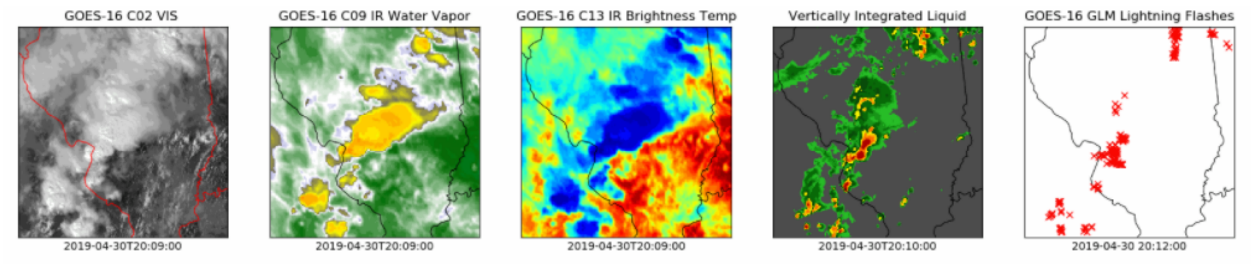
Storm EVent ImageRy (SEVIR) - Transfer Learning with respect to Nowcasting and Synthetic Weather Radar Data Generation

Summary	In this codelab, we present to you how to implement transfer learning on the SEVIR data to generate Synthetic Weather Radar Data for training and Nowcasting Analysis.
URL	sevir-nowcasting-synrad-bdsia
Course	Big Data Systems & Intelligence Analysis
Environment	Google Cloud Storage, Google Colab (Python 3.7)
Status	Published
Feedback Link	
Author	Vachana Belgavi, Akshata Nangappa, Akshay Bhosale
Author LDAP	
Analytics Account	

Introduction

Storm EVent ImageRy (SEVIR)

meteorological satellite imagery weather


GOES-16 C02 VIS
GOES-16 C09 IR Water Vapor
GOES-16 C13 IR Brightness Temp
Vertically Integrated Liquid
GOES-16 GLM Lightning Flashes
2019-04-30T20:09:00
2019-04-30T20:09:00
2019-04-30T20:09:00
2019-04-30T20:10:00
2019-04-30 20:12:00

Last Updated: 2022-02-10

What is the SEVIR Dataset?

Maintained and curated by National Oceanic And Atmospheric Administration's National Centers for Environmental Information, the dataset currently holds data from January 1950 to October 2021. SEVIR is a dataset of approximately 15,000 demographically and temporally aligned image sequences from the GOES-16 satellite and NextGen radar mosaics. One such sequence is shown above. Each sequence, or "event" in SEVIR contains a 384 km x 384 km regional images that span for 4 hours.

SEVIR FILE DETAILS:

The total size of the entire SEVIR dataset is almost close to 1TB considering the fact that it contains images fo 384km x 384km for every 4 hours for years.
There are a total of 107 files in HDF5 format containing imagery:

- Files separated by image type and date range for easier access.
- File sizes ranging between 1GB and 20 GB

There are 4 types of images available in the dataset, namely:

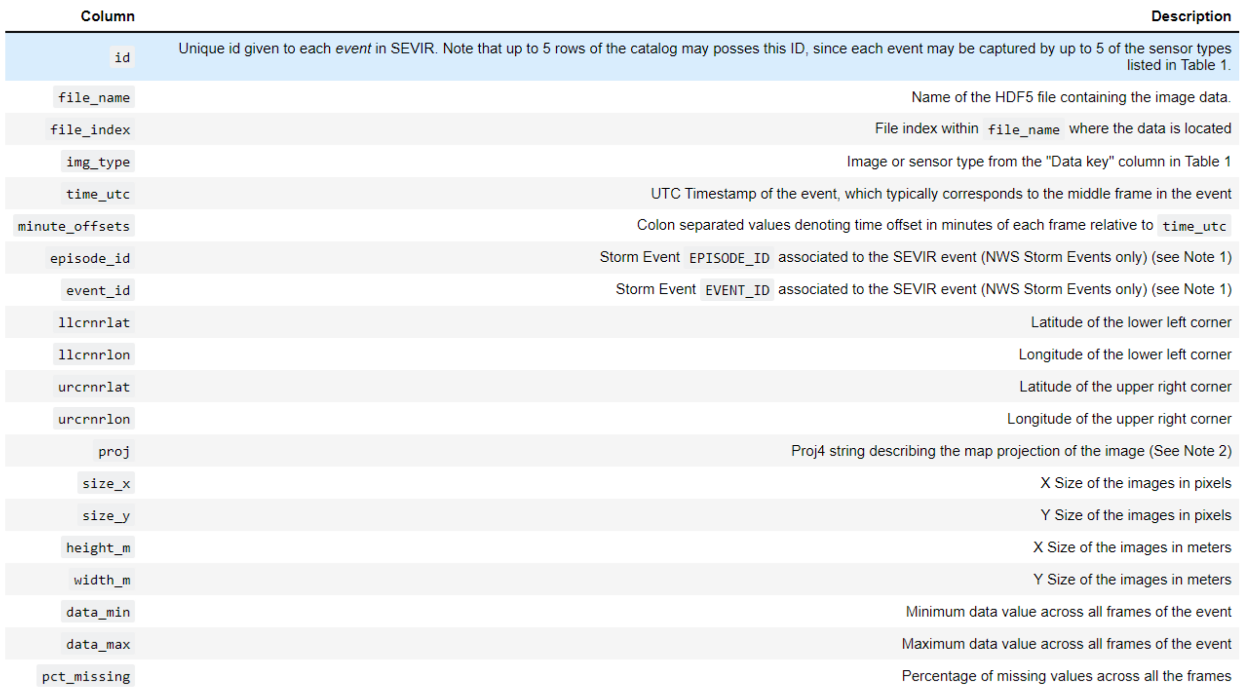
Sensor	Data key	Description	Spatial Resolution	Patch Size
GOES-16 C02 0.64	vis	Visible Satellite	0.5 km	768x768

		Imagery		
GOES-16 C09 6.9	ir069	Infrared Satellite Imagery - Water vapor	2 km	192x192
GOES-16 C13 10.7	ir107	Infrared Satellite Imagery-Window	2 km	192x192
Vertically Integrated Liquid (VIL)	vil	NEXRAD radar mosaic of VIL	1 km	384x384

CATALOG.csv

Since we are focusing on testing the pre-trained models by generating images for the year 2018 and 2019 our catalog file contains the metadata specific to those years, for all events, including:

- Unique EVENT_ID for each event
- Timestamp for each image
- Latitude and Longitude coordinates for each slice of image.
- Map projection and image extent in projection coordinates for exact georeferencing
- NOAA's National Centers for Environmental Information (NCEI) Storm Event ID and Episode ID for non-random event cases.



Column	Description
`id`	Unique id given to each *event* in SEVIR. Note that up to 5 rows of the catalog may posses this ID, since each event may be captured by up to 5 of the sensor types listed in Table 1.
`file_name`	Name of the HDF5 file containing the image data.
`file_index`	File index within `file_name` where the data is located
`img_type`	Image or sensor type from the "Data key" column in Table 1
`time_utc`	UTC Timestamp of the event, which typically corresponds to the middle frame in the event
`minute_offsets`	Colon separated values denoting time offset in minutes of each frame relative to `time_utc`
`episode_id`	Storm Event `EPISODE_ID` associated to the SEVIR event (NWS Storm Events only) (see Note 1)
`event_id`	Storm Event `EVENT_ID` associated to the SEVIR event (NWS Storm Events only) (see Note 1)
`llcrnrlat`	Latitude of the lower left corner
`llcrnrlon`	Longitude of the lower left corner
`urcrnrlat`	Latitude of the upper right corner
`urcrnrlon`	Longitude of the upper right corner
`proj`	Proj4 string describing the map projection of the image (See Note 2)
`size_x`	X Size of the images in pixels
`size_y`	Y Size of the images in pixels
`height_m`	X Size of the images in meters
`width_m`	Y Size of the images in meters
`data_min`	Minimum data value across all frames of the event
`data_max`	Maximum data value across all frames of the event
`pct_missing`	Percentage of missing values across all the frames

Table 2: SEVIR Catalog columns

Applications & Use Cases:

These images and the entire dataset serves as a motivation for Machine Learning & AI researchers to study and improve algorithms to accurately predict and forecast the weather and storm events to avoid the fatalities caused by these natural events. There are several use-cases for a excellent performing model, to name a few:

- Forecasting for the US Government
- Aviation Industry
- Marine Industry
- NOAA(National Oceanic And Atmospheric Administration)
- Weather News / Media issued Warnings
- Energy Sector
- Agricultural Industry
- Banking & Insurance Industry
- Consumer Market Applications

Methodology:

Duration: 2:00

Well everyone by now is familiar with Google Colab's environment and benefits for Machine Learning and AI, which is running Python 3.7 enabling us to import libraries like h5py for reading the H5 files, matplotlib for plotting intuitive graphs and tensorflow for Transfer Learning.

Transfer Learning:

In brief, transfer learning is storing models trained on a particular data and using them elsewhere on new data but related data. To put it in layman's terminology, we use the knowledge acquired by training on a particular domains data and apply it in the same domains different datasets. Since training the model with apt knowledge of both weather and statistics (evaluate model performance) is a challenge in itself, followed by the next obstacle that is computational power we implement transfer learning where we refer and use MIT's trained Neural Network model for generating images for our desired years and check their accuracy against the actual images.

Artificial Neural Networks:

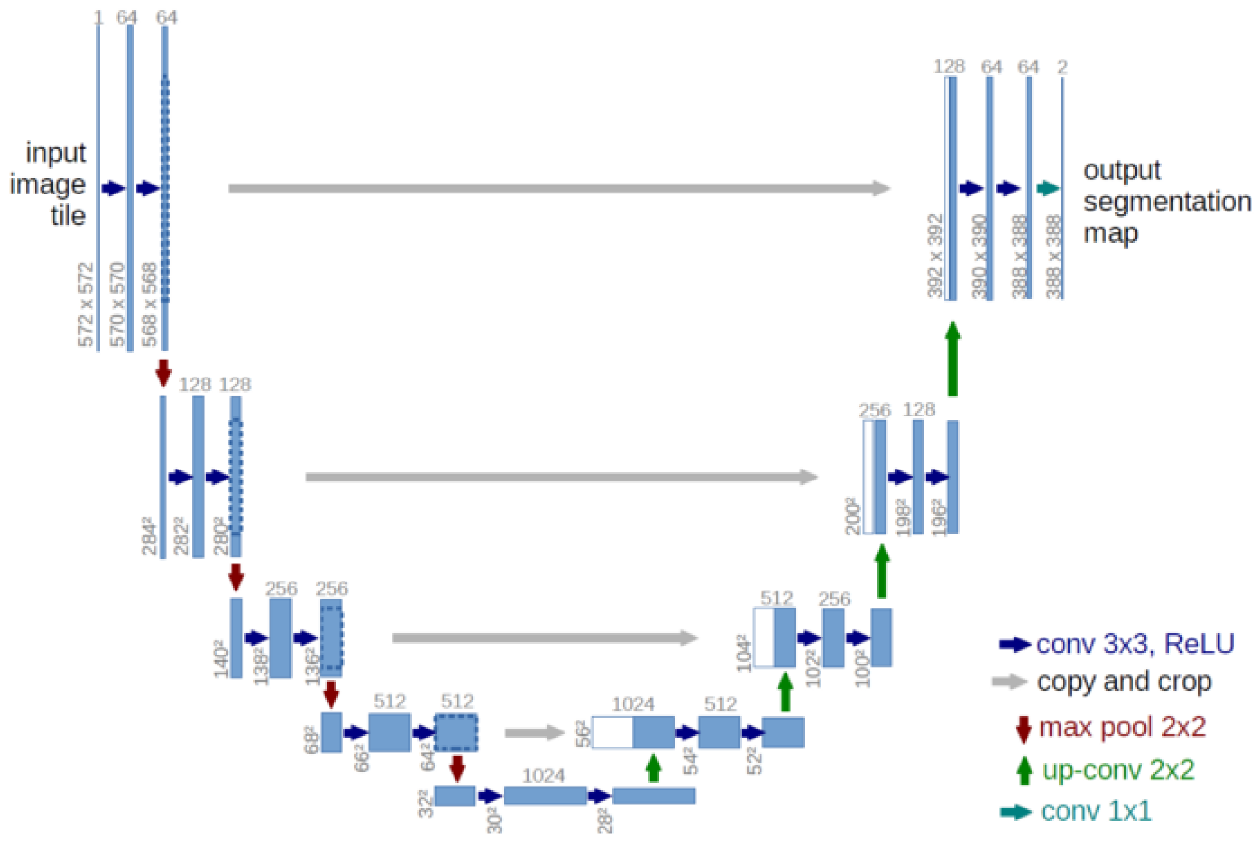
The kickstarter of AI, Artificial Neural Networks are a series of algorithms that generate intelligence by finding relationships between data points. Very similar to the neural networks of the brain, these ANN train and learn through a series of forward and backward propagation, where the only exposed layers to us are the input layer and the output layer. They are a series of hidden layers that generate random weights, perform feature extraction, calculate errors and reduce these errors to the minimal point before giving us the output. The geniuses behind the trained models have implemented very sophisticated Neural Network models, performing trial and error on their loss function and accuracy before finding the best suited model and saving it for future use by others.
The Neural Network Algorithms used in these application of Nowcasting and Synthetic Weather Radar Data generation are as follows:

- cGAN: Conditional Generative Adversarial Network.
    - UNet : A symmetric U-shaped Fully Convolutional Network Architecture used along with GAN for generator and discriminator functionality.
- VGG16 : Visual Geometry Group's 16 Layered Convolutional Neural Network.

cGAN - Conditional Generative Adversarial Network & U-Net:

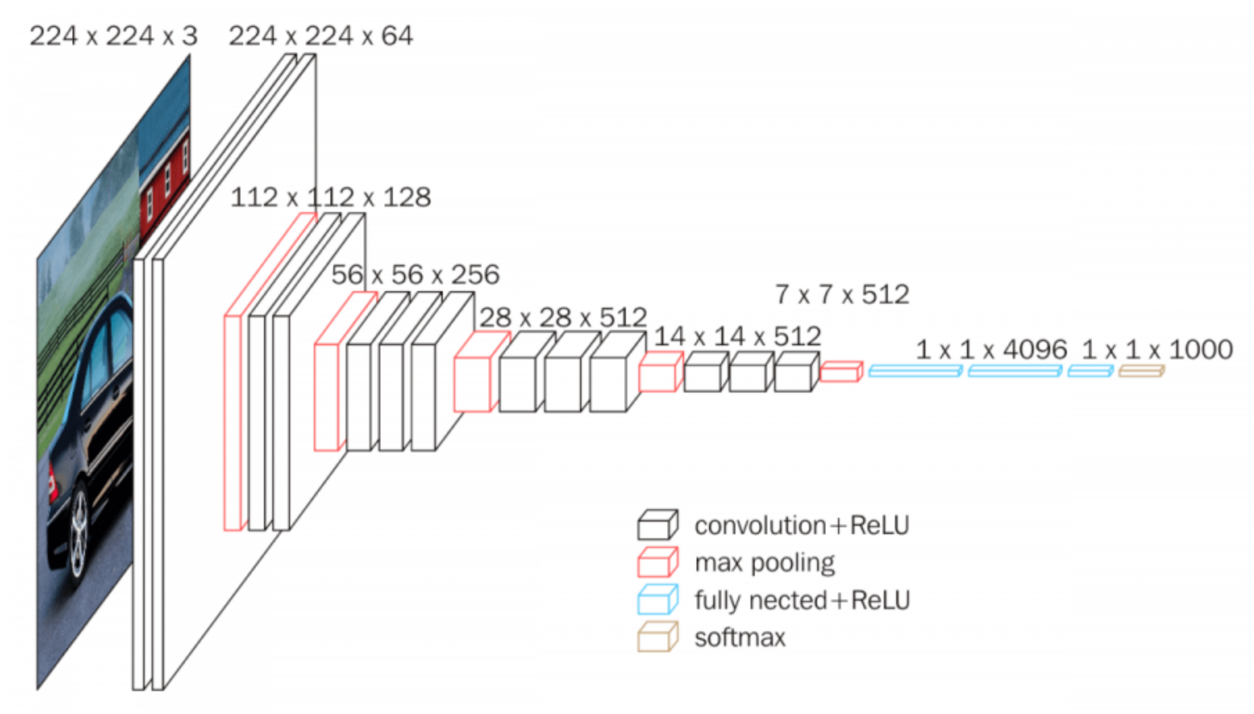
To begin with GAN is an unsupervised learning model for generative modeling where it discovers and learns patterns in the inputs and hence is used to generate or output new data that has been drawn as an inference of the input data. GAN uses bth generator and discriminator functionalities to generate new samples and discriminate them by classification of whether the generated samples are real or fake. The generator and discriminator models train together until the discriminator classifies a generated sample as real.

In this case we the geniuses at MIT have used U-Net as both generator and discriminator considering U-Net's advantage of being a Fully Convolutional Network for pixel segmentation in bio-medical fields. Since the network architecture of U-Net follows both contracting and expanding paths in a U-shape they can be used for predictions using upsampling and classification while contracting since feature information is increased thereby providing pixel clarity.



VGG16 - Visual geometry Groups 16 Layered Network:

A development over the ImageNet, VGG16 comprises of a combination of 16 layers of inferential learning using convolutional neural networks with ReLU, max-pooling layers, fully connected convolutional networks and soft-max layer at the end. Each comes with a benefit of its own when it comes to image segmentation.


224 x 224 x 3
224 x 224 x 64
112 x 112 x 128
56 x 56 x 256
28 x 28 x 512
14 x 14 x 512
7 x 7 x 512
1 x 1 x 4096
1 x 1 x 1000
convolution+ReLU
max pooling
fully nected+ReLU
softmax

Metrics:

Duration: 1:00

Loss Functions:

The loss functions used in modeling by the MIT geniuses are as following:

1. Reconstruction Loss:

   L_{MSE} or L_{MAE}
2. VGG16 Content Loss:

   $$L_{content}(Y, \hat{Y}; \ell) = \|\phi^{\ell}(Y) - \phi^{\ell}(\hat{Y})\|_2$$
3. VGG16 Style Loss:

   $$L_{style}(Y, \hat{Y}; \mathbf{w}) = \sum_{\ell} w^{\ell} \|G^{\ell}(Y) - G^{\ell}(\hat{Y})\|_2$$
4. Conditional GAN Loss:

   $$L_{cGAN}(X, Y, \hat{Y}) = \mathbb{E}_{x,y}\left[D(x, y)\right] + \mathbb{E}_{x}\left[1 - D(x, G(x))\right]$$

Evaluation & Confusion Matrix:

HITS
Prediction = 1
Truth = 1
	Misses:
	Prediction = 0
	Truth = 1

False Alarm
Prediction = 1
Truth = 0
	Correct Rejection:
	Prediction = 0
	Truth = 0

Probability of Detection (POD): #hits / (#hits + #misses)

Success Ratio (SUCR): #hits / (#hits + #false alarm)

Critical Success Index (CSI): #hits / (#hits + #misses + #false alarm)

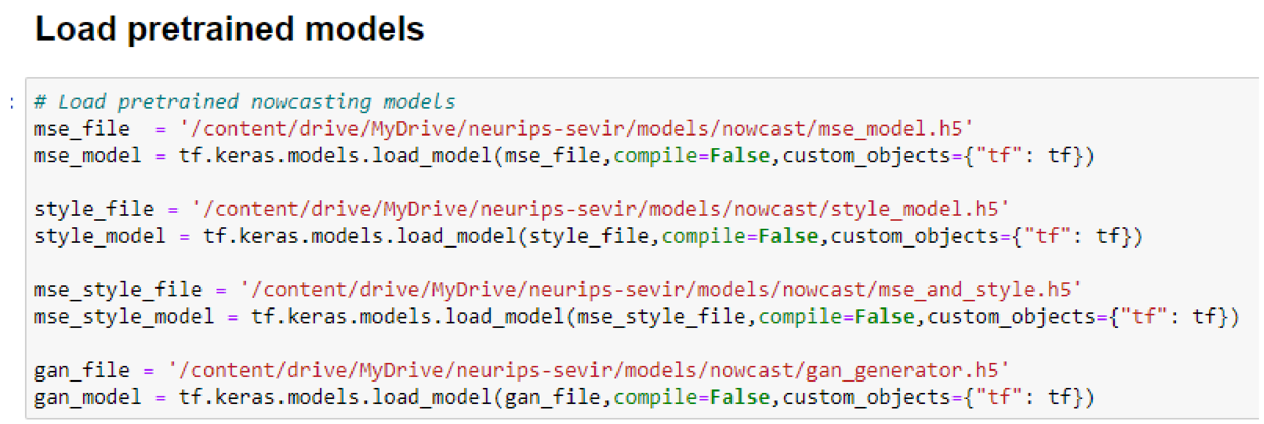
BIAS = (#hits + #false alarm) / (#hits + #misses)

Nowcasting:

Duration: 1:00

Transfer Learning - Loading the Pre-trained Models:

After carefully following the steps from previous codelabs to mount the dataset on our google bucket and extract on the files required for our generative modeling, we shall go ahead and import the necessary libraries in the same notebook, followed by loading the individual trained models, each differentiated by the loss function used in its training.



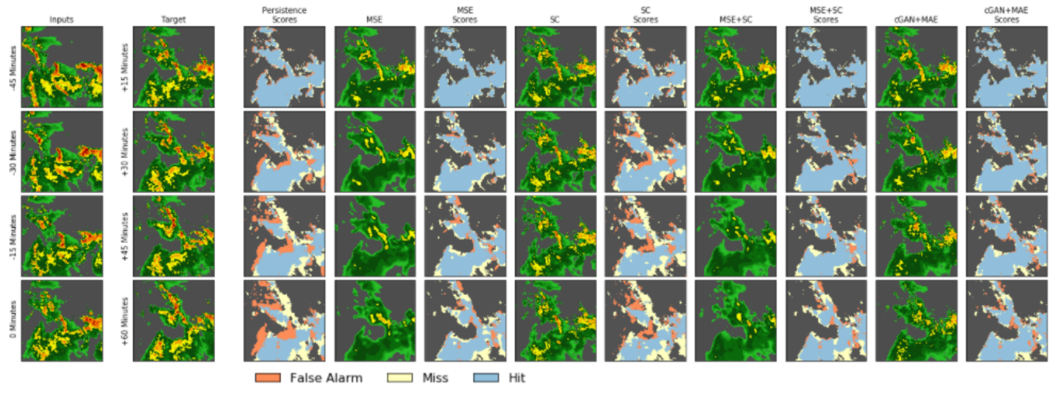
Load pretrained models

```
# Load pretrained nowcasting models
mse_file  = '/content/drive/MyDrive/neurips-sevir/models/nowcast/mse_model.h5'
mse_model = tf.keras.models.load_model(mse_file,compile=False,custom_objects={"tf": tf})

style_file = '/content/drive/MyDrive/neurips-sevir/models/nowcast/style_model.h5'
style_model = tf.keras.models.load_model(style_file,compile=False,custom_objects={"tf": tf})

mse_style_file = '/content/drive/MyDrive/neurips-sevir/models/nowcast/mse_and_style.h5'
mse_style_model = tf.keras.models.load_model(mse_style_file,compile=False,custom_objects={"tf": tf})

gan_file = '/content/drive/MyDrive/neurips-sevir/models/nowcast/gan_generator.h5'
gan_model = tf.keras.models.load_model(gan_file,compile=False,custom_objects={"tf": tf})
```

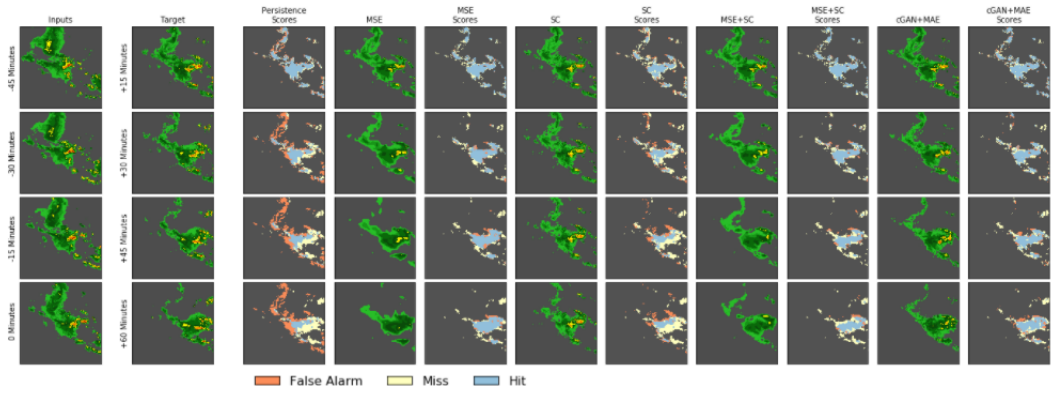
Well, we are aware that Nowcasting is a prediction task of generating weather forecasts like radar echoes, precipitation, cloud coverage for the above mentioned applications using meteorological knowledge.
The models here take input of 13 VIL images, each sampled at every 5 minutes, train and generate the next 12 images in the sequence for the following hour.
Since MSE as a loss function was not clear enough they tried VGG16 with losses of individual Style and Content to improve texture details while failing to capture motion, which is an important aspect considering the cloud movements and their respective impact. The third model had a loss function that was a combination of the previous two, improving both motion tracking and texture before moving onto the U-Net powered cGAN.

Plotting Test Cases:

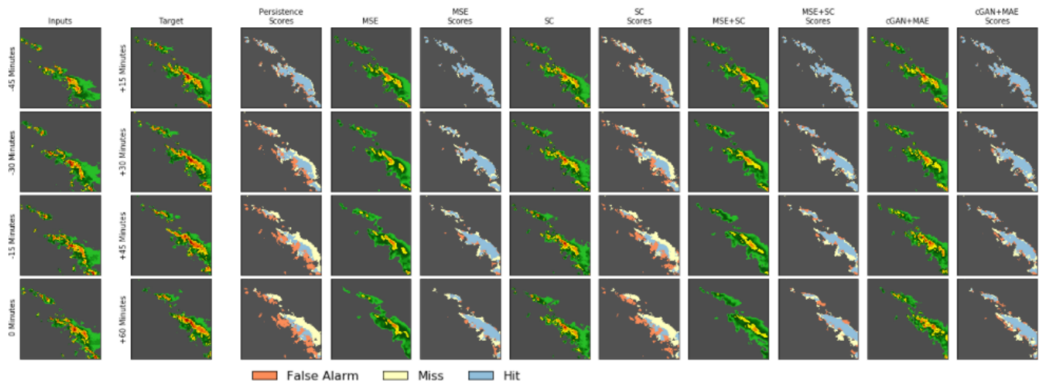
```
idx=25
fig,ax = plt.subplots(4,13,figsize=(24,8), gridspec_kw={'width_ratios': [1,.2,1,.2,1,1,1,1,1,1,1,1,1]})
visualize_result([mse_model,style_model,mse_style_model,gan_model],x_test,y_test,idx,ax,labels=['MSE','SC','MSE+SC','cGAN+MAE'])
```



```
idx=45
fig,ax = plt.subplots(4,13,figsize=(24,8), gridspec_kw={'width_ratios': [1,.2,1,.2,1,1,1,1,1,1,1,1,1]})
visualize_result([mse_model,style_model,mse_style_model,gan_model],x_test,y_test,idx,ax,labels=['MSE','SC','MSE+SC','cGAN+MAE'])
```



```
idx=32
fig,ax = plt.subplots(4,13,figsize=(24,8), gridspec_kw={'width_ratios': [1,.2,1,.2,1,1,1,1,1,1,1,1,1]})
visualize_result([mse_model,style_model,mse_style_model,gan_model],x_test,y_test,idx,ax,labels=['MSE','SC','MSE+SC','cGAN+MAE'])
```



Generating Synthetic Weather Radar Data:

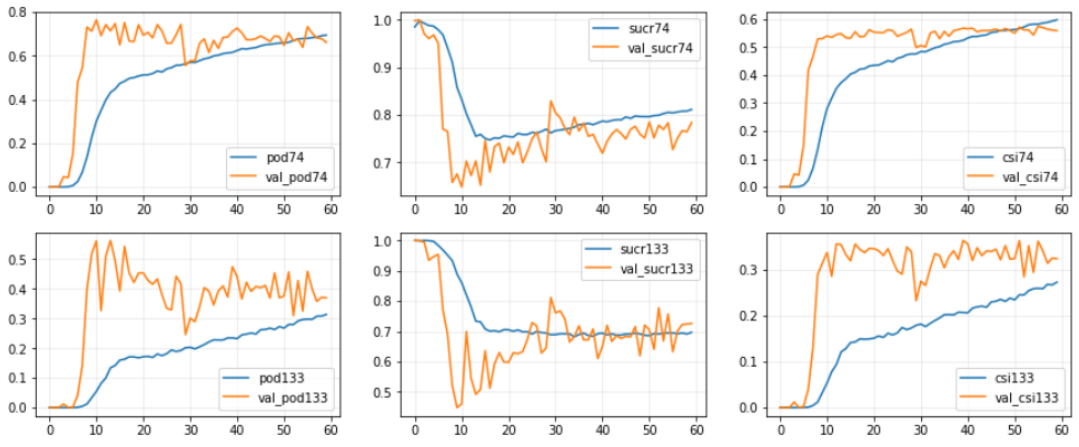
Duration: 1:00

Plotting Metrics:

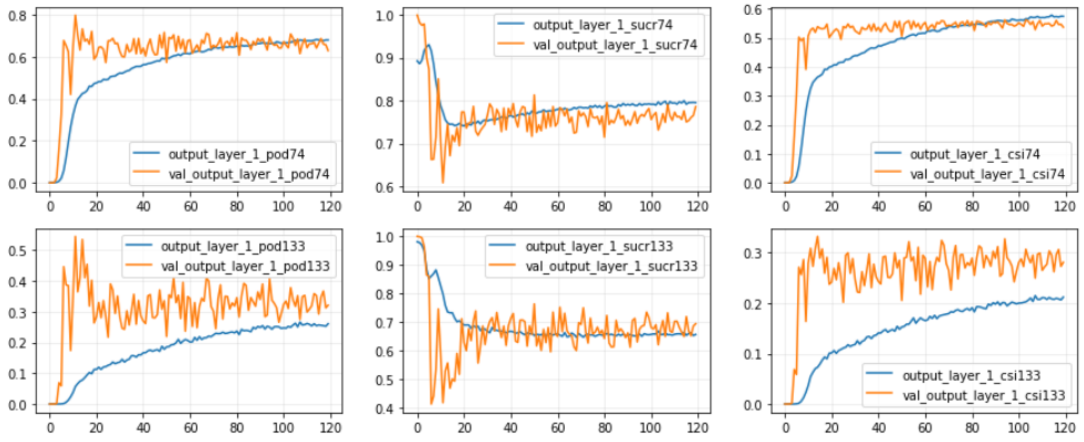
Similar to the Nowcasting approach, we load the required libraries, mount the data to our drive, and an addition is that we load metrics.csv, also made available by the MIT authors, to track the above defined metrics during training. Further we plot the metrics for each different model of neural network, tracking their individual losses:

```
# Read the metrics output during training
log_files= {
    'mse':'/content/drive/MyDrive/neurips-2020-sevir/logs/sample-mse/metrics.csv',
    'mse+vgg':'/content/drive/MyDrive/neurips-2020-sevir/logs/sample-mse-vgg/metrics.csv',
    'gan':'/content/drive/MyDrive/neurips-2020-sevir/logs/sample-gan-mae/metrics.csv'
}
metrics={ k:pd.read_csv(v) for k,v in log_files.items()}
```

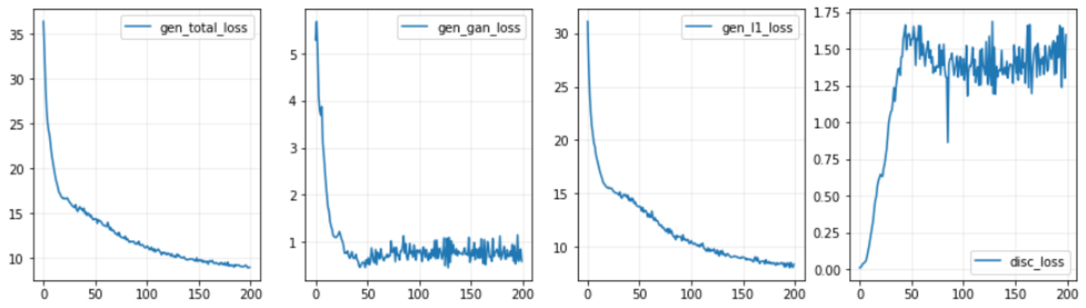
```
# MSE
fig,ax=plt.subplots(2,3,figsize=(15,6))
def plot_metrics_row(df,metric_names,ax):
    for k,m in enumerate(metric_names):
        df[[m,'val_'+m]].plot(ax=ax[k])
        ax[k].grid(True,alpha=.25)

plot_metrics_row(metrics['mse'],['pod74','sucr74','csi74'],ax[0])
plot_metrics_row(metrics['mse'],['pod133','sucr133','csi133'],ax[1])
```



```
# mse+mse
fig,ax=plt.subplots(2,3,figsize=(15,6))
def plot_metrics_row(df,metric_names,ax):
    for k,m in enumerate(metric_names):
        df[[m,'val_'+m]].plot(ax=ax[k])
        ax[k].grid(True,alpha=.25)

plot_metrics_row(metrics['mse+vgg'],['output_layer_1_pod74','output_layer_1_sucr74','output_layer_1_csi74'],ax[0])
plot_metrics_row(metrics['mse+vgg'],['output_layer_1_pod133','output_layer_1_sucr133','output_layer_1_csi133'],ax[1])
```

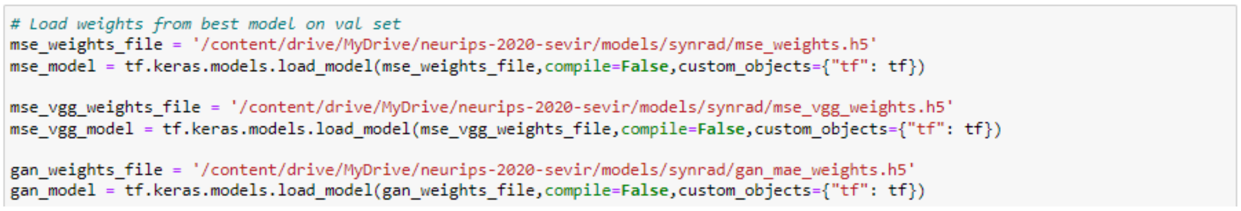


```
# gan
fig,ax=plt.subplots(1,4,figsize=(15,4))
def plot_metrics_row(df,metrics,ax):
    for k,m in enumerate(metrics):
        df[[m]].plot(ax=ax[k])
        ax[k].grid(True,alpha=.25)

plot_metrics_row(metrics['gan'],['gen_total_loss', 'gen_gan_loss', 'gen_l1_loss', 'disc_loss'],ax)
```



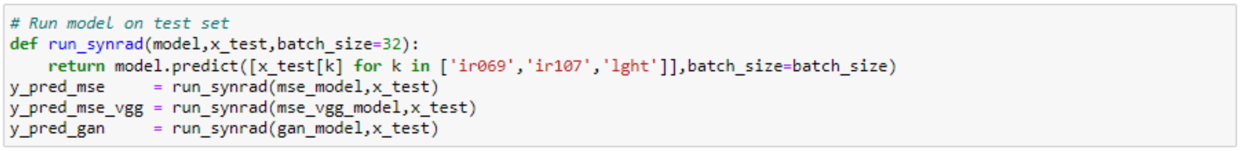
Generating Sample Images & Visualizing Test Cases:

Now we load the models with their respective weights on the validation set, as compiled by the authors and visualize them using a few test cases of our choice.

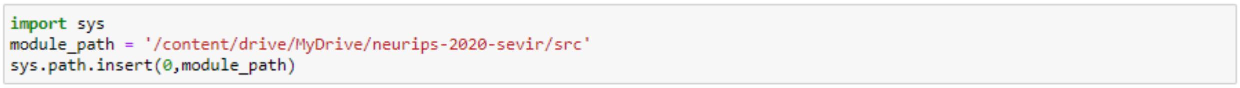


```
# Load weights from best model on val set
mse_weights_file = '/content/drive/MyDrive/neurips-2020-sevir/models/synrad/mse_weights.h5'
mse_model = tf.keras.models.load_model(mse_weights_file,compile=False,custom_objects={"tf": tf})

mse_vgg_weights_file = '/content/drive/MyDrive/neurips-2020-sevir/models/synrad/mse_vgg_weights.h5'
mse_vgg_model = tf.keras.models.load_model(mse_vgg_weights_file,compile=False,custom_objects={"tf": tf})

gan_weights_file = '/content/drive/MyDrive/neurips-2020-sevir/models/synrad/gan_mae_weights.h5'
gan_model = tf.keras.models.load_model(gan_weights_file,compile=False,custom_objects={"tf": tf})
```

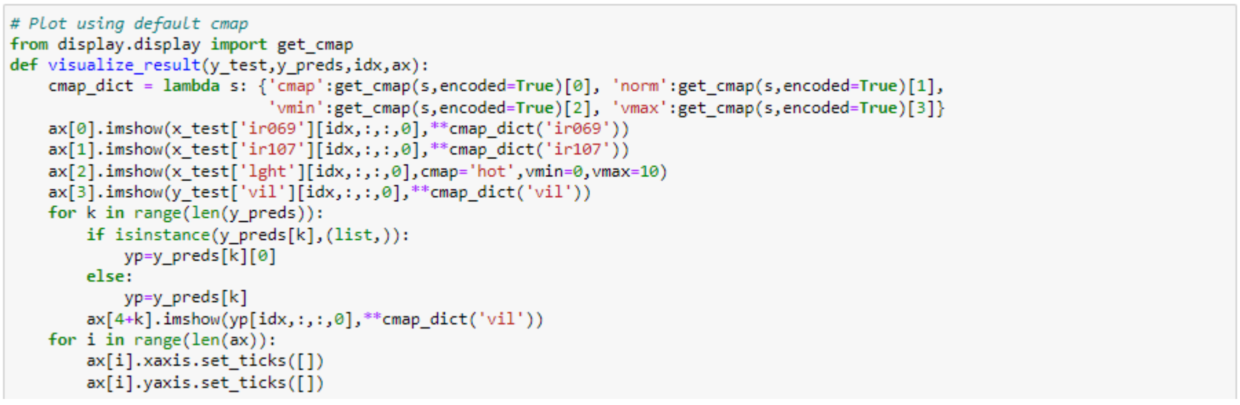
Visualize results on some test samples



```
# Run model on test set
def run_synrad(model,x_test,batch_size=32):
    return model.predict([x_test[k] for k in ['ir069','ir107','lght']],batch_size=batch_size)
y_pred_mse     = run_synrad(mse_model,x_test)
y_pred_mse_vgg = run_synrad(mse_vgg_model,x_test)
y_pred_gan     = run_synrad(gan_model,x_test)
```

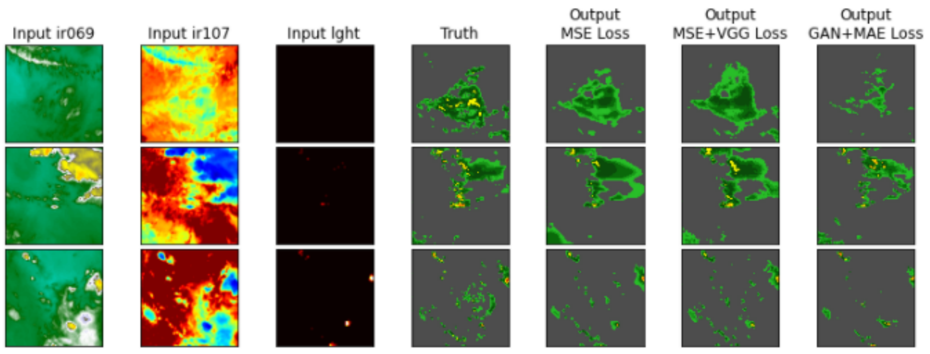


```
import sys
module_path = '/content/drive/MyDrive/neurips-2020-sevir/src'
sys.path.insert(0,module_path)
```



```
# Plot using default cmap
from display.display import get_cmap
def visualize_result(y_test,y_preds,idx,ax):
    cmap_dict = lambda s: {'cmap':get_cmap(s,encoded=True)[0], 'norm':get_cmap(s,encoded=True)[1],
                           'vmin':get_cmap(s,encoded=True)[2], 'vmax':get_cmap(s,encoded=True)[3]}
    ax[0].imshow(x_test['ir069'][idx,:,:,0],**cmap_dict('ir069'))
    ax[1].imshow(x_test['ir107'][idx,:,:,0],**cmap_dict('ir107'))
    ax[2].imshow(x_test['lght'][idx,:,:,0],cmap='hot',vmin=0,vmax=10)
    ax[3].imshow(y_test['vil'][idx,:,:,0],**cmap_dict('vil'))
    for k in range(len(y_preds)):
        if isinstance(y_preds[k],(list,)):
            yp=y_preds[k][0]
        else:
            yp=y_preds[k]
        ax[4+k].imshow(yp[idx,:,:,0],**cmap_dict('vil'))
    for i in range(len(ax)):
        ax[i].xaxis.set_ticks([])
        ax[i].yaxis.set_ticks([])
```

```
test_idx = [123,456,789]
N=len(test_idx)
fig,ax = plt.subplots(N,7,figsize=(12,4))
for k,i in enumerate(test_idx):
    visualize_result(y_test,[y_pred_mse,y_pred_mse_vgg,y_pred_gan], i, ax[k] )

ax[0][0].set_title('Input ir069')
ax[0][1].set_title('Input ir107')
ax[0][2].set_title('Input lght')
ax[0][3].set_title('Truth')
ax[0][4].set_title('Output\nMSE Loss')
ax[0][5].set_title('Output\nMSE+VGG Loss')
ax[0][6].set_title('Output\nGAN+MAE Loss')
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.05,
                    wspace=0.35)
```



References:

Duration: 0:00

Reference docs

- https://proceedings.neurips.cc/paper/2020/file/fa78a16157fed00d7a80515818432169-Paper.pdf
- https://papers.nips.cc/paper/2020/file/fa78a16157fed00d7a80515818432169-Supplemental.pdf
- https://sevir.mit.edu/sites/default/files/About_SEVIR.pdf
- . https://github.com/MIT-AI-Accelerator/neurips-2020-sevir