Multi-Modal Multi-Task Data Compression

Narek Alvandian
EPFL
narek.alvandian@epfl.ch

Roman Bachmann
EPFL
roman.bachmann@epfl.ch

***Abstract*—Data compression is an essential part of our everyday computer workflow, making our communication systems and hardware more effective and efficient. Neural Networks (NNs) are a powerful tool for dealing with compression, significantly outperforming traditional methods. To the best of our knowledge, however, they all have been trained to compress one type of information, like natural-looking RGB images. At the same time, other fields of Machine Learning greatly benefit from working with multiple modalities, exploiting connections between different types of data. With this work, we bring those ideas to compression. We also show that naively applying existing Multi-modal Multi-task (MT) methods for compression might bring additional overhead in the number of bits and reconstruction error, instead of bringing benefits. We address these problems to design solutions that carefully handle compression-related specifics. We leverage MT architectures to get better compression for multi-task datasets while disentangling latent representations to allow storage and distribution of a subset of all tasks in the dataset, which would not be possible when using classical MT methods.**

I. Introduction

Data compression is an essential part of our everyday computer use. It saves both space on our computers, increasing hardware efficiency, and communication speed - reducing the number of bits to be transferred through the internet.

Neural Networks (NNs) are a powerful tool for data compression [29, 5, 4, 6, 32], significantly outperforming traditional codecs like JPEG, HEVC, and MP3. For images in particular, besides showing better objective metrics, the reconstructions are also more perceptually appealing and do not have unnatural artifacts like JPEG when compressed for lower bit-rates [4].

To the best of our knowledge, however, all learning-based compression techniques, so far, have been tailored for a single modality (e.g. RGB images). In other applications of machine learning, multi-modal (multiple inputs: RGB, depth map, audio, text, etc.) [8, 7] and multi-task (multiple outputs: RGB, normals, sentiment analysis, etc.)[26, 10, 31, 27] architectures are leveraged to improve performance on the downstream tasks. Different modalities often provide very different information about the data, so it is not surprising that multi-modal models can learn richer representations. It is hard to think of reasons why the same approach wouldn't bring benefits to compression. Having said that, it is important to understand *why* do we expect to see benefits and whether something needs to be changed for *compression* models.

Any compression algorithm transforms initial data into a code of lesser size (in the number of bits) and then recovers the same data from that code. We can look at this as if there are two functions: one extracts and encodes the core information about data into the code, the other uses that core information to decode the initial data. Thus, if we want to store the least amount of bits - it is important to avoid having duplicate information in the code.

In our setting though, multi-task datasets certainly have that "duplicate information" across tasks, like depth/color/normal discontinuities are all a result of the same underlying phenomena, but are still represented in multiple images when working with multi-task data. In a MT setting we utilize this understanding and construct models to learn representations for downstream tasks. We could also use such architecture to construct a compressor neural network.

For such a compressor to be more useful, one should be able to choose which tasks to store and decode. This would require the latents to be task-specific so that we can store and distribute only a subset of tasks. That requirement, unfortunately, is not satisfied in a default MT setting, where the learned representation (code) is a single multidimensional vector for all tasks, and it is hard to separate task-specific parts in that code without losing much performance.

Notice the arising dichotomy: we want to share information across modalities to learn better representations, but we do not want to share too much of it, so we can still have task-specific codes and decoders.

In our work, we suggest solutions that carefully address this problem. We leverage multi-modal multi-task architectures to get better compression for multi-task datasets while disentangling latent representations to enable working with a subset of codes.

In section II we review the Information Theoretic background of the problem, previous work on Neural Networks for Compression, and core ideas of Multi-taskness and Multi-modality. In section III we describe suggested architectures used for experiments and explain the rationale behind their design. In section IV we describe our setup: the dataset, hyper-parameters of our models, multi-task loss balancing, and optimizers used for experiments. Finally, section VI has discussion of potential extensions and shortcomings of current work.

II. Related Work

A. Data Compression

The goal of data compression is to derive a representation or a *code* for initial data, which takes up fewer bits to store but from which we would be able to reconstruct our initial data back. If we expect reconstructions to be perfect - the problem is that of *Lossless Data Compression*, if we allow approximate reconstructions, then we are in a *Lossy Data Compression* setting.

In this work, we use Neural Networks for Lossy Data Compression, but Lossless algorithms are an essential part of that pipeline, so in the next section, we review both.

For **Lossless Data Compression** Consider $\{0,1\}^*$ to be the notation for a set (or a bit-string) of 0s and 1s of arbitrarily (but finite) length. Let $X = (x_1, x_2, ..., x_N)$ where $x_i \in \{0,1\}^*$ be a random source generating values x_i and p_X to be a probability density function of that random process. Our goal is then to construct a uniquely decodable function $c : X \to \{0,1\}^*$ s.t. the following expected length of the code - the **bitrate** - is minimized.

$$\min_c E[l(c(X))] = \min_c \sum_i^N p_X(x_i) l(c(x_i))$$

where $l : \{0,1\}^* \to \mathbb{Z}^+$ - is the function that maps a bit-rate to its length.

From the work of Claude Shannon [20] we know that the lower bound of this value is the entropy $H(X)$ of the source X, defined as follows:

$$H(X) = \sum_i^N p_X(x_i) \log \frac{1}{p_X(x_i)}$$

This is why algorithms for lossless compression are referred to as **entropy coding** algorithms. From the equation above we can see that the optimal length of a code for some bit-string x_i is the logarithm of its inverse probability. There exist classical algorithms such as Huffman Coding, Arithmetic Coding, and Bits-Back Coding [25] that given this distribution p_x can achieve bitrates close to the entropy.

However, for real-world data - we never know the real distribution p_X, and the best we can do is approximate it with some distribution q_X. In this case, **Cross-Entropy** becomes the new lower bound:

$$CE(X) = \sum_i^N p_X(x_i) \log \frac{1}{q_X(x_i)} = H(X) + D_{KL}(p_x||q_x)$$

Notice how we can rewrite it as the sum of the entropy $H(X)$ and the Kullback-Leibler divergence between two distributions p_X and q_X: $D_{KL}(p_x||q_x) = \sum_i p_X(x_i) \log \frac{p_X(x_i)}{q_X(x_i)}$. Since we cannot influence the entropy of the process - it is clear that the problem of lossless data compression can be re-framed as one of density estimation, since $D_{KL}(p_x||q_x) = 0 \iff p_X = q_X$. In our setting, this density estimation is performed by Neural Networks.

As for the **Lossy Data Compression** - it relaxes the problem above and allows to reconstruct the data approximately.

In this case, we optimize the so-called Rate-Distortion objective:

$$\min_{e,d} R(e(X)) + \lambda \cdot E_p[D(X, d(e(X)))]$$

where e and d are the so-called analysis and synthesis functions (often referred to as g_a and g_s), which encode and decode the data respectively, and $\lambda \in \mathbb{R}$ is a trade-off parameter. The rate R is defined as Cross Entropy, and distortion D is some measure of the difference between decoded data $\hat{X} = d(e(X))$ and initial data X. This pipeline is often referred to as **transform coding** because we first transform the data into a more compressible representation. Here function e is used to decorrelate parts of the data, and then losslessly compress those new representations. Note, however, that to losslessly compress some distribution, we need it to be discrete. If this is not the case by default, then an additional quantization function q is used on the outputs of e.

To sum up, **Lossy Data Compression pipeline** has following steps:

- Encode initial data X into a representation Y using analysis (encoder) function $e(.)$;
- Use q to map (possibly) continuous values Y to a set of discrete values $\hat{Y}$;
- Fit such a Probability Density Function $q_{\hat{Y}}$ that minimizes $D_{KL}(p_{\hat{Y}}||q_{\hat{Y}})$ with $p_{\hat{Y}}$ being the true distribution of the encoded data;
- Use an entropy coder c to losslessly compress $\hat{Y}$ into (or later decompress from) a bitstring;
- Get a reconstruction $\hat{X}$ from $\hat{Y}$ using synthesis (decoder) function $d(.)$.

One classical transform-coding algorithm for Lossy Compression is JPEG. It uses Discrete Cosine Transform as $e(.)$ and it's inverse as $d(.)$. Then it losslessly encodes the transformed representations into bitstrings and decodes them back.

In general, though, there is no reason to be constrained by linear transform functions. This too is where Neural Networks come in.

B. Neural Networks for Data Compression

Although neural networks have been used for image compression since 1980s [13] we will only discuss methods starting from the introduction of the first End-to-End learnable compression framework [4], since these are more related to our approach. For those who are interested in learning more, however, the following reviews [29, 5] cannot be recommended enough.

The words compression and decompression should've hinted at the use of Autoencoders (AEs) and the fact that we need to estimate some densities should've narrowed the search

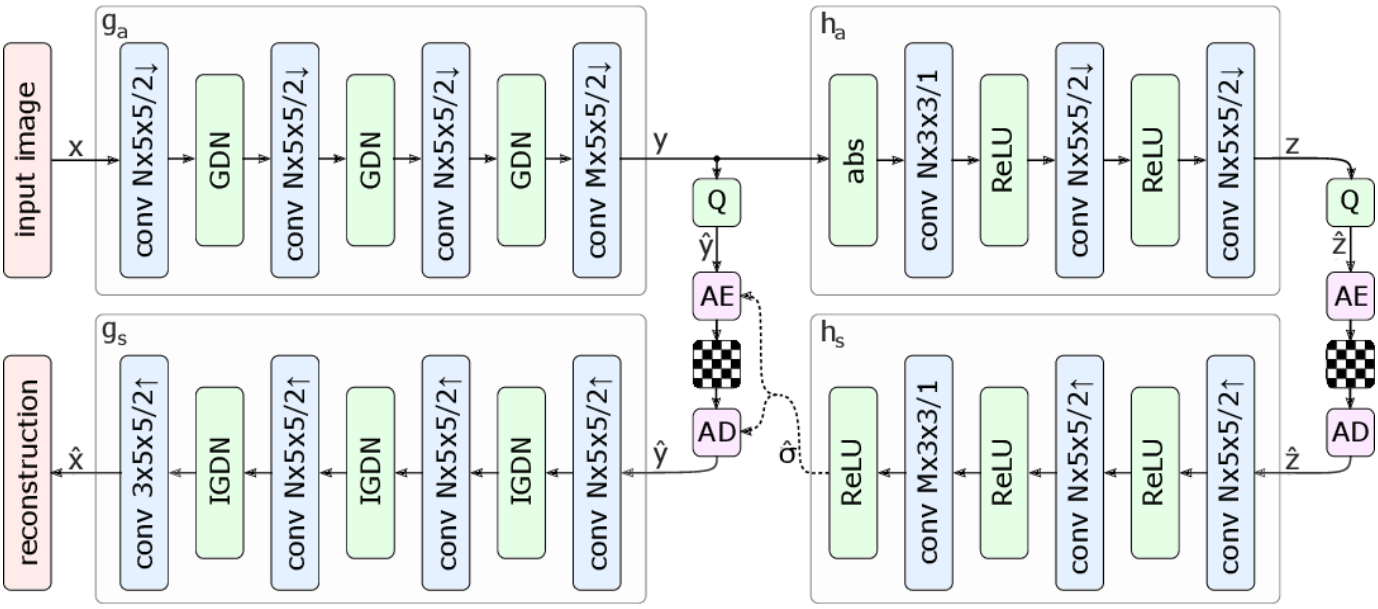


Fig. 1: Scale Hyperprior architecture used in our work as a backbone compressor. Q is the quantization step, AE/AD stands for Arithmetic Encoder/Arithmetic Decoder - a common entropy coding technique and the black-and-white checkerboard notation illustrates the binary code. The figure is taken from the original paper [6]

to Variational Autoencoder (VAEs) [18]. These indeed are the core models behind modern compression methods.

Autoencoders for Compression. In the first work on End-to-End learnable image compression [4], authors chose to use convolutional/upsampling layers as encoder and decoder functions respectively. The choice of non-linearity fell on Generalized Divisive Normalisation (GDN) and its inverse (IGDN), since it was previously shown to perform well in density estimation tasks [4, 3, 2].

The same Convolutional Neural Network approach was chosen later in several works [22, 6, 17, 9], making the networks deeper, and eventually moving to visual transformers [32].

As for the **Loss Function**, an instance of the Rate-Distortion objective is minimized:

$$\min_{\theta,\psi} H(\hat{Y}) + \lambda \cdot D\big(X, \hat{X}\big)$$

where $\hat{x}_i = d_\psi(\hat{y}_i) = d_\psi\Big(q(y_i)\Big) = d_\psi\Big(q(e_\theta(x_i))\Big)$, θ and ψ are learnable parameters of the network.

Measuring distortion can be done by using Mean Squared Error, L1 loss, MS-SSIM [28], etc.

In the majority of work [4, 22, 6, 17, 9, 32], λ is fixed during training and inference process, meaning that if we want to change the compression quality - we have to learn/use a new model. Although, it doesn't have to be that way. There exists an RNN-based family of methods [24] which allows for easy change of the rate-distortion trade-off without the need to retrain a model. As a side note - it was quite interesting to note how similar that work is to diffusion models [21, 12, 19], first developed around the same time - but having their popularity surge in the last couple of years for applications to generative modeling (which has many intersections with compression).

Notice that in the loss function, we have an entropy term $H(\hat{Y})$ which means that we need probabilities or likelihoods for each $\hat{y}_i$. Before estimating any probability measures, we need to discretize the values which come from the encoder function $e(x_i)$. Although Differential Entropy also exists - reporting those is not quite fair because such estimates are noticeably more optimistic [4].

Discretizing Continuous-Valued Vectors by rounding each dimension's scalar to the nearest integer would result in derivatives being zero almost everywhere, or even undefined at some points during backpropagation. This is why there exist multiple techniques to make quantization differentiable [23] [1] [4]

A widely used one is to add noise sampled uniformly at random from $[-0.5, 0.5]$ during training and exactly round values at each dimension during inference. It was shown in [4] [6] that this kind of quantization followed by a point-wise convolution with a Uniform $[-0.5, 0.5]$ kernel can be an infinitely precise, yet smooth approximation of our naive rounding function.

Learning a Probability Density Function over Quantized Latents is essential to compress those discrete values losslessly to a bit-rate close to entropy. We need to fit a PMF q over those discrete values as close to p as possible, which can be done in many ways.

Each latent dimension $\hat{y}_i$ is usually modeled with a separate distribution. In [4] each marginal is modeled with a piece-wise linear non-parametric function, which uses 10 sampling points per unit interval and the learned parameters are just probabilities of each value (basically fits a histogram to our

data).

It was noted [6], though, that such latents are not decorrelated properly - neighboring distributions would have similar scales (which is not optimal, since any correlations in the latent cause inefficiencies for entropy coding). And interestingly enough, the authors showed that this problem is not solved by simply scaling up the model (in the hope of increased learning capabilities).

To solve this, a so-called Hyperprior was introduced. Now each marginal was modeled by a zero-mean Gaussian with a learned scale (standard deviation) and another non-parametric density estimation was performed to learn the distributions of those scales. A general architecture of a Scale Hyperprior Network is shown on Figure 1. Later that idea was extended to also learn the distribution of non-zero means and to learn them auto-regressively [17] and even using transformers [32].

It's worth noting, that the general outline of networks didn't change at all since the Scale Hyperprior work. Noteworthy performance improvements came from using a different entropy coding technique, modeling the prior and hyperprior differently, and of course - introducing Transformers to all possible parts of the network.

As a side note. It's also interesting to see that the relaxed rate–distortion optimization problem bears some resemblance to those used to fit generative image models, and in particular, Variational Autoencoders (VAEs) [18], but differs in the constraints we impose. Nevertheless, if quantization is made with uniform noise and MSE is used as a distortion metric, then this is exactly the setup of a classical VAE with Gaussian prior [6].

C. Multi-modal and Multi-task Learning

Multi-modal learning is concerned with constructing representations from multiple modalities. Such methods are expected to combine the unique information that each modality provides to better perform the downstream task.

Our solution is indeed Multi-modal, but it is also **Multi-task**, which means that we have multiple outputs. The main idea here is quite similar to Multi-modality, in that - we want to combine information from different sources, except the implementation is symmetrically mirrored. A classical example of a multitask problem would've been to predict the depth, semantic mask, and bounding boxes of objects, given only an RGB image for input.

III. Method

We want to design a Multi-modal Multi-task algorithm, which will be able to leverage connections between different modalities, with the possibility to store and decode only the tasks of interest while getting the benefits of finding connections between all of the shown tasks. Such a compressor, not only will be more useful for the community, because of its flexibility for stored data but we also expect it to outperform other learning-based compressors trained on a single modality even in their domains.

Initially, it might seem that it suffices to choose a MT architecture, quantization, and add an Entropy term to the loss. Although this would certainly work, we show that a noticeable compression overhead is expected.

Also, assume we want to compress a multi-million image multi-task dataset, like Taskonomy [30] or ones that can be generated using Omnidata [11]. In that case - it will be highly desirable to be able to choose a subset of tasks to store, distribute and decode.

We can achieve selective decoding with a naive out-of-the-box solution by passing the latent codes only to the decoders of specified tasks. Despite wasted computation, selective decoding indeed seems easy without much change to default Multi-learning setups.

Selective storage of the codes, however, is more complicated.

A. Architectures

We propose the following two solutions to separate the codes by task, while still leveraging multi-modality.

We will refer to the first one as **Disjoint Latent Model**. A schema with a description of this method's workflow is shown on Figure 2. The idea here is to have a unified encoder that takes all modalities as inputs and produces a latent with M channels, where M channels are equally distributed across T tasks. This task-wise channel specialization can be achieved by passing a *subset* of $\frac{M}{T}$ channels to each of the task-specific decoders. For example, consider a latent of size $M = 60$ and $T = 3$ tasks. Then each of the 3 decoders gets 20 channels as input. Thus, during backpropagation, channels will only receive gradient updates related to one task, but since the encoder works with all modalities - we still expect to see benefits from Multi-taskness and Multi-modality.

In an MT setting each task is expected to have some information, that is useful to multiple (maybe even all) tasks plus some task-specific information. To successfully solve the task, *each* of the disjoint latents of size $\frac{M}{T}$ must store *both* shared and task-specific information resulting in duplicate storage of the "shared" information.

This intuition is also grounded from the Information Theoretic point of view. Consider Y_1 and Y_2 to be two distributions of those disjoint latents. Then the amount of information that we store is approximately equal to:

$$H(Y_1) + H(Y_2)$$

where H is the entropy of a random variable.

Now consider Y_1 and Y_2 joint distributions s.t.

$$\begin{aligned} p(Y_1) &= p(\tilde{Y}_1, Y_s) \\ p(Y_2) &= p(\tilde{Y}_2, Y_s) \end{aligned} \tag{1}$$

where Y_s can be viewed as a random variable that carries some information useful for both tasks, and $\tilde{Y}_i$ is a r.v. with task-specific information. Then by definition, our total entropy can be rewritten in the following way:

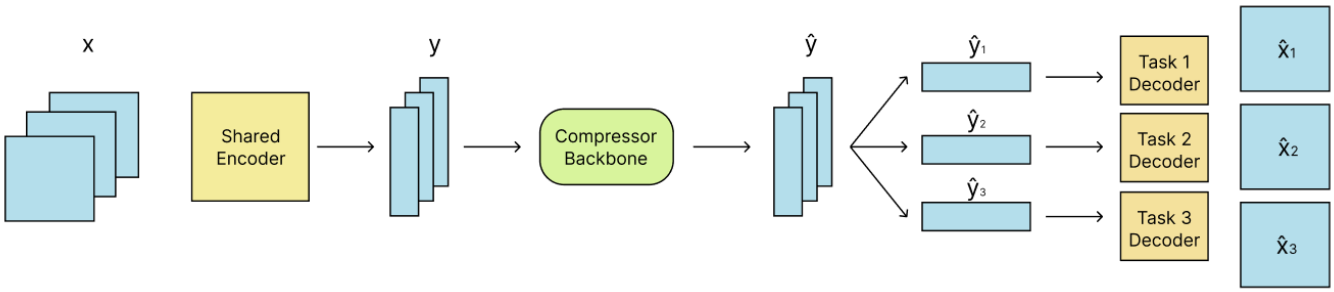


Fig. 2: **Disjoint Latent Compressor**. All 3 input modalities go through a shared encoder. This encoder produces latents y with the number of channels being a multiple of the number of tasks (on the figure it's 1 channel per each task for simplicity). y then goes through a compressor backbone (for example - Scale Hyperprior) where it is quantized, compressed into a binary string, and then recovered into $\hat{y}$. Then those disjoint groups of channels (here, again, one per task for simplicity) are inputs to task-specific decoders, which produce according reconstructions $\hat{x}_i$

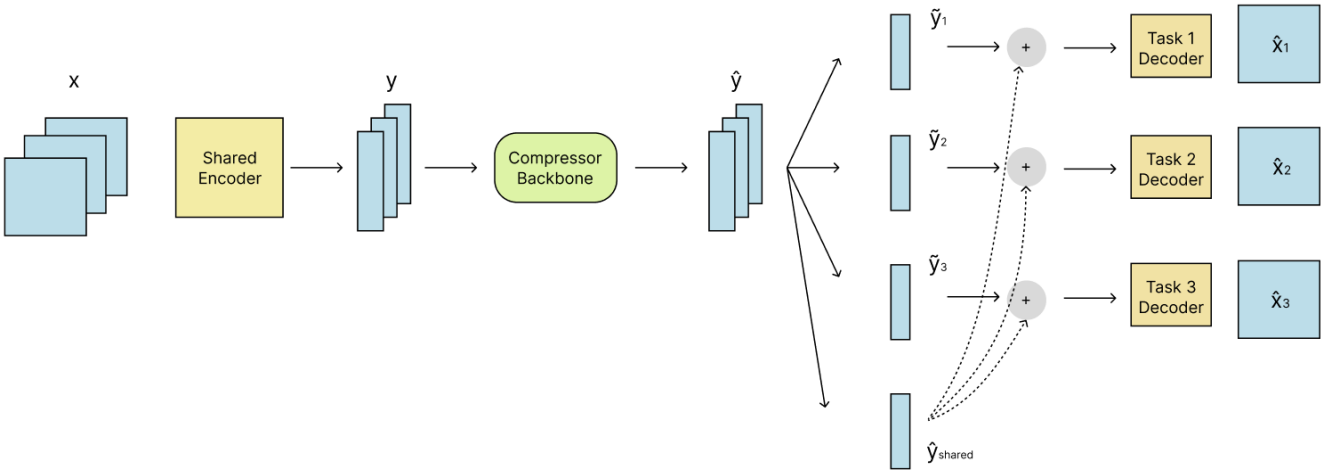


Fig. 3: **Shared Latent Compressor**. We begin similarly to Disjoint Latent Compressor. First, we get latents y by doing a forward pass of all inputs through a shared encoder. Then those latents go as input to the compressor backbone, which produces their reconstructed version $\hat{y}$. Now, unlike the Disjoint Latent model, the task-specific subset $\hat{y}_i$ of $\hat{y}$ is concatenated with the shared subset $\hat{y}_{shared}$ and passed to a task-specific decoder to get an according reconstruction $\hat{x}_i$. Note that in this setting $\hat{y}_{shared}$ contributes to all tasks, while $\hat{y}_i$ remain task-specific.

$$
\begin{aligned}
&H(Y_1) + H(Y_2) = \\
&H(\tilde{Y}_1, Y_s) + H(\tilde{Y}_2, Y_s) = \\
&H(\tilde{Y}_1|Y_s) + H(Y_s) + H(\tilde{Y}_2|Y_s) + H(Y_s)
\end{aligned} \tag{2}
$$

And we see that we store the shared information $H(Y_s)$ twice in the case of two tasks.

Guided by this intuition we propose the **Shared Information Model**. The idea is to learn a single latent with information useful for all tasks, and also learn T task-specific latents. In this case, each of the T decoders gets as input a combination of this shared latent and it's task-specific one. Note that here we store shared information only once. This architecture is shown on Figure 3. We still have a single encoder that takes all modalities as input. It then produces a latent vector with M channels, then this latent vector is split to $T + 1$ vectors with equal number of channels (T for task-specific ones and +1 shared). *Each* decoder then gets as input the concatenation of a task-specific and shared latent. Thus, during backpropagation, we expect that the shared latent will get parameter updates from all the tasks, while the task-specific latents will only get updates from an according task. Note that in this case, simillarly to the Disjoint Latent model each of the decoders gets the same M/T number of channels.

B. Metrics

As reconstruction metrics we used: Mean Square Error and Peak Signal to Noise Ration (PSNR), where PSNR was computed in the following way.

$$PSNR(x, \hat{x}) = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE(x, \hat{x})} \right)$$

and MAX_I is the maximum value of the image pixel.

For compression we measured the Bits Per Pixel (BPP). Which is the number of bits that one image's latent takes (after being compressed with a classical codec) divided by the resolution of the image.

IV. Experimental setup

We worked with 3 modalities/tasks: RGB images, per-pixel depths and normals. Although we focused on these specifically, out approach can be extended to any choice and number of modalities.

A. Data

All experiments were performed on an in-house pseudo-labeled version of the CLEVR [14] dataset. We had 50k training and 5k validation points per task.

All images were rescaled from 512x512 to 256x256. RGB and Normals were also normalized to be in [0, 1].

B. Models

We used three additional setups for our baselines:

- First, we used a publicly available Scale-Hyperprior network trained on a large set of natural-looking RGB images. Authors published 8 versions with increasing reconstruction quality. In our experiments we used only 5 of them, because other versions had twice the number of parameters.

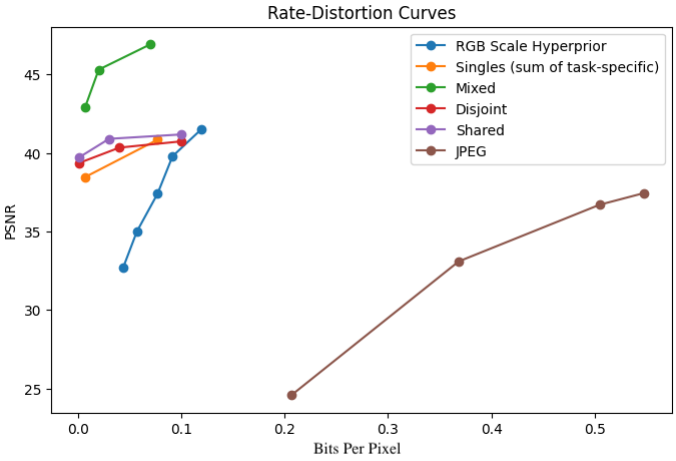
- Second, we trained multiple single-task compressors for each of the modalities. One model for one task. We refer to these as **Single Task Compressors**.

- Finally, we also trained a Multi-modal Multi-task compressor, but one that doesn't have separable latents. This is a multi-headed AutoEncoder with T input and T output heads. Here we do not have any task separation in the latents. We refer to this model as **Mixed Latent Compressor**.

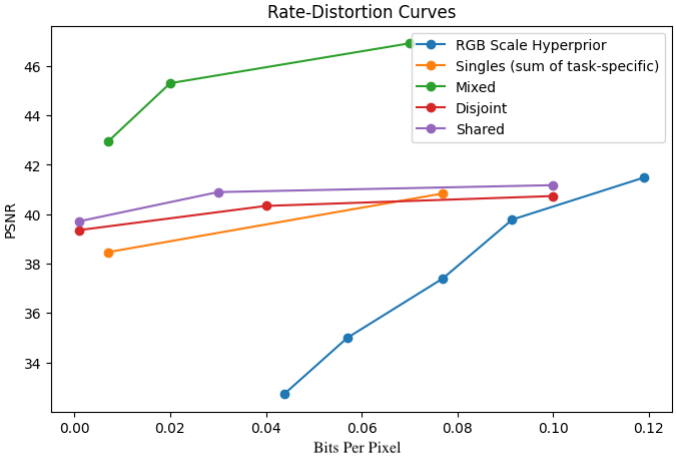
For all of our setups we used Scale Hyperprior [6] network as a backbone compressor.

We tried to make performance comparison as fair as possible by proxying the "learning ability" of the networks by their sizes. For all of our setups we maintained approximately same number of parameters ∼5M.

To achieve that, all multi-task networks (Mixed Latent, Disjoint Latent, Shared Latent) have the same latent code size $L = 300$ and approximately the same number of convolutional filters per layer C ($C = 32$ for Mixed and $C = 42$ for Shared and Disjoint networks due to specifics of the architecture). The pretrained Scale Hyperprior network also has ∼5M parameters. For the setup, where we trained one compressor network per task, we made it such that the sum of all T ($T = 3$ in our case) network parameters is ∼5M, which resulted into $C = 48$.



(a) Rate-distortion curve for all models across tasks



(b) Rate-distortion curve for all models except JPEG

Fig. 4: Rate-Distortion performance across models, averaged over tasks

C. Training

We trained our models using Adam[16] optimizer with a batch size of 64 for 2200 epochs. The learning rate would go from 10^{-5} to 10^{-8} with cosine decay scheduling.

All three tasks were learned using the previously-mentioned Rate-Distortion objective, where distortion was measured as MSE$*255*255$ with 3 different values of $\lambda = 0.1, 0.01, 0.001$ choosing this way the Rate-Distortion trade-off. We also used uncertainty weighting for balancing the multi-task loss. [15].

V. Results

A. Compression Performance

Consider the Rate-Distortion plots on Figure 4a and Figure 4b. Figure 4a shows how much better all of the learning-based models are compared to JPEG.

To better see the differences in performance between learning-based models, we can use Figure 4b which excludes the JPEG curve.

There ares several questions that can be answered by looking at this figure.

First, it's clear that three smaller single-task models outperform a general RGB-learned network. The natural RGB Scale Hyperprior was only trained on a single modality, this way normals and depths should be very much out-of-distribution, resulting into worse compression performance.

Second, we see that all three multi-task networks have either comparable or noticeably better performance than the sum of the three single-task compressors.

We also notice that the best performance *across all models* comes from the Mixed Latent Compressor. We expected such results since the Mixed Latent model fully leverages multi-taskness and multi-modality without storing duplicate information in the latent. All M latent channels are shared between task-specific decoder heads.

Continuing with a similar line of though, we expected the Disjoint Latent Model to have a a comparable, or maybe slightly better performance, than the three Single Task Compressors. This model has an encoder that is able to leverage multi-modality, but it was also expected to perform noticeably worse than the Mixed Latent model, because to successfully solve all tasks it should store a copy of the shared information in each task's latents.

Finally, we see that we indeed obtained a better compressor by adapting our multi-task architecture from a naive Disjoint Latent to the Shared Latent one. This change in performance, however, is barely noticeable. We expected our metrics to be close to the Mixed Latent Model, arguing that both models leverage multi-modality, while leaving no room for duplicate information.

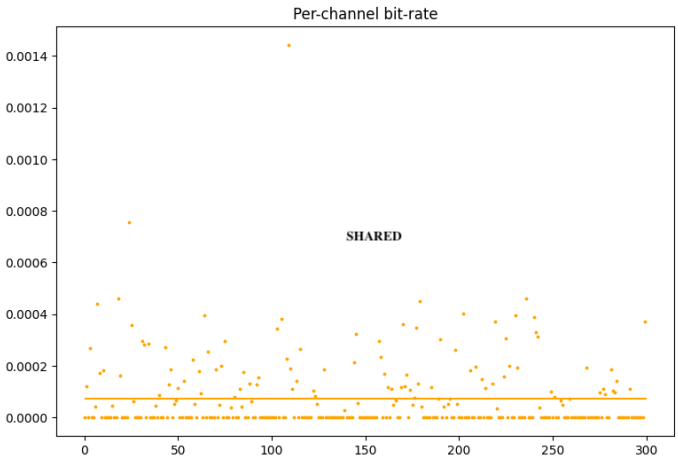
To better understand these results we further investigate the shared latent model.

B. What does the shared latent encode?

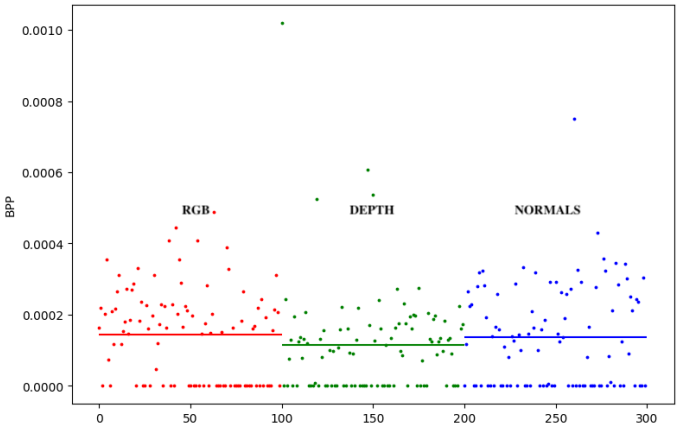
Figure 5 shows the distribution of information across the channels of our multi-task models ($\lambda = 0.001$) with coloring by tasks. Note that many of the channels have 0 bits of information. This is because we're operating in the most aggressive compression setting, so big chunk of channels has constant values resulting into 0 bits of information.

As expected, in the Mixed Latent case the distribution looks homogenious across channels.

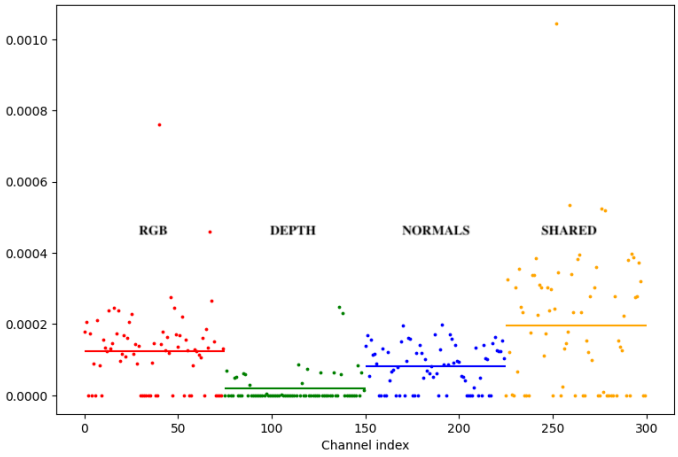
Comparing Disjoint and Shared models is more interesting. As it was expected for the shared model all three tasks have less information compared to the Disjoint case, because some parts of it moved to the shared latents. However, the difference in that decrease is not uniform. RGB values clearly decreased the least, while normals and especially depth have noticeably less information. This can be more clearly seen on Figure 6a and 6b. The most striking in this figure is that depth channels have almost no information left.



(a) Mixed Latent Compressor

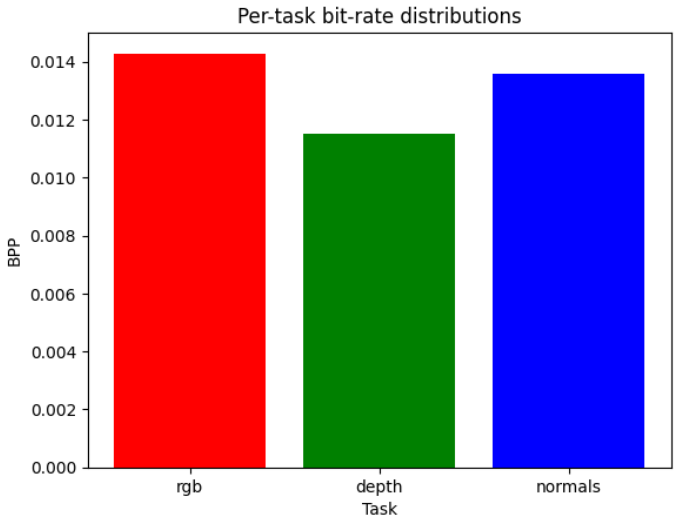


(b) Disjoint Latent Compressor

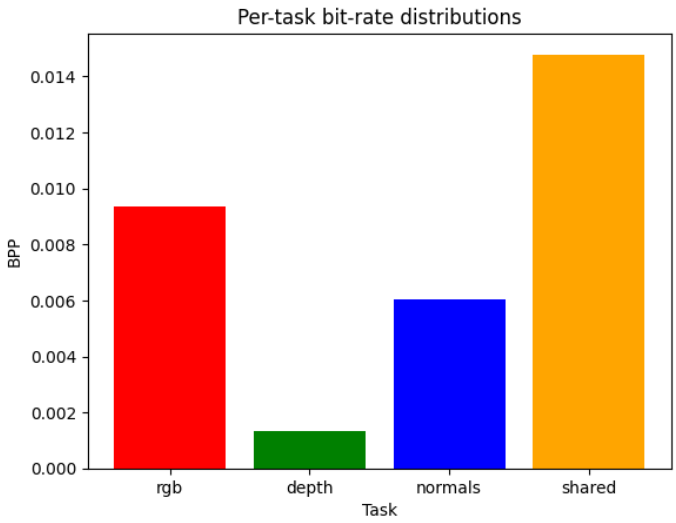


(c) Shared Latent Compressor

Fig. 5: Per-channel bit-rates of Multi-task Compression Networks. Colors show the task, stored in the according channels of the latent. Solid lines show the per-task mean BPP.



(a) Disjoint Latent Compressor



(b) Shared Latent Compressor

Fig. 6: Bar plots showing the distribution of bits over task channels for Disjoint Latent and Shared Latent models

Intuitively, this makes sense. We know that depth and normals represent almost the same information but in different forms. We also understand that some of the depth/normal information can be infered from the RGB image. These observation support the idea that is suggest by the plot, namely "the depth information is shared across tasks". However, numerically it is not yet obvious that this is the case. Ideally, for the Shared Latent Compressor we would want to see compression rates close to the Mixed Latent model to get better support of out hypothesis.

VI. Discussion

In this work, we suggested architectures that perform Multi-task and Multi-modal data compression, noticeably outperforming those trained for a single task, especially at extremely low bit-rates. This could be useful for compressing multi-task datasets for more effective storage and distribution.

We also attempted to make our compressor more user-friendly by adding a feature of selective task storage and decoding. As expected and partially explained by our analysis, such a feature would add noticeable overhead to the compression rate, so we suggested a better architecture for compressing with separable latents. Although we were able to get a slight improvement over the naive approach, the results were far from the expected outcome and don't perfectly align with the hypothesis suggested in section III.

Further investigation of the shared latents and per-task compression rates is needed to obtain more conclusive results and explanations. Would be interesting to look at the per-task rate-distortion curves and to substitute the shared latents with different values to see the changes in reconstructions.

References

[1] Eirikur Agustsson et al. "Soft-to-hard vector quantization for end-to-end learning compressible representations". In: *Advances in neural information processing systems* 30 (2017).

[2] Johannes Ballé. "Efficient nonlinear transforms for lossy image compression". In: *2018 Picture Coding Symposium (PCS)*. IEEE. 2018, pp. 248–252.

[3] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. "Density Modeling of Images using a Generalized Normalization Transformation". In: (2015). DOI: 10.48550/ARXIV.1511.06281. URL: https://arxiv.org/abs/1511.06281.

[4] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. "End-to-end Optimized Image Compression". In: (2016). DOI: 10.48550/ARXIV.1611.01704. URL: https://arxiv.org/abs/1611.01704.

[5] Johannes Ballé et al. "Nonlinear Transform Coding". In: *CoRR* abs/2007.03034 (2020). arXiv: 2007.03034. URL: https://arxiv.org/abs/2007.03034.

[6] Johannes Ballé et al. *Variational image compression with a scale hyperprior*. 2018. DOI: 10.48550/ARXIV.1802.01436. URL: https://arxiv.org/abs/1802.01436.

[7] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. *Multimodal Machine Learning: A Survey and Taxonomy*. 2017. DOI: 10.48550/ARXIV.1705.09406. URL: https://arxiv.org/abs/1705.09406.

[8] Khaled Bayoudh et al. "A survey on deep multimodal learning for computer vision: advances, trends, applications, and datasets". In: *The Visual Computer* (2021), pp. 1–32.

[9] Zhengxue Cheng et al. "Learned image compression with discretized gaussian mixture likelihoods and attention modules". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 7939–7948.

[10] Michael Crawshaw. *Multi-Task Learning with Deep Neural Networks: A Survey*. 2020. DOI: 10.48550/ARXIV.2009.09796. URL: https://arxiv.org/abs/2009.09796.

[11] Ainaz Eftekhar et al. "Omnidata: A scalable pipeline for making multi-task mid-level vision datasets from 3d scans". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10786–10796.

[12] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6840–6851.

[13] J Jiang. "Image compression with neural networks–a survey". In: *Signal processing: image Communication* 14.9 (1999), pp. 737–760.

[14] Justin Johnson et al. "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2901–2910.

[15] Alex Kendall, Yarin Gal, and Roberto Cipolla. *Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics*. 2017. DOI: 10.48550/ARXIV.1705.07115. URL: https://arxiv.org/abs/1705.07115.

[16] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[17] David Minnen, Johannes Ballé, and George D Toderici. "Joint autoregressive and hierarchical priors for learned image compression". In: *Advances in neural information processing systems* 31 (2018).

[18] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. *Stochastic Backpropagation and Approximate Inference in Deep Generative Models*. 2014. DOI: 10.48550/ARXIV.1401.4082. URL: https://arxiv.org/abs/1401.4082.

[19] Robin Rombach et al. "High-resolution image synthesis with latent diffusion models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 10684–10695.

[20] Claude E Shannon. "A mathematical theory of communication". In: *The Bell system technical journal* 27.3 (1948), pp. 379–423.

[21] Jascha Sohl-Dickstein et al. "Deep unsupervised learning using nonequilibrium thermodynamics". In: *International Conference on Machine Learning*. PMLR. 2015, pp. 2256–2265.

[22] Lucas Theis et al. "Lossy image compression with compressive autoencoders". In: *arXiv preprint arXiv:1703.00395* (2017).

[23] George Toderici et al. "Full resolution image compression with recurrent neural networks". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 5306–5314.

[24] George Toderici et al. "Variable rate image compression with recurrent neural networks". In: *arXiv preprint arXiv:1511.06085* (2015).

[25] James Townsend, Tom Bird, and David Barber. "Practical lossless compression with latent variables using bits back coding". In: *arXiv preprint arXiv:1901.04866* (2019).

[26] Simon Vandenhende et al. "Multi-Task Learning for Dense Prediction Tasks: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: 10.1109/tpami.2021.3054719. URL: https://doi.org/10.1109%2Ftpami.2021.3054719.

[27] Willem Waegeman, Krzysztof Dembczynski, and Eyke Huellermeier. *Multi-Target Prediction: A Unifying View on Problems and Methods*. 2018. DOI: 10.48550/ARXIV.1809.02352. URL: https://arxiv.org/abs/1809.02352.

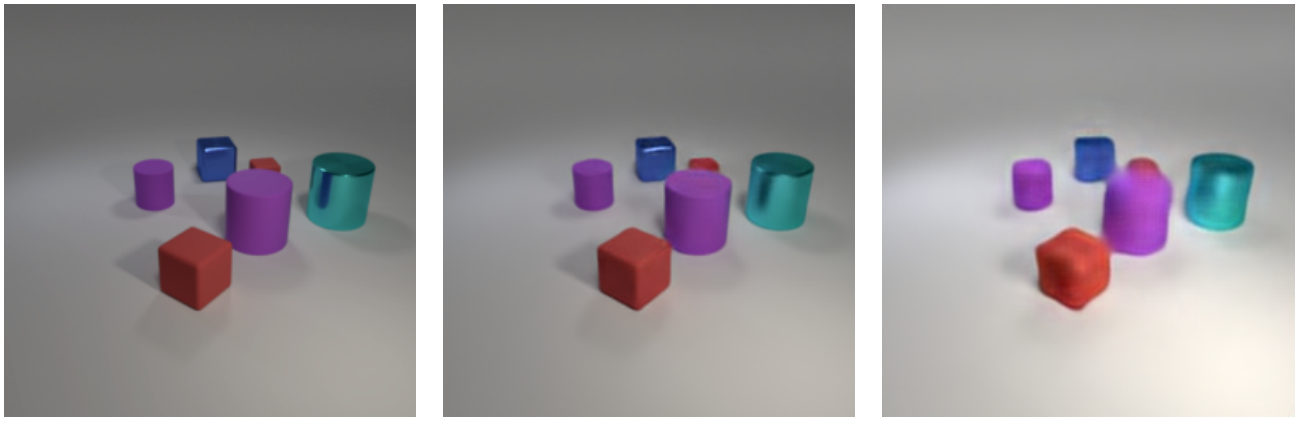
[28] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. "Multiscale structural similarity for image quality assessment". In: *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*. Vol. 2. Ieee. 2003, pp. 1398–1402.

[29] Yibo Yang, Stephan Mandt, and Lucas Theis. *An Introduction to Neural Data Compression*. 2022. DOI: 10.48550/ARXIV.2202.06533. URL: https://arxiv.org/abs/2202.06533.

[30] Amir R Zamir et al. "Taskonomy: Disentangling task transfer learning". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 3712–3722.

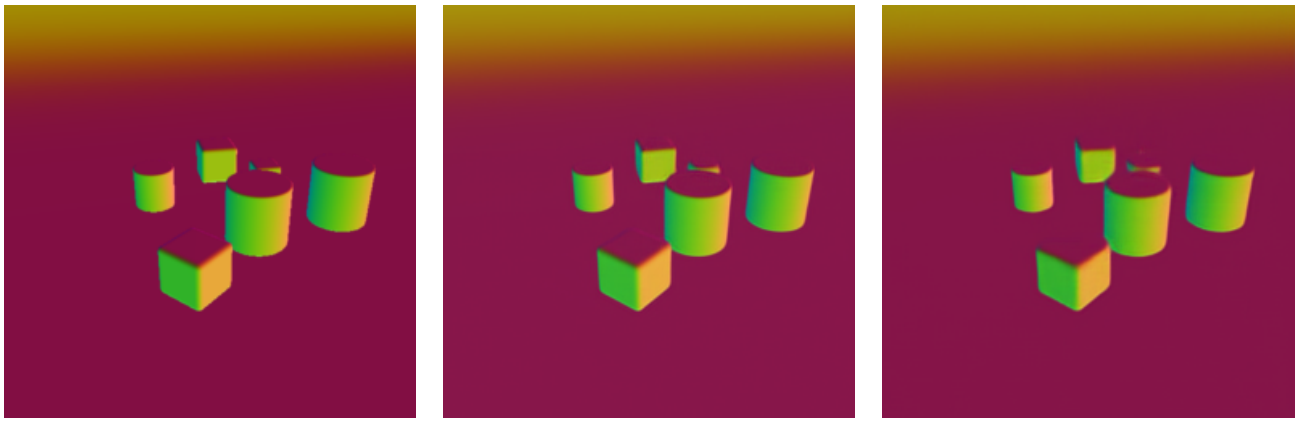
[31] Yu Zhang and Qiang Yang. "A survey on multi-task learning". In: *IEEE Transactions on Knowledge and Data Engineering* 34.12 (2021), pp. 5586–5609.

[32] Yinhao Zhu, Yang Yang, and Taco Cohen. "Transformer-based Transform Coding". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=IDwN6xjHnK8.



(a) Original (b) MSE * 255 * 255 = 3.1 | BPP = 0.2 (c) MSE * 255 * 255 = 11 | BPP = 0.04

Fig. 7: Examples of RGB reconstructions with according metric values



(a) Original (b) MSE * 255 * 255 = 1.7 | BPP = 0.2 (c) MSE * 255 * 255 = 4 | BPP = 0.04

Fig. 8: Examples of normal reconstructions with according metric values

(a) Original (b) MSE * 255 * 255 = 0.7 | BPP = 0.2 (c) MSE * 255 * 255 = 2.7 | BPP = 0.04

Fig. 9: Examples of depth reconstructions with according metric values