COMP 6731

Pattern Recognition

Implementation of a model for Pan-sharpening

Implementation based on

Paper: “Deep Gradient Projection Networks for Pan-sharpening” by S. Xu, J. Zhang, Z. Zhao, K. Sun, J. Liu and C. Zhang

Report Submitted To: Adam Krzyzak

Date: December 7, 2022

Ganesh Arun Masurkar

40017722

ABSTRACT

This conference paper [1] combines classical model approach with deep learning to create HRMS (High spatial resolution multispectral image having ‘B’ bands) from LRMS (Low spatial resolution multispectral image having ‘B’ bands) and PAN (Panchromatic image having ‘b’ band). This paper uses some assumed prior knowledge (deep prior – since it is derived from the data using neural network) of HRMS to generate two optimization problems (LRHS-aware and PAN-aware problems, also known as MS-block and PAN-block respectively).

The prior is calculated by assuming that LRMS images is obtained by blurring and down sampling of HRMS images and PAN images is obtained by linear combination of the bands of HRMS image. Each of these transformations accrue some loss of information which is learned during training using supervised data.

The blocks form a layer in the GPP (Gradient projection-based pan-sharpening) neural network. Initially we generate our prior $HRMS^0$ from LRMS images and feed it to MS-block whose output $HRMS^{0.5}$ is fed to PAN-block which uses it as prior and generates an output image $HRMS^1$. This is then fed to the MS-block in the next layer and so on. The output of PAN-block from n^{th} layer is $HRMS^n$, which is our output.

TABLE OF CONTENTS

List of Abbreviations

- HRMS: High spatial resolution multispectral image
- LRMS: Low spatial resolution multispectral image
- PAN: Panchromatic image
- GPPNN: Gradient projection-based pan-sharpening neural network
- MS: Multi-spectral
- HS: Hyperspectral
- MHNet: Multispectral and hyperspectal image fusion network
- PSNR: Peak Signal-to-noise-ratio

INTRODUCTION AND OBJECTIVE

In recent years, there has been a need to remotely monitor environments. The reasons for monitoring are to observe effects of climate change on natural resources like water, forest and so on, to study movements of hurricanes or storms, to study terrains of extra-terrestrial planets, to survey environments for illegal infrastructures.

Remote monitoring ranges from ground level to space level. Monitoring from within the atmosphere of planets is very inefficient due to atmospheric drag and larger ground to monitor. Satellite monitoring solves both issues; we can cover more ground faster. For monitoring from space, we need to use high resolution camera to capture minute details. But high-resolution images come with drawback of large memory requirement. Memory requirements increase with increase in number of bands. Additionally, high-resolution images require higher bandwidth for efficient transmission.

For all the above reasons, we need a better alternative to high-resolution multi-spectral (multiple bands) images. LRMS images solves the issues of higher memory requirements, but we lose minute details. PAN images capture minute details and require less memory than HRMS images, but we lose spectral information. LRMS and PAN images complement each other and together can be used to regenerate HRMS images. Using LRMS and PAN images to regenerate HRMS image is referred to as pan-sharpening.

BACKGROUND AND RELATED PROBLEMS

Many of the current approaches involved using CS algorithms or convolution networks for pan-sharpening.

Classical CS algorithms involve performing decomposition or applying high pass filter on LRMS and PAN images. But these processes are slow and require accurate filters for pan-sharpening to work properly. Convolution approach involves creation of intuitive networks and require a lot of trained samples.

Model based approach is the bridge between classical and convolution approach. It doesn’t require complicated filters, intuitive networks, or large number of samples. There is one model-

based approach ‘MHNet’ [2] described in the paper [1] which works with MS and HS images containing at-most 10 bands, but the model described in paper [1] works for MS images with ‘B’ number of bands.

GPPNN MODEL [1]

The model described in the paper [1] has two optimizations problems which work collectively to regenerate output HRMS image. The paper [1] describes optimization problem(s) and the network created by repeatedly applying the optimization problems.

Optimization problems (Blocks)

To derive the optimization problems, we start with observation model. One observation being a PAN (P) image is generated by spectral decomposition (S) of HRMS (H) image and other observation being LRMS (L) images is generated by consecutive blurring (K) followed by down-sampling (D) of HRMS (H) image (as indicated in Figure 1a [1]).

The optimization problem for PAN image is $\min \frac{1}{2}||P - HS||^2 + deepPrior(H)$ i.e.: square of absolute error constrained by H being the prior. The resulting update rule based on optimization problem for PAN image is referred to as PAN block (as seen in Figure 1b and 1c [1]).

Similarly, the optimization problem for LRMS image is $\min \frac{1}{2}||L - DKH||^2 + deepPrior(H)$ i.e.:

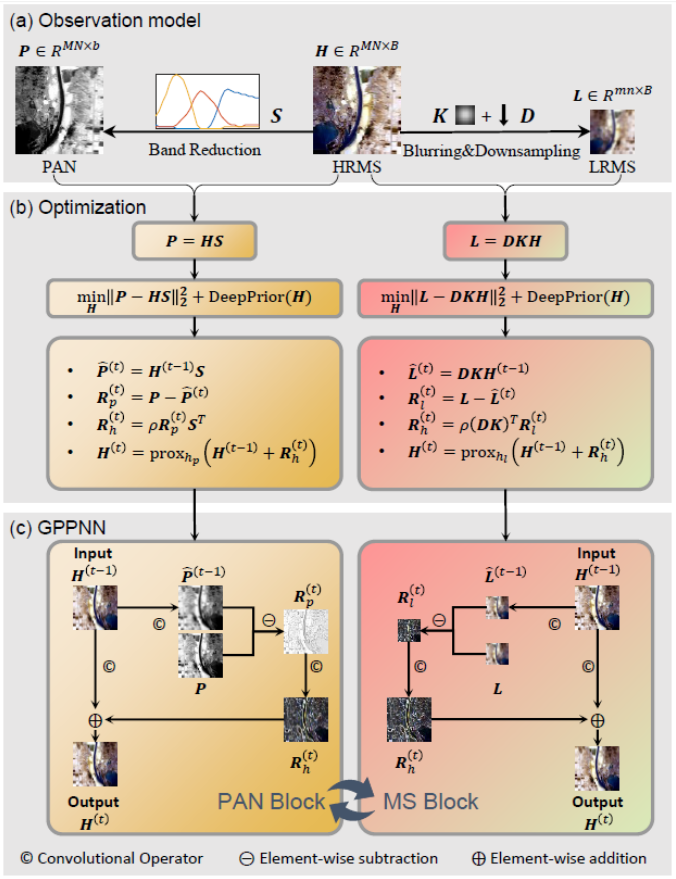


Figure 1. (a) The observation models for LRMS and PAN images. (b) Two formulated optimization problems and iterative steps of the gradient projection algorithm. (c) The main blocks in our proposed GPPNN.

Figure 1 [1]

square of absolute error constrained by H being the prior. The resulting update rule based on optimization problem for LRMS image is referred to as MS block (as seen in Figure 1b and 1c [1]).

Each of the convolution operator in Figure 1c are of type *conv(ReLU(conv(.; in, C); C, out)*, where 'in' and 'out' for each block is a function of number of bands.

GPPNN

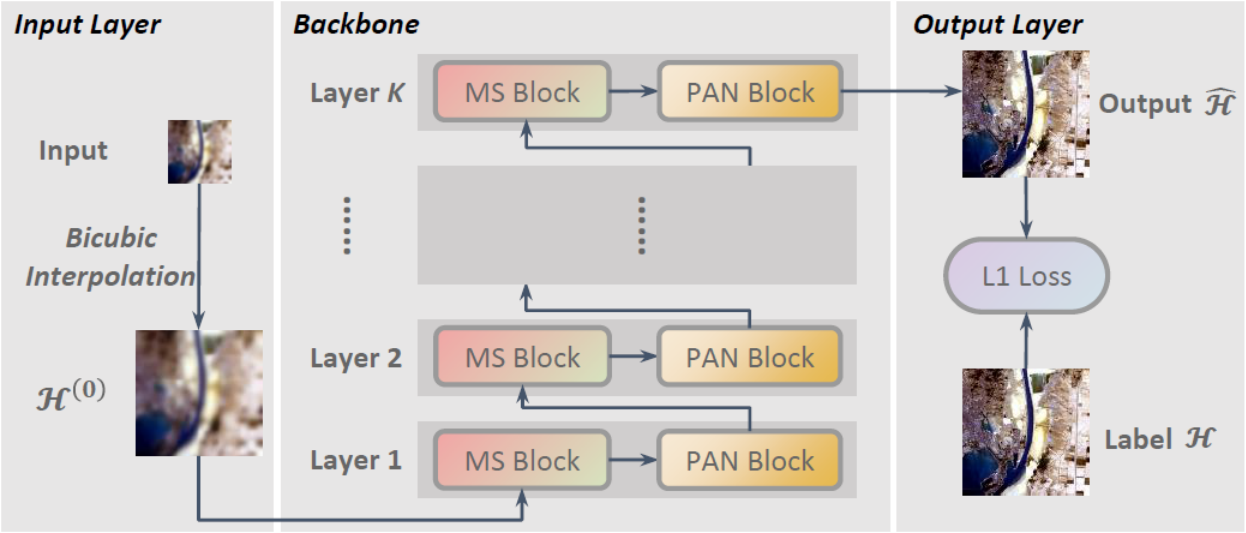


Figure 2. The structure of GPPNN.

Figure 2 [1]

The paper created a network from these optimization problems. The network includes input, backbone, and output layer (as seen in figure 2 [1]).

The input layer takes the LRMS image and perform bicubic interpolation (up-sampling) to generate our initial prior H^0.

The backbone layer consists of 'k' number of layers, where each layer contains an MS and PAN block. The initial prior H^0 is used as input to MS block in layer 1, output of that MS block is used as input to PAN block in layer 1 and then output of that PAN block is used as input to MS block in layer 2 and so on. At k^{th} layer the output of PAN block is the output H^k.

In output layer, we use the output H^k to train the model using L1 loss.

IMPLEMENTATION

Gathering dataset:

I started by collecting training and testing samples. I used GaoFen2 dataset but instead of using the entire dataset which has thousands of large samples, I used a single satellite image to generate small patches of training and testing HRMS samples (they were used due to time limit). To generate these HRMS patches, I used pytorch crop function as listed below:

croppedImg = transforms.functional.crop(img, …, …, size, size)

To generate LRMS images from these HRMS patches, I used pytorch GaussianBlur function followed by Resize function as listed below:

transforms.GaussianBlur(…)

transforms.Resize(…)

To generate PAN images from these HRMS patches, I used pytorch Grayscale function as listed below:

transforms.Grayscale(1)

I generated two set of datasets:

- Dataset 'Dataset (128px)' and 'Test Dataset (128px)' are for training and testing respectively. It contains HRMS/PAN images with resolution 128x128 and LRMS images with resolution 64x64. The resolution ratio is 2.
- Dataset 'Dataset (200px)' and 'Test Dataset (200px)' are for training and testing respectively. It contains HRMS/PAN images with resolution 200x200 and LRMS images with resolution 50x50. The resolution ratio is 4.

The structure of each dataset is as follows: one folder 'HRMS' (containing HRMS samples), one folder 'LRMS' (containing LRMS samples) and one folder 'PAN' (containing PAN samples).

Implementation of algorithm:

I implemented my project using pytorch in python (because I have experience building neural networks using pytorch) and measured PSNR using torchmetrics library (to compare performance of my trained model with model tested in the paper [1]).

Before training or testing is performed, I preload and preprocess the samples by calling 'loadData' function. It is done beforehand to make the process more efficient. I load and apply required transformations (such as converting to tensor, normalization and unsqueeze) on LRMS, HRMS and PAN images and store them. In this preloading step, I generate the initial prior $HRMS^0$ by up-sampling LRMS images using bicubic interpolation and store it.

I did not use ‘skorch’ because I am not aware how to override ‘forward’ function to allow it to accept more than one argument. Instead I created my own class ‘Net’ (which inherits nn.Module) and in this class I created 3 instance variables BCB, BCb and bCB (1st one is used in both blocks so I created a main copy that was accessible to both blocks). Each of the 3 instance variables are initialized as follows (‘in’ and ‘out’ channels are ‘B’ and ‘B’ respectively for BCB, ‘B’ and ‘b’ respectively for BCb and ‘b’ and ‘B’ respectively for bCB):

```
nn.Sequential(
        nn.Conv2d(in_channels=_, out_channels=C, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(in_channels=C, out_channels=_, kernel_size=3, padding=1)
)
```

In ‘Net’ class, I override ‘forward’ function to except 3 arguments; LRMS, prior $HRMS^0$ generated from LRMS (by up-sampling) and PAN. In this function I have implemented the backbone of GPPNN which includes the two optimization blocks (I implemented the entire backbone in this function to prevent unnecessary duplication of code).

In MS block, the algorithm requires performing blurring followed by down-sampling/up-sampling but I accidentally did the opposite and most of the experiments were carried out on this algorithm (trained model name containing ‘B’ at the end are trained on this algorithm). I carried out more experiments using the correct algorithm and performance of the correct model was similar to incorrect model.

<u>CLI and user manual:</u>

Run ‘main.py’ executable to get CLI and follow the instructions (You can refer demo file submitted with the report for more information on how to perform testing of trained model and generate a HRMS image from a single sample of LRMS and PAN image). You need to install ‘torch’, ‘torchvision’, ‘PIL’ and ‘torchmetrics’ python libraries.

In my implementation I provide a simple CLI interface to allow users to train a new model. Before training a model, you need to initialize following variables in the code; ‘numberOfSamples’ (to indicate the number of samples you want to train the model on), ‘normalizeImg’ (to indicate whether the images should be normalized for training and testing), ‘C’ (to indicate number of features), ‘K’ (to indicate the number of layers in GPPNN),

'numOfEpochs' (to indicate the number of epochs for training), 'baseDataFolder' (to indicate location of training samples), and 'baseTestDataFolder' (to indicate location of testing samples). The CLI also allows users to test a trained model on test dataset and measure the performance of trained model. CLI also allows users to test a single LRMS and PAN sample and see the generated HRMS image. For testing, the trained model needs to have configuration information embedded in the name, the format is; *128By64(number of samples,C,Epoch,k,normalization,after)*, '128By64 ' refers to '128px' dataset (you can skip that when using '200px' dataset). 'Number of samples', 'C', 'Epoch', and 'k' are integers except 'normalization' which should be T or F and 'after' should be A or B ('after' indicates whether blurring should be done after or before down-sampling/up-sampling). These should at the end (For reference, refer naming convention of trained model submitted with the report).

EXPERIMENTS AND RESULTS

I performed experiments with different number of training samples, different configurations (i.e.: C, k, epochs) and different number of features. The performance is measured using average peak signal to noise ratio (Avg. PSNR). I used training and testing set.

Trained model number	**Trained model name (number of samples, C, Epoch, k, normalization, after)**	**Training set (Avg. PSNR)**	**Testing set (Avg. PSNR)**
1a	TrainedModel(500,30,10,3,F,B)	30.95	30.32
1b	TrainedModel(500,32,50,4,F,B)	32.92	33.97
2aa	TrainedModel128By64(500,30,10,3,F,B)	35.06	34.74
2ab	TrainedModel128By64(500,30,10,3,T,A)	34.42	34.25
2ba	TrainedModel128By64(500,32,50,4,F,B)	35.18	36.72
2bb	TrainedModel128By64(500,32,50,4,T,A)	37.42	36.58
3aa	TrainedModel128By64(1400,30,10,3,F,B)	36.25	36.01
3ab	TrainedModel128By64(1400,30,10,3,T,A)	35.58	36.11
3b	TrainedModel128By64(1400,32,50,4,F,B)	36.95	37.92
4	TrainedModel128By64(1400,32,25,6,T,B)	37.92	38.54

The models were tested on testing set containing 300 images and I calculated the average PSNR of all 300 images. The result is listed above in the table.

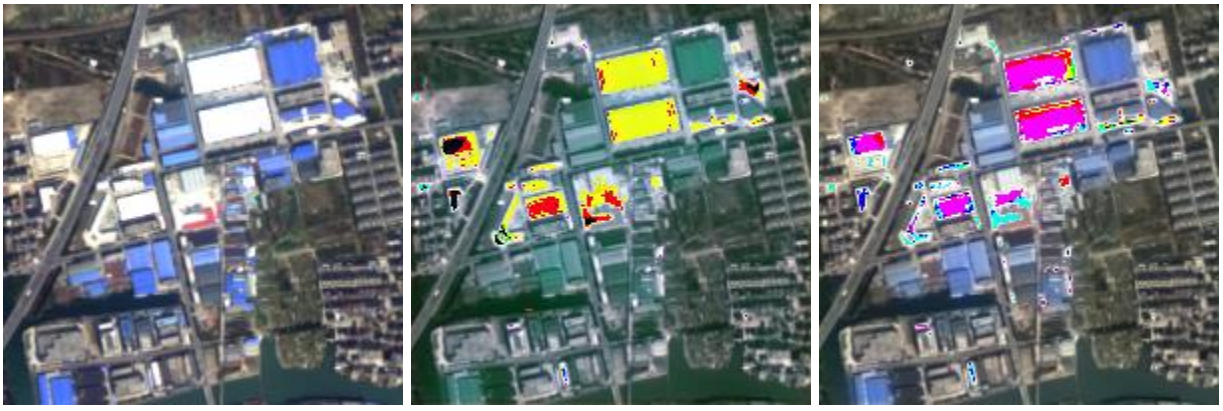
As per the table both incorrect (model name containing ‘B’) and correct model (model name containing ‘A’) perform approximately the same. The best trained model is in red.

Observation 1 from above table: From models ‘1a’, ‘1b’, ‘2aa’, ‘2ab’, ‘2ba’, ‘2bb’, ‘3aa’, ‘3ab’, and ‘3b’, we observe that model performance improves when number of features are increased. We also observe that optimal number of features for indicated number of samples is greater than or equal to number of features used in these models.

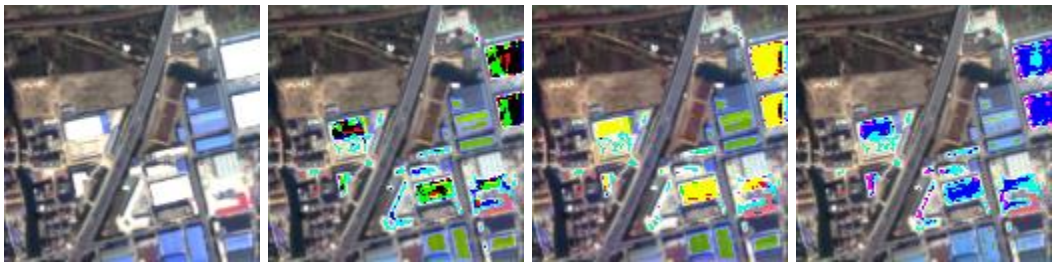
Observation 2 from above table: From models ‘1a’, ‘1b’, ‘2aa’, ‘2ab’, ‘2ba’, ‘2bb’, ‘3aa’, ‘3ab’, and ‘3b’, we observe that model performance improves when number of samples is increased.

Observation 3 from above table: From models ‘1a’, ‘1b’, ‘2aa’, ‘2ab’, ‘2ba’, ‘2bb’, ‘3aa’, ‘3ab’, and ‘3b’, we observe that model performance improves when number of epochs is increased.

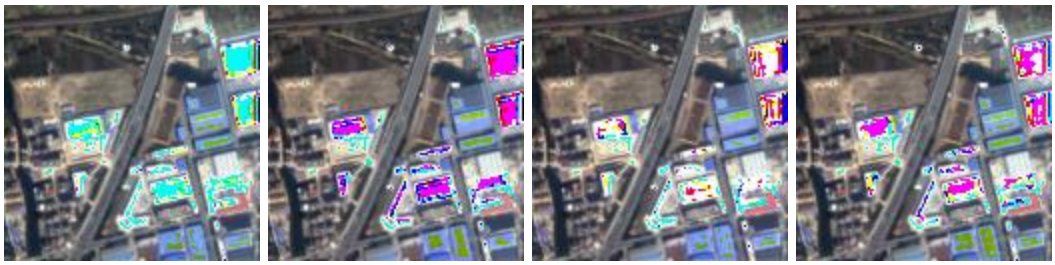
Below are set of output images generated by different trained model (identified by their model number) and the original image.



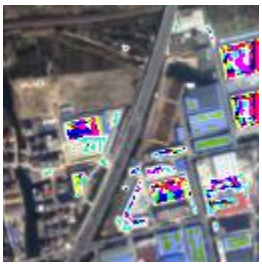
Original 1a 1b



Original 2aa 2ab 2ba



2bb 3aa 3ab 3b



4

Observation 4 from above images: From images for models ‘2ab’ and ‘3ab’ we observe that model performs well on images containing white/bright colors when number of samples (containing target images) are increased.

Observation 5 from above images: From images for models ‘1a’, ‘2aa’, and ‘2ab’, we observe that model performance is better with lower resolution ratio.

Observation 6 from above images: From images for models ‘1a’ and ‘1b’, we observe that performance of model that were trained on dataset with high resolution ratio improves with increase in number of features and epochs.

Below are examples HRMS images generated by model number ‘4’.

Observation 7 from below images: Model works well for images containing darker colors. More training is required to process images containing white/bright colors effectively.

Original Generated Original Generated

CONCLUSION

Based on the experiments, the model works well with all range of resolution ratios. For higher resolution ratio the performance of the model depends on the pixels retained in LRMS. If LRMS is too small, some of the pixels will be lost. Performance is also dependent on the number of features and epochs used while training.
Models trained on fewer samples and fewer number of epochs works well for images with darker pixels.
For models to work well on all types of images (i.e.: images with bright or dark pixels), model needs to be trained on large dataset and high number of epochs.

FUTURE WORKS

Randomizing training samples before each epoch could be done to prevent overfitting. Randomization also helps in making a more robust model.
We can train multiple models each on different types of images such as grasslands, rural, urban, mountainous region etc. This way we can auto classify an input image based on its performance on different models. Additionally, training different types of images separately will improve the model performance. This was evident from my experiments, where the perform of the model was hindered by differences in training samples i.e.: performance of grasslands or mountainous regions while training was better than performance of rural or urban regions (latter needs more features, samples, and higher number of epochs).
We can split training set into training and validation set, this way we can find the optimal number of epochs that leads to a best model for a specific configuration.

REFERENCES

[1] S. Xu, J. Zhang, Z. Zhao, K. Sun, J. Liu and C. Zhang, "Deep Gradient Projection Networks for Pan-sharpening," 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 1366-1375, doi: 10.1109/CVPR46437.2021.00142.

[2] Qi Xie, Minghao Zhou, Qian Zhao, Deyu Meng, Wangmeng Zuo, Zongben Xu; Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 1585-1594.