 22.0911 - val_mse: 22.0911 - val_mae: 1.7635 - val_root_mean_squared_error: 4.7001
Epoch 7/50
6/6 [==============================] - 2s 298ms/step - loss: 20.0218 - mse: 20.0218 - mae: 1.6698 - root_mean_squared_error: 4.4746 - val_loss: 21.8850 - val_mse: 21.8850 - val_mae: 1.7979 - val_root_mean_squared_error: 4.6781
Epoch 8/50
6/6 [==============================] - 2s 306ms/step - loss: 19.8205 - mse: 19.8205 - mae: 1.6829 - root_mean_squared_error: 4.4520 - val_loss: 21.6470 - val_mse: 21.6470 - val_mae: 1.8063 - val_root_mean_squared_error: 4.6526
Epoch 9/50
6/6 [==============================] - 2s 299ms/step - loss: 19.5945 - mse: 19.5945 - mae: 1.6959 - root_mean_squared_error: 4.4266 - val_loss: 21.3785 - val_mse: 21.3785 - val_mae: 1.7833 - val_root_mean_squared_error: 4.6237
Epoch 10/50
6/6 [==============================] - 2s 302ms/step - loss: 19.3613 - mse: 19.3613 - mae: 1.6972 - root_mean_squared_error: 4.4002 - val_loss: 21.1159 - val_mse: 21.1159 - val_mae: 1.7738 - val_root_mean_squared_error: 4.5952
Epoch 11/50
...
Epoch 50/50
6/6 [==============================] - 2s 298ms/step - loss: 14.5707 - mse: 14.5707 - mae: 1.4677 - root_mean_squared_error: 3.8172 - val_loss: 16.8718 - val_mse: 16.8718 - val_mae: 1.5163 - val_root_mean_squared_error: 4.1075
(188, 1319040)
```



5.2 CNN-FT

Output for 50 epoch run:

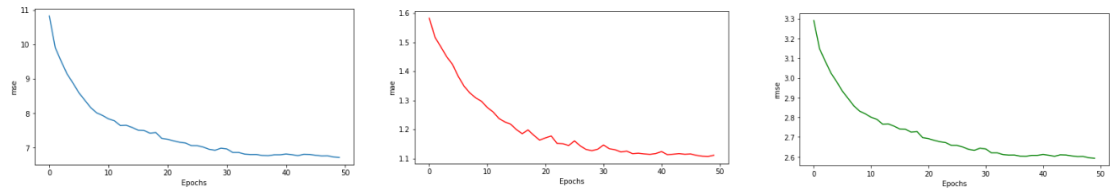
```
Epoch 1/50
7/7 [==============================] - 12s 526ms/step - loss: 15.8348 - mse: 15.8348 - mae: 1.5538 - root_mean_squared_error: 3.9793 - val_loss: 14.9121 - val_mse: 14.9121 - val_mae: 1.4694 - val_root_mean_squared_error: 3.8616
Epoch 2/50
7/7 [==============================] - 2s 274ms/step - loss: 15.1802 - mse: 15.1802 - mae: 1.5191 - root_mean_squared_error: 3.8962 - val_loss: 14.8288 - val_mse: 14.8288 - val_mae: 1.5511 - val_root_mean_squared_error: 3.8508
Epoch 3/50
7/7 [==============================] - 2s 281ms/step - loss: 14.9246 - mse: 14.9246 - mae: 1.5203 - root_mean_squared_error: 3.8632 - val_loss: 14.4663 - val_mse: 14.4663 - val_mae: 1.4529 - val_root_mean_squared_error: 3.8035
Epoch 4/50
7/7 [==============================] - 2s 276ms/step - loss: 14.7526 - mse: 14.7526 - mae: 1.4701 - root_mean_squared_error: 3.8409 - val_loss: 14.3912 - val_mse: 14.3912 - val_mae: 1.4526 - val_root_mean_squared_error: 3.7936
Epoch 5/50
7/7 [==============================] - 2s 275ms/step - loss: 14.5453 - mse: 14.5453 - mae: 1.4979 - root_mean_squared_error: 3.8138 - val_loss: 14.3321 - val_mse: 14.3321 - val_mae: 1.4875 - val_root_mean_squared_error: 3.7858
Epoch 6/50
7/7 [==============================] - 2s 280ms/step - loss: 14.3930 - mse: 14.3930 - mae: 1.4934 - root_mean_squared_error: 3.7938 - val_loss: 14.4079 - val_mse: 14.4079 - val_mae: 1.4468 - val_root_mean_squared_error: 3.7958
Epoch 7/50
7/7 [==============================] - 2s 276ms/step - loss: 14.3524 - mse: 14.3524 - mae: 1.4846 - root_mean_squared_error: 3.7885 - val_loss: 14.3020 - val_mse: 14.3020 - val_mae: 1.4728 - val_root_mean_squared_error: 3.7818
Epoch 8/50
7/7 [==============================] - 2s 278ms/step - loss: 14.2609 - mse: 14.2609 - mae: 1.4888 - root_mean_squared_error: 3.7764 - val_loss: 14.2373 - val_mse: 14.2373 - val_mae: 1.4570 - val_root_mean_squared_error: 3.7732
Epoch 9/50
7/7 [==============================] - 2s 283ms/step - loss: 14.2168 - mse: 14.2168 - mae: 1.4911 - root_mean_squared_error: 3.7705 - val_loss: 14.2942 - val_mse: 14.2942 - val_mae: 1.4683 - val_root_mean_squared_error: 3.7808
Epoch 10/50
7/7 [==============================] - 2s 281ms/step - loss: 14.1767 - mse: 14.1767 - mae: 1.4911 - root_mean_squared_error: 3.7652 - val_loss: 14.2845 - val_mse: 14.2845 - val_mae: 1.4591 - val_root_mean_squared_error: 3.7795
Epoch 11/50
...
Epoch 50/50
7/7 [==============================] - 2s 273ms/step - loss: 13.6989 - mse: 13.6989 - mae: 1.4963 - root_mean_squared_error: 3.7012 - val_loss: 13.9934 - val_mse: 13.9934 - val_mae: 1.4984 - val_root_mean_squared_error: 3.7408
```



5.3 CNN-GRU

Output for 50 epoch run:

```
Epoch 1/50
5/5 [==============================] - 10s 688ms/step - loss: 10.8289 - mse: 10.8289 - mae: 1.5825 - root_mean_squared_error: 3.2907 - val_loss: 9.8625 - val_mse: 9.8625 - val_mae: 1.5110 - val_root_mean_squared_error: 3.1405
Epoch 2/50
5/5 [==============================] - 1s 297ms/step - loss: 9.9096 - mse: 9.9096 - mae: 1.5164 - root_mean_squared_error: 3.1480 - val_loss: 9.4059 - val_mse: 9.4059 - val_mae: 1.4739 - val_root_mean_squared_error: 3.0669
Epoch 3/50
5/5 [==============================] - 1s 294ms/step - loss: 9.5142 - mse: 9.5142 - mae: 1.4837 - root_mean_squared_error: 3.0845 - val_loss: 8.9799 - val_mse: 8.9799 - val_mae: 1.4329 - val_root_mean_squared_error: 2.9966
Epoch 4/50
5/5 [==============================] - 1s 289ms/step - loss: 9.1457 - mse: 9.1457 - mae: 1.4499 - root_mean_squared_error: 3.0242 - val_loss: 8.6886 - val_mse: 8.6886 - val_mae: 1.4062 - val_root_mean_squared_error: 2.9476
Epoch 5/50
5/5 [==============================] - 1s 299ms/step - loss: 8.8798 - mse: 8.8798 - mae: 1.4235 - root_mean_squared_error: 2.9799 - val_loss: 8.2878 - val_mse: 8.2878 - val_mae: 1.3574 - val_root_mean_squared_error: 2.8789
Epoch 6/50
5/5 [==============================] - 1s 301ms/step - loss: 8.5985 - mse: 8.5985 - mae: 1.3821 - root_mean_squared_error: 2.9323 - val_loss: 8.0892 - val_mse: 8.0892 - val_mae: 1.3229 - val_root_mean_squared_error: 2.8442
Epoch 7/50
5/5 [==============================] - 1s 290ms/step - loss: 8.3818 - mse: 8.3818 - mae: 1.3484 - root_mean_squared_error: 2.8951 - val_loss: 7.9940 - val_mse: 7.9940 - val_mae: 1.2998 - val_root_mean_squared_error: 2.8274
Epoch 8/50
5/5 [==============================] - 1s 288ms/step - loss: 8.1663 - mse: 8.1663 - mae: 1.3252 - root_mean_squared_error: 2.8577 - val_loss: 7.8528 - val_mse: 7.8528 - val_mae: 1.2843 - val_root_mean_squared_error: 2.8023
Epoch 9/50
5/5 [==============================] - 1s 290ms/step - loss: 8.0159 - mse: 8.0159 - mae: 1.3085 - root_mean_squared_error: 2.8312 - val_loss: 7.7427 - val_mse: 7.7427 - val_mae: 1.2733 - val_root_mean_squared_error: 2.7826
Epoch 10/50
5/5 [==============================] - 1s 296ms/step - loss: 7.9388 - mse: 7.9388 - mae: 1.2966 - root_mean_squared_error: 2.8176 - val_loss: 7.7006 - val_mse: 7.7006 - val_mae: 1.2597 - val_root_mean_squared_error: 2.7750
Epoch 11/50
...
Epoch 50/50
5/5 [==============================] - 1s 287ms/step - loss: 6.7198 - mse: 6.7198 - mae: 1.1110 - root_mean_squared_error: 2.5923 - val_loss: 7.0474 - val_mse: 7.0474 - val_mae: 1.1436 - val_root_mean_squared_error: 2.6547
```

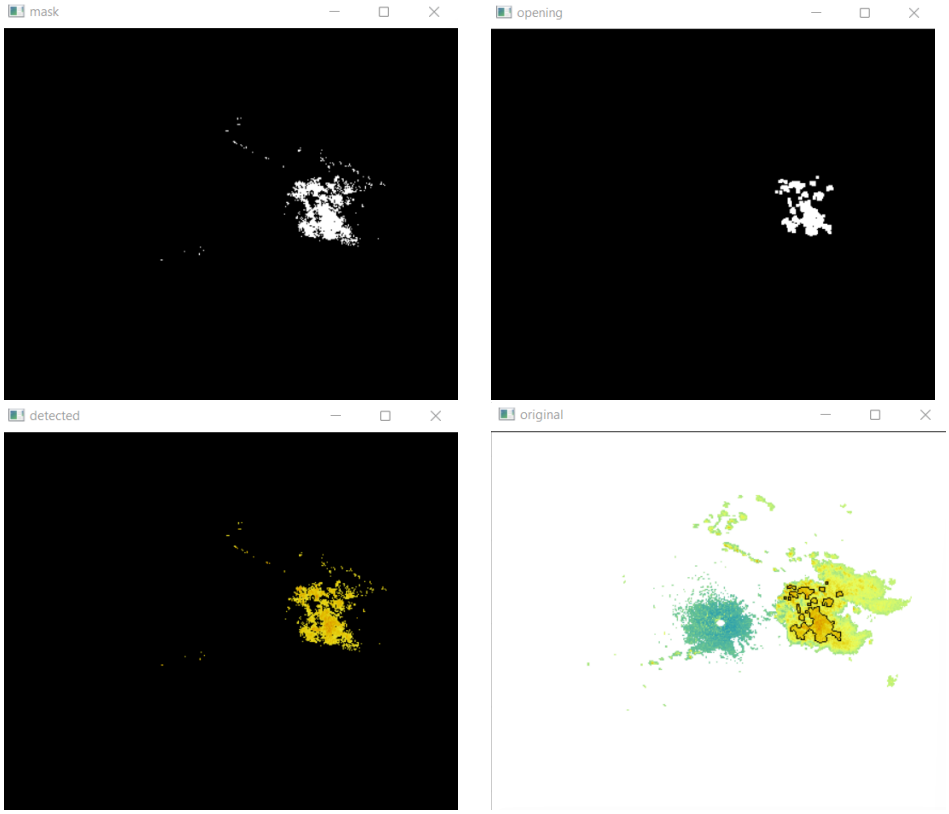


6 Supplementary Work

We tried to classify the precipitation data using radar reflectivity weather maps, the classes being whether the data points offer large-scale or small-scale rainfall over the radar coverage area.

The threshold value was set to be 400, and if the contour area over the weather map is greater than 400 then it is labelled as large-scale, otherwise as small-scale.

Contour Area was calculated as follows:




```
76/76 [==============================] - 5943s 78s/step - loss: 595.5771 - accuracy: 0.8774 - val_loss: 0.1254 - val_accuracy: 0.9692
Epoch 2/5
76/76 [==============================] - 4804s 63s/step - loss: 0.0401 - accuracy: 0.9949
Epoch 3/5
76/76 [==============================] - 4926s 65s/step - loss: 0.0337 - accuracy: 0.9952
Epoch 4/5
76/76 [==============================] - 4721s 62s/step - loss: 0.0127 - accuracy: 0.9987
Epoch 5/5
76/76 [==============================] - 4792s 63s/step - loss: 0.0315 - accuracy: 0.9958
```

7 Experimental Result Comparisons

The Base CNN trained over Miami region results in a MSE of 14.634 which is acceptable given the total range of valid values being [-32.0, 96.0] and trained only with 3 months of data (weather nowcasting typically requires years of data).
Simple, Fine Tuned Transfer Learning Model like CNN-FT doesn't reduce the loss and the reason could be because Transfer Learning Model needs solid pre-trained weights.
However, adding new layers (2 GRU layers) introduces a memory element into picture making the CNN-GRU Nowcasting Model more problem specific, thus resulting in a MSE of 6.814.

Evaluation Metrics/Model	Base CNN	CNN-FT	CNN-GRU
Mean Squared Error	14.634	13.728	6.814
Root Mean Squared Error	2.891	2.764	1.556
Mean Absolute Error	1.423	1.495	1.124
R2 Score	0.316	0.316	0.317

8 Challenges Faced

One of the major challenges faced while executing the project was that the Neural Network Model was very computationally expensive leading to hours of training time even while using a Graphics Processing Unit (GPU). Since the data set used occupied high storage and CNN-GRU model required step-wise processing approach, the training process halted due to RAM overloading.

9 Research Contributions and Future Work

The base CNN model was the Benchmark Model[1]. The CNN-FT model was tested and produced acceptable results. However, the CNN-GRU model that we built definetly resulted in a highly improvised performance due to its short-term forecasting (prediction using a memory concept) characteristic, which is great for NLP, Weather Forecasts or any Time-series application.

Future work should focus on operational implementation of the nowcasting models, especially on how to the real-time observations can be incorporated into the learning schemes. Performance wise, dividing the data point into smaller windows and batch processing those individual frames using memory based deep learning techniques to finally provide a combined result could lead to better outputs.

10 Conclusion

Conventional Nowcastings rely on extrapolation of data and multisource weather data availability, and the nowcasting performance is often hindered by underlying assumptions in the physical conditions that may not be sufficient enough to represent the rapidly varying atmospheric state. Deep learning is expected to enhance radar nowcasting through extracting the complex spatial–temporal features of precipitation and it is proved to be effective for regional applications. As Gated Recurrent Unit can be used to improve the memory capacity of a neural network and as well as provide the ease of training a model the CNN-GRU model provides the most accurate nowcasting result over all the other models.

11 References

1. Advancing Radar Nowcasting Through Deep Transfer Learning

2. A Machine Learning Nowcasting Method based on Real-time Reanalysis Data

3. Deep Learning for Precipitation Nowcasting: A Benchmark and A New Model - RNN, LSTM, Convolutional GRU and Trajectory GRU

4. Py-ART Radar object and indexing

5. Dynamic Reflectivity Plots-SciPy201t-OpenAccessRadar