Compression Methods for Extracted Satellite Image Features

Code available at https://github.com/zy-f/sat-featvec-compressors

Christos Polzak (clcp)
Stanford University
clcp@stanford.edu

1 Introduction

Satellite images are an important remote sensing data source that allow us to predict and track socially meaningful metrics over time via machine learning (ML) techniques. For example, ML can be used to evaluate humanity's progress toward the United Nations' Sustainable Development Goals (SDGs), which has historically been limited to measurement via surveys and censuses [1]. However, storing high-resolution satellite imagery of large swaths of the Earth over multiple time-steps for such uses is very space-intensive [2]. Fortunately, for many machine learning applications, storing the original images themselves is not strictly necessary. At Stanford's SustainLab, we have been investigating a pipeline where we instead use a deep neural network to extract generally-useful features from satellite images, and then run a compressor that further reduces the space required to store the feature vectors, which are then only saved in their compressed state. Then, for downstream tasks (such as tracking SDG progress), researchers could simply decompress the feature vectors and train small ML models on the features (pipeline summarized in Figure 1).

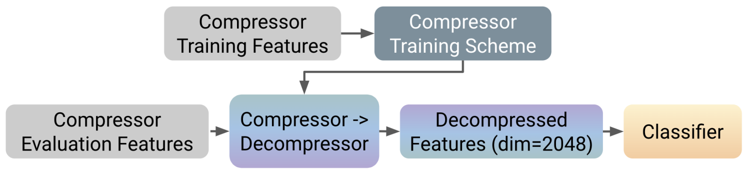

Compressor Training Features
Compressor Training Scheme
Compressor Evaluation Features
Compressor -> Decompressor
Decompressed Features (dim=2048)
Classifier

Figure 1: Satellite Image Compression Pipeline

In this paper, we investigate this second step in the pipeline, seeking to answer the question: **What is the best compressor to decrease the space requirements for satellite image features while simultaneously preserving decompressed downstream performance?** In particular, we investigate compressors acting along two axes: **dimensionality reduction** and **quantization**. Here, we emphasize that since these compressors must provide features that can be applied to a range of downstream tasks, we intentionally avoid compression methods that would involve using specific downstream classification labels to train the compression algorithm.

2 Methods

2.1 Datasets

Functional Map of the World (fMoW). The fMoW dataset [3] provides temporally-repeated satellite images corresponding to 63 different classes of buildings and structures (for example, "flooded road", "zoo", or "electricity substation"), where each location has multiple associated images with metadata annotations (including time and geolocation) and at least one bounding box identifying a structure of interest in the image. For this project, we utilize only the RGB images without any other metadata. There are approximately

360,000 training samples and 50,000 validation samples. We use fMoW-extracted features as our compressor training data and for our in-distribution evaluation.

UCMerced Land Use (UCM) UCMerced Land Use [4] is a small, high-resolution satellite image dataset with 100 images each of 21 classes of land use (e.g. "harbor", "forest", "runway"). We use the UCM dataset to evaluate the performance of our compressors when taken out of their training distribution.

2.2 Feature Extractor

For this project, we do not experiment with the underlying deep feature extractor. We utilize SustainLab's pretrained Geography-Aware SSL (GeoSSL) model [5]), which was pretrained using a contrastive self-supervised learning process on high-resolution RGB satellite images from the fMoW dataset. It outputs 32-bit float features z of dimension $d = 2048$, which we use as our compressor inputs.

2.3 Compression Methods

Entropy Bottleneck (EB). In the current SustainLab pipeline, we had utilized the end-to-end compressor introduced by Ballé et al. [6], as it is implemeted in the CompressAI library[7]. Ballé et al.'s method consists of convolutional analysis transformation that bottlenecks into a uniform quantizer and then gets decompressed via a convolutional synthesis transform that seeks to reverse the analysis transformation after quantization. For the purposes of this project, we can interpret this entropy bottleneck as an autoencoder that jointly reduces dimensionality and quantizes its compressed representation. Notably, the entropy bottleneck was originally designed to compress images themselves. Its convolutional filters are designed to take advantage of spatial associations between nearby pixels—associations that do not necessarily exist in the feature vector outputs of a deep feature extractor. To determine the optimal analysis and synthesis transforms, we compute a joint rate-distortion loss. Let z be our original feature vector, $\hat{z}$ be our reconstructed feature vector after the synthesis transform, x be the dimensionally-reduced analysis representation (before quantization), and $q(\cdot)$ be a relaxed, smoothed representation of the probability distribution of different decompressed values under the uniform quantizer (the strict version of the distribution would be a sum of Dirac delta functions at the medians of the quantization bins). Then, our loss function is:

$$\mathcal{L}_{\text{EB}}(z, \hat{z}) = -\log q(x) + \lambda \|z - \hat{z}\|_2^2$$

The first term represents our rate loss: the model is penalized for creating analyzed representations x that are unlikely under the quantization distribution (which makes them more entropically costly). The second term is our distortion loss: the model is penalized for ultimately synthesizing representations $\hat{z}$ that are "distorted" far away from the original representation The first term represents our rate loss: the model is penalized for creating analyzed representations x that are unlikely under the quantization distribution (which makes them more entropically costly). The second term is our distortion loss: the model is penalized for ultimately synthesizing representations $\hat{z}$ that are "distorted" far away from the original feature vector z; we measure this as mean-squared error. We use the hyperparameter λ to trade-off between the two loss terms. Notably, the EB compressor uses an entropy coder to store the representation q(x), so we do not have direct control over the final compression rate—it depends on the likelihoods of the feature vectors, which matters when we generalize the compressor to a new dataset, for which the entropy coding may be suboptimal due to a different distribution under the different quantization bins.

Uniform Quantization (Quant). To benchmark the quality of the EB compressor against simpler methods, we implemented simple, element-wise uniform quantization. As a first step, we clip outliers to the minimum or maximum of an acceptable range—empirically, we do a line search on possible clipping thresholds and choose reasonable values so that the total amount of values clipped is less than .05%. Then, given N bits per element of storage after quantization, we simply divide the acceptable range into 2^N equally-sized bins, using the following formula:

$$Q(z_i) = \text{Int}\left(\frac{\min(z_i, z_{\max} - \epsilon) - z_{\min}}{z_{\max} - z_{\min}} \times 2^N\right)$$

where $z_{\min}$ and $z_{\max}$ are the minimum and maximum clipping values, respectively, z_i is the i-th element in the feature vector, and ϵ is a small value that just ensures that $z = z_{\max}$ stays within the $[0, 1)$ range but stays in the largest bin. To de-quantize, we then approximate each value as the median of its bin:

$$\hat{z}_i = \frac{Q(z_i) + 1/2}{2^N} \times (z_{\max} - z_{\min}) + z_{\min}$$

Principal Components Analysis (PCA). PCA is simple method for dimensionality-reduction that maximally preserves the (unsupervised) variance in the data (notably, we do not make use of labels in this process, instead focusing on the variance of the input data itself). To perform a k-dimensional PCA, we first compute the sample covariance matrix $S \in \mathcal{R}^{d \times d}$:

$$\begin{aligned} S &= \frac{1}{m-1} \sum_{i=0}^{m-1} (z_i - \bar{x})(z_i - \bar{x})^T \\ &= \frac{1}{m-1} \tilde{Z}^T \tilde{Z} \end{aligned}$$

where m is the number of data samples, $\bar{z} \in \mathcal{R}^d$ is the sample mean for our features, and $\tilde{Z} \in \mathcal{R}^{m \times d}$ is the centered (de-meaned) matrix of samples, where each row is a sample data.
Then, since the covariance matrix is positive semi-definite, we can diagonalize to get $S = U \Lambda U^T$, where Λ is a diagonal matrix of the eigenvalues of S ordered from largest to smallest, and column u_i of U represents the corresponding eigenvector for the eigenvalue at Λ_{ii}. Then, the first k eigenvalues in Λ and corresponding columns of U represent the k orthogonal directions in d-dimensonal space that maximally preserve the variance of the original data matrix Z. Thus, finally, to dimensionally-reduce a data point to dimension $k < d$, we can utilize projection matrix $U_p = [u_1\ u_2\ ...u_k] \in \mathcal{R}^{d \times k}$. To reduce the dimensionality of a data point, we compute $z_p = U_p^T (z - \bar{z}) \in \mathcal{R}^k$. We then reconstruct our approximation as $\hat{z} = U_p z_p$. (NB: we train directly on the centered data without shifting the input back to its mean, as this is generally good practice for neural network training reasons anyway.)

PCA + Quant. To emulate the compression pipeline generated by the EB compressor, we also try compressing our data with PCA first, and then separately performing quantization on the reduced-dimensionality representation, again choosing clipping values to only distort outliers in the outer .05% tails of the distribution. In other words, to compress, we project to a lower dimension $k < d$, then quantize the k-dimensional data. To decompress, we de-quantize the data and then reconstruct it in higher-dimensional space.

Autoencoder (AE). Inspired by prior work where multi-layer perceptron (MLP) autoencoders have been used to reduce the dimensionality of tabular or vectorized scientific data [8], we build a simple autoencoder architecture where the original $d = 2048$ dimensonal data is passed through an analysis transform of multiple feed-forward neural network layers of decreasing size down to a bottleneck dimension k. We then use a sigmoid activation function to obtain our compressed representation x. Then, the compressed representation x is passed through a synthesis transform consisting of feed-forward neural network layers of increasing size back to the original dimension d, producing our reconstruction $\hat{z}$. To create effective analysis and synthesis transforms, we train the neural network on the reconstruction error $||z - \hat{z}||_2^2$—this is the same as the distortion loss from the EB compressor. Whereas PCA tries to maximally preserve the variance of data under a *linear* transformation down to k-dimensional space, the idea of an autoencoder is to force a neural network to determine a *non-linear* transformation into k-dimensional space that can then be reconstructed back into the original input.

2.4 Compressor Performance Evaluation

Our compressors are evaluated on two primary metrics: their ability to reduce the storage requirements for the feature vectors, and the usefulness of the decompressed vectors in downstream performance.

Compressed Data Size. First, to determine the compressive effectiveness of each of our compressors, we compute the filesize required to store the fMoW training and validation splits in their most compressed form under the compression method. For the EB compressor, that is the $q(x)$ representation; for quantization, it is simply the quantized version of the data; for PCA, we store the z_p data; and for the autoencoder, we store the data under its Sigmoid(x) representation. We store the data as pickle-based binary Pytorch files, and use the built-in Linux *du* command to determine file size.

Classifiers. To evaluate the downstream usefulness of the decompressed vectors, we use two simple neural classifiers to learn to perform downstream classification tasks:

- A linear classifier, consisting of a trainable affine transformation from dimension d to the number of classes, followed by a softmax operation to compute per-class probabilities
- An MLP classifier consisting of 3 layers of affine transformation with non-linear ReLU activations, with the final layer similarly being softmaxed to produce per-class probabilities

To determine classifier performance, we use validation set accuracy.

2.5 Baseline

As a baseline, we use our deep feature extractor to obtain feature vectors for both the fMoW dataset and the UCM dataset, save them off, and then train our classifiers on the uncompressed features. For data size, we find that the fMoW feature vectors take **3.2GB** of space. On the fMoW dataset, our linear classifier achieves 69.1% accuracy, while the MLP classifier achieves 70.9% accuracy. On the UCM dataset, the linear classifier achieves 94.5% accuracy. (Due to time constraints, we did not use the MLP classifier on the UCM dataset.)

3 Results

Experimental Details. For all our compressors, we train any parameters on the fMoW training dataset only, then evaluate that compressor both in-distribution on the validation fMoW dataset as well as out-of-distribution on the UCM dataset. We train our two classifiers on the training datasets (fMoW and UCM) and evaluate them on their respective validation sets. Our hyperparameters for the compressors are detailed in the respective sections below. Our MLP classifier has two hidden dimensions of size 2048 followed by 1024, with ReLU activations and 50% dropout after each of the two hidden layers active during training. For our fMoW classifiers, we use a learning rate of $1e-4$, a batch size of 256, no weight decay, the Adam optimizer, and train for 100 epochs. On UCM data, the only difference is that we train for 1000 epochs, since the dataset is smaller.

EB. We use $\lambda = 4e-2$, a learning rate of 0.2, a batch size of 128, and train for 10 epochs. Our results are summarized in Table 1. Since this is the extant compressor we use at SustainLab, further experiments center on trying to beat the benchmark set by this EB compressor.

Quant. We experiment with 8-bit quantization ($N = 8$) (compressed data stored as the Pytorch unsigned 8-bit integers) and 15-bit quantization ($N = 15$) (compressed data stored as Pytorch signed 16-bit integers). We use a lower clipping bound of 0 (the features come from the outputs of a ReLU function, so nothing gets clipped off) and an upper bound of 0.6. Our results are summarized in Table 1. Notably, we find that even when we drop from 32-bit floats to 8-bit ints, quantization is lossless. Thus, in further experiments, we ceased exploring 15-bit quantization.

PCA. We first coarsely explore dimensionality reductions of $k \in \{10, 30, 100, 300, 100\}$. Witnessing diminishing returns, we then experiment more finely in the range $k \in \{75, 100, 125, 150, 175, 200\}$ (see Figure 2). Since their performance approaches EB pretty well, we focus full-evaluation efforts on $k = 150$ and $k = 200$. Our results for these are summarized in Table 1.

PCA + Quant. Making use of prior results, we focus on $N = 8$-bit quantization of PCA projections of size $k = 150$ and $k = 200$. The PCA outputs fall into a slightly wider distribution; we set a lower clipping bound of -1 and an upper clipping bound of +1.5 to avoid distorting most of our data (¿99.9%).

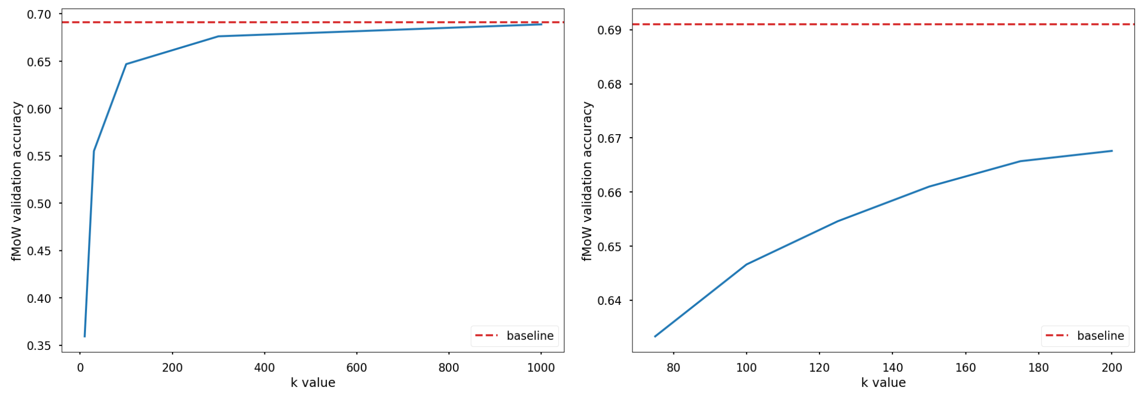
AE. Again making use of prior results, we focus on bottleneck dimensions of $k = 150$ and $k = 200$, seeking to outperform PCA of equivalent dimensionality reduction. We tried two architectures, each with 50% dropout at all hidden layers:

- **Thin+Deep:** 4-layer encoder (and decoder architecture), with fully-connected layers of dimensions $d \to 512 \to 512 \to k$ (in the reverse order for the decoder); and,
- **Wide+Shallow:** 3-layer encoder with fully-connected layers with dimensionality $d \to 1024 \to k$.

For all encoders, we train for 60 epochs with with a learning rate of $1e-4$, Adam optimization, batch size of 256, and no weight decay. Our AE-specific results are summarized in Table 2. Notably, we observe that the shallower, wider autoencoder performs far better, but that none of our autoencoders were able to surpass the equivalent PCA dimensionality reduction. For this reason, we stopped pursuing neural autoencoders further for the sake of this project (such as trying AE + Quant). We include our best-case autoencoder in our primary results for comparison (the $k = 200$ Wide+Shallow autoencoder).

Compressor	fMoW Size (MB)	fMoW Linear* (%)	fMoW MLP* (%) (Δ% vs Linear)	UCM Linear* (%)
Baseline	3200	69.1	70.9 (+1.8)	94.5
Quant (N=8)	818	*69.1*	*71.0 (+1.9)*	*94.5*
Quant (N=15)	1600	*69.1*	-	*94.5*
EB	124	**67.0**	68.1 (+1.1)	**93.6**
PCA (K=200)	338	66.8	**70.5 (+3.7)**	**93.6**
PCA (K=150)	258	66.1	-	**93.6**
PCA (K=100)	179	64.6	-	92.9
PCA (K=200) + Quant (N=8)	83	66.7	70.3 (+3.6)	93.3
PCA (K=150) + Quant (N=8)	63	66.1	69.9 (+3.8)	**93.6**
PCA (K=100) + Quant (N=8)	**44**	64.6	69.2 (+4.6)	93.1
Autoencoder (K=200)	322	66.4	69.5 (+3.1)	92.9

Table 1: Compression Rates and Downstream Performances of Compressors (*Peak Validation Accuracy)
Italicized = matches baseline; **Bold = best performance with significant compression**



(a) Coarse Exploration (b) Fine Exploration

Figure 2: In-Distribution Decompressed Performance of Different PCA Reductions

Type	Bottleneck Dim (k)	fMoW Linear Val Accuracy (%)
Thin+Deep	150	52.5
Wide+Shallow	150	66.1
Wide+Shallow	200	66.4

Table 2: Autencoder Architectures and Performances

4 Discussion

Surprisingly, we find that, in this satellite image feature vector use case, simple quantization methods are more than sufficient to keep up with and even surpass neural network-based compression schemes, with far less overhead required to prepare the compressor. First of all, we find that simple, uniform quantization from 32-bits down to 8-bits is effectively lossless. Even when using a deeper classifier to try to extract more nuanced decision boundaries, where we thought the loss of information from quantization might limit the potential of a deeper model, or when generalizing, where we thought the clipping boundaries or bins might be poorly set for the different distribution, we find that quantization fully keeps up with baseline uncompressed performance.

We observe a similar story playing out with the PCA + Quant compressors, where we find that having a deeper classifier extracts significantly more performance from the input decompressed features, quantizing over just PCA is still essentially lossless, and despite the totally different feature inputs, PCA projection generalizes well to a new dataset. With an MLP fMoW classifier, *all* of the PCA + Quant compressors surpass the EB compressor, which benefits very little from a deeper classifier, suggesting that the quantizer built into the EB compressor might be too aggressive, while not getting many compressive gains as the satellite features can already be compressed with a uniform quantizer lossless-ly. Importantly, we find that these improvements in classification are possible despite the fact that all the PCA + Quant compressors are able to store the fMoW dataset as much as nearly 3x more efficiently than the EB compressor. Highlighting the k=150 PCA + Quant compressor, we are able to achieve less than 1% performance degradation, both with a deeper classifier and when generalizing to a new dataset, with a compression rate of over 50x.

In conclusion, we find that, at least under these relatively constrained conditions, using PCA and a simple uniform quantizer can provide similar downstream performance to more advanced neural compression methods while being much easier to implement and allowing for even higher compression gains than the neural methods.

5 Future Work

We believe future work should investigate along three main avenues:

- Simple convex classification methods that can directly solve for the global optimum to verify the true potential of our decompressed features
- Further hyperparameter and architecture investigation of the autoencoder architecture, which may have had its potential limited by the constraints of the class project's relatively short research time
- Test the compression methods on datasets or tasks that are further from the original training dataset (for example. regression, or a dataset other than UCM that is harder than UCM or more different from fMoW)

6 Acknowledgements

This is an extension on research I am performing through Stanford University's SustainLab on satellite image compression methods, led by Professors Stefano Ermon, David Lobell, and Marshall Burke.

References

[1] "Transforming our world: The 2030 agenda for sustainable development." https://sustainabledevelopment.un.org/post2015/transformingourworld/publication, September 2015. Accessed: 28 October 2022.

[2] D. Mohney, "Terabytes from space: Satellite imaging is filling data centers." https://datacenterfrontier.com/terabytes-from-space-satellite-imaging-is-filling-data-centers, 28 April 2020. Accessed: 28 October 2022.

[3] G. Christie, N. Fendley, J. Wilson, and R. Mukherjee, "Functional map of the world," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[4] Y. Yang and S. Newsam, "Bag-of-visual-words and spatial extensions for land-use classification," in *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, (New York, NY, USA), p. 270–279, Association for Computing Machinery, 2010.

[5] K. Ayush, B. Uzkent, C. Meng, K. Tanmay, M. Burke, D. Lobell, and S. Ermon, "Geography-aware self-supervised learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 10181–10190, October 2021.

[6] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," *CoRR*, vol. abs/1611.01704, 2016.

[7] J. Bégaint, F. Racapé, S. Feltman, and A. Pushparaja, "Compressai: a pytorch library and evaluation platform for end-to-end compression research," *arXiv preprint arXiv:2011.03029*, 2020.

[8] T. Liu, J. Wang, Q. Liu, S. Alibhai, T. Lu, and X. He, "High-ratio lossy compression: Exploring the autoencoder to compress scientific data," *IEEE Transactions on Big Data*, vol. 9, no. 1, pp. 22–36, 2023.