Supplementary Material

A Dataset Supplementals

Telescope	Dataset ID	Location	Central Wavelength (Å)	Bandpass Filter	Resolution (arcsec / pix)	Image Sizes (pix×px)	Total # Arrays	Total Dataset Size (GB)	Approx. Total Pixels
LCO	GBI-16-2D-Legacy	multiple sites Earth	varies	B, V, rp, ip	0.58	3136×2112	9	0.1	5.5×10^7
Keck	GBI-16-2D	HI, USA, Earth	varies, 4500–8400	B, V, R, I	0.135	2248×2048; 3768×2520	137	1.5	7.4×10^8
Hubble	SBI-16-2D	LEO, Space	5926	F606W	0.05	4144×2068	2140	69	1.8×10^{10}
JWST	SBI-16-3D	L2, Space	19840	F200W	0.031	2048×2048 × #Groups	1273	90	7.0×10^9
SDSS	GBI-16-4D	NM, USA, Earth	3543, 4770, 6231, 7625, 9134	all of $[u, g, r, i, z]$	0.396	800×800×5 × #Timesteps	500	158	1.6×10^{10}

Table 4: Datasets summary. Total pixels computed using each 16-bit data point as one "pixel."

A.1 Naming conventions

Throughout this work, the data are referred to in three different ways depending on the context; we elucidate the reference conventions here.

A.1.1 Dataset ID

The adopted Dataset ID naming convention provides a shorthand description of the origin and form of the data:

(*origin*)(*data taking mode*)-(*number of bits per pixel*)-(*dimensionality*),

where *origin* is either space-based (SB) or ground-based (GB), and *data taking mode* refers to the primary objective of the exposure, only I (imaging[1]). -2D implies each instance is a 2-dimensional array. -3D may be thought of as a temporal sequence of images (ie., movie) taken in roughly the same portion of the sky. -4D may be thought of as a temporal sequence of images taken in the same portion of the sky at different wavelengths. Clearly, -3D and -4D may be decomposed into individual 2D images. All data in the corpus are `uint16` but future datasets may be added with different bit depths.

This is a standardized way of naming our datasets that include functional details, allowing one to quickly search for or identify the kind of data from the name alone.

A.1.2 Telescope or Survey Name

For brevity and ease of reading, we use the "telescope" column as nicknames to refer to each of our published datasets throughout our paper.

A.1.3 Experimental dataset names

For several reasons, some of our benchmark models were trained on a subset of a dataset. To this end, we added additional descriptors to the dataset nicknames as a way to describe these subselections. LCO, Keck, and Hubble were unchanged and used fully. JWST was used in two forms: a first frame model from JWST-2D (`jwst[t=0]` for every image cube), and JWST-2D-Res (we used all of the residual images to train a separate residual coding model, i.e., `jwst[t=i+1] − jwst[t=i]` for

[1]Other data taking modes such as spectroscopy may be added in future work.

all i, for every image cube). SDSS was split into three datasets: SDSS-2D (`sdss[t=0][λ=2]`), SDSS-3Dλ (`sdss[t=0][λ=1,2,3]`), SDSS-3DT (`sdss[t=0,1,2][λ=2]`).

A.2 ACCESSIBILITY

All datasets herein are released under the AstroCompress project as HuggingFace `datasets` and may be accessed as `numpy` nd-arrays in Python:

```
import numpy as np
from datasets import load_dataset

dataset = load_dataset(f"AstroCompress/{name}", config, split=split, \
                       streaming=True, trust_remote_code=True)
ds = dataset.with_format("np", columns=["image"], dtype=np.uint16)
ds[0]["image"].shape # -> (tuple with shape of the numpy array)
```

Here `name` is one of the five datasets described below, `config` is either "tiny" (small number of files for code testing purposes; default value) or "full" (full dataset), and `split` is one of "train" or "test". To use the datasets with `pytorch`:

```
import torch
from torch.utils.data import DataLoader

dataset = load_dataset(f"AstroCompress/{name}", config, split=split, \
                       streaming=True, trust_remote_code=True)
dst = dataset.with_format("torch", columns=["image"], dtype=torch.uint16, \
                          device=device)
dataloader = DataLoader(dst, batch_size=batch_size, num_workers=num_workers)
next(iter(dataloader)) # -> yields the first batch of images
```

where `device` is the pytorch device to use (e.g., "gpu", "mps:0").

A.3 DATASET COLLECTION AND SCIENTIFIC DETAILS

A.3.1 DATASET COLLECTION OVERVIEW

All of the data that we have pulled is public data from various astronomical archives. The specific licenses under which the data have been released are noted in the acknowledgment section. Here we detail the selection and curation process for each dataset. Table 4 provides a broad overview of the corpus.

At a high level: for each dataset, we generally come up with a set of logical filters and pull a list of *observations* via API and/or direct download from the archives where the data are disseminated. An observation refers to one telescope's "visit" to a certain place in the sky, integrated over a short period of time (typically less than 20 min). Each observation can result in multiple versions of that observation in the archives, as many archives store various processed versions of observations along with the associated calibration data used. For a given observation, we download the one file in raw form (`uint16`; processed data is usually `float32`). Finally, we select a subset of the downloaded data that contains no pairwise overlapping of the image footprint on the sky. This ensures that compression models trained on the training set do not overfit on regions of the sky present in the test data. It also allows users to safely create their own validation sets from subsets of our training sets.

A.3.2 GBI-16-2D (KECK)

This is a 2-D optical imaging dataset from the ground-based W. M. Keck Observatory, obtained in a variety of observing conditions, filters, and across exposure times from 30 seconds to >10 min. The data are all selected from the Low Resolution Imaging Spectrometer (LRIS; Oke et al. (1995)) and are from scientific observations (as opposed to calibration exposures) obtained by one of the co-authors of this paper (J.S.B.) from 2005 to 2010. Since LRIS has a dichroic optical element, which splits the incoming beam at a designated wavelength, the data are obtained either with the blue or

red side camera. Each camera has its own filter wheel allowing for imaging in a variety of different bandpasses. The raw data of a given observation are stored in FITS format (Pence et al., 2010). The LRIS CCDs are readout using 2 amplifiers which are stored in distinct logical and memory sections (called Header Data Units; HDUs) within the FITS files. The blue side camera has two CCDs and the redside camera had 1 CCD before June 2009 and 2 thereafter (Rockosi et al., 2010). In addition to storing the light-exposed regions, LRIS FITS files usually include small regions of virtually read pixels, called overscan regions, which aid image processing.

Collection

We used the Keck Observatory Archive (KOA) to identify raw LRIS science images obtained under the principal investigator program by J.S.B. These data were then downloaded and checked for potential footprint overlaps using the positional information of the telescope pointings from the FITS header. The data assembly code for the HuggingFace GBI-16-2D dataset handles the variety of FITS formats and emits 2D images of size 2248×2048 or 3768×2520 pix^2. An example image from this dataset, with two amplifier reads, is shown in Fig. 1.

A.3.3 SBI-16-2D (HUBBLE)

This dataset is based on data from the Wide Field Channel (WFC) instrument of the Advanced Camera for Surveys (ACS) onboard the Hubble Space Telescope (HST). Unlike the GBI-16-2D dataset, all observations in SBI-16-2D are obtained in space and with the same bandpass filter (F606W), providing a more uniform point-spread function across the dataset. Our goal with assembling SBI-16-2D is two-fold. First, we aim to provide a large (> 50 GB), raw 2-D optical imaging dataset from space. Second, space-based CCD imaging, unlike ground based imaging, suffers significantly from charge particle collisions with the detectors (called "cosmic-ray [CR] hits"). Such random hits add spurious counts to the affected pixels, corrupting the scientific utility of the observations. WFC CCDs also demonstrate large amounts of charge transfer inefficiency leading to correlated streaks in the vertical direction (see Fig. 1).

Collection

We first compile a list of observations that fit our criteria, including instrument, filter choice, and integration time ranges. This data was then downloaded from the Mikulski Archive for Space Telescopes (MAST): `https://mast.stsci.edu/search/ui/#/hst`. For reproducibility, we provide a script that performs this in our HuggingFace repository located at `SBI-16-2D/utils/pull_hubble_csv.py`. Explanations for each filtering process can be found in the code comments therein. Each FITS file contains two images each of size 4144×2068; these images are stored in the 1st and 4th headers in the FITS files. After removing overlapping regions in the sky, the final dataset amounts to 4282 images of size 4144×2068.

A.3.4 SBI-16-3D (JWST)

This dataset comes from the NIRCAM instrument on the James Webb Space Telescope (JWST). NIRCAM conducts up-the-ramp imaging of various objects in several infrared wavelengths. During up-the-ramp sampling, the aperture is exposed to a region in space, and light continuously accumulates charge on the array. The array repeatedly measures this cumulative charge for multiple frames, thus creating a 3-D time-series image tensor. Every N frames is averaged to create a "group", each of which is stored in our arrays in the T dimension (where N is determined by the "read pattern"). We strongly encourage interested readers to learn more about the JWST readout patterns on the official documentation page [2]. It is worth noting that the time gap between subsequent observations varies, depending on readout pattern, but is roughly in the realm of twenty seconds to two minutes. For this reason, we expect that most physical conditions present in the field of view should remain static across time steps.

Collection

Similar to Hubble, we pulled the JWST observations list from the JWST section of MAST. A script version of this can be found at `SBI-16-3D/utils/pull_jwst_csv.py`. Explanations for

[2] `https://jwst-docs.stsci.edu/jwst-near-infrared-camera/nircam-instrumentation/nircam-detector-overview/nircam-detector-readout-patterns`

each filtering process can be found in the code comments there. We provide 1273 images of size 2048 by 2048 by T, where T represents time, under the F200W wavelength filter.

A.3.5 GBI-16-4D (SDSS)

This dataset is assembled from "Stripe 82" of the Sloan Digital Sky Survey (SDSS; York et al. (2000)). Stripe 82, a $\sim$250 sq. deg. equatorial region was repeatedly imaged in 5 optical bandpasses with 30 total CCDs over the course of many months for several years, with supernova discovery as the main science objective (Sako et al., 2018). Images are obtained in drift scan mode by fixing the telescope in declination and letting the sky naturally move across the field of view as the CCDs are readout at the sidereal rate. For a given nightly "run" across Stripe 82, the SDSS image reduction pipeline (Stoughton et al., 2002) creates (for each of the 6 camera columns, "camcols") "field" images spanning a similar declination (ie., north-south) and right ascension (RA; ie., east-west). The images served by the legacy SDSS archive[3] are 2048$\times$1489 `uint16` pixels, calibrated with a world coordinate system (WCS) that maps pixel location to sky position.

In total, we release 500 GBI-16-4D cubes as part of this dataset, with an uncompressed size of $158GB$. At a fixed wavelength, the images in time mimic the raw `uint16` time sequence produced by imaging satellites, like Kepler, TESS, and CuRIOS (Gulick et al., 2022). Exploiting the correlations in time (and wavelength) should improve the effective compression compared to the single-slice capabilities.

We note that the gap in time between subsequent timestep images may be many nights, resulting in very different background sky conditions. While the background sky levels across nights are relatively uncorrelated, the pixels containing astrophysical sources exhibit significant correlation.

Collection

We queried the SDSS Stripe 82 database to assemble a table of 160k unique Stripe 82 field images deemed to be of top quality (`quality` flag = 3) and then randomly sampled fields and determined the center of the field sky positions. For each such field, we queried the SDSS Stripe 82 database for other images (regardless of `quality`) that overlap the field center and downloaded those images. Since field images from different runs are not aligned in RA, we also downloaded the two fields immediately adjacent and created a stitched-together mosaic of those three images. We then used the WCS in each mosaic image to cut out the same position across all available runs across all available filters.

The assembled data are 4-D cube representations of the same 800$\times$800 pix^2 portion of the sky observed from t=1 up to $\sim$90 times in f=5 filters (u, g, r, i, z). Given the excellent but inherently noisy process of WCS fitting, the $t \times f$ image slices in a given cube are spatially aligned to $<$1 pix. As a result, for a fixed pixel location in a given cube there are high correlations in the pixel value across time and wavelength. While the effective integration time is identical across all image slices, there are slices that are of higher signal-to-noise than others in the same cube and all have varying background levels. In the few cases where an image slices does not fully overlap the central region of the anchor field, we fill the missing region with values of zero.

A.3.6 GBI-16-2D-LEGACY

This small ground-based dataset obtained on multiple CCDs across many different telescopes is reproduced from the public corpus released by Maireles-González et al. (2023) and assembled by us in HuggingFace `dataset` format. Our experiments only made use of the subset of data from the Las Cumbres Observatory (**LCO**).

Collection

We retrieved all of these files from a download page of Maireles-González et al. (2023) in a `.raw` format and used a script (`GBI-16-2D-Legacy/raw_to_fits.py` in HuggingFace) to convert the images to FITS format. Unlike in the other datasets, we did not check for nor remove potential overlapping images.

[3]These images are from the reduction pipeline 40/41 ("rerun") and were part of data release 7 from SDSS. Newer reductions of the same raw data were released with different calibrations in `float32` format.

A.4 DATASET ASSEMBLY

Rejection of overlapping images After the initial download of these raw images from various astronomy databases, a significant portion of the necessary data curation is to ensure that the imagery does not overlap on the sky. It is particularly essential to ensure that the spatial footprint of all data in the train set does not overlap with that of the test set. Because we anticipate that future use of this data will split it into validation sets as well, we conservatively ensure that all of our images have pairwise zero overlap.

This was done in two stages, the implementations for which can be found in the `utils/`

Stage 1. Before downloading raw imagery, we first download a metadata file that contains a list of observations and the right ascension (RA) and declination (DEC) of each observation. These are akin to standard spherical coordinates θ and ϕ. The corresponding pixel for the given RA and DEC varies depending on the data source; it can be the image center pixel, the RA and DEC of the target celestial object of interest, or some other pixel within the image entirely. In order to filter out images in this overlapping set, we ran a hierarchical agglomerative clustering (HAC) algorithm via `sklearn.cluster.AgglomerativeClustering` on a matrix of precomputed pairwise angular distance matrix via `astropy.coordinates.angular_separation`. We made a small modification to the astropy source code in order to allow numpy vectorization. The threshold for clustering was selected as `2*FOV`, where `FOV` was the field of view of any given telescope. Within each cluster, a subset of well-separated images was downloaded.

Stage 2. After downloading the data, we were able to use the World Coordinate System (WCS) of each image to map any given pixel in an image to an RA and DEC, using `astropy.wcs`. The Python library `spherical-geometry` is used to compute spherical polygons from the four corners of the image in RA/DEC, and computes overlaps between these polygon objects. In some cases, such as for Hubble, there are two images contained within a single FITS file, so we need to compute spherical polygons for both and then use the union of those two polygons for overlap calculations. Using these algorithms, we further filter out overlapping images within each cluster.

Code to download raw data from source and filter it can be found in the `utils` folder of each HuggingFace dataset. We hope these files, in combination with the observation list assembly code will facilitate efforts for future experts to expand on our dataset from the astronomy data sources themselves.

Disclaimers: Because the Keck dataset did not have WCS information, we did not run stage 2, and instead used a more stringent clustering threshold in stage 1 and took only one image from each cluster. No filtering was done on the very small Legacy dataset; it also contained no WCS, RA or DEC data.

B BASELINE ALGORITHMS AND IMPLEMENTATION

From any dataset parent folder on HuggingFace, run `python utils/eval_baselines.py 2d` to get 2D evaluations of all algorithms. For JWST and SDSS, this `2d` argument may be changed to several other options that can be found in the command arguments `help` docstring. These options allow for JWST residual ("diffs", as stated in the code) compression, compressing entire 3D tensors for JWST and SDSS, as well as some unique SDSS experiments in which we compressed the top 8 bits and bottom 8 bits separately (`2d_top` and `2d_bottom`). We also attempted with poor results to compress 2D SDSS arrays composed of a single spatial pixel, but all wavelengths and timesteps (`tw`). The exact functionalities are documented in the script.

The script also saves the compression ratio, read time, and write time for every single image into a `.csv` file. We have already performed this and uploaded the CSV files for each dataset to the HuggingFace parent repository. Additionally, running statistics on mean values are printed to the console.

Implementations for all the non-neural baseline algorithms were adapted from the Python library `imagecodecs=2024.1.1`. All codecs in this library call the native C codec, specifically: JPEG-XL via `libjxl`, JPEG-LS via `charls`, JPEG-2000 via `OpenJPEG`, and RICE forked from `cfitsio=3.49`.

All the non-neural baseline algorithms natively support 16-bit inputs.

JPEG-XL and JPEG-2000 both use "reversible color transforms" to decorrelate different channels from each other before compressing a multi-channel image. JPEG-LS can only compress each channel as an independent 2D image. RICE is a simple histogram-based codec that flattens N-dimensional arrays before encoding.

In our specific implementations, JPEG-XL and JPEG-2000 supported multi-channel compression directly. To evaluate multichannel arrays using JPEG-LS and RICE, we applied `.reshape((H, -1))` to convert them to single-channel. This was performed only on SDSS data, which resulted in an image height of $H = 800$. For reporting compression ratio, we did not consider the bit-rate cost of transmitting the data shape needed for decoding, as this overhead is negligible for even the smallest image in our dataset.

Compute hardware was used as following: We ran classical compression codecs on a single thread with a `Intel(R) Xeon(R) Silver 4112 CPU @ 2.60GHz` processor and neural compression networks on a single `NVIDIA RTX 6000 ADA`.

C Details on neural compression methods

In this section, we provide an in-depth look at our experiments and modifications to existing neural compression methods to work with our astronomical image data. Our implementation can be found at https://github.com/tuatruog/AstroCompress.

C.1 Data format

As mentioned in the main text, most neural image compression methods are designed to handle 3-channel 8-bit RGB images, so we made minor modifications to the neural compression methods to handle the 16-bit data of AstroCompress.

At a high level, we experimented with two approaches: (1) adding support for 16-bit input directly; (2) treating the 16-bit input as the concatenation of two 8-bit inputs – the most significant byte (MSB) and least significant byte (LSB). We used the better of the two when reporting results, and generally found approach (2) to perform similarly or better than approach (1). As an example, we list the compression ratios obtained with either approach for **PixelCNN++** in Table 5.

- To implement approach (1), we make the following modifications to support 16-bit input directly: For **IDF**, we change the input scaling coefficient from 2^8 to 2^{16}, so that it models the set $\mathbb{Z}/65536$ (instead of $\mathbb{Z}/256$). For **L3C** and **PixelCNN++**, which both use a discretized logistic mixture likelihood model (Salimans et al., 2017), we increase the number of bins from $2^8 - 1$ to $2^{16} - 1$.
- To implement approach (2), we generally convert 16-bit input into 8-bit input while doubling the number of channels. For a 2D (single-frame) image of shape $1 \times H \times W$, this corresponds to treating it as an 8-bit tensor of shape $2 \times H \times W$ where the first channel contains the least significant byte (LSB) and the second channel contains the most significant byte (MSB). For a 3D (multi-frame) image of shape $3 \times H \times W$, this corresponds to treating it as an 8-bit tensor of shape $6 \times H \times W$, where the first three channels contain the LSBs of the original input and the remaining channels contain the MSBs of the original input. The neural compression models are modified accordingly to support the increased number of channels:
  - For **IDF**, we simply doubled the number of input/output channels while keeping the rest of the architecture the same;
  - For **L3C** and **PixelCNN++**, we model 2-channel 8-bit images by using only the "R" and "G" parts of the original RGB autoregressive logistic mixture likelihood model, and we model 6-channel 8-bit images by performing the same autoregressive modeling as the RGB case for the first 3 (LSB) channels, and similarly for the remaining 3 (MSB) channels (so the LSB and MSB channels are modeled separately).

Experiment	2-channel 8-bit	1-channel 16-bit
LCO	1.41	**2.02**
Keck	**2.08**	1.83
Hubble	**3.13**	2.71
JWST-2D	**1.44**	**1.44**
SDSS-2D	**3.35**	1.84

Table 5: Comparison of compression ratios for PixelCNN++ using the 2-channel 8-bit format v.s. 1-channel 16-bit format. The better result is highlighted in bold.

C.2 Architecture and training details

IDF We adopted the implementation from Hoogeboom et al. (2019) at https://github.com/jornpeters/integer_discrete_flows. The network and training hyper-parameters are also set to be consistent with the default configurations from (Hoogeboom et al., 2019), which we find to yield the best results. We train on patches of 32×32 with a batch size of 256. We use a learning rate of 1×10^{-3} and an exponential decay scheduler with rate 0.999.

L3C We adopted the implementation from Mentzer et al. (2019) at https://github.com/fab-jul/L3C-PyTorch. We largely followed the default model configuration provided by (Mentzer et al., 2019) to train the model on 2D image data across all datasets. For 3D experiments, we increased the base convolution filter size and adjusted the latent channel size to 96 and 8, respectively. We trained on 32×32 patches, with a learning rate of 1×10^{-4} and an exponential decay scheduler with rate 0.9.

PixelCNN++ We adopted the implementation from https://github.com/pclucas14/pixel-cnn-pp and adopted the same model architecture as in the default configuration (5 resnet blocks, 160 filters, 12 logistic mixture components, and a learning rate of 2×10^{-4}). We also explored different ways of formatting/modeling 8-bit data (converted from 16-bit input), such as training two separate models for the LSB and MSB, or concatenating the LSB and MSB across the width dimension instead of channel dimension, but did not observe significant improvements compared to the basic approach of stacking the LSB and MSB along the channel dimension (as described in Section C.1).

VDM We adopt our implementation from https://github.com/addtt/variational-diffusion-models/tree/main which is directly based on the official implementation from Kingma et al. (2021) at https://github.com/google-research/vdm. We use a scaled-down version of the denoising network from the VDM paper (Kingma et al., 2021), using a U-Net of depth 4, consisting of 4 ResNet blocks in the forward direction and 5 ResNet blocks in the reverse direction, with a single attention layer and two additional ResNet blocks in the middle. We keep the number of channels constant throughout at 128. We train on patches of size 64×64 (using Adam and a learning rate of 2×10^{-4}) and evaluate on patches of size 256×256 to give our model additional context at test time.

D Additional Data Explorations

What determines how easily images can be compressed? We answer this question via inspiration from a study by Pence et al. 2009, who suggested that the compressibility of astronomy imagery is largely determined by the noise level of background pixels, which are not part of "source" objects such as stars and galaxies. The majority of pixels in our images are background pixels.

Shannon's source coding theorem establishes that the entropy of a data source defines the lower bound for the optimal bitrate. Pence et al. 2009 show that if we assume that background pixels are drawn from a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, then the corresponding per-pixel entropy in bits is proportional to $\log_2(\sigma)$. We plot this quantity below, against JPEG-LS bits per pixel.

While background pixels are not exactly drawn from a Gaussian, and JPEG-LS is not a perfect codec, the linear relationship on this plot suggests that background noise levels significantly influence an

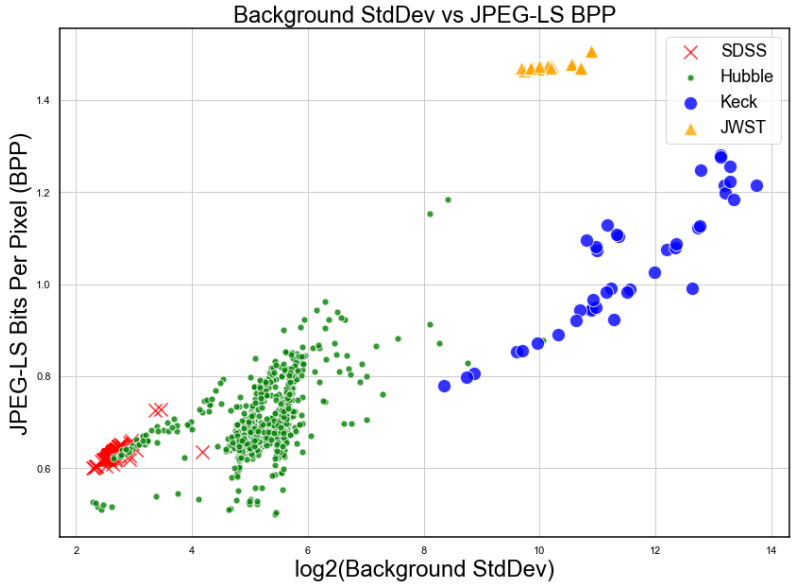

Figure 5: Correlation between background noise variation and compressibility. Each point represents a full-size image. Horizontal axis: log standard deviation of the lower 99 percent of pixel values in a given image. Vertical axis: JPEG-LS bits per pixel, as a representative codec. Single-frame datasets only.

image's compressibility. Moreover, Figure 2 reinforces this insight: as exposure time increases, the number of noise bits in background pixels rises, thereby reducing the image's compression potential.

We note here that the SDSS and Keck datasets show a much tighter linear relationship than Hubble images, suggesting that source pixels are more frequent and more relevant, since we have verified that the background pixels appear quite Gaussian. This corresponds well with intuition, as Hubble is known to sometimes take deep, long exposures of dense stellar fields. JWST images appear as outliers, as it uses HgCdTe detectors for infrared bands, compared to the CMOS detectors for optical imaging of all other telescopes.

E Evaluating Pretrained Models

We briefly perform a preliminary exploration of how well a neural compression method trained on RGB images generalizes to our astronomy data. Compression models designed for RGB images cannot directly handle our 16-bit data; therefore, we implement a workaround by reformatting each 16-bit pixel as consisting of 3 channels of 8 bits like [MSB, MSB, LSB], where we duplicate the most significant bits (MSB) across the first two channels and fill the last channel with the least significant bits (LSB). As an alternative method, we also tried filling the last channel with only zeros [MSB, LSB, **0**]. We will call these workarounds "duplicated" and "zero-padded," respectively.

We pretrained the L3C model on RGB images from OpenImages (generally around 500 pixels in at least one dimension) and ImageNet64 datasets and evaluated on our single-frame datasets. We also trained IDF on RGB images from Cifar10 dataset and evaluate in the same manner. Overall, we found our RGB-pretrained model to give surprisingly robust performance on astronomy data. When compared to the L3C model pre-trained on the respective astronomy datasets, the RGB-pretrained model did worse by 15% - 25% on KECK, HST, and JWST-2D datasets but actually performed better by 10% - 28% on LCO and SDSS datasets in terms of compression ratio. The detailed results are shown in table 6.

Model / Dataset	L3C-OpenImages (zero-padded)	L3C-ImageNet64 (zero-padded)	L3C-OpenImages (duplicated)	L3C-ImageNet64 (duplicated)	IDF-Cifar10 (zero-padded)	IDF-Cifar10 (duplicated)
LCO	1.78	2.16	2.04	2.05	1.72	1.84
Keck	1.26	1.44	1.27	1.38	0.66	0.59
Hubble	1.77	2.18	1.88	1.90	0.59	1.08
JWST	1.08	1.19	1.06	1.18	0.62	0.58
SDSS	2.03	2.58	2.30	2.14	1.68	1.62

Table 6: Compression ratio of RGB-trained models across different astronomy datasets.

F Further Motivation

We present below a brief amount of quantitative evidence on the explosion of the astronomy data scale and the need for advances in its processing:

Astronomy data volumes are growing at an apparently superexponential rate (see Figure 1 of (Maireles-González et al., 2023)). A growth rate of about 10x every 10 years has become nearly 1000x every 10 years. The ability to transmit and store this data will become a massive burden preventing this growth rate from continuing. We hope for advanced compression to alleviate some of this problem.

The current state-of-the-art supercomputers administered by the United States Department of Energy were only recently made ready to handle exascale computing in 2023[4]—specifically Frontier and Aurora, the current two most powerful systems in the world according to `top500.org`. While this "exascale" refers to exaflops rather than exabytes, the need for compute scale is built specifically to address data scale.

In sum, we repeat the breadth of data sources that will soon reach exabyte scale: radio astronomy, satellite imagery[5], genomes, brain mapping and beyond. We note the massive economic potential seen in some of these fields. Lossless methods can be applied without fear, but very carefully crafted lossy designs may soon bring forth orders of magnitude more performant pipelines.

[4] https://science.osti.gov/-/media/bes/besac/pdf/202212/7-Helland--BESAC-Panel.pdf

[5] https://www.earthdata.nasa.gov/s3fs-public/2022-02/ESDS_Highlights_2019.pdf