Chapter 5

Deep Learning based Compression Techniques

5.1 Convolution Neural Network Based Lossy Compression of Hyperspectral Images

** The content of this chapter has been published by Yaman Dua, Ravi Shankar Singh, Smit Lunagariya, Kshitij Parwani and Vinod Kumar. "Convolution Neural Network Based Lossy Compression of Hyperspectral Images." Signal Processing: Image Communication 95 (2021): 116255.* `https://doi.org/10.1016/j.image.2021.116255` *(Elsevier, IF: 3.256)*

5.1.1 Introduction

This chapter introduces the concept of autoencoder for compression of HSIs. Deep learning algorithms for both supervised and non-supervised learning applications have gained attention due to the flexibility of deep networks. More complex functions can be accomplished by adding more units in a layer or by adding multiple layers within a network. It helps to automatically extract feature vectors from given large datasets

without labeling the attributes in case of big data. CNN is used in deep learning for image processing applications [170]. CNNs have the ability to learn from high dimensional, complex, nonlinear, and extensive collection mappings of input data [171–173]. It uses a linear mathematical operation named "convolution" in neural networks rather than matrix multiplication. Trained CNN models can be directly used for processing similar datasets, so these models are portable from one location to another after being tested and validated. It forms the basis for the selection of CNN for compression of images since compression and decompression are generally performed at different locations in transmission applications.

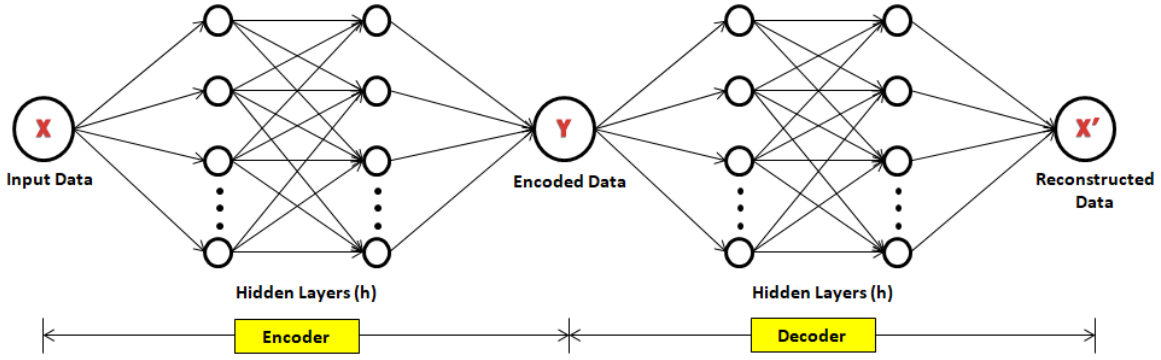

Figure 5.1: Parts of an autoencoder

A trained neural network that intends to copy the useful properties of the input data to its output layer with an intermediate set of hidden layers is called an Autoencoder [174]. It consists of two parts: Encoder ($A_{\text{encoder}}(h|x)$) represented by Equation 5.1 and Decoder $A_{\text{decoder}}(x|h)$ represented by Equation 5.2, where $'x'$ is the input dataset, and $'h'$ is the set of hidden layers in the network. An Encoder maps the high dimensional input (x) to a lower-dimensional encoded data (y) while passing through hidden layers(h). In contrast, the decoder reconstructs the input data (x') with some loss represented by($x-x'$), by mapping encoded data (y) with weights of a similar set of hidden layers. These details are represented diagrammatically in Figure 5.1.

$$X(\text{input data}) \overset{\text{hidden layer(h)}}{\Longrightarrow} Y(\text{encoded data}) \tag{5.1}$$

$$Y(\text{encoded data}) \overset{\text{hidden layer}(h)}{\Longrightarrow} X'(\text{reconstructed data}) \tag{5.2}$$

Autoencoders can be tuned and trained using the same techniques used for training modern deep learning models like gradient descent and recirculation [175]. It learns by comparing the weights of original input with the weights of reconstructed data, as it is more biologically appealing than back-propagation. The objective of the autoencoder model is to reduce the loss obtained from the difference between reconstructed data and original data. Deep learning models have been used in many big data applications for feature extraction, classification, and dimensionality reduction, giving the best performance in each domain. In this work, also perform application-oriented analysis of the reconstructed image by using state of the art HSI classification algorithm.

5.1.2 Methodology

The concept of the autoencoder is used in this article to reduce the dimension of HSI dataset that has two components encoder and decoder. Each component uses a set of convolution layers, max-pooling layer, and the transpose of convolution layer on a complete HSI. The architecture of the proposed algorithm is diagrammatically represented in Figure 5.2. We haven't considered partitioning of images into small patches as it leads to reduced compression performance for the following reasons [140]:

- Each patch is considered as an independent image, thus a header is required for each one
- Less number of neighbors results in limited prediction at the junction of each patch
- Less adaptive compression due to the presence of small patches (fewer data to learn)

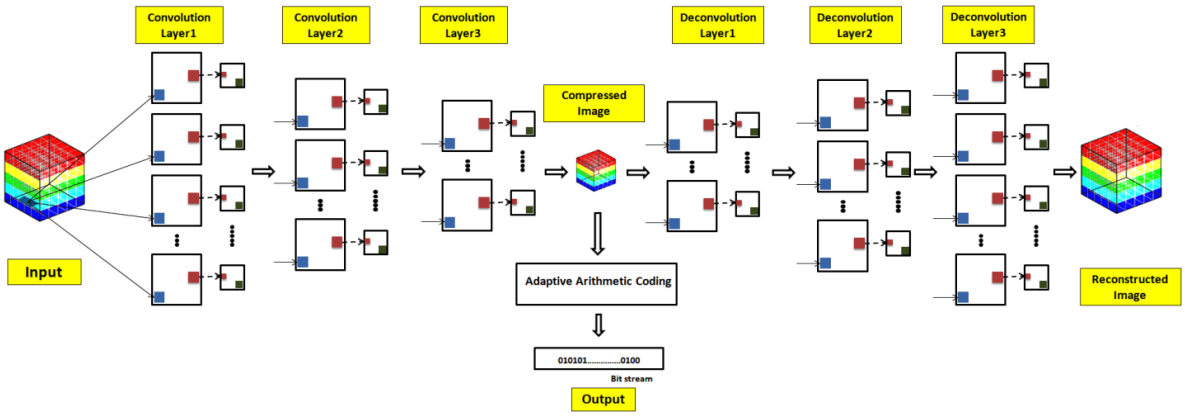

Figure 5.2: Architecture of the proposed algorithm

Various steps of the algorithm are explained in the following subsections.

5.1.2.1 Encoder

The pixel values of an input HSI are first normalized to make the mean equals to zero and variance equals to one by subtracting the mean from elements row-wise and then dividing the difference by its standard deviation. Thus forming a new 3D tensor, this is considered as an input layer in the deep network. It is followed by a combination of the convolution layer, with the rectified linear unit (RELU) as an activation function, and pooling layer.

Convolution Layer- It is the first layer in a CNN that comprises of a set of kernels; it is also known as the building block of CNN. Each kernel performs a mathematical operation on the input data called convolution. Convolution can be defined as an operation always performed on two functions with a real-valued argument, represented by $(*)$. Mathematically, it can be found by Equation 5.3 for a 2D input, where the 2D kernel is $\varphi\,(p,q)$ and $I\,(p,q)$ is the 2D input signal.

$$C(p,q) = (\varphi * I)(p,q) = \sum_i \sum_j I(p-i, q-j)\varphi(p,q) \tag{5.3}$$

Convolution can be continuous or discrete, depending upon the need of application

and input signal. The size of kernel is much smaller than the size of the input image resulting in the sparse convolution matrix. The first convolution layer in CNN captures features such as gradient orientation, color, edge, etc. [176]. Convolution results in reduced dimensionality of the input signal that is taken care by padding extra bits around the input image.

Activation Function- It is an intermediary function that transforms the output of a convolution layer and transfers it to the next layer. We have used two different activation functions in the proposed algorithm: ReLU and Hyperbolic Tangent (tanh). [177] ReLU function removes the negative values from the input and converts it to zero without saturating at any positive value. The function of ReLU is $\max(0, x)$, which makes it very computationally efficient. Tanh is a nonlinear activation function that is bounded in the range (-1, +1), mathematically, it is calculated using Equation 5.4. Instead of ReLU Generalized Divisive Normalization (GDN) is used in end to end compression systems [178] to improve the performance. It is not included in the proposed work because it can only be executed across different channels, and not across different spatial positions. Adam [179] is used as optimization function.

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{5.4}$$

Pooling Layer- It is responsible for the extraction of dominant features that are translational, positional, and rotational invariant. It has many other features like controlling over-fitting, reduction of the size of convolved feature by replacing the output of the feature at a location with the statistics of neighboring outputs. Two important pooling operations are max pool and average pool. The former replaces elements of a fixed area by the maximum element of that area, and the latter replaces the elements with the arithmetic mean or weighted average of the elements of an area.

The proposed model uses three layers in the encoder section, starting from the convolution layer with ReLU activation function, number of filters in the first layer is

Algorithm 5.1: Autoencoder based compression algorithm

Result: Encoded bitstream of compressed image

Input: Hyperspectral Image $I(R, C, B)$ of size $(R \times C \times B)$, where number of rows = R, number of columns = C, and number of spectral bands = B

Representation: $norm_{img}[x, y, z]$ = normalized image; $\mu[i]$ = arithmetic mean of the elements of the row (i), $\sigma[y]$ = Standard deviation of the elements of the row (i)

Initialization: Adam optimization algorithm [100] is used to optimize the neural network with the parameters, $\alpha = 0.0009$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, where α represents the learning rate of the model, β_1 and β_2 represent the exponential decay rate for 1^{st} and 2^{nd} moment estimates.

Algo:

for *z = 1 to B* **do**
  for *y = 1 to R* **do**
    for *x = 1 to C* **do**
      $norm_{img}[x, y, z] = (I[x, y, z] - \mu[y])/\sigma[y]$; `/* Normalize the image bandwise */`
    end
  end
end

Transform $norm_{img}(R, C, B)$ as $norm_{img}(B, R, C)$ to use the "channel first" parameter in CNN

`// Define the encoder model`

Add conv2D layer with # of filters = 128 and activation_function = "ReLU" ; `/* Remove the layer if` $B <= 128$ `*/`

Add maxpool layer ; `/* Remove the layer if` $B <= 128$ `*/`

Add conv2D layer with # of filters = 64 and activation_function = "ReLU" ;

Add maxpool layer ;

Add conv2D layer with # of filters = 32 and activation_function = "ReLU" ;

Add maxpool layer;

`// Define the decoder model`

Add Conv2DTranspose layer with # of filters = 64 and activation_function = "ReLU";

Add Conv2DTranspose layer with # of filters = 64 and activation_function = "ReLU";

Add Conv2DTranspose layer with # of filters = 128 and activation_function = "ReLU" ; `/* Remove the layer if` $B <=128$ `*/`

Add Conv2DTranspose layer, # of filters = 128, activation_function = "ReLU" ; `/* Remove the layer if` $B <=128$ `*/`

Add Conv2DTranspose layer, # of filters = B(# of bands), activation_function = "ReLU";

Add Conv2DTranspose layer with # of filters = B and activation_function = "tanh";

Train the model for # of epochs = 120;

`// Use the encoder to encode given test HSI`

128 for HSIs with the number of bands greater than 128. It is followed by a max pool layer with the size of kernel$= (2, 2)$. The second major layer is a convolution layer with the number of filters equal to 64, followed by a max pool layer. The third layer to be added is convolution layer with tanh activation function, number of kernels, or filters in this layer = 32, which is again followed by max pool layer.

5.1.2.2 Decoder

The output of the last layer of the encoder is taken as the input in the decoder model that has six convolution transpose layers. The output of decoder is a reconstructed or regenerated image with the shape similar to that of the input image.

Deconvolution Layer - It is also known as transpose convolution layer that reverses the phases of convolution layer combining conv2D and upsampling layers [180]. This layer is used to regain the shape of the input, maintaining the convolution operation. The same size of components can be used for deconvolution that maps features to their pixel values. Also referred to as fractional, upward, backward, or sub-pixel convolution layer.

The proposed model is trained for 120 epochs to minimize the loss, and the best model that gives the minimum loss is used for compression of HSIs. Various steps with learning parameters are given in Algorithm 5.1, along with the initialization parameters.

5.1.2.3 Entropy Coding

The compressed image from the proposed autoencoder is not the same as the input image due to presence of error in the model. It can be stored in a comparatively lower number of bits per pixels per band, thus giving a better compression performance. We have used AAC [181] for entropy coding of the compressed image that can be transmitted or stored with comparatively reduced size.

The best autoencoder structure in terms of compression ratio and image quality is

the proposed three layer structure. It was identified experimentally that using more than three layers for this dataset reduces the image quality to an unacceptable level. In contradiction to our expectation, the image quality got reduced as the decompression starts from bottleneck stage in which dimensions of the compressed image starts decreasing critically. On the other hand, if number of convolution layers in the proposed model are reduced, optimal compression advantage could not be achieved due to increment in the number of training parameters. Another possible reason for such variation can be the increment in complexity for deeper networks that make the algorithm difficult to train and it may stuck at local optima [182]. The method has been tested to compress arbitrary datasets after training the model with similar type of HSI captured from similar sensor/device. The major limitation of the model is that it needs to be re-trained for different category of datasets to be compressed. For HSIs from same sensor of same location, the model need not learn before prediction. It helps to reduce the learning time for similar category prediction. We chose loss and accuracy as two prime parameters to obtain better results in terms of quality of decompressed image. In a near-lossless algorithm, there always exists a trade-off between compression ratio and the quality of reconstructed image (or loss). If the objective is to increase the compression ratio, the quality has to be compromised a bit. This dependency on the quality or loss (mean square error, in our case) makes compression ratio, a dependent variable. That can vary by changing the parameters of the proposed model. Other parameters of the model have been optimized by sequence of experiments, so for other datasets the model needs optimization. The parameters taken by our model to encode and decode any HSI depends upon the number of pixels per band and presence of number of channels in it. It provides an add-on to the results obtained in terms of compression ratio and PSNR. The proposed method supports input HSIs of 16-bit depth per pixel, spatial resolutions ranging from 1.3 metres to 30 metres, and spectral resolutions from 400 to 2500 nm. Some of them have been used in the experiment to

test the performance of the model. The details of these datasets can be found in the next section.

5.1.3 Experiment Results

This section provides simulation results of the proposed algorithm, along with a discussion on various outcomes of the results. The following sub-sections contain detailed information about the dataset used, the implementation environment, and simulation results of various experiments.

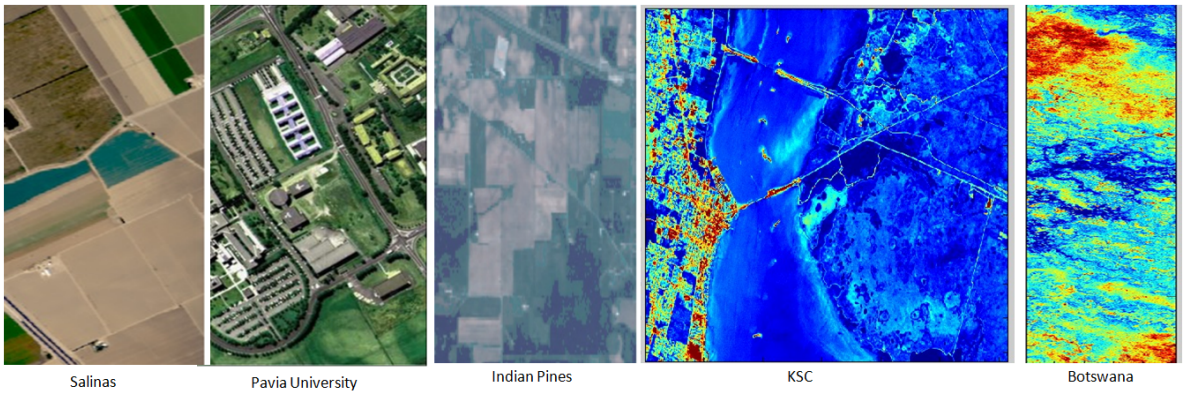

Figure 5.3: RGB image of datasets used in the experiment

5.1.3.1 Dataset

The algorithm was evaluated on the five HSI datasets: Salinas's scene acquired by AVIRIS sensor, Pavia University acquired by ROSIS sensor, Indian Pines acquired by AVIRIS sensor, KSC acquired by AVIRIS sensor, and Botswana acquired by Hyperion sensor. Dataset is available for research purpose from [47]. The dataset was divided into three parts for training, validation, and testing phase with split ratio of 0.7, 0.15, and 0.15, respectively. RGB converted image of the datasets can be seen in Figure 5.3. These images were selected for experiment as they are collected from a variety of sensors. Also, due to the availability of ground truth images that can aid in analysis of compression on supervised classification. Details of these images are provided in Table

5.7.

Table 5.1: Description of the dataset used to evaluate the algorithm

Dataset	Number of rows	Number of columns	Number of bands
SALINAS	512	217	204
PAVIA UNIVERSITY	610	340	103
INDIAN PINES	145	145	200
KSC	512	614	176
BOTSWANA	1476	256	145

Salinas - It was collected from Salinas Valley in California by a high-resolution sensor that captures 224 spectral bands with a 3.7 meter spatial resolution per pixel. The image contains radiance information of vineyard fields, bare soils, and vegetables. 20 spectral bands were removed from the original image that was responsible for the absorption of water. The ground truth image of Salinas contains 16 classes.

Pavia University - It was collected from Pavia city in Italy by ROSIS sensor during flight operation. The spatial size of original image was 610×610 with a resolution of 1.3 meters that was reduced to 610×340 after removing unnecessary samples. Its ground truth contains nine different classes.

Indian Pines - The image was captured from a test site in Indiana by AVIRIS sensor in June containing 220 bands out of which 20 water absorption bands were removed in preprocessing. Its ground truth contains non-mutually exclusive sixteen different classes.

KSC - The dataset was captured on Mar 23, 1996, from a location in Florida named KSC by AVIRIS instrument of NASA. The spectral wavelength range of the sensor varies from 400 nm to 2500 nm, each band having a bandwidth of 10 nm out of which 48 low SNR and water absorption bands were removed in preprocessing. Each pixel contains information of 18 meters captured from 20 km altitude. Different land

cover types represented by 13 classes represented in the ground truth image.

Botswana - It was acquired by EO-1 satellite of NASA equipped with a Hyperion sensor from Okavango Delta in Botswana. The spatial resolution of the sensor is 30 meters with spectral bandwidth of 10 nm calculating to 242 bands. After preprocessing at UT center for space research number of bands was reduced to 145, representing 14 different classes with land cover information of the area like drier woodlands, occasional and seasonal swamps.

5.1.3.2 Implementation Environment and Evaluation Metrics

The proposed algorithm along with the state-of-the-art techniques like 3D-DWT + SVR [69] and NFTD + PCA [183] have been implemented on a personal computer powered by Windows 10 platform having Intel(R) Xeon(R) CPU @ 2.30 GHz with four cores and a RAM of 16 GigaBytes. Programs are coded in Python 3.7 language using neural network libraries like Keras, TensorFlow, and PyTorch. The evaluation metrics used to analyze the effectiveness of the proposed algorithm are CR, PSNR, and Classification Performance.

5.1.3.3 Simulation of the proposed algorithm

Experiment 1: The performance of the proposed model is evaluated in this experiment by calculating the loss as a function of number of epochs. After tuning the model, it is experimentally found that the model shows optimum performance when number of epochs = 120. Figure 5.4 shows the graph of loss obtained (on y-axes) versus number of epochs (on x-axes) for Salinas, Figure 5.5 for Pavia, Figure 5.6 for Indian Pines, Figure 5.7 for Botswana datasets. It can be observed from the graph that training loss start declining gradually after 20 epochs. It attained a static level at 120 epochs after which no significant change can be observed, so 120 epoch is selected to be an optimal choice. It is worth mentioning here that the size of the filter/kernel has to be modified

according to the size of the input image. Since we didn't reduce the size of the image to make the algorithm more scalable to other HSI

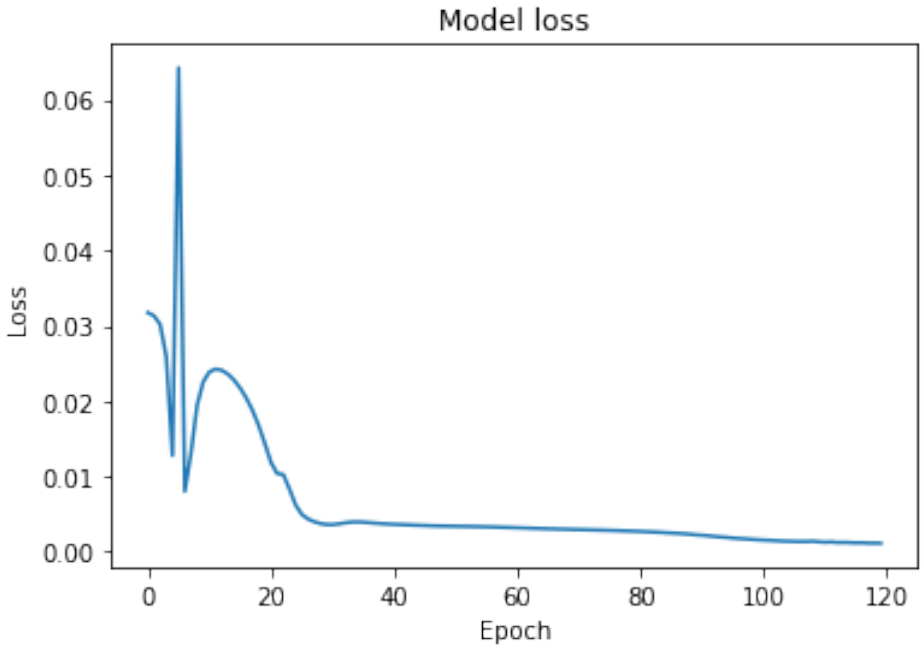

Figure 5.4: Performance of the proposed autoencoder for Salinas' dataset

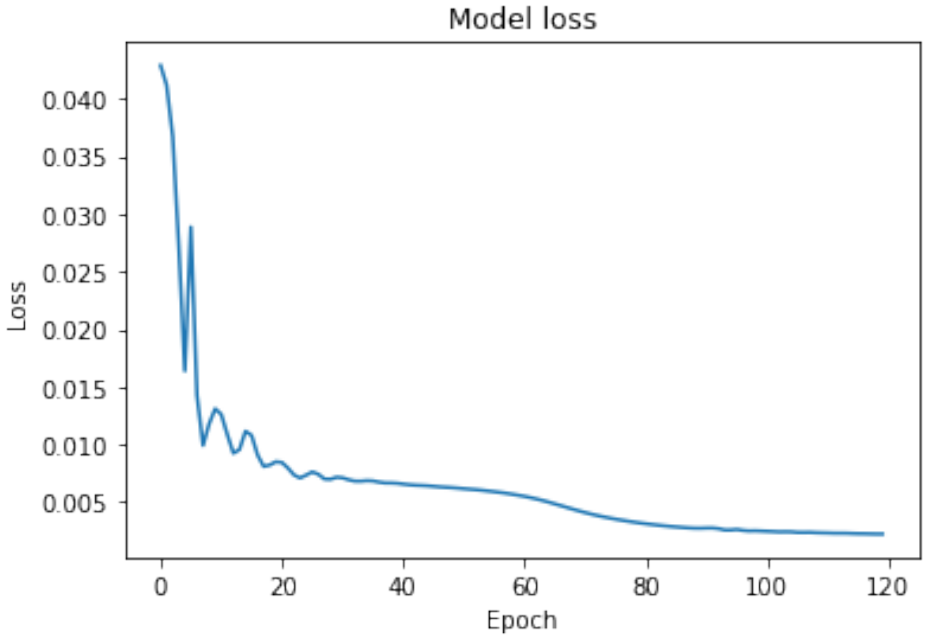

Figure 5.5: Performance of the proposed autoencoder for Pavia dataset

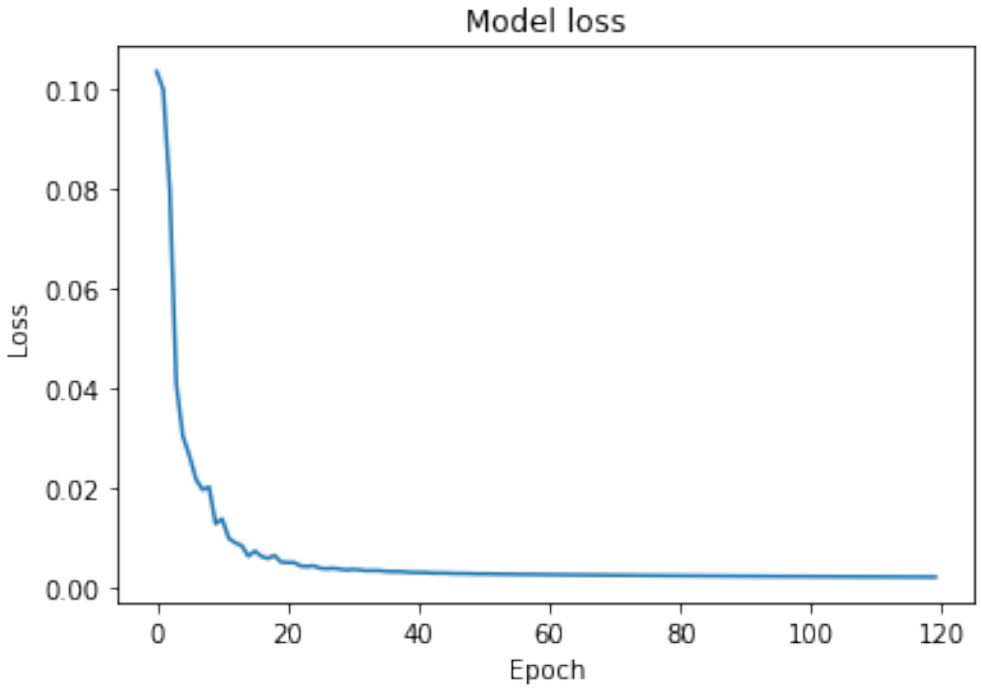

Figure 5.6: Performance of the proposed autoencoder for Indian Pines dataset

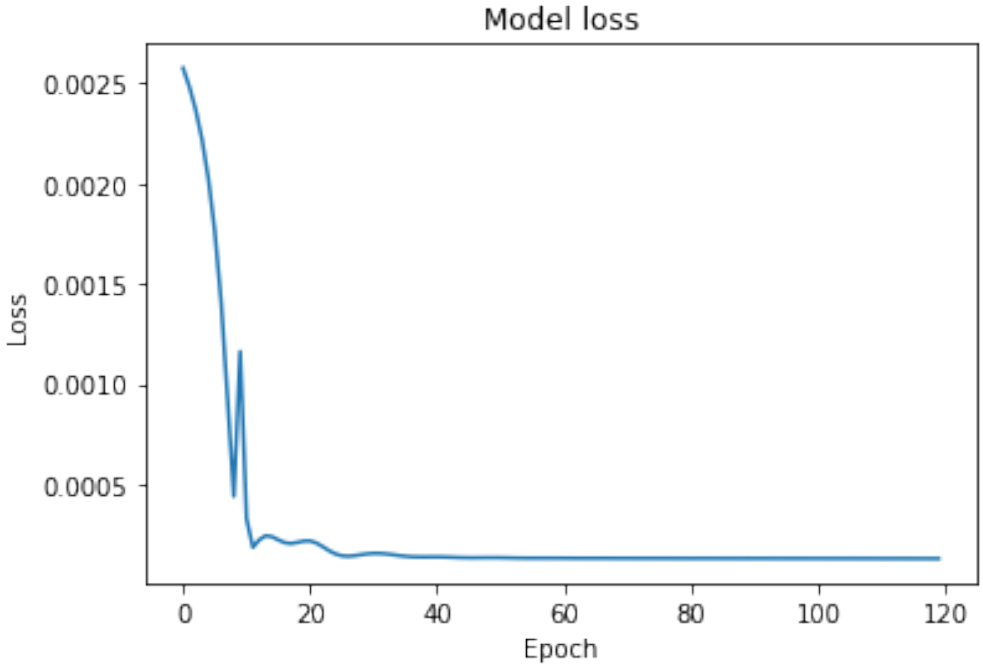

Figure 5.7: Performance of the proposed autoencoder for Botswana dataset

Table 5.2: Compression results obtained from different algorithms

Dataset/Attributes	3D DWT+SVR		NFTD+PCA		Proposed Technique		
	CR	PSNR	CR	PSNR	CR	PSNR	Execution Time (s)
SALINAS	21.53	42.28	272.23	46.47	**447.42**	**54.46**	416.51
PAVIA UNIVERSITY	25.95	43.59	137.51	47.11	**157.71**	**55.26**	656.79
INDIAN PINES	27.02	44.19	**130.25**	49.68	128.33	**55.19**	315.92
KSC	26.35	41.56	235.12	47.06	**371.67**	**53.32**	898.48
BOTSWANA	23.04	45.68	193.25	48.72	**313.47**	**56.14**	1772.38

Experiment 2: In this experiment, we have evaluated the efficiency of the proposed method in terms of compression ratio with the other two state of the art algorithms. Second method of the NFTD + PCA algorithm is implemented with principal components = 16 due to a large number of spectral bands present in this dataset. Results obtained for different datasets are represented in Table 5.2, where it can be observed that the proposed method gives an increment of 65% CR from NFTD + PCA method and 21 times better than 3D DWT+SVR method for Salinas dataset. The change observed is due to the fact that the model identifies the pattern of variations in pixel, supported by AAC encoding. It helps to provide a symbolic representation to the pixel with maximum frequency with reduced per bit storage. Salinas, KSC, and Botswana dataset are equipped with the maximum spectral and spatial correlations to favor compression algorithm for better performance.

The performance of the proposed method for Indian pines dataset is weak as compared to NFTD + PCA due to less number of pixels per band, i.e., reduced spatial information. Similar is the case with PU dataset, where the proposed method can't provide more than 15% increment due to less number of spectral bands. This analysis proves that reduction in either parameters leads to poor performance like in Indian pines and Pavia dataset. The graphical representation of the results is given in Figure 5.8 that confirms the efficiency of proposed compression algorithm. Green bars represent

the performance of the proposed model with other two state-of-the-art techniques.

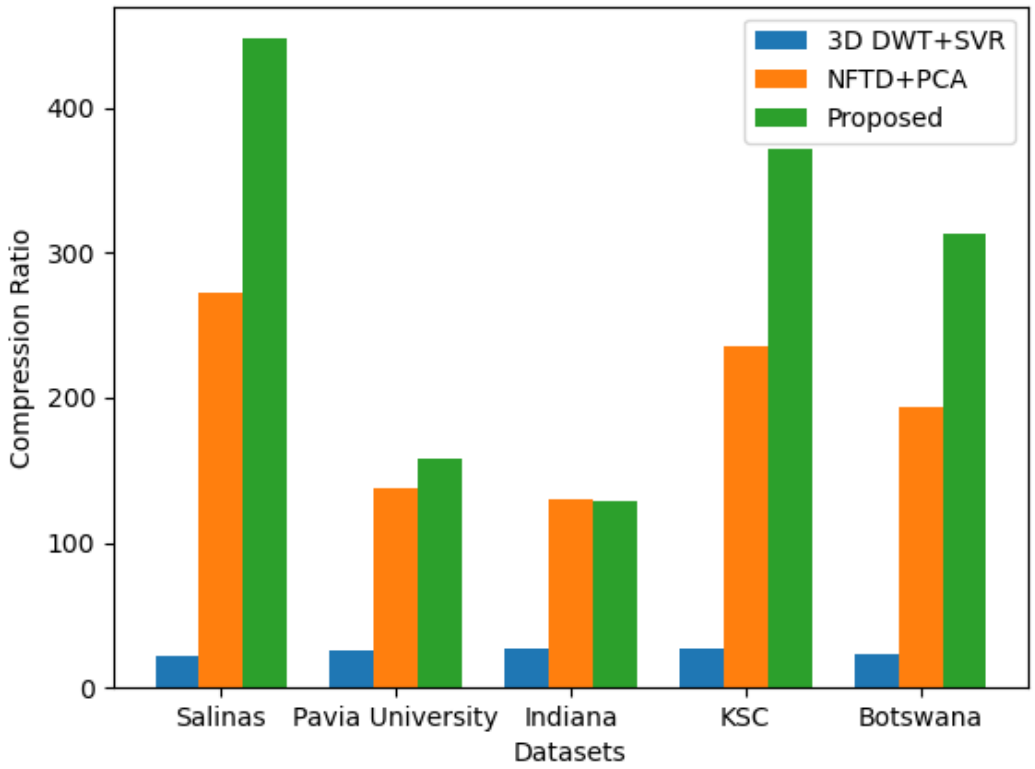

Figure 5.8: Comparison of compression ratio with state of the art algorithm

Experiment 3: The second metric used to analyze the performance of the proposed method is PSNR. It checks the quality of reconstructed image in mathematical terms, which should be higher for a better compression algorithm. Figure 5.9 represents the PSNR of the HSIs obtained from multiple algorithms. Results prove that the proposed method outperforms the existing state of art algorithms for all the datasets. A comparative result is given in Table 5.2, where it can be observed that compression ratio and PSNR are somewhat related except for the case of intrinsic property of dataset. For higher compression ratio like in Salinas image, PSNR is typically lower than others. Similarly for the case of Botswana maximum PSNR is obtained with reduced compression ratio. This situation do not corroborate with Pavia and Indian pines dataset due

to reduced spectral or spatial correlation. The execution time of the proposed model on these datasets is also provided in the last column of Table 5.2 in seconds. It can be observed that execution time only depends on the size of the input image, where it is minimum for Indian pines and maximum for Botswana dataset. Figure 5.10 represents the 7^{th} band of original and decompressed PU and IP dataset.

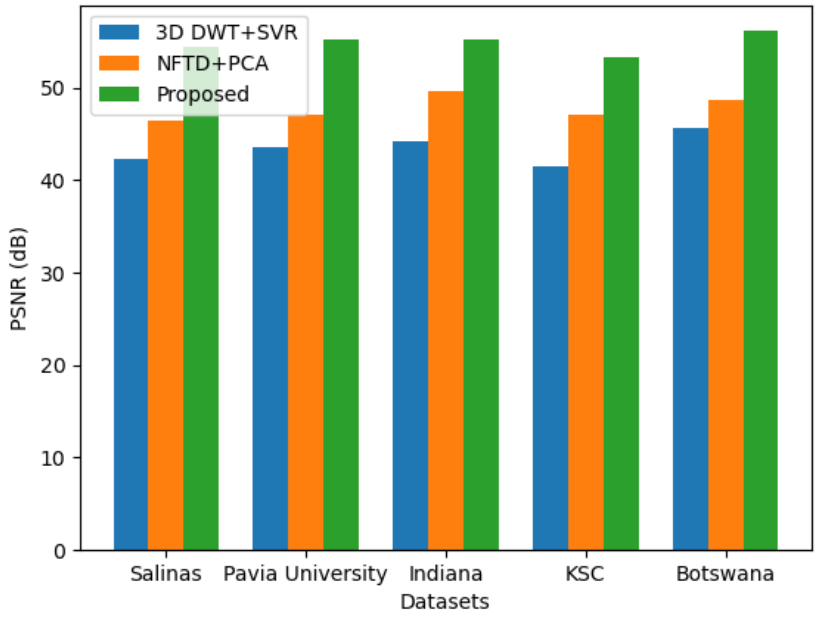

Figure 5.9: Comparison of PSNR with state of the art algorithm

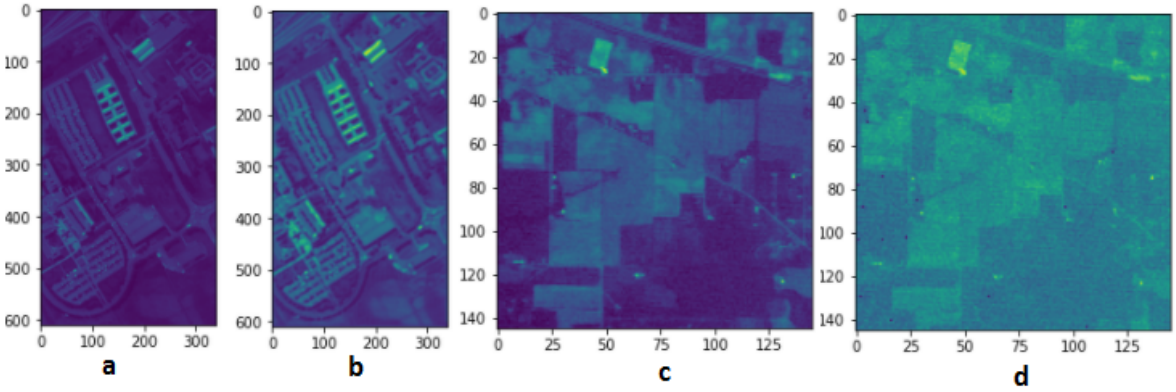

Figure 5.10: 7^{th} band of a) Original PU, b) Decompressed PU, c) Original IP, d) Decompressed IP

5.1.3.4 Comparison Results with Classification Accuracy

The HSI has been compressed with a deep learning model that reduces its dimension even up to 32 bands, where each band is left with only a few pixels, and the original image has been reconstructed from that depth using the decoder model. In this experiment, we evaluate the effect of compression on the inherent features of the image by classifying the original and reconstructed image using the state of the art HSI classification algorithm (HybridSN) proposed in [184]. The authors claim to have obtained more than 99.5% accuracy for some datasets. So, we classify our original image with this algorithm, compress it using the method proposed in this work, and again classify the reconstructed image with the same algorithm.

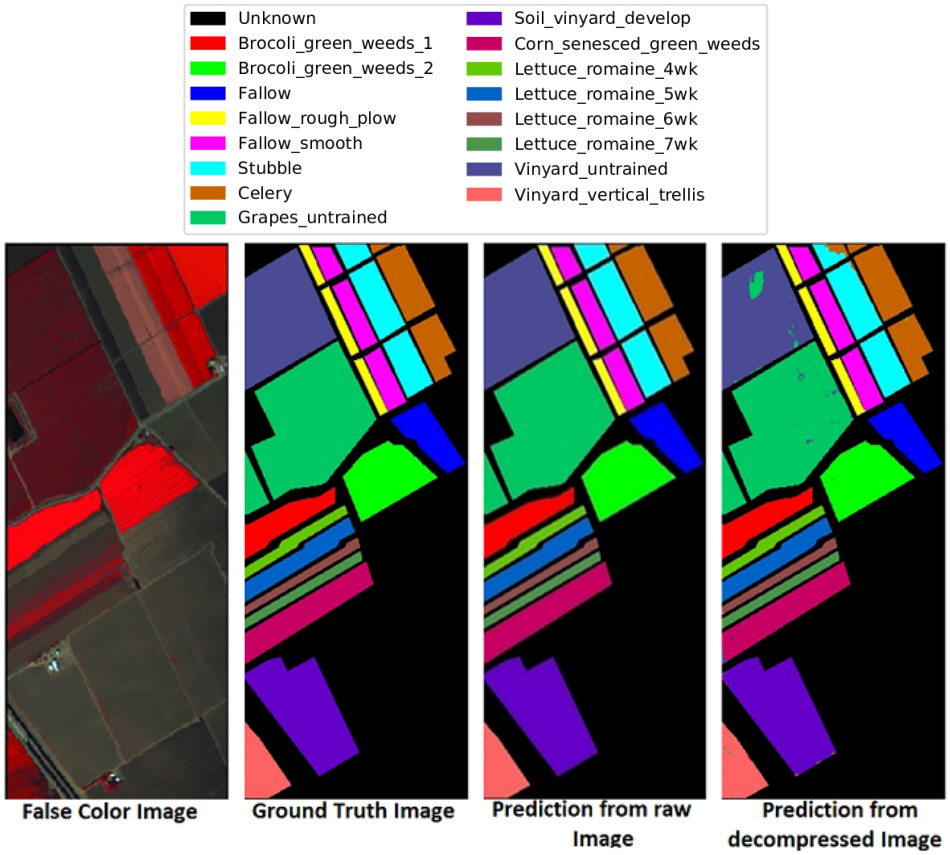

Figure 5.11: Classification results for Salinas dataset

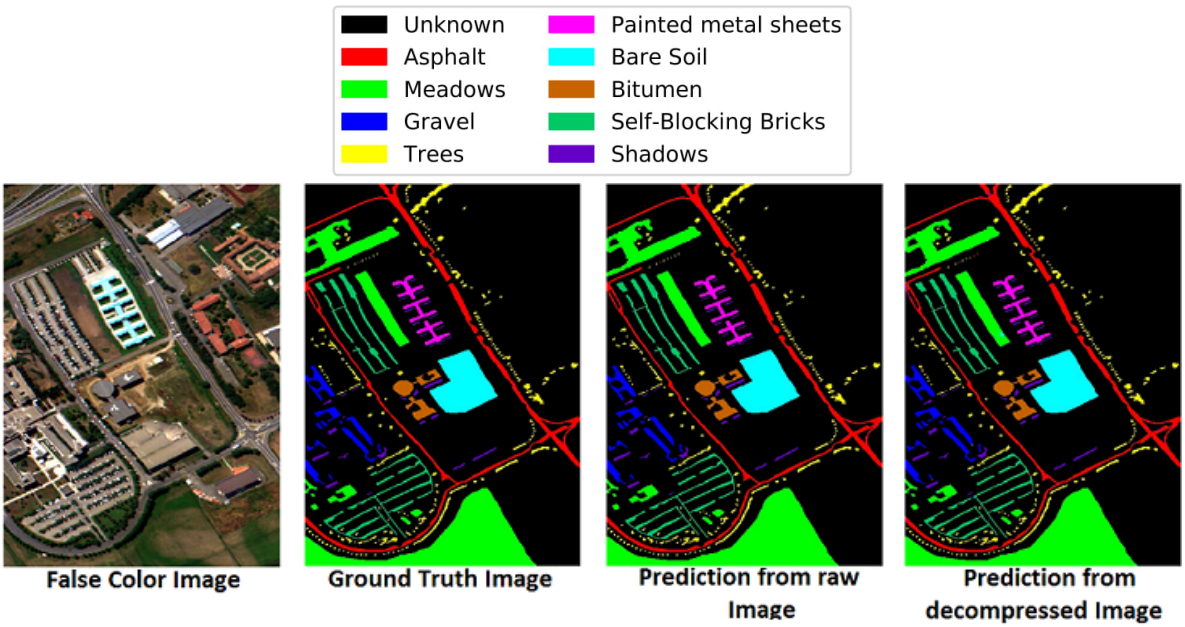

Figure 5.12: Confusion matrix for original Pavia University dataset

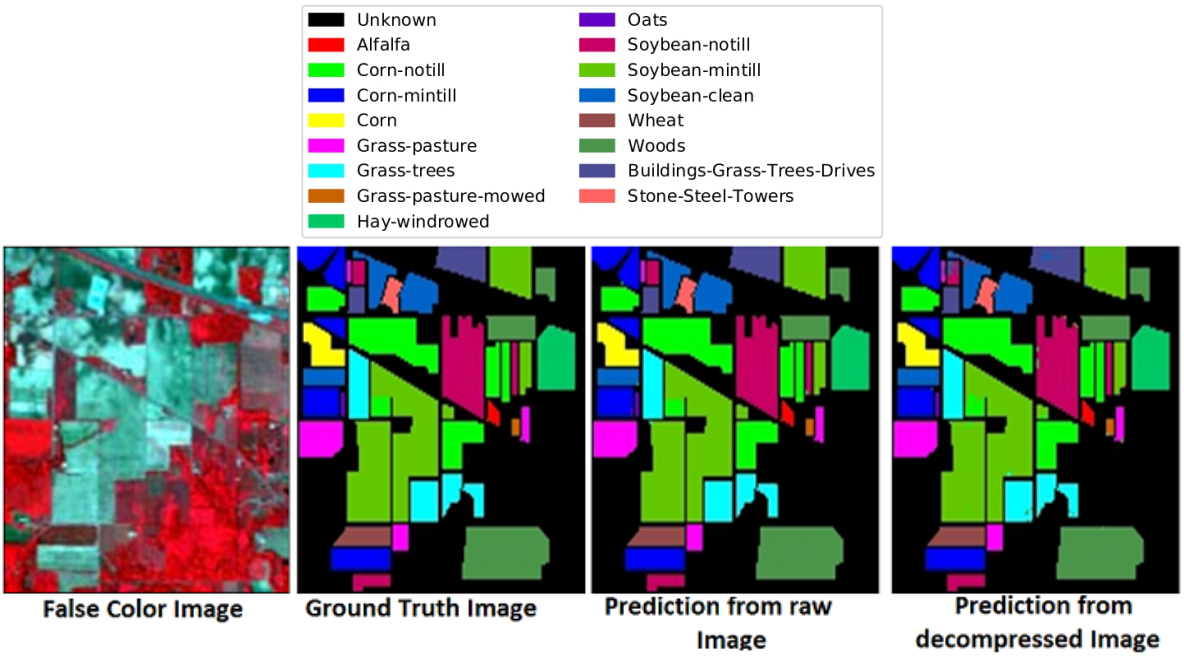

Figure 5.13: Confusion matrix for reconstructed Indian Pines

Experiment 1: This experiment evaluates the Overall Accuracy (OA), Kappa accuracy (KA), and the Average Accuracy (AA) for original and reconstructed image

obtained by the HybridSN algorithm for all the datasets. Results are shown in Table 5.3, where it can be observed that for Pavia University data, the difference between OA of original and reconstructed image is less than 0.2%. The maximum deviation of 1.78% is obtained in OA of the KSC dataset due to sparse location of classes in the ground truth. The pictorial representation of the false-color image, ground truth image, predicted image from original HSI, and image predicted from reconstructed HSI can be found in Figure 5.11 for Salinas, Figure 5.12 Pavia University, and Figure 5.13 for Indian pines dataset.

Table 5.3: Classification results for Original and reconstructed HSI datasets

DatasetAttributes	Overall Accuracy (%)		Kappa Accuracy (%)		Average Accuracy (%)	
	Original	Reconstructed	Original	Reconstructed	Original	Reconstructed
SALINAS	99.9393	98.5221	99.9324	98.3538	99.8692	98.9625
PAVIA UNIVERSITY	99.9432	99.7495	99.9247	99.6681	99.8646	99.6286
INDIAN PINES	99.6516	98.9562	99.6027	98.9929	99.8022	99.2107

Experiment 2: This experiment examines the effect of compression on the class-wise Accuracy of different images. The results of Salinas' dataset are given in Table 5.4, Pavia University Data are represented in Table 5.5, and Table 5.6 presents IP results. Classification results are obtained by implementation of HybridSN technique on original and decompressed images of Pavia University, Indian Pines, Salinas, Botswana and KSC datasets. Name and support values of the classes are available as a part of the dataset on [47]. It gives an in-depth analysis of the classes that have been affected by compressing HSI with such a large compression ratio. The precision of Asphalt and Gravel class has reduced to 0.99 and 0.98, respectively.

Table 5.4: Class-wise analysis of Original and reconstructed Salinas dataset

Classes	Support	Original			Reconstructed		
		Precision	Recall	F1-score	Precision	Recall	F1-score
Brocoli_green_weeds_1	1406	1	1	1	1	1	1
Brocoli_green_weeds_2	2608	1	1	1	1	1	1
Fallow	1383	1	1	1	1	1	1
Fallow_rough_plow	976	1	0.98	0.99	1	0.99	1
Fallow_smooth	1875	0.99	1	0.99	1	1	1
Stubble	2771	1	1	1	1	0.95	0.97
Celery	2505	1	1	1	0.95	1	0.97
Grapes_untrained	7890	1	1	1	0.97	0.99	0.98
Soil_vinyard_develop	4342	1	1	1	1	0.99	0.99
Corn_senesced_green_weeds	2295	1	1	1	0.99	0.98	0.99
Lettuce_romaine_4wk	748	1	1	1	1	1	1
Lettuce_romaine_5wk	1349	1	1	1	1	0.99	0.99
Lettuce_romaine_6wk	641	1	1	1	0.98	1	0.99
Lettuce_romaine_7wk	749	1	1	1	0.99	1	0.99
Vinyard_untrained	5088	1	1	1	0.98	0.96	0.97
Vinyard_vertical_trellis	1265	1	1	1	1	0.98	0.99

Table 5.5: Class-wise analysis of Original and reconstructed Pavia University dataset

Classes	Support	Original			Reconstructed		
		Precision	Recall	F1-score	Precision	Recall	F1-score
ASPHALT	4642	1.00	1.00	1.00	0.99	1.00	1.00
MEADOWS	13055	1.00	1.00	1.00	1.00	1.00	1.00
GRAVEL	1469	1.00	1.00	1.00	0.98	0.99	0.99
TREES	2145	1.00	1.00	1.00	1.00	0.99	1.00
PAINTED METAL SHEETS	942	1.00	1.00	1.00	1.00	1.00	1.00
BARE SOIL	3520	1.00	1.00	1.00	1.00	1.00	1.00
BITUMEN	931	1.00	1.00	1.00	1.00	0.99	1.00
Self-Blocking BRICKS	2577	1.00	1.00	1.00	1.00	0.99	0.99
SHADOWS	663	1.00	1.00	1.00	1.00	1.00	1.00

(a) Original image

(b) Reconstructed image

Figure 5.14: Confusion matrix for Pavia University dataset

Table 5.6: Class-wise analysis of Original and reconstructed Indian Pines data

Classes	Support	Original			Reconstructed		
		Precision	Recall	F1-score	Precision	Recall	F1-score
Alfalfa	32	1	1	1	0.97	0.97	0.97
Corn-notill	1000	0.99	0.99	0.99	0.98	0.95	0.94
Corn-mintill	581	1	0.99	1	1	0.96	1
Corn	166	1	1	1	1	1	1
Grass-pasture	338	1	1	1	1	0.99	1
Grass-trees	511	1	1	1	1	1	1
Grass-pasture-mowed	20	1	1	1	1	1	1
Hay-windrowed	335	1	1	1	0.94	1	1
Oats	14	0.78	1	0.88	0.77	0.89	0.87
Soybean-notill	680	1	1	1	1	1	1
Soybean-mintill	1719	0.99	1	1	0.99	1	1
Soybean-clean	415	1	1	1	1	1	1
Wheat	143	1	1	1	0.99	1	1
Woods	886	1	1	1	1	1	1
Buildings-Grass-Trees-Drives	270	1	1	1	1	1	1
Stone-Steel-Towers	65	0.98	1	0.99	0.96	0.98	0.97

Experiment 3: It is also essential to observe the individual misclassification rate from the reconstructed and original image. Figure 5.14 gives a view of the confusion matrix obtained from the classification of original and reconstructed/ decompressed HSI using HybridSN.

5.1.4 Conclusion

In this chapter, deep learning based compression has been used to reduce the size of HSI. It is a lossy compression technique that used the concept of autoencoder to extract the features of an HSI and store it into a small-sized 3D tensor. Original size image can be reconstructed using the decoder that reverses the steps of CNN using the transpose convolution layer. The compressed image can be transmitted to the decoder at a different location after entropy coding the pixels using AAC that generated the stream of bits. Bit-streams can also be used to store the information of an HSI that can be regenerated at any point of time using the decoder. The experiment results proved that the method outperforms the existing state of the art algorithms in terms of compression ratio and PSNR. The proposed model is generally applicable to the HSI of similar sensors, for other datasets the network should be trained again. The weight parameters of the network training have been considered during evaluation to avoid any impact of retraining on the performance. The experiment results, as observed, vary for different datasets due to differential properties of each image. The effect of compression on classification performance has also been evaluated by comparing the OA, KA, and AA obtained by classification of the original and reconstructed image using HybridSN algorithm. A minimum drop of 0.2% and a maximum drop of 1.78% in OA of the original and reconstructed image is observed that is negligible since there is no significant change in class-wise accuracy.

5.2 WaveTraNet: HSI Compression Model based on Wavelet Transform and Convolution Neural Network

** The content of this chapter has been submitted for publication by Yaman Dua, Ravi Shankar Singh, and Vinod Kumar "WaveTraNet: A HSI compression model based on Wavelet Transform and ConvNet." IEEE Transactions on Signal Processing (IEEE, IF: 4.9)*

5.2.1 Introduction

In this work, we propose a controlled lossy compression technique that consists of wavelet decorrelation followed by CNN prediction. Our primary goal is to develop an efficient lossy algorithm and improve its performance in terms of total bits required to store compressed image (compression ratio) and decompressed image quality. The coding algorithm consists of wavelet transform, quantization step, feature extraction, and loss minimization. Simulation results prove the efficacy of our approach in terms of various parameters on three widely used publicly available datasets. Since the application of HSI is far from the human psycho-visual system, we analyze the efficiency of decompressed images from the proposed technique on classification accuracy using Hybrid-SN model proposed in [184].

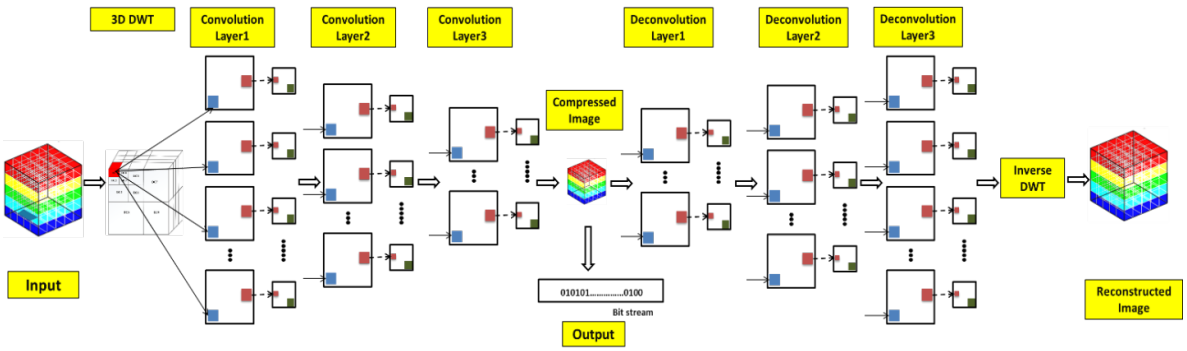

Figure 5.15: Framework of the proposed algorithm

5.2.2 Proposed Algorithm

We consider a HSI ($H=\langle X_0, X_1, ..., X_j\rangle$) as a set of bands X_j, $j=0, ..., N_C-1$, where each band X_j is a set of array of pixels $X_j=[X_j(0,0), ..., X_j(x,y), ..., X_j(N_H-1, N_W-1)]$ at spatial coordinates (x, y). The pixels in band X_j are modeled as occurrences of an identically distributed random process r_j. Since most of the hyperspectral sensors generate integer data of 16 or 12-bit per pixel, a discrete-valued random process r_j is assumed to take values from alphabet $\chi=\{0, 1, ..., 2^{16}-1\}$ with cardinality $|\chi|=2^{16}$. 16-bit signed integers can be represented by adding -2^{-15} to each sample, which maps to values from $\chi=\{-2^{-15}, ..., 0, 1, ..., 2^{15}-1\}$. Symbols resemble matrices unless stated. Figure 5.15 represents the framework of the proposed Wavetranet model. It comprises of two major parts: an encoder $\mathfrak{F}:\mathbb{R}^N\mapsto\mathbb{R}^M$, and a decoder $\mathfrak{G}:\mathbb{R}^M\mapsto\mathbb{R}^N$: The compressor consists of seven steps*: wavelet transform, quantization, normalization, convolution layer, activation layer, pooling layer, and loss minimization.*

Wavelet Transform

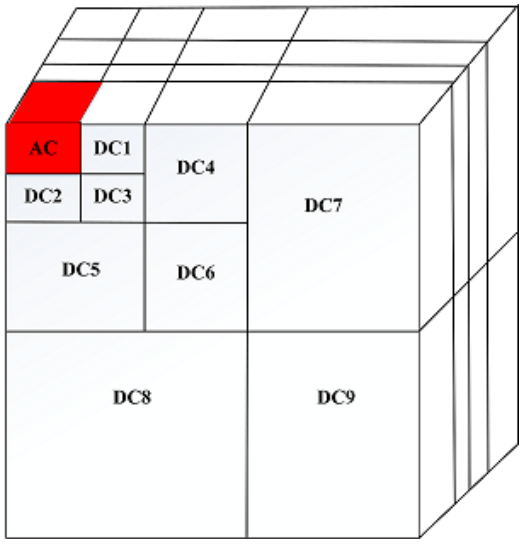

Figure 5.16: Three level decomposition of DWT coefficients

A Discrete Bi-orthogonal Wavelet Transform of a function $p(t)$, represented by the coefficients $\hat{p}_{m,n}$, is its decomposition on expansions and translations of a mother func-

tion $\bar{\psi}(t)$, such that:

$$\psi_{m,n}(t) = 2^{-\frac{m}{2}}\psi(2^{-m}t - n) \tag{5.5}$$

$$\hat{p}_{m,n} = \hat{p}(2^m, n2^m) = \int_{-\infty}^{\infty} 2^{-\frac{m}{2}}\bar{\psi}(2^{-m}t - n)p(t)dt \tag{5.6}$$

In terms of images, filter banks of DWT converge to a continuous wavelet bases and provide a set of approximate ($a_n^{(s)}$) and detail coefficients ($d_n^{(s)}$) with the help of low and high pass filter banks, $f(u)$ and $g(u)$ respectively [185], and $\{\psi_{m,n} : m, n \in \mathbb{Z}\}$. These coefficients can be calculated using equation 5.7,5.8:

$$a_n^{(s+1)} = \sum_{u=0}^{N-1} f_u a_{2n-u}^{(s)} \qquad s = 1, 2, ..., J \tag{5.7}$$

$$d_n^{(s+1)} = \sum_{u=0}^{N-1} g_u a_{2n-u}^{(s)} \qquad a_n^{(0)} = p_n \tag{5.8}$$

Split level is represented by s, J being its maximum limit, N represents the total sub-bands generated.

The basic principle of wavelet transform used in our approach is that splitting the signal into approximation and detail coefficients achieves efficient decorrelation. These two half subsequences represent low-pass and high-pass frequency components of the spectrum respectively. It has been observed that in all real-world signals, including HSIs, most of the higher energy coefficients are condensed around lower frequency component and vice versa. The split process can be iteratively applied over this significant component with the help of filter bank scheme that results in good energy compaction. This concept gives rise to the concept of coding gain in wavelet analysis, i.e. should there be a general threshold on the level of split iterations for all signals or the selection is an adaptive process? Coding gain is a performance metric to measure the efficiency of transforms [186]. The objective function for 's' decomposition levels, $G_{s\to(s+1)}$ is

defined by equation 5.9:

$$G_{s\to(s+1)} = \frac{\sigma_s^2}{\left(\prod_{u=1}^{N} \sigma_{s+1,u}^2\right)^{\frac{1}{N}}} \tag{5.9}$$

where σ_s^2 denotes the variance of transformed coefficients in a sub band at split level s, and $\sigma_{s+1,u}^2$ denotes the variance of $u = 1, .., N$ subbands. $\prod_{u=1}^{N}$ denotes the geometric mean of variances of subband. If $G_{s\to(s+1)} \leq 1$, current split at s is retained otherwise $s+1$ level is accepted.

Frequency localization capability of a wavelet filter ($f(u)$ with unit energy (equation 5.10) is measured by the mean square difference between its spatial width, according to equation 5.11:

$$\sum_u |f(u)|^2 = 1 \tag{5.10}$$

$$\sigma_t^2 = \sum_u (u-\bar{u})^2 |f(u)|^2, \quad \text{where } \bar{u} = \sum_u u|f(u)|^2 \tag{5.11}$$

The DWT inherently has sound spatial frequency localization in both low and high frequency sub bands, as required for a high coding gain. To a larger extent it still depends on the filter's choice for perfect reconstruction. The term 'sub band' in our scenario is a three dimensional tensor representing splits at various level. We selected Daubechies filter (db1) filter in our model which is asymmetric, orthogonal, and bi-orthogonal. Split level is analyzed as a trade-off between quality of reconstructed image and the compression ratio, and optimum performance was obtained at $s = 3$, i.e. we employ 3D wavelet decomposition.

Quantization

The approximate coefficients (AC) ($a_n^{(s)}$) and detail filter coefficients (DC) ($d_n^{(s)}$) of 3D-DWT obtained from the previous step are in the floating-point format. A total of 10 different sub bands are generated after three successful split levels out of which nine sub bands represent DC, three from each division and remaining one represent AC from third division (see Figure 5.16). Quantization follows a different rule for AC and DC.

All the coefficients of the former one are retained, but rounded off to nearest integer by following the given Equation 5.12. While the strategy used to quantize the latter one is to replace values greater than a specific threshold τ_n by 1 and discard other values by replacing it with zero. In [187], threshold function for 2D transform is provided that can be easily extended for 3D DWT in our case. Equation 5.13 represents the process mathematically.

$$a_n^{(s)} \longmapsto A_n^{(s)} \in \mathbb{Z}^+ \tag{5.12}$$

$$d_n^{(s)} \longmapsto D_n^{(s)} \in \{0, 1\} \tag{5.13}$$

$$D_n = \begin{cases} 1, & \text{if } \ d_n > \tau_n \\ 0, & \text{otherwise} \end{cases}$$

$$\tau_n = \frac{\Theta(W(x, y, z))}{4 \cdot \Theta(x, y, z)}$$

$\Theta(\cdot)$ is a function that gives a measure of wavelet coefficients generated.

Quantization generates a sparse matrix with very few values compared to transformed coefficients. It is represented as:

$$Q[x, y, z] = [A_n, D_n] \tag{5.14}$$

$[\cdot, \cdot]$ represents concatenation operation.

The quantization step is followed by normalization to make the data fit in Gaussian curve within a range $(-1, +1)$. It is done mainly on the matrix Q to make mean $\mu[Q]$ and variance $\sigma^2[Q]$ as 0 and 1 respectively. Equation 5.15 represents the term mathematically:

$$Q_{norm}[x, y, z] = \frac{Q[x, y, z] - \mu[y])}{\sigma[y]} \tag{5.15}$$

$Q[x, y, z]$ = quantized input image, $Q_{norm}[x, y, z]$ = normalized image, $\mu[y]$ = arithmetic mean of the elements of the row (y), $\sigma[y]$ = Standard deviation of the elements of the row (y), where $y = \{0, 1, ..., N_C - 1\}$. Normalization helps to efficiently train the neural network model discussed in the next subsection. Normalized cube will act as an input to the 2D convolutional neural network.

Convolution Neural Network

A CNN represents sub-class of feed forward interconnected neural network, which comprises of a series of convolutional and pooling layers to extract the main features from the images responding the best to the final objective. It is a multilayer perceptron, specially designed to identify 2D image features. Each neuron, locally connected, learns the local feature by weight-sharing mechanism using the same convolution kernel. Kernel is a small matrix of integers containing receptive fields that operate upon each pixel using algebraic multiplication generate a feature map. Another important element of CNN is the bias vector that doesn't depend upon the input data. It corresponds to the output of a network when it has zero input. It helps to implement complex logic gates in the network. The deep CNN architecture learns by updating the parameters of all the layers in the network iteratively through an optimizer.

2D Convolution Layer

The convolutional product, represented by $(\circledast)$, of an image and kernel, is a 2D matrix where each element is the sum of the element-wise multiplication of the cube (filter) and the subcube of the given image. In the usual 2D-Convolution, if input data is 3D, then it will be convoluted with a 3D-kernel. The number of bands in the kernel is the same as the number of bands in the input. The output of the 2D-Convolution is always a series of 2D image or matrix. Number of kernels used in convolution determine the third dimension of the output image. Mathematically, the l^{th} convolution layer can

be denoted as: $\forall n \in [1, 2, ..., N_C^l]$:

$$(q^{l-1} \circledast K^n)_{x,y} = \Psi^l\left(\sum_{i=1}^{N_H^{l-1}} \sum_{j=1}^{N_W^{l-1}} \sum_{k=1}^{N_C^{l-1}} K_{i,j,k}^n \cdot q_{x+i-1,y+j-1,k}^{l-1} + b_n^l\right)$$

$$q^l = \left[\Psi^l(q^{l-1} \circledast K^1), \Psi^l(q^{l-1} \circledast K^2), ..., \Psi^l(q^{l-1} \circledast K^{(N_C^l)})\right] \tag{5.16}$$

where q^{l-1} is the input for l^{th} layer with size $(N_H^{l-1}, N_W^{l-1}, N_C^{l-1})$, q^0 is the original input image, K^n is the matrix of 2D kernel of size (f^l, f^l) with N_C^{l-1} entries, b_n^l is the bias for n^{th} convolution, and Ψ^l is the activation function defined in the following subsection. An output matrix q^l of size (N_H^l, N_W^l, N_C^l) is obtained post convolution at l^{th} layer.

Activation Layer (ReLU)

The activation function is the nonlinear transformation over the input signal. This transformed output is then transmitted to the next layer of neurons as input. It is generally placed after convolution layer to decide if neuron would fire the output or not. It does not modify the shape of the input (q and r have the same size), it is a truncation performed individually for every element in the input. Rectified Linear Unit (ReLU) is most widely used activation function that converts all negative inputs to zero and do not activates the neuron in such cases. Also, it is computationally efficient as all the neurons are not activated at the same time and is six times faster than its counterparts (tanh and sigmoid).

$$r_{i,j,k} = \max\{0, q_{i,j,k}^l\} \tag{5.17}$$

with $0 \leq i < N_H^l = N_H^{l+1}$, $0 \leq j < N_W^l = N_W^{l+1}$, and $0 \leq k < N_C^l = N_C^{l+1}$. No learning is required in ReLU layer, since it doesn't contain any parameters. Therefore, the above equation 5.17 becomes:

$$\frac{dr_{i,j,k}}{dq_{i,j,k}^l} = [\![q_{i,j,k}^l > 0]\!] \tag{5.18}$$

where $[\![\cdot]\!]$ represents an indicator operation, its value is one if the argument returns true and 0 otherwise. Hence,

$$\left[\frac{\partial z}{\partial q^l}\right]_{i,j,k} = \begin{cases} \left[\frac{\partial z}{\partial r}\right]_{i,j,k}, & \text{if } \; q^l_{i,j,k} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{5.19}$$

The image contains semantic information, which is a non-linear mapping of the input values of a pixel. ReLU, a non-linear function, is utilized to improve the non-linearity property of CNN to effectively map the highly non-linear semantic input to output feature. If we treat $q^l_{i,j,k}$ as one of the $N^l_H \times N^l_W \times N^l_C$ features extracted by CNN layers 1 to $l-1$, which can be positive, zero or negative. If a specific region inside the input image has certain patterns then $q^l_{i,j,k}$ may be positive and if the region does not contain those patterns, it will either be zero or negative. In this case, the ReLU layer will keep all negative values to 0, i.e., activate $r^l_{i,j,k}$ only if the image possess that pattern in particular region. Instinctively, this property is useful for recognizing complex patterns and objects.

Pooling Layer (Maxpool)

Pooling is a local operator, and its forward computation is pretty straightforward. A pooling layer operates upon each channel in q^l independently. It requires no parameter (w^i is null, hence parameter learning is not required at this layer). Let $q^l \in \mathbb{R}^{N^l_H \times N^l_W \times N^l_C}$ be the input to the l^{th} pooling layer, $(N_H \times N_W)$ is pre-specified spatial limit of the pooling, such that N_H divides N^l_H, N_W divides N^l_W and the stride equals spatial extent. The pooling layer's output (r or equivalently q^{l+1} will be an order 3 tensor of size $N^{l+1}_H \times N^{l+1}_W \times N^{l+1}_C$, with:

$$N^{l+1}_H = \frac{N^l_H}{N_H}, \qquad N^{l+1}_W = \frac{N^l_W}{N_W}, \qquad N^{l+1}_C = N^l_C \tag{5.20}$$

In each band, the 2D matrix with $N^l_H \times N^l_W$ elements gets split into $N^{l+1}_H \times N^{l+1}_W$ non-

overlapping sub-regions, where each region is of $N_H \times N_W$ size. The pooling function maps each subregion into a corresponding single number. Maxpooling operator is utilized in our model that projects a subregion to its maximum value. Mathematically, this can be represented as,

$$r_{i^{l+1},j^{l+1},k} = \max_{0<=i<N_H, 0<=j<N_W} q^{l}_{i^{l+1}\times N_H+i,\ j^{l+1}\times N_W+j,\ k} \tag{5.21}$$

where $0 <= i^{l+1} < N_H^{l+1}$, $0 <= j^{l+1} < N_W^{l+1}$, and $0 <= k < N_C^{l+1} = N_C^l$.

Loss Minimization

The next step in compression model is loss minimization using back propagation. Mean squared error (MSE) is an optimal measure of oversight in a multi-dimensional HSI input. Back propagation is a technique is which the difference between predicted value and desired output is transmitted back to the neurons of consecutive layers to update the weight matrix and bias. Each iteration follows back propagation until the MSE reduces to a target threshold (in order of 10^3). Mathematically, this objective is defined using a cost function $J(\theta)$. Gradient of $J(\theta)$ updates the parameter in its reverse direction $\nabla_\theta J(\theta)$. Equation 5.22 defines the objective function:

$$\min_{w_j,q'} J(\theta) = \frac{1}{2}\sum_{j=1}^{l}(q_j - \hat{q}_j)^2 \tag{5.22}$$

where $\hat{q}_j = w_j^T, q'$ is the j^{th} output of the network with $w_j = [w_{j1}\ w_{j2}\ ...\ w_{ji}\ ...\ w_{jc}]^T$ and $q' = [q'_1\ q'_2\ ...\ q'_i\ ...\ q'_c]^T$

Optimization algorithm defines the criterion to update the network parameters (set of weights and biases) to obtain optimal results. It can be done either by minimizing or maximizing the cost function in search of the best candidate solution having lowest or highest score. Typically, it is to reduce the error. In our case, the well-known "Adam optimization" algorithm proves to be the perfect choice due to certain properties:

1. In each iteration, it follows intuitive learning by considering the actual step size

which is approximately bounded to the step size of hyper-parameter.

2. It is well suited to the areas with very small gradients as the step size is invariable of gradient's magnitude.
3. It is designed to function for broader range of tasks as it has the benefits of RMSprop, Adagrad and SGD with momentum. It works well for sparse gradients and applications with online settings.

Adam: The weight update function depends on the mean and variance of gradient of objective function, first moment and second moment, respectively. The average of past gradient ($\mathfrak{a}_t$) and past squared gradient ($\mathfrak{u}_t$) are exponentially decaying values calculated as follows:

$$\mathfrak{a}_t = \beta_1 \mathfrak{a}_{t-1} + (1-\beta_1) d_t \qquad \mathfrak{u}_t = \beta_2 \mathfrak{u}_{t-1} + (1-\beta_2) d_t^2 \tag{5.23}$$

where β_1 and β_2 represent learning rate adaptively chosen according to the convergence. When β_1 and β_2 are close to zero, the $\mathfrak{a}_t$ and $\mathfrak{u}_t$ are biased towards zero. It is updated with modified bias term to vanquish this condition as:

$$\hat{\mathfrak{a}}_t = \frac{\mathfrak{a}_t}{1-\beta_1'} \qquad \hat{\mathfrak{u}}_t = \frac{\mathfrak{u}_t}{1-\beta_2'} \tag{5.24}$$

Using equation 5.23 and 5.24, the Adam weight upgrade rule is given by:

$$\Theta_{t+1} = \Theta_t - \frac{\eta}{\sqrt{\hat{\mathfrak{u}}_t + \epsilon}} \hat{\mathfrak{a}}_t \tag{5.25}$$

Final predicted image obtained as output from the convolution network forms our compressed image along with the model parameters. This reduced image can either be converted into one-dimensional vector using row wise shifting or stored/transmitted as three dimensional tensor. No significant impact could be observed, either way, on the compression performance, but complexity improved at the encoder end due to the

unnecessary conversion of tensor to vector. So, the suggestible approach allows 3D input to the decompressor having three steps: *Deconvolution layer, de-normalization, and inverse wavelet transform.*

Deconvolution Layer

Deconvolution is a mathematical operation that reverses the effect of convolution. Convolution networks follow a bottom-up approach in which a series of convolution, activation, and pooling layer process the input signal. In contrast deconvolution layer follows top-down approach which attempts to reconstruct input signal by performing convolution over feature maps and learned filters/kernels. A multi-component deconvolution problem is to be solved for inferring feature maps from the given set of filters and input [188].

In order to explain our deconvolution network, let us consider an input image r^i, composed of K_0 channels $r^i_1, ..., r^i_{K_0}$, which is applied to a single deconvolution layer. Each channel c of r^i is represented as a linear sum of K_l convolution of latent feature maps z^i_k and filters $\gamma_{k,c}$ using equation 5.26.

$$\sum_{k=1}^{K_l} z^i_k \circledast \gamma_{k,c} = r^i_c \tag{5.26}$$

If r^i_c is an $N_H \times N_W \times N_C$ image and the size of each filter is $H \times H$, then size of latent feature maps are $(N_H + H - 1) \times (N_W + H - 1)$. It do not contain sparsity, as equation 5.26 being an under-determined system. A regularization term need to be introduced on z^i_k to generate a unique solution. The overall cost function after regularization leads to:

$$C_l(r^i) = \frac{\lambda}{2} \sum_{c=1}^{N_C} \| \sum_{k=1}^{K_1} z^i_k \circledast \gamma_{k,c} - r^i_c \|^2_2 + \sum_{k=1}^{K_l} |z^i_k|^P \tag{5.27}$$

Equation 5.27 provides feature maps with sparsity for a single layer multi-band image that can be accumulated for a multi-layer architecture. It can be achieved by considering l^{th} layer feature maps as input for $l + 1$ layer. As the input for l^{th} layer

is supposed to be an image with K_{l-1} channels, but in this case it will be number of feature maps at layer $l-1$. A generalized cost function C_l for layer l will be given by:

$$C_l(r) = \frac{\lambda}{2}\sum_{i=1}^{l}\sum_{c=1}^{K_{l-1}}\|\sum_{k=1}^{K_1}\Omega_{k,c}^{l}(z_k^i \circledast \gamma_{k,c}^i) - z_{c,+l-1}^i\|_2^2 + \sum_{i=1}^{l}\sum_{k=1}^{K_l}|z_{k,l}^i|^P \tag{5.28}$$

where $z_{c,l-1}^i$ represent the previous layer feature maps, and $\Omega_{k,c}^l$ represent the connectivity among successive layer feature maps through a fixed binary matrix, i.e. if $z_{k,l}^i$ and $z_{c,l-1}^i$ are connected or not. We assume that $\Omega_{k,c}^l$ in layer $l=-1$ is always 1, but sparse in other higher layers. Bottom-up method is used for training the hierarchy, it means learnings on $C_{l-1}(r)$ is given as a result to $z_{c,l-1}^i$. We reconstruct r', after successful convergence of $z_{k,l}$, by back projection of the image via $\gamma_{k,c}^l$ and $\gamma_{k,c}^{l-1}$ using the equation 5.29:

$$\bar{r}' = \sum_{k=1}^{K_{l-1}}\Omega_{k,c}^{l-1}(\sum_{b=1}^{K_l}\Omega_{b,k}^{l}(z_{b,l} \circledast \gamma_{b,k}^l)) \circledast \gamma_{k,c}^{l-1} \tag{5.29}$$

Inverse Wavelet

Inverse discrete wavelet transform (IDWT) is the process of reconstructing original HSI back with the help of coefficients obtained after quantization (in our case predicted output of deconvolution network) and a set of similar filter banks. The first step towards inverse calculation is to obtain denormalized image with pixel values in 16-bit integers. It is followed by inverse calculation in terms of the inverse of equation 5.6 using below equation 5.30. The image obtained post inversion represents the final decompressed/reconstructed image used for evaluation.

$$p(t) = \sum_m\sum_n \hat{p}_{m,n}\psi(2^{-m}t - n) \tag{5.30}$$

GPU Optimization

The computational complexity of CNN is a hurdle in its general acceptance besides

high performance in most of the applications. Two significant reasons for gradual increment in execution time are: depth of the network (large number of hidden layers) and large size of training data in multidimensional arrays, multichannel texts, images, videos. Researchers suggest various methods to reduce the complexity, in turn, decrease training time [189]. The most popular technique is to accelerate training using GPUs by exploiting the multi-level parallelism existing in the training process and massive floating point operations. The computing model of GPU is suitable for such computations, which led to the development of multiple open-source libraries for CPU-GPU and multi-GPU accelerations. Our model contains few convolution layers and small training data that enabled us to optimize the network using simple GPU modifications. We utilize GPU for hyper-parameter tuning along with training of the model for a set of epochs until a stable loss could be obtained.

Classification Technique

HSI classification is one of the widely used image processing tasks to label every pixel in an HSI. The quality of the decompressed image cannot be fully perceived with human vision system and its related parameters. This concept gives rise to the process of application-oriented analysis of the compression system. In this work, we compare the accuracy of reconstructed HSI with that of original HSI through the classification model as presented in [184]. It proposes a hybrid model by assembling 3D-CNN and 2D-CNN layers to achieve maximum possible accuracy by utilizing the spatial and spectral feature map to their full extent. Readers should refer to the original article [184] for more insights on the technique. To illustrate our simulation results on classification, we utilize three evaluation parameters, namely, OA, AA, κ accuracy.

To train our CNN compression module, loss and accuracy have been selected as two important criterions to obtain better performance in terms of reconstructed image quality. Compression ratio is considered to be a dependent variable that relies upon MSE loss function, which in turn can be modified by updating the network's parameters.

The size of filters considered to compress and decompress HSI in our neural network depends upon number of bands and total number of pixels per band. It provides additional benefit to the results observed in terms of compression ratio (CR), structural similarity index (SSIM), MSE, normalized root mean square error (NRMSE) and PSNR. The proposed method works well for HSIs with spectral and spatial resolution of 400 to 2500 nm and 1.3 m to 30 m respectively, represented by 16-bit depth per pixel. Details about dataset used to evaluate the performance of proposed model are given in next section.

5.2.3 Experiments

The datasets considered for the experiments are three open source HSI namely, SA, IP, and PU. Table 5.7 contains description of these datasets. We have performed all the simulations on a personal computer powered by Windows 10 platform having Intel(R) Xeon(R) CPU @ 1.90 GHz with four cores and a 16 GB RAM, and NVIDIA Tesla P100 GPU with stacked memory capacity of 16 GB and 21 teraFLOPS of 16-bit floating-point (FP16) performance. In order to determine the relative performance of the proposed technique, we compare the results with state-of-the-art learning based compression algorithms including SVR+DWT and autoencoder compression model.

Table 5.7: Description of the dataset used to evaluate the algorithm

Dataset	Number of rows	Number of columns	Number of bands
Salinas	512	217	204
IP	145	145	200
PU	610	340	103

Most of the techniques in existing literature trim the input HSIs to a fixed shape in the pre-processing phase to make compression efficient that may result in loss of critical information around edges. The model proposed in this work accepts images of all shape

and size, to avoid any data leakage. All the simulations have been performed without manipulating the original shape of the input images to keep the model realistic. Our first experiment evaluates the trade-off between reconstruction error and coding gain by changing split level (s) of DWT. Three level DWT selected as the optimum choice for all three datasets. The next step is to train the model by analyzing the error as a function of total number of epochs. It is experimentally found after tuning the model that it gives optimum performance around 210 epochs. We observe from the training graph that training loss start reducing gradually after 20 epochs and attained a static level around 210 epochs after which no significant improvement could be observed, so 210 epochs is considered to be an optimal choice.

Analysis using Salinas dataset

The training loss for the model calculates to 0.1951% for SA data. The following experiment evaluates the relative compression performance of proposed technique with other techniques using compression ratio and quality of reconstructed image using four parameters. We can observe from Table 5.8, that our approach outperforms both SVR and Autoencoder compression schemes. A significant increment in CR represents the benefit of using wavelet correlation before CNN that significantly decreases the total number of trainable parameters and shape of compressed image. Our compression model increases the quality of reconstructed image significantly in terms of reference based parameters compared to SVR technique.

In case of SA, compressed image obtained at the end of compression stage has a shape [83×34×13]. Almost every feature of HSI is retained in this lower dimension tensor. To verify our hypothesis, we perform application specific analysis in terms of classification accuracy. It is worth to mention the spatial patch size selected for SA dataset is [15×15] and 30% samples are utilized for training. A minor change in OA of 0.0228%, 0.0254% in Kappa coefficient, and 0.0206% in AA proves that our model can efficiently be utilized in the applications dealing with very delicate accuracy.

Table 5.8: Compression results on different datasets

Data Set	Parameters	SVR+DWT	Autoencoder	Proposed
SA	SSIM	0.9219	0.9855	**0.9897**
	MSE	0.0036	0.0008	**0.0004**
	NRMSE	0.2412	0.1597	**0.1208**
	PSNR	42.28	54.46	**54.58**
	CR	21.53	50.06	**307.46**
IP	SSIM	0.9296	0.9813	**0.9845**
	MSE	0.0046	0.0021	**0.0010**
	NRMSE	0.3719	0.1418	**0.1002**
	PSNR	44.41	54.19	**55.25**
	CR	27.02	49.16	**100.11**
PU	SSIM	0.9116	0.9694	**0.9729**
	MSE	0.0064	0.0025	**0.0021**
	NRMSE	0.3962	0.2506	**0.2234**
	PSNR	43.59	55.25	**55.59**
	CR	25.95	43.66	**374.78**

Analysis using Indian Pines dataset

In this subsection, we analyze our model on IP dataset, which is known for low spatial correlation corresponding to its spectral counterpart. This feature leads to lower CR for our model as compared to other datasets. Numerical results of the experiment are available in Table 5.8, where structural similarity of the proposed technique is far better than that of SVR and Autoencoder. Reconstruction error seems to decrease further by quarter and half respectively. The training loss for the compression model after 210 epochs calculates to 0.4104%, which is comparatively more than SA data but insignificant as a whole.

The shape of the compressed image for the IP dataset is [22×22×13], which stores all the features. On decompression, all the classes can be restored with negligible disturbance. The patch size selected for classification, giving optimal performance, is [25×25] as suggested in HybridSN technique. The difference between prediction by original and decompressed HSIs in terms of OA is 0.404%, Kappa is 0.461%, and AA is 0.53%. A similar impact was seen in terms of class-wise accuracy and predicted image.

Analysis using Pavia University dataset

The simulation results available in Table 5.8 correspond to the performance on PU dataset that contains 103 spectral bands. It provides maximum compression ratio compared to other datasets with relatively higher reconstruction quality in terms of PSNR. Though an insignificant rise is visible compared to autoencoder-based compression, the structural index and error seem promising. Model training loss evaluates to 0.13% after 210 epochs that proves it to be a better data for the network. Shape of the compressed image is [13×74×41], which results in a compression ratio of 374.78.

In another experiment, we evaluate classification accuracy of original and reconstructed PU dataset. Table 5.9 contains the values of OA, Kappa, and AA on multiple datasets. OA of original and reconstructed image differ by 0.5577%, for Kappa it is 0.739%, and AA is 0.714%. Due to limitation in number of pages, we only represent the pictorial view of false color image, decompressed image, ground truth, and predicted image from original and reconstructed data of PU in Figure 5.17. It can be analyzed that the numerical difference in accuracy does not correspond meaningful change to prediction quality.

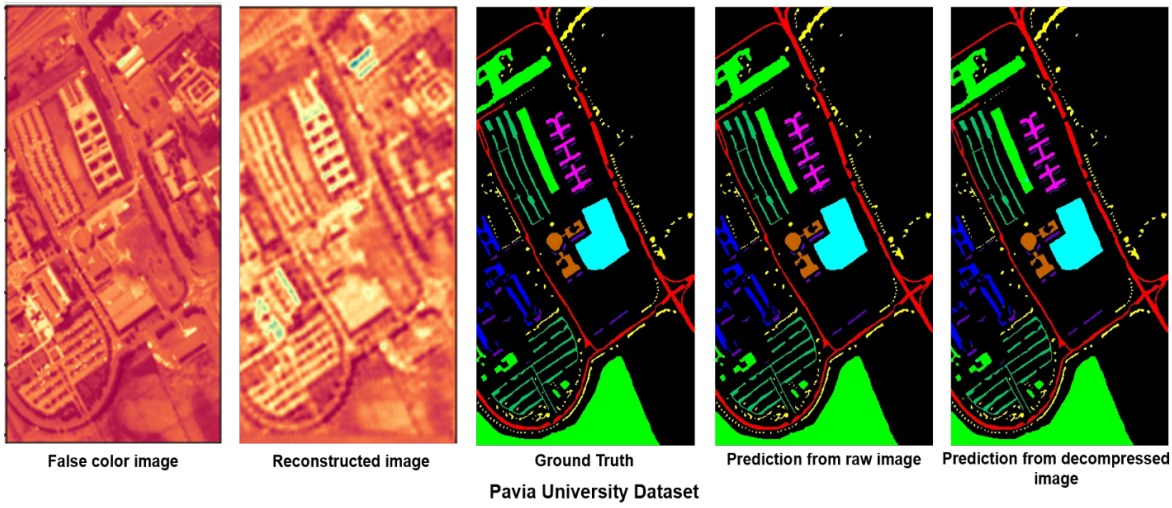

Figure 5.17: Compression and Classification results for Pavia University dataset

(a) Original image (b) Reconstructed image

Figure 5.18: Confusion matrix for Pavia University dataset

However, the question may arise for the impact of compression on class-wise prediction capability of the classification module. We conduct another experiment to assess this comparable property and present its result in Table 5.10 for PU data. Class name and support values were part of the dataset, classification by original and reconstructed image in terms of precision, recall and f1 score are calculated. The precision and f1 score

of Gravel class reduce to 0.95 and 0.97, recall of Trees to 0.97 reported as the minimum change. We obtain similar analysis for other datasets too. Figure 5.18 represents the confusion matrix and individual misclassification rate of each class from original and reconstructed PU data using HybridSN algorithm.

Table 5.9: Classification results for Original and reconstructed HSI datasets on different training values

Data Set	Train Data(%)	Original			Reconstructed		
		20%	30%	40%	20%	30%	40%
SA	**OA(%)**	99.9954	99.9973	100	99.9745	99.9973	100
	Kappa(%)	99.9948	99.9971	100	99.9717	99.9961	100
	AA(%)	99.9916	99.9992	100	99.9781	99.9987	100
IP	**OA(%)**	99.0487	99.6516	99.7033	98.8170	99.2479	99.5735
	Kappa(%)	98.9161	99.6027	99.7398	98.6509	99.1419	99.6260
	AA(%)	99.2913	99.8022	99.8182	98.4853	99.2722	99.4174
PU	**OA(%)**	99.8722	99.9432	100	99.4360	99.3855	99.8675
	Kappa(%)	99.9035	99.9247	100	99.2522	99.1857	99.8244
	AA(%)	99.8047	99.8744	100	99.0685	99.1605	99.7508

Table 5.10: Class-wise analysis of Original and Reconstructed Pavia University dataset

Classes	Support	Original			Reconstructed		
		Precision	Recall	F1-score	Precision	Recall	F1-score
ASPHALT	4642	1.00	1.00	1.00	0.99	0.99	0.99
MEADOWS	13055	1.00	1.00	1.00	1.00	1.00	1.00
GRAVEL	1469	1.00	1.00	1.00	0.95	0.99	0.97
TREES	2145	1.00	1.00	1.00	1.00	0.97	0.98
Painted Metal Sheets	942	1.00	1.00	1.00	1.00	1.00	1.00
BARE SOIL	3520	1.00	1.00	1.00	1.00	1.00	1.00
BITUMEN	931	1.00	1.00	1.00	0.98	1.00	0.99
Self-Blocking Bricks	2577	1.00	1.00	1.00	0.99	0.99	0.99
SHADOWS	663	1.00	1.00	1.00	0.99	0.99	0.99

5.2.4 Conclusion

We propose a novel approach of HSI compression based on the integration of transform domain and deep learning. The energy compaction property of discrete wavelet transform is utilized for spectral decorrelation followed by feature extraction attribute of convolution network. A series of convolution, ReLU and maxpooling layer generate a 3D tensor of smaller size with minimum parameters. The set of parameters (weights and biases) of the network need to be transmitted/stored along with compressed image to regenerate the original HSI with some acceptable loss. The sizes of these weights have been calculated during assessment to avoid the impact of re-training on performance. The experiment results vary for different datasets due to diverse characteristics of each input image. We evaluate its performance on application oriented domain, say classification accuracy, to better understand the impact on image quality in terms of OA, KA, and AA. In future, we plan to evaluate our model on other HSI processing applications like video processing, time-series analysis, anomaly detection. Another approach to improve the usability of our model is to incorporate transfer learning and avoid retraining the model repeatedly for each category of datasets. We are also planning to introduce the proposed model for compression of medical HSI and observe the effect of disease monitoring with the compressed image.