NM391 The Ones n Zeros

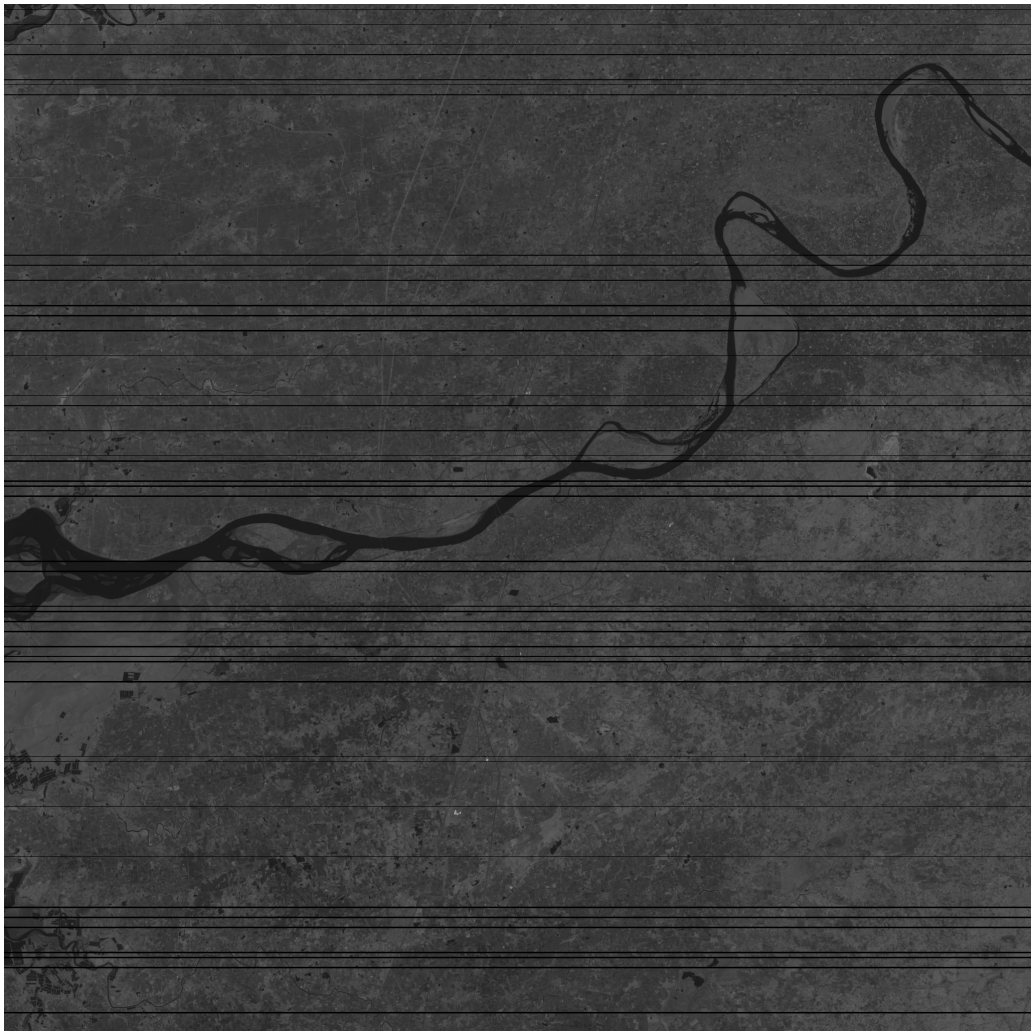
NM391-ISRO Reconstruction of missing data in Satellite Imagery : Short Wave Infra-Red(SWIR) detectors used in satellite imaging cameras suffer from dropouts in pixel and line direction in raw data. Develop software to reconstruct missing parts of a satellite image so that observers are unable to identify regions that have undergone reconstruction.

Smart India Hackathon 2020

Final Evaluation Results :

- The final Reconstructed Image generated by our 16-bit GANs Reconstruction Algorithm with 359 epochs and segmented processing is as follows :

- Input Image



- Final Output

[CLICK ON IMAGE TO VIEW FULL RESOLUTION TIFF/PNG]

USAGE INSTRUCTIONS

Download Link : MIRROR - mega.nz

- Download Weights from the given Mirror and Place them in `NM391_The-Ones-n-Zeros/gans_reconstruction/logs`
- Thus, a new Directory will be created which enlists the files as follows :

```
NM391_The-Ones-n-Zeros/
└── gans_reconstruction
    └── logs
        └── model_weights
            ├── checkpoint
            ├── here.txt
            ├── snap-122000.data-00000-of-00001
            ├── snap-122000.index
            └── snap-122000.meta
```

```
3 directories, 5 files
```

- In case you have trouble with the Redirect, Direct Link : https://mega.nz/folder/3t0wTaqS#XV5PhaO-eNDGJoDWxqGDHw

Requirements

- The following modules are required in order to ensure proper functioning of the Program :
  - Python 3.× : https://python.org
  - tensorflow v1.5.0 through v1.7.0
  - tensorflow toolkit neuralgym : `pip install git+https://github.com/JiahuiYu/neuralgym`

Execution

- ***All the source Files are Stored under*** `gans_reconstruction` ***for easier access. The file tree is as shown below :***

```
gans_reconstruction
├── __pycache__
│   ├── reconstruction_model.cpython-37.pyc
│   ├── test2.cpython-37.pyc
│   └── utils.cpython-37.pyc
├── batch_test.py
├── data_flist
│   ├── train_shuffled.flist
│   └── validation_shuffled.flist
├── driver.py
├── flist.py
├── guided_batch_test.py
├── logs
│   └── model_weights
│       └── here.txt
├── neuralgym
│   ├── __init__.py
│   ├── __pycache__
│   │   └── __init__.cpython-37.pyc
│   ├── callbacks
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-37.pyc
│   │   │   ├── callbacks.cpython-37.pyc
```

```
│   │   │   ├── hyper_param_scheduler.cpython-37.pyc
│   │   │   ├── model_restorer.cpython-37.pyc
│   │   │   ├── model_saver.cpython-37.pyc
│   │   │   ├── model_sync.cpython-37.pyc
│   │   │   ├── npz_model_loader.cpython-37.pyc
│   │   │   ├── secondary_multigpu_trainer.cpython-37.pyc
│   │   │   ├── secondary_trainer.cpython-37.pyc
│   │   │   ├── summary_writer.cpython-37.pyc
│   │   │   └── weights_viewer.cpython-37.pyc
│   │   ├── callbacks.py
│   │   ├── hyper_param_scheduler.py
│   │   ├── model_restorer.py
│   │   ├── model_saver.py
│   │   ├── model_sync.py
│   │   ├── npz_model_loader.py
│   │   ├── secondary_multigpu_trainer.py
│   │   ├── secondary_trainer.py
│   │   ├── summary_writer.py
│   │   └── weights_viewer.py
│   ├── data
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-37.pyc
│   │   │   ├── data_from_fnames.cpython-37.pyc
│   │   │   ├── dataset.cpython-37.pyc
│   │   │   └── feeding_queue_runner.cpython-37.pyc
│   │   ├── data_from_fnames.py
│   │   ├── dataset.py
│   │   └── feeding_queue_runner.py
│   ├── models
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-37.pyc
│   │   │   └── model.cpython-37.pyc
│   │   └── model.py
│   ├── ops
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-37.pyc
│   │   │   ├── gan_ops.cpython-37.pyc
│   │   │   ├── image_ops.cpython-37.pyc
│   │   │   ├── layers.cpython-37.pyc
│   │   │   ├── loss_ops.cpython-37.pyc
│   │   │   ├── summary_ops.cpython-37.pyc
│   │   │   └── train_ops.cpython-37.pyc
│   │   ├── gan_ops.py
│   │   ├── image_ops.py
│   │   ├── layers.py
│   │   ├── loss_ops.py
│   │   ├── summary_ops.py
│   │   └── train_ops.py
```

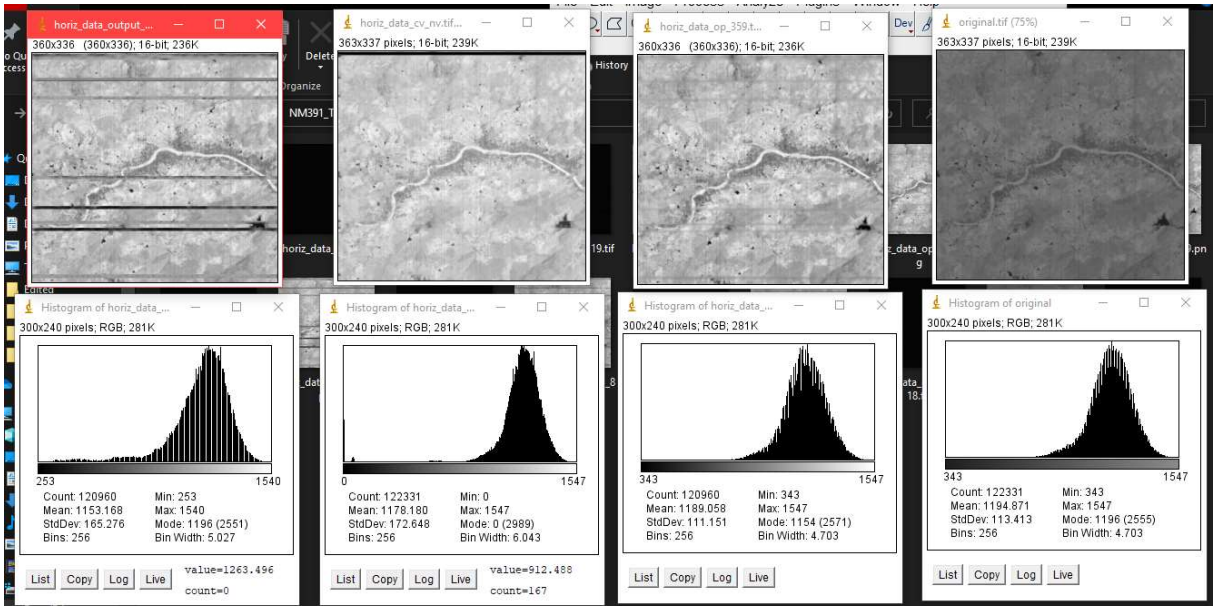
```
│   ├── server
│   │   ├── __init__.py
│   │   └── __pycache__
│   │       └── __init__.cpython-37.pyc
│   ├── train
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-37.pyc
│   │   │   ├── multigpu_trainer.cpython-37.pyc
│   │   │   └── trainer.cpython-37.pyc
│   │   ├── multigpu_trainer.py
│   │   └── trainer.py
│   └── utils
│       ├── __init__.py
│       ├── __pycache__
│       │   ├── __init__.cpython-37.pyc
│       │   ├── config.cpython-37.pyc
│       │   ├── data_utils.cpython-37.pyc
│       │   ├── gpus.cpython-37.pyc
│       │   ├── logger.cpython-37.pyc
│       │   └── tf_utils.cpython-37.pyc
│       ├── config.py
│       ├── data_utils.py
│       ├── gpus.py
│       ├── logger.py
│       └── tf_utils.py
├── reconstruction_model.py
├── sat_reconstruction.yml
├── test.py
├── testing_examples
│   └── example
│       ├── input.tiff
│       ├── original.tiff
│       └── output.tiff
├── train.py
├── training_data
│   └── training
│       ├── training
│       │   └── here.txt
│       └── validation
│           └── here.txt
└── utils.py

26 directories, 90 files
```

- In order to execute the program after placong the desired weights in the correct location, the syntax to run the program is `python driver.py --input <input_file_path> --output <output_file_path>` .

Comparison

This Section aims to provide a direct comparison of our lates model with other Available Solutions and our past algorithms.

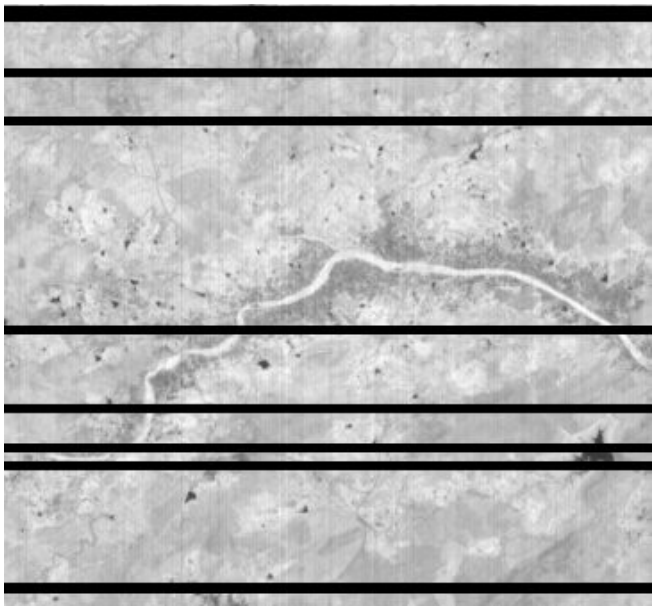


- ***From Left,***
  i. Old Algorithm Using GANS with old 8-bit Data.
  ii. Old Algorithm Using OpenCV.
  iii. **New Algorithm Using GANS with 16-bit Data.**
  iv. Original Image.

Progress of New 16-bit GANS

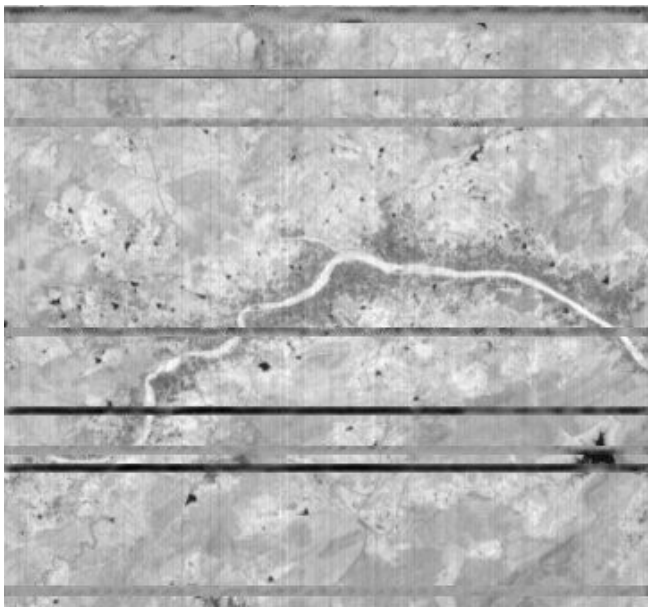
This section aims to track the progress of our neural network plotted in accordance with the epochs developed.

- ***INPUT IMAGE***



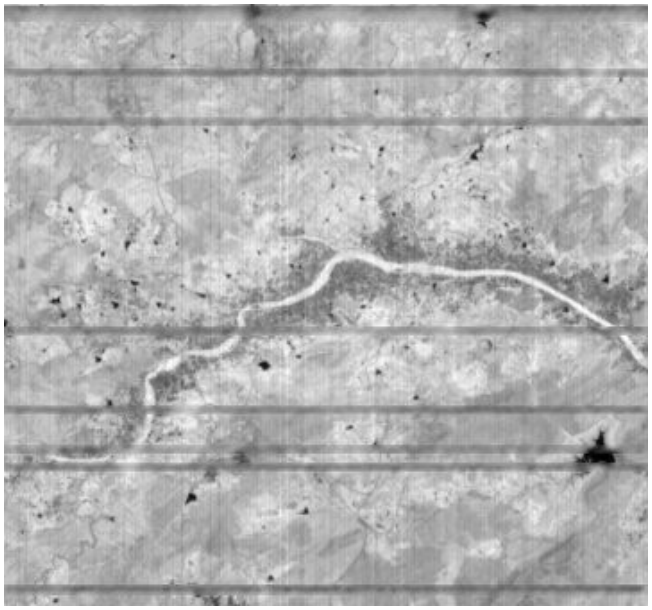
- SSIM with Ground Truth : 0.9314753871768957

- Old 8-bit GANS



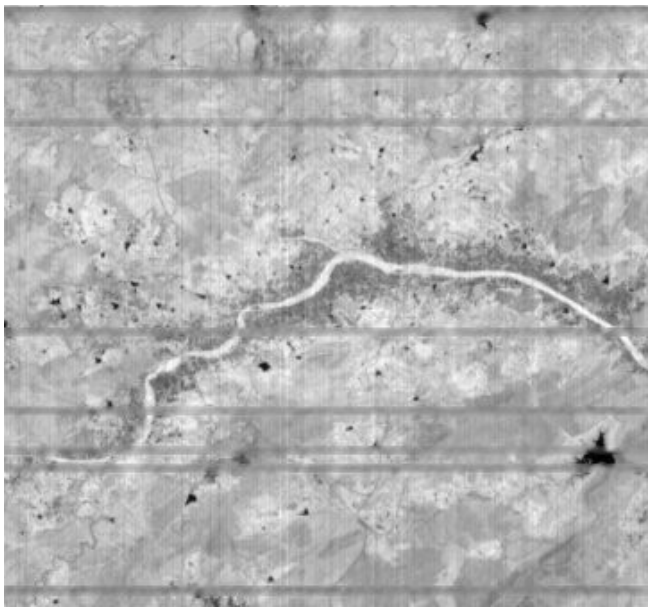
- SSIM with Ground Truth : 0.9942820439040533

- 16-bit GANS - 82 epochs



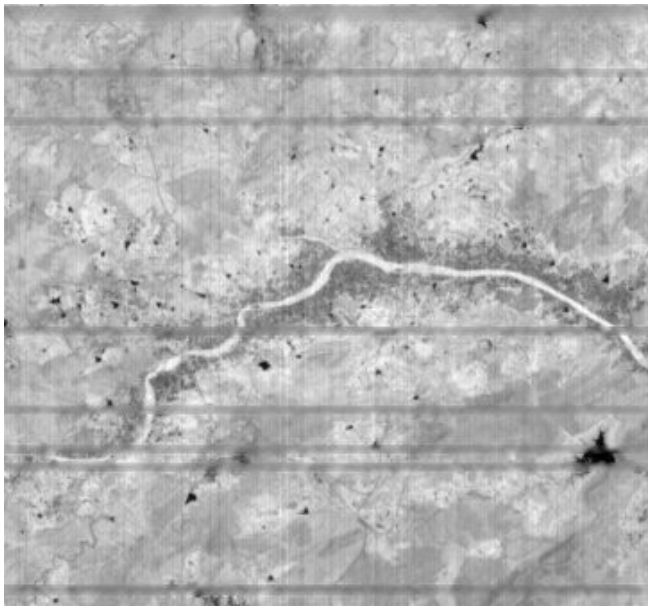
- SSIM with Ground Truth : 0.9986725530035611

- 16-bit GANS - 119 epochs



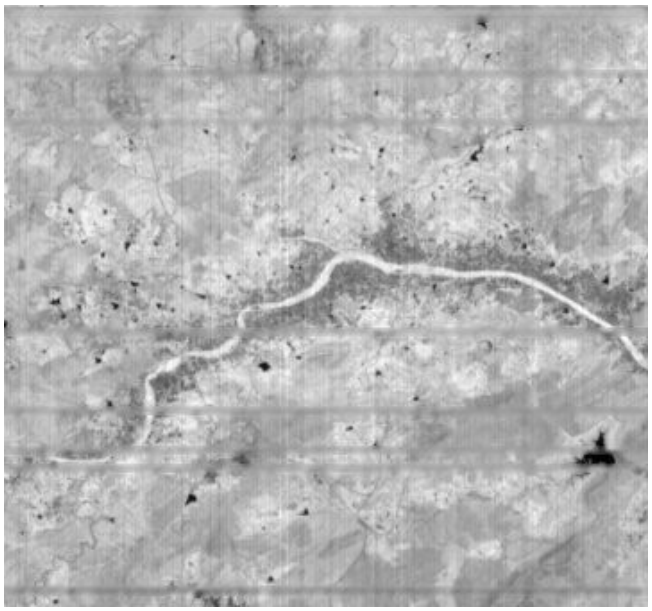
- SSIM with Ground Truth : 0.9993279604107491

- 16-bit GANS - 201 epochs



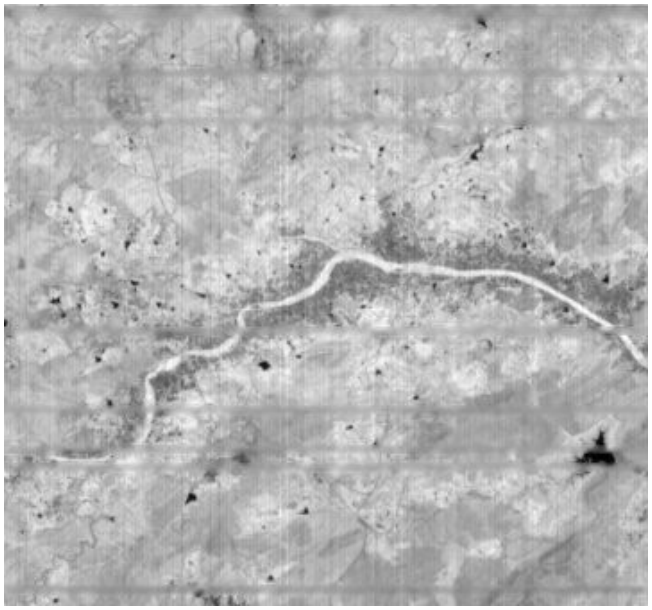
- SSIM with Ground Truth : 0.9993420019395316

- 16-bit GANS - 237 epochs



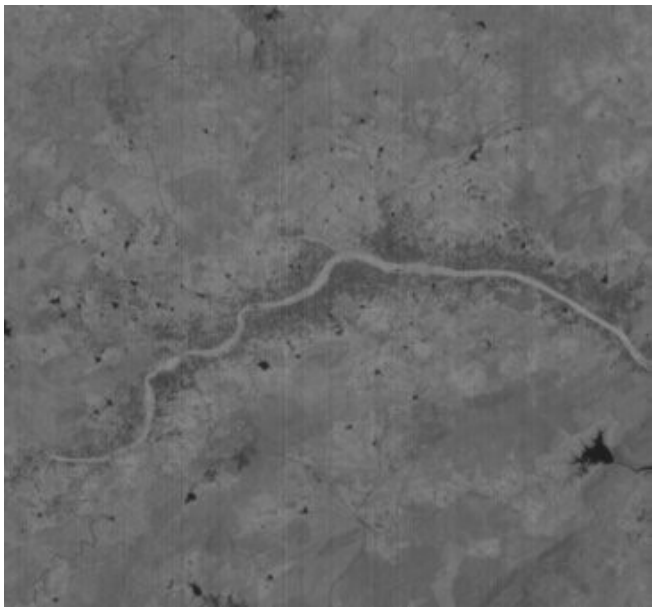
- SSIM with Ground Truth : 0.9996917811413732

- 16-bit GANS - 359 epochs



```
- SSIM with Ground Truth : 0.9996847497074818
```

- Ground Truth



Sample Output

- https://github.com/its-charan-here/NM391_The-Ones-n-Zeros/tree/master/final_output

Satellite Imagery

- Satellite imagery depicts the Earth's surface at various spectral, temporal, radiometric, and increasingly detailed spatial resolutions, as is determined by each collection system's sensing device, and the orbital path of its reconnaissance platform.

- Satellite images are one of the most powerful and important tools used by the meteorologist. They are essentially the eyes in the sky. These images reassure forecasters to the behavior of the atmosphere as they give a clear, concise, and accurate representation of how events are unfolding.

- There are three main types of Satellite Imagery used today :

    - *VISIBLE IMAGERY* : Visible images represent the amount of sunlight being scattered back into space by the clouds, aerosols, atmospheric gases, and the Earth's surface.

    - ***INFRARED IMAGERY[1] : IR or infrared satellite imagery is sort of a temperature map. The weather satellite detects heat energy in the infrared spectrum (infrared energy is invisible to the human eye). Since temperature, in general, decreases with increasing height, high altitudes will appear whiter than low altitudes.***

    - *WATER VAPOUR IMAGERY* : Water vapor satellite pictures indicate how much moisture is present in the upper atmosphere (approximately from 15,000 ft to 30,000 ft). The highest humidities will be the whitest areas while dry regions will be dark. Water vapor imagery is useful for indicating where heavy rain is possible.

[1] For the sake of efficiency, we shall constrain our analysis to INFRARED IMAGERY only, in particular, Short Wave Infrared Imagery or SWIR.

Short Wave Infrared Images (SWIR)

- SWIR refers to non-visible light falling roughly between 1400 and 3000 nanometers (nm) in wavelength. Visible light, on the other hand, typically corresponds to the 400 to 700 nm range. Immediately adjacent to visible light is near infrared, or NIR, within the 700 to 1400 nm range, and SWIR is adjacent to NIR.

- Collecting satellite imagery in SWIR wavelengths has unique benefits, including improved atmospheric transparency and material identification. Because of their chemistries, many materials have specific reflectance and absorption features in the SWIR bands that allow for their characterization from space.

Errors in SWIR

- A digital remotely sensed image is typically composed of picture elements (pixels) having Digital Number (DN) or Brightness Value (BV) located at the intersection of each row and column of each band in imagery. A smaller number indicates low radiance from the area and high number is an indicator of high radiant properties of the area. Raw digital images usually contain distortions or various types of error so they cannot be used directly without proper processing. Sources of these distortions range from variation in the altitude and velocity of the sensor to Earth's rotation and curvature etc. Corrections are thus necessary for such satellite images. Programs to rectify Satellite imagery aim to correct distorted or degraded image data to create a reliable representation of the original scene.

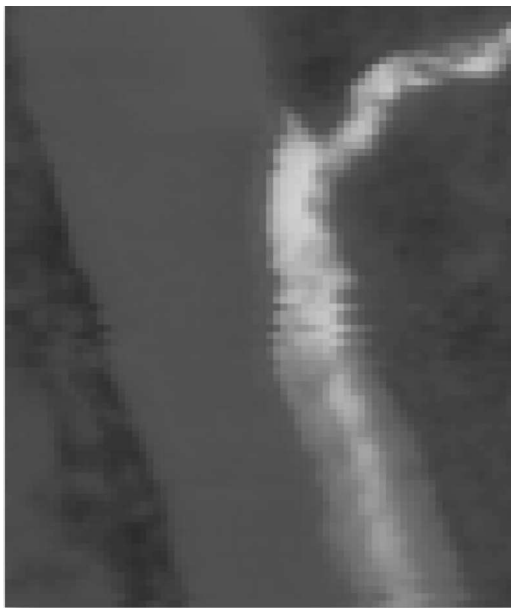
- Broadly Categorised, there are two types of errors, Radiometric and Geometric Errors, for the scope of the project we shall be limited to Radiometric Errors involving missing data.

- Radiometric Errors are caused by inconsistencies in the capturing sensor and atmospheric conditions which lead to incorrect or no brightness values being recorded.

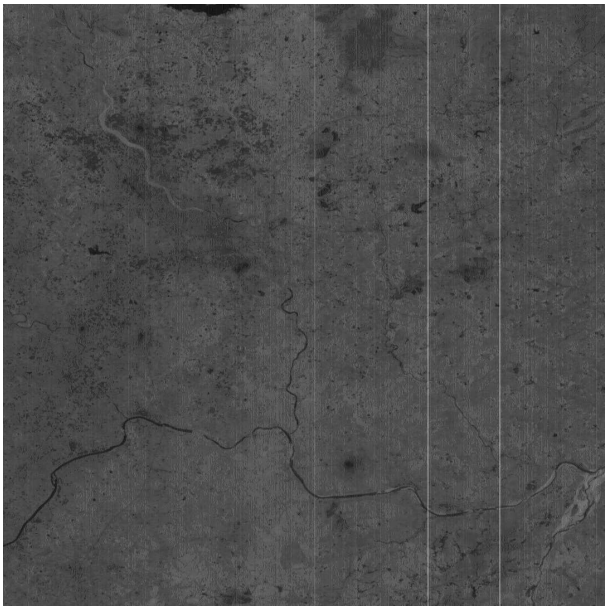
- Such errors include :

    - Random bad pixels (shot noise).

    - Line-start/stop problems.



- ***Line or column drop-outs.*** [2]



- Partial line or column drop-outs.

- Line or column striping.

[2] In order to have a fixed scope to facilitate easier understanding, we shall be constrained to Line & Column Dropouts only for Data Reconstruction.

Reconstruction

- The process of building or forming (something) again after it has been damaged or destroyed is known as reconstruction, in this case we shall refer to reconstructing the missing data in Satellite Images caused by Radiometric Errors.

Proposed Solution

Traditional methods

Traditional methods of rectifying the errors in question are listed as follows :

- Replacement by others Pixels in Vicinity : the brightness value of the missing pixels is replaced by the value of the pixel on immediately preceding or succeeding line.
- Replacement by Averaging : the brightness value of the missing pixels is calculated by taking the mean of the pixels in its proximity.
- In essence, simple mathematical operations are used in order to estimate the missing data in terms of numerical values.

New Technique

The proposed technique we plan to implement aims to utilise the newly unveiled power of Machine Learning and Artificial Intelligence to provide Optimum, Efficient and Accurate rectification methods by reconstructing the missing data using Neural Networks.

Realisation & Implementation of Proposed Solution

Parsing Satellite Imagery

- The first challenge faced by our team was reading the satellite Raster Data and converting it into a Programmatically Modifiable format in order for our Neural Network to be able to Reconstruct the Missing Data.
- In order to accommodate this our first solution was to read the binary data from the raw Raster via native python methods and to convert it into a Numpy Matrix with pixel values indicating the Luminosity of each pixel. Which gave rise to the problem of having unsigned 16 bit integers as the datatype for the image.
- Having 16-bit Images made it difficult for openly Available solutions to be able to read the data, hence we came up.with a conversion process to convert the data to unsigned 8 bit integers.

```
>>> matrix16
array([[1265, 1245, 1274, ..., 1286, 1291, 1220],
       [1297, 1282, 1272, ..., 1265, 1293, 1240],
       [1306, 1302, 1327, ..., 1254, 1311, 1269],
       ...,
       [1181, 1146, 1169, ..., 1274, 1277, 1240],
       [1251, 1189, 1203, ..., 1314, 1271, 1228],
       [1303, 1205, 1198, ..., 1223, 1209, 1171]], dtype=uint16)
```

```
>>> matrix8 = matrix16//256
>>> matrix8 = matrix8.astype(np.uint8)
>>> matrix8
array([[4, 4, 4, ..., 5, 5, 4],
       [5, 5, 4, ..., 4, 5, 4],
       [5, 5, 5, ..., 4, 5, 4],
       ...,
       [4, 4, 4, ..., 4, 4, 4],
       [4, 4, 4, ..., 5, 4, 4],
       [5, 4, 4, ..., 4, 4, 4]], dtype=uint8)
```

- This caused a reduction in data thus making our solution less reliable and preferable at its job. Creating a new challenge for us.
- ***In order to have a reliable and efficient solution, we reverted to binary parsing and kept the data to its Predefined 16 bit format and Started working on a neural network which would be able to process this 16 bit data.***
- ***This eliminated the option for us to borrow and reference code from other commercial and free solutions being used to reconstruct day to day images as they supported only 8 bit data.***

Developing a Neural Network for the Reconstruction

- With the ability of sharing and Referencing pre-existing Neural Networks and Deep Learning Reconstruction Solutions out of scope, we went back to the roots of Machine Learning and Image Processing.
- In order to implement the research done by many Enthusiasts over the years for our solution, we started by implementing the solution they provided and stripping it down to form a peocessinf algorithm for matrices od pixel values.
- Once we had the barebones of our algorithm ready, we rigorously trained it with openly available datasets and the Given data.
- After ensuring that we had generated enough epochs and weights to properly process 8-bit data, ***we rebased the program to test 16-bit data.***
- After Various Modifications to every layer of the Neural Network, we had a model that could now train on 16-bit Satellite Imagery.
- Using openly available LANDSAT data, and Datasets from Kaggle combined with other sources, we **trained our model with 16-bit SWIR TIFF Images.**

Creating a Mask for Reconstruction

- After importing the given 16-bit image as a Numpy Matrix, it was only a matter of time and basic programming to create an iterating algorithm that identifies dropped out pixels by a thresholding system.
- This algorithm worked well, except in images with actual black regions and for large data, this algorithm was inefficient and rather slow due to the number of iterations.
- Thus, we redesigned this algorithm to incorporate a ***content aware system that could identify line and column dropouts*** instead of iterating over the whole image. This improved our benchmark time and computing efficiency.

Reconstruction of Missing Data with said Program

- Since we did not have a 16-bit Image Processing Progam ready, we tried to work around the problem and incorporate a downscaling factor which would convert the 16-bit input to 8-bit.
- While creating this downscaling factor, we noticed that the algorithm did not work correctly and we lost all data. Upon brainstorming we came up with a solution to this.
- Since the Luminosity of the Pixels was a relative factor, which can be deduced by the interconvertibility of 16-bit to 8-bit data,we incorporated a relativity factor in our program which would convert the images to 8-bit properly.

```
>>> matrix16 = n
>>> matrix16
array([[1265, 1245, 1274, ..., 1286, 1291, 1220],
       [1297, 1282, 1272, ..., 1265, 1293, 1240],
       [1306, 1302, 1327, ..., 1254, 1311, 1269],
       ...,
       [1181, 1146, 1169, ..., 1274, 1277, 1240],
       [1251, 1189, 1203, ..., 1314, 1271, 1228],
       [1303, 1205, 1198, ..., 1223, 1209, 1171]], dtype=uint16)
>>> relativity_factor = np.amax(matrix16)/256
>>> relativity_factor
6.04296875

>>> matrix8 = (matrix16//relativity_factor).astype(np.uint8)
>>> matrix8
array([[209, 206, 210, ..., 212, 213, 201],
       [214, 212, 210, ..., 209, 213, 205],
       [216, 215, 219, ..., 207, 216, 209],
       ...,
       [195, 189, 193, ..., 210, 211, 205],
```

```
       [207, 196, 199, ..., 217, 210, 203],
       [215, 199, 198, ..., 202, 200, 193]], dtype=uint8)
```

- Once converted to 8-bit, we could run it through our Algorithm and Reconstruct the data and Upscale it to 16-bit by the relative factor calculated earlier.

```
>>> matrix8
array([[209, 206, 210, ..., 212, 213, 201],
       ...,
       [207, 196, 199, ..., 217, 210, 203],
>>> output16 = matrix8 * relativity_factor // 1
       [1293., 1281., 1269., ..., 1262., 1287., 1238.],
       [1178., 1142., 1166., ..., 1269., 1275., 1238.],
```

- Since we discovered that this was a lossy means of conversion, ***we discarded this whole Program and started afresh with more experience and knowledge.***
- After training the Neural Network on 16-bit Images, in order to have an efficient and quick algorithm, ***we decided to furnish the program with only the necessary data in order for reconstruction, the rest of the image would stay as it is.***
- ***Thus, we divided the images into sections or layers and differentiated each layer using the mask developed earlier, once we had identified which sections needed reconstruction, we would supply those to the GANS algorithm which would generate the reconstructed image of the partial data we provided to it.***
- ***Once we had the Reconstructed Image, we would patch together all the different sections from the Original Image and the Output produced by GANS over the partial data and generate our final output file.***
- Thus Accomplishing ***Lossless 16-bit Input and 16-bit Output.***

Citations

```
@article{yu2018free,
  title={Free-Form Image Inpainting with Gated Convolution},
  author={Yu, Jiahui and Lin, Zhe and Yang, Jimei and Shen, Xiaohui and Lu, Xin
and Huang, Thomas S},
  journal={arXiv preprint arXiv:1806.03589},
  year={2018}
}
```

Instructions for dropouts_mask.py :

- Syntax : `python dropouts_mask.py (input_file) [OPTIONAL = output_extension png|tiff|tif]`