A General and Adaptive Robust Loss Function

Jonathan T. Barron
barron@google.com

Abstract

We present a generalization of the Cauchy/Lorentzian, Geman-McClure, Welsch/Leclerc, generalized Charbonnier, Charbonnier/pseudo-Huber/L1-L2, and L2 loss functions. By introducing robustness as a continous parameter, our loss function allows algorithms built around robust loss minimization to be generalized, which improves performance on basic vision tasks such as registration and clustering. Interpreting our loss as the negative log of a univariate density yields a general probability distribution that includes normal and Cauchy distributions as special cases. This probabilistic interpretation enables the training of neural networks in which the robustness of the loss automatically adapts itself during training, which improves performance on learning-based tasks such as generative image synthesis and unsupervised monocular depth estimation, without requiring any manual parameter tuning.

Many problems in statistics and optimization require *robustness* — that a model be less influenced by outliers than by inliers [18, 20]. This idea is common in parameter estimation and learning tasks, where a robust loss (say, absolute error) may be preferred over a non-robust loss (say, squared error) due to its reduced sensitivity to large errors. Researchers have developed various different robust penalties with particular properties, many of which are summarized well in [3, 38]. In gradient descent or M-estimation [17] these losses are often interchangeable, so researchers may experiment with different losses when designing a system. This flexibility in shaping a loss function may be useful because of non-Gaussian noise, or simply because the loss that is minimized during learning or parameter estimation is different from how the resulting learned model or estimated parameters will be evaluated. For example, one might train a neural network by minimizing the difference between the network's output and a set of images, but evaluate that network in terms of how well it hallucinates random images.

In this paper we present a single loss function that is a superset of many common robust loss functions. A single continuous-valued parameter in our general loss function can be set such that it is equal to several traditional losses, and can be adjusted to model a wider family of functions. This allows us to generalize algorithms built around a fixed robust loss with a new "robustness" hyperparameter that can be tuned or annealed to improve performance.

Though new hyperparameters may be valuable to a practitioner, they complicate experimentation by requiring manual tuning or time-consuming cross-validation. However, by viewing our general loss function as the negative log-likelihood of a probability distribution, and by treating the robustness of that distribution as a latent variable, we show that maximizing the likelihood of that distribution allows gradient-based optimization frameworks to *automatically* determine how robust the loss should be without any manual parameter tuning. This "adaptive" form of our loss is particularly effective in models with multivariate output spaces (say, image generation or depth estimation) as we can introduce independent robustness variables for each dimension in the output and thereby allow the model to independently adapt the robustness of its loss in each dimension.

The rest of the paper is as follows: In Section 1 we define our general loss function, relate it to existing losses, and enumerate some of its useful properties. In Section 2 we use our loss to construct a probability distribution, which

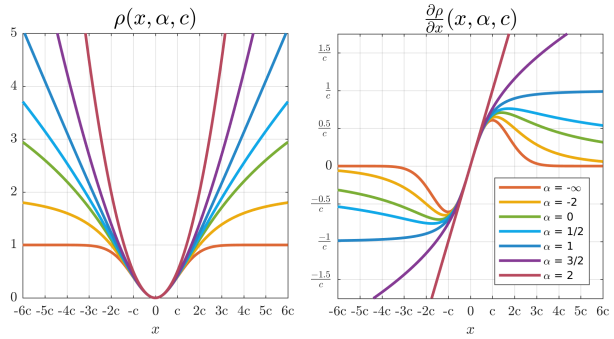

Figure 1. Our general loss function (left) and its gradient (right) for different values of its shape parameter α. Several values of α reproduce existing loss functions: L2 loss ($\alpha = 2$), Charbonnier loss ($\alpha = 1$), Cauchy loss ($\alpha = 0$), Geman-McClure loss ($\alpha = -2$), and Welsch loss ($\alpha = -\infty$). See `desmos.com/calculator/pckurs3uut` for an interactive version of this plot.

requires deriving a partition function and a sampling procedure. Section 3 discusses four representative experiments: In Sections 3.1 and 3.2 we take two deep learning models (variational autoencoders for image synthesis and self-supervised monocular depth estimation), replace their losses with the negative log-likelihood of our general distribution, and demonstrate that allowing our distribution to automatically determine its own robustness can improve performance without introducing any additional manually-tuned hyperparameters. In Sections 3.3 and 3.4 we use our loss function to generalize algorithms for the classic vision tasks of registration and clustering, and demonstrate the performance improvement that can be achieved by introducing robustness as a hyperparameter that is annealed or manually tuned.

1. Loss Function

The simplest form of our loss function is:

$$f\left(x,\alpha,c\right)=\frac{\left|2-\alpha\right|}{\alpha}\left(\left(\frac{\left(x/c\right)^2}{\left|2-\alpha\right|}+1\right)^{\left(\alpha/2\right)}-1\right) \quad (1)$$

Here $\alpha \in \mathcal{R}$ is a shape parameter that controls the robustness of the loss and $c>0$ is a scale parameter that controls the size of the loss's non-robust quadratic bowl near $x=0$.

Though our loss is undefined when $\alpha=2$, it approaches L2 loss (squared error) in the limit:

$$\lim_{\alpha\to 2} f\left(x,\alpha,c\right)=\frac{1}{2}\left(x/c\right)^2 \quad (2)$$

When $\alpha=1$ our loss is a smoothed form of L1 loss:

$$f\left(x,1,c\right)=\sqrt{\left(x/c\right)^2+1}-1 \quad (3)$$

This is often referred to as Charbonnier loss [6], pseudo-Huber loss (as it resembles the classic Huber loss [19]), or L1-L2 loss [38] (as it behaves like L2 loss near the origin and like L1 loss elsewhere).

Our loss's ability to express L2 and smoothed L1 losses is shared by the "generalized Charbonnier" loss [34], which has been used in flow and depth estimation tasks that require robustness [7, 24] and is commonly defined as:

$$\left(x^2+\epsilon^2\right)^{\alpha/2} \quad (4)$$

Our loss has significantly more expressive power than the generalized Charbonnier loss, which we can see by setting our shape parameter α to nonpositive values. Though $f\left(x,0,c\right)$ is undefined, we can take the limit of $f\left(x,\alpha,c\right)$ as α approaches zero:

$$\lim_{\alpha\to 0} f\left(x,\alpha,c\right)=\log\left(\frac{1}{2}\left(x/c\right)^2+1\right) \quad (5)$$

This yields Cauchy (aka Lorentzian) loss [2]. By setting $\alpha=-2$, our loss reproduces Geman-McClure loss [14]:

$$f\left(x,-2,c\right)=\frac{2\left(x/c\right)^2}{\left(x/c\right)^2+4} \quad (6)$$

In the limit as α approaches negative infinity, our loss becomes Welsch [21] (aka Leclerc [26]) loss:

$$\lim_{\alpha\to-\infty} f\left(x,\alpha,c\right)=1-\exp\left(-\frac{1}{2}\left(x/c\right)^2\right) \quad (7)$$

With this analysis we can present our final loss function, which is simply $f\left(\cdot\right)$ with special cases for its removable singularities at $\alpha=0$ and $\alpha=2$ and its limit at $\alpha=-\infty$.

$$\rho\left(x,\alpha,c\right)=\begin{cases}\frac{1}{2}\left(x/c\right)^2 & \text{if } \alpha=2\\ \log\left(\frac{1}{2}\left(x/c\right)^2+1\right) & \text{if } \alpha=0\\ 1-\exp\left(-\frac{1}{2}\left(x/c\right)^2\right) & \text{if } \alpha=-\infty\\ \frac{\left|2-\alpha\right|}{\alpha}\left(\left(\frac{\left(x/c\right)^2}{\left|2-\alpha\right|}+1\right)^{\left(\alpha/2\right)}-1\right) & \text{otherwise}\end{cases} \quad (8)$$

As we have shown, this loss function is a superset of the Welsch/Leclerc, Geman-McClure, Cauchy/Lorentzian, generalized Charbonnier, Charbonnier/pseudo-Huber/L1-L2, and L2 loss functions.

To enable gradient-based optimization we can derive the derivative of $\rho\left(x,\alpha,c\right)$ with respect to x:

$$\frac{\partial\rho}{\partial x}\left(x,\alpha,c\right)=\begin{cases}\frac{x}{c^2} & \text{if } \alpha=2\\ \frac{2x}{x^2+2c^2} & \text{if } \alpha=0\\ \frac{x}{c^2}\exp\left(-\frac{1}{2}\left(x/c\right)^2\right) & \text{if } \alpha=-\infty\\ \frac{x}{c^2}\left(\frac{\left(x/c\right)^2}{\left|2-\alpha\right|}+1\right)^{\left(\alpha/2-1\right)} & \text{otherwise}\end{cases} \quad (9)$$

Our loss and its derivative are visualized for different values of α in Figure 1.

The shape of the derivative gives some intuition as to how α affects behavior when our loss is being minimized by gradient descent or some related method. For all values of α the derivative is approximately linear when $|x|<c$, so the effect of a small residual is always linearly proportional to that residual's magnitude. When $\alpha=2$, the derivative's magnitude stays linearly proportional to the residual's magnitude — larger residuals have correspondingly larger effects. When $\alpha=1$ the derivative's magnitude saturates to a constant $1/c$ as $|x|$ grows larger than c, so as residuals increase their effect never decreases but never exceeds a fixed amount. When $\alpha<1$ the derivative's magnitude begins to decrease as $|x|$ grows larger than c (in the language of M-estimation [17], the derivative, aka "influence", is "redescending") so as the residual of an outlier increases, that outlier has *less* effect during gradient descent. The effect

of outliers diminishes as α becomes more negative, and as α approaches $-\infty$ outliers whose residual magnitudes are larger than $4c$ are almost completely ignored.

We can also intuit how α affects behavior by thinking in terms of averages. Because the mean minimizes squared error and the median minimizes absolute error, minimizing our loss with $\alpha = 2$ is equivalent to estimating a mean, and with $\alpha = 1$ is similar to estimating a median. Minimizing our loss when $\alpha = -\infty$ is equivalent to local mode-finding [35]. Values of α between these extents can be thought of as smoothly interpolating between these three kinds of averages during estimation.

Our loss function has several useful properties that we will take advantage of. The loss is smooth (*i.e.*, in C^∞) with respect to x, α, and $c > 0$, and is therefore suited to gradient-based optimization over its input and its hyperparameters. The loss is zero at the origin, and increases monotonically with respect to $|x|$:

$$\rho\left(0, \alpha, c\right) = 0 \qquad \frac{\partial \rho}{\partial |x|}\left(x, \alpha, c\right) \geq 0 \tag{10}$$

The loss is invariant to a simultaneous scaling of c and x:

$$\forall_{k>0} \quad \rho(kx, \alpha, kc) = \rho(x, \alpha, c) \tag{11}$$

The loss increases monotonically with respect to α:

$$\frac{\partial \rho}{\partial \alpha}\left(x, \alpha, c\right) \geq 0 \tag{12}$$

This is convenient for graduated non-convexity [4] — we can initialize α such that our loss is convex and then gradually reduce α (and therefore reduce convexity and increase robustness) during optimization, thereby enabling robust estimation that (often) avoids local minima.

We can take the limit of the loss as α approaches infinity:

$$\lim_{\alpha \to +\infty} \rho\left(x, \alpha, c\right) = \exp\left(\frac{1}{2}\left({}^{x}/_{c}\right)^2\right) - 1 \tag{13}$$

Because of Eq. 12 this is an upper bound on our loss, which is useful when constructing our probability distribution.

We can bound the magnitude of the gradient of the loss:

$$\left|\frac{\partial \rho}{\partial x}\left(x, \alpha, c\right)\right| \leq \begin{cases} \frac{1}{c}\left(\frac{\alpha-2}{\alpha-1}\right)^{\left(\frac{\alpha-1}{2}\right)} \leq \frac{1}{c} & \text{if } \alpha \leq 1 \\ \frac{x}{c} & \text{if } \alpha \leq 2 \end{cases} \tag{14}$$

This allows us to better reason about exploding gradients.

L1 loss is not expressible by our loss, but if c is much smaller than x we can approximate it with $\alpha = 1$:

$$f\left(x, 1, c\right) \approx \frac{|x|}{c} - 1 \quad \text{if } c \ll x \tag{15}$$

See the appendix for other potentially-useful properties of our loss that are not used in our experiments.

2. Probability Density Function

With our loss function we can construct a general probability distribution, such that the negative log-likelihood (NLL) of its PDF is a shifted version of our loss function:

$$p\left(x \mid \mu, \alpha, c\right) = \frac{1}{cZ\left(\alpha\right)} \exp\left(-\rho\left(x - \mu, \alpha, c\right)\right) \tag{16}$$

$$Z\left(\alpha\right) = \int_{-\infty}^{\infty} \exp\left(-\rho\left(x, \alpha, 1\right)\right) \tag{17}$$

where $p\left(x \mid \mu, \alpha, c\right)$ is only defined if $\alpha \geq 0$, as $Z\left(\alpha\right)$ is divergent when $\alpha < 0$. For some values of α the partition function is relatively straightforward:

$$\begin{aligned} Z\left(0\right) &= \pi\sqrt{2} & Z\left(1\right) &= 2eK_1(1) \\ Z\left(2\right) &= \sqrt{2\pi} & Z\left(4\right) &= e^{1/4}K_{1/4}(1/4) \end{aligned} \tag{18}$$

where $K_n(\cdot)$ is the modified Bessel function of the second kind. For any rational positive α (excluding a singularity at $\alpha = 2$) where $\alpha = {}^{n}/_{d}$ with $n, d \in \mathbb{N}$, we see that

$$\begin{aligned} Z\left(\frac{n}{d}\right) &= \frac{e^{\left|\frac{2d}{n}-1\right|}\sqrt{\left|\frac{2d}{n}-1\right|}}{(2\pi)^{(d-1)}} G^{0,0}_{p,q}\left(\begin{matrix}\mathbf{a_p}\\ \mathbf{b_q}\end{matrix} \;\middle|\; \left(\frac{1}{n} - \frac{1}{2d}\right)^{2d}\right) \\ \mathbf{b_q} &= \left\{\frac{i}{n}\middle| i = -\frac{1}{2}, ..., n - \frac{3}{2}\right\} \cup \left\{\frac{i}{2d}\middle| i = 1, ..., 2d-1\right\} \\ \mathbf{a_p} &= \left\{\frac{i}{n}\middle| i = 1, ..., n-1\right\} \end{aligned} \tag{19}$$

where $G(\cdot)$ is the Meijer G-function and $\mathbf{b_q}$ is a multiset (items may occur twice). Because the partition function is difficult to evaluate or differentiate, in our experiments we approximate $\log(Z\left(\alpha\right))$ with a cubic hermite spline (see the appendix for details).

Just as our loss function includes several common loss function as special cases, our distribution includes several common distributions as special cases. When $\alpha = 2$ our distribution becomes a normal (Gaussian) distribution, and when $\alpha = 0$ our distribution becomes a Cauchy distribution. These are also both special cases of Student's t-distribution, though they appear to be the only two points where these two families of distributions intersect. Our distribution resembles the generalized Gaussian distribution [29, 33], except that it is "smoothed" so as to approach a Gaussian distribution near the origin regardless of the shape parameter α. The PDF and NLL of our distribution for different values of α can be seen in Figure 2.

In later experiments we will use the NLL of our general distribution $-\log(p(\cdot|\alpha, c))$ as the loss for training our neural networks, not our general loss $\rho\left(\cdot, \alpha, c\right)$. Critically, using the NLL allows us to treat α as a free parameter, thereby allowing optimization to automatically determine the degree of robustness that should be imposed by the loss being

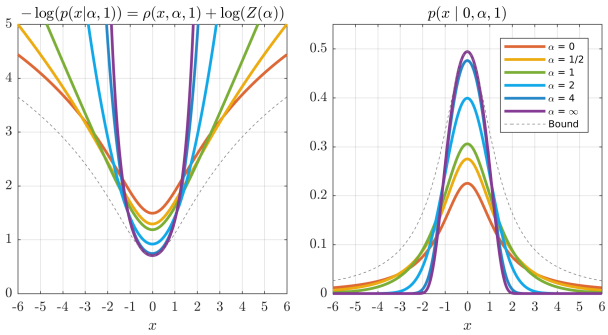

Figure 2. The negative log-likelihoods (left) and probability densities (right) of the distribution corresponding to our loss function when it is defined ($\alpha \geq 0$). NLLs are simply losses (Fig. 1) shifted by a log partition function. Densities are bounded by a scaled Cauchy distribution.

used during training. To understand why this is necessary, consider a training procedure in which we simply minimize our $\rho(\cdot, \alpha, c)$ with respect to α and our model weights. In this scenario, the monotonicity of our general loss with respect to α (Eq. 12) means that optimization can trivially minimize the cost of outliers by setting α to be as small as possible. Now consider that same training procedure in which we minimize the NLL of our distribution instead of our loss. As can be observed in Figure 2, reducing α will decrease the NLL of outliers but will *increase* the NLL of inliers. During training, optimization will have to choose between reducing α, thereby getting "discount" on large errors at the cost of paying a penalty for small errors, or increasing α, thereby incurring a higher cost for outliers but a lower cost for inliers. This tradeoff forces optimization to judiciously adapt the robustness of the NLL being minimized. As we will demonstrate later, allowing the NLL to adapt in this way can increase performance on a variety of learning tasks, in addition to obviating the need for manually tuning α as a fixed hyperparameter.

Sampling from our distribution is straightforward given the observation that $-\log(p(x \mid 0, \alpha, 1))$ is bounded from below by $\rho(x, 0, 1) + \log(Z(\alpha))$ (shifted Cauchy loss). See Figure 2 for visualizations of this bound when $\alpha = \infty$, which also bounds the NLL for all values of α. This lets us perform rejection sampling using a Cauchy as the proposal distribution. Because our distribution is a location-scale family, we sample from $p(x \mid 0, \alpha, 1)$ and then scale and shift that sample by c and μ. This sampling approach is efficient, with an acceptance rate between $\sim 45\%$ ($\alpha = \infty$) and 100% ($\alpha = 0$). Pseudocode for sampling is shown in Algorithm 1.

3. Experiments

We will now demonstrate the utility of our loss function and distribution with four experiments. None of these results are intended to represent the state-of-the-art for any particular task — our goal is to demonstrate the value of our loss and distribution as useful tools in isolation. We will show that across a wide variety of tasks, just replacing the loss function of an existing model with our general loss function can enable significant performance improvements.

Algorithm 1 Sampling from our general distribution

Input: Parameters for the distribution to sample $\{\mu, \alpha, c\}$
Output: A sample drawn from $p(x \mid \mu, \alpha, c)$.

while True:
  $x \sim \text{Cauchy}(x_0 = 0, \gamma = \sqrt{2})$
  $u \sim \text{Uniform}(0, 1)$
  if $u < \frac{p(x \mid 0,\alpha,1)}{\exp(-\rho(x,0,1)-\log(Z(\alpha)))}$:
    return $cx + \mu$

In Sections 3.1 and 3.2 we focus on learning based vision tasks in which training involves minimizing the differences between images: variational autoencoders for image synthesis and self-supervised monocular depth estimation. We will generalize and improve models for both tasks by using our general distribution (either as a conditional distribution in a generative model or by using its NLL as a loss) and allowing the distribution to *automatically* determine its own degree of robustness. Because robustness is automatic and requires no manually-tuned hyperparameters, we can even allow for the robustness of our loss to be adapted individually for each dimension of our output space — we can have a different degree of robustness at each pixel in an image, for example. As we will show, this approach is particularly effective when combined with image representations such as wavelets, in which we expect to see non-Gaussian, heavy-tailed distributions.

In Sections 3.3 and 3.4 we will build upon existing algorithms for two classic vision tasks (registration and clustering) that both work by minimizing a robust loss that is subsumed by our general loss. We will then replace each algorithm's fixed robust loss with our loss, thereby introducing a continuous tunable robustness parameter α. This generalization allows us to introduce new models in which α is manually tuned or annealed, thereby improving performance. These results demonstrate the value of our loss function when designing classic vision algorithms, by allowing model robustness to be introduced into the algorithm design space as a continuous hyperparameter.

3.1. Variational Autoencoders

Variational autoencoders [23, 30] are a landmark technique for training autoencoders as generative models, which can then be used to draw random samples that resemble training data. We will demonstrate that our general distribution can be used to improve the log-likelihood performance of VAEs for image synthesis on the CelebA dataset [27]. A

common design decision for VAEs is to model images using an independent normal distribution on a vector of RGB pixel values [23], and we use this design as our baseline model. Recent work has improved upon this model by using deep, learned, and GAN[16]-like loss functions [9, 25]. Though it's possible that our general loss or distribution can add value in these circumstances, to more precisely isolate its contribution we will explore the hypothesis that the baseline model of pixel representations and normal distributions can be improved significantly with the small change of just modeling a linear transformation of a VAE's output with our general distribution. Again, our goal is not to advance the state of the art for any particular image synthesis task, but is instead to explore the value of our distribution in a minimal and experimentally controlled setting.

In our baseline model we give each pixel's normal distribution a variable scale parameter $\sigma^{(i)}$ that will be optimized over during training, thereby allowing the VAE to adjust the scale of its distribution for each output dimension. We can straightforwardly replace this per-pixel normal distribution with a per-pixel general distribution, in which each output dimension is given a distinct shape parameter $\alpha^{(i)} \in (0, 2)$ in addition to its scale parameter $c^{(i)}$ (*i.e.*, $\sigma^{(i)}$). By letting the $\alpha^{(i)}$ parameters be free variables alongside the scale parameters, training is able to adaptively select both the scale *and robustness* of the VAE's posterior distribution over pixel values. We also compare against a baseline in which we use a Cauchy distribution as an example of a fixed robust distribution. Because Cauchy and normal distributions correspond to $\alpha = 0$ and $\alpha = 2$ in our model, our model can be seen as selecting its own degree of robustness between those two extremes.

Regarding implementation, for each output dimension i we construct unconstrained real TensorFlow variables $\{\alpha_\ell^{(i)}\}$ and $\{c_\ell^{(i)}\}$ and define

$$\alpha^{(i)} = (2 - 2\epsilon_\alpha)\,\text{sigmoid}\left(\alpha_\ell^{(i)}\right) + \epsilon_\alpha, \quad \epsilon_\alpha = 10^{-3} \tag{20}$$

$$c^{(i)} = \text{softplus}\left(c_\ell^{(i)}\right) + \epsilon_c, \quad \epsilon_c = 10^{-5} \tag{21}$$

The ϵ_c offset avoids degenerate optima where likelihood is maximized by having $c^{(i)}$ approach 0, and the ϵ_α offset avoids numerical instability near $\alpha^{(i)} = 0$ and $\alpha^{(i)} = 2$. The variables are initialized such that initially all $\alpha^{(i)} = 1$ and $c^{(i)} = 0.01$, and are optimized simultaneously with the autoencoder's weights using the same Adam [22] optimizer instance. To avoid exploding gradients we multiply the loss being minimized by ϵ_c (the minimum value that any $c^{(i)}$ can reach), thereby bounding gradient magnitudes by residual magnitudes as per Eq. 14.

Though modeling images using independent distributions on pixel intensities is a popular choice due to its simplicity, classic work in natural image statistics suggest that images are better modeled with heavy-tailed distributions on wavelet-like image decompositions [10, 28]. We therefore train additional models in which our decoded RGB per-pixel images are linearly transformed into spaces that better model natural images before computing the NLL of our distribution. For this we use the DCT [1] and the CDF 9/7 wavelet decomposition [8], both with a YUV colorspace. These representations resemble the JPEG and JPEG 2000 compression standards, respectively.

Our results can be seen in Table 1, where we report the validation set evidence lower bound (ELBO) for the cross product of our three distributions and our three image representations, and in Figure 3, where we visualize

	Normal	Cauchy	Ours
Pixels + RGB	8,616 ± 14	10,227 ± 3.4	10,229 ± 1.6
DCT + YUV	31,259 ± 7.1	31,282 ± 1.4	32,777 ± 0.6
Wavelets + YUV	30,965 ± 5.3	35,770 ± 1.5	36,301 ± 1.4

Table 1. Validation set ELBOs (higher is better) for our variational autoencoders. Models using our general distribution better maximize the likelihood of unseen data than those using normal or Cauchy distributions (both special cases of our model) for all three image representations. The best and second best performing techniques for each representation are colored orange and yellow respectively. We report the mean and standard deviation for 3 random trials.

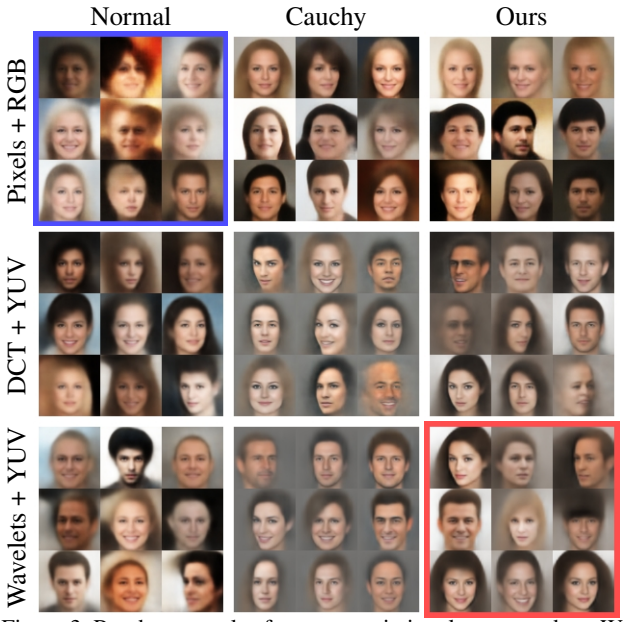

Figure 3. Random samples from our variational autoencoders. We use either normal, Cauchy, or our general distributions (columns) to model the coefficients of three different image representations (rows). Because our distribution can adaptively interpolate between Cauchy-like or normal-like behavior for each coefficient individually, using it results in sharper and higher-quality samples — particularly when using DCT or wavelet representations. The baseline technique we build upon is highlighted in blue, and our best-performing model is highlighted in red.

samples from these nine models. Table 1 shows that our general distribution maximizes ELBOs across all representations (albeit with a narrow margin with respect to Cauchy for the "Pixels + RGB" representation). For all representations, VAEs trained with our general distribution produce sharper and more detailed samples than those using normal distributions. Models trained with Cauchy distributions preserve high-frequency detail and work well on pixel representations, but systematically fail to synthesize low-frequency image content as evidenced by the sample's gray backgrounds. Comparing performance across image representations shows that the "Wavelets + YUV" representation best maximizes the validation set ELBO — though if we were to limit our model to only normal distributions the "DCT + YUV" model would appear superior, suggesting that there is value in reasoning jointly about distributions and image representations. Overall, we see that just changing the output distribution of a VAE from a normal distribution on RGB pixel values to our general distribution on YUV wavelet coefficients increasing validation ELBO by $\sim 4.2\times$. After training we see shape parameters $\alpha^{(i)}$ that span $(0, 2)$, suggesting that a mixture of normal-like and Cauchy-like distributions are useful in modeling natural images. Note that this adaptive robustness is just a consequence of allowing $\{\alpha_\ell^{(i)}\}$ to be free variables during training, and requires no manual parameter tuning. See the appendix for more samples and reconstructions from these models, and a review of our experimental procedure.

3.2. Unsupervised Monocular Depth Estimation

Due to the difficulty in acquiring ground-truth direct depth observations, there has been recent interest in "unsupervised" monocular depth estimation, in which stereo pairs and geometric constraints are used to directly train a neural network [11, 12, 15, 40] We use [40] as a representative model from this literature, which is notable for estimating depth *and* camera pose. This model is trained by minimizing the differences between two images in a stereo pair, where one image has been warped to match the other according to the depth and pose predictions of a neural network. In [40] that difference between images is defined as the absolute difference between RGB values. We will replace that loss with different varieties of our general loss, and demonstrate that using annealed or adaptive forms of our loss can improve performance.

The absolute loss in [40] is equivalent to maximizing the likelihood of a Laplacian distribution with a fixed scale on RGB pixel values. We replace that fixed Laplacian distribution with our general distribution, keeping our scale fixed but allowing the shape parameter α to vary. Following our observation from Section 3.1 that YUV wavelet representations work well when modeling images with our loss function, we impose our loss on a YUV wavelet decomposition

	lower is better					higher is better		
	Avg	AbsRel	SqRel	RMS	logRMS	<1.25	$<1.25^2$	$<1.25^3$
Baseline [40] as reported	0.407	0.221	2.226	7.527	0.294	0.676	0.885	0.954
Baseline [40] reproduced	0.398	0.208	2.773	7.085	0.286	0.726	0.895	0.953
Ours, fixed $\alpha = 1$	0.356	0.194	2.138	6.743	0.268	0.738	0.906	0.960
Ours, fixed $\alpha = 0$	0.350	0.187	2.407	6.649	0.261	0.766	0.911	0.960
Ours, fixed $\alpha = 2$	0.349	0.190	1.922	6.648	0.267	0.737	0.904	0.961
Ours, annealing $\alpha = 2 \rightarrow 0$	0.341	0.184	2.063	6.697	0.260	0.756	0.911	0.963
Ours, adaptive $\alpha \in (0, 2)$	0.332	0.181	2.144	6.454	0.254	0.766	0.916	0.965

Table 2. Results on unsupervised monocular depth estimation using the KITTI dataset [13], building upon the model from [40] ("Baseline"). By replacing the per-pixel loss used by [40] with several variants of our own per-wavelet general loss function in which our loss's shape parameters are fixed, annealed, or adaptive, we see a significant performance improvement. The top three techniques are colored red, orange, and yellow for each metric.

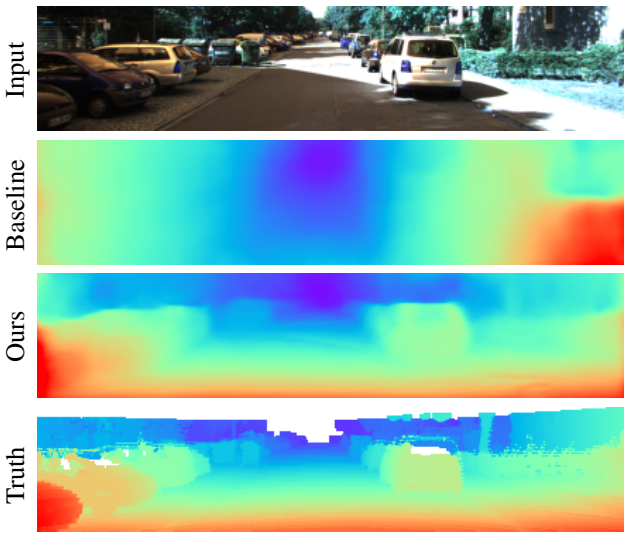

Figure 4. Monocular depth estimation results on the KITTI benchmark using the "Baseline" network of [40]. Replacing only the network's loss function with our "adaptive" loss over wavelet coefficients results in significantly improved depth estimates.

instead of the RGB pixel representation of [40]. The only changes we made to the code from [40] were to replace its loss function with our own and to remove the model components that stopped yielding any improvement after the loss function was replaced (see the appendix for details). All training and evaluation was performed on the KITTI dataset [13] using the same training/test split as [40].

Results can be seen in Table 2. We present the error and accuracy metrics used in [40] and our own "average" error measure: the geometric mean of the four errors and one minus the three accuracies. The "Baseline" models use the loss function of [40], and we present both the numbers in [40] ("as reported") and our own numbers from running the code from [40] ourselves ("reproduced"). The "Ours" entries all use our general loss imposed on wavelet coefficients, but for each entry we use a different strategy for setting the shape parameter or parameters (while keeping our loss's scale c fixed to 0.01, matching the fixed scale assump-

tion of the baseline model and roughly matching the shape of its L1 loss as per Eq. 15). For the "fixed" models we use a constant value for α for all wavelet coefficients, and observe that though performance is improved relative to the baseline, no single value of α is optimal. The $\alpha = 1$ entry is simply a smoothed version of the L1 loss used by the baseline model, suggesting that just using a wavelet representation improves performance. In the "annealing $\alpha = 2 \rightarrow 0$" model we linearly interpolate α from 2 (L2) to 0 (Cauchy) as a function of training iteration, which outperforms all "fixed" models. In the "adaptive $\alpha \in (0, 2)$" model we assign each wavelet coefficient its own shape parameter as a free variable and we allow those variables to be optimized alongside our network weights during training as was done in Section 3.1. This "adaptive" strategy outperforms the "annealing" and all "fixed" strategies, thereby demonstrating the value of allowing the model to adaptively determine the robustness of its loss during training. Note that though the "fixed" and "annealed" strategies only require our general loss, the "adaptive" strategy requires that we use the NLL of our general distribution as our loss — otherwise training would simply drive α to be as small as possible due to the monotonicity of our loss with respect to α, causing performance to degrade to the "fixed $\alpha = 0$" model. Comparing the "adaptive" model's performance to that of the "fixed" models suggests that, as in Section 3.1, no single setting of α is optimal for all wavelet coefficients. Overall, we see that just replacing the loss function of [40] with our adaptive loss on wavelet coefficients reduces average error by $\sim 17\%$.

In Figure 4 we compare our "adaptive" model's output to the baseline model and the ground-truth depth, and demonstrate a substantial qualitative improvement. See the appendix for many more results, and for visualizations of the per-coefficient robustness selected by our model.

3.3. Fast Global Registration

The Fast Global Registration (FGR) algorithm of [39] finds the rigid transformation $\mathbf{T}$ that aligns point sets $\mathbf{P}$ and $\mathbf{Q}$ by minimizing the following loss:

$$\sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \rho_{gm}\left(\|\mathbf{p} - \mathbf{T}\mathbf{q}\|, c\right) \tag{22}$$

where $\rho_{gm}(\cdot)$ is Geman-McClure loss. By using the Black and Rangarajan duality between robust estimation and line processes [3] FGR is capable of producing high-quality registrations at high speeds. Because Geman-McClure loss is a special case of our loss, and because we can formulate our loss as an outlier process (see appendix), we can generalize FGR to an arbitrary shape parameter α by replacing $\rho_{gm}(\cdot, c)$ with our $\rho(\cdot, \alpha, c)$ (where setting $\alpha = -2$ reproduces FGR).

	Mean RMSE $\times 100$			Max RMSE $\times 100$		
$\sigma =$	0	0.0025	0.005	0	0.0025	0.005
FGR [39]	0.373	0.518	0.821	0.591	1.040	1.767
shape-annealed gFGR	0.374	0.510	**0.802**	0.590	0.997	1.670
gFGR*	**0.370**	**0.509**	0.806	**0.545**	**0.961**	**1.669**

Table 3. Results on the registration task of [39], in which we compare their "FGR" algorithm to two version of our "gFGR" generalization.

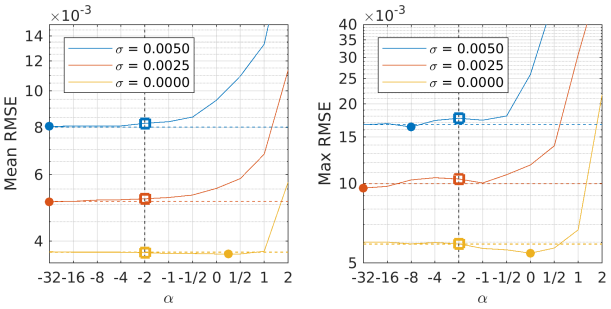

Figure 5. Performance (lower is better) of our gFGR algorithm on the task of [39] as we vary our shape parameter α, with the lowest-error point indicated by a circle. FGR (which is equivalent to gFGR with $\alpha = -2$) is shown as a black dashed line and a square, and shape-annealed gFGR for each noise level is shown as a dotted line.

This generalized FGR (gFGR) enables algorithmic improvements. FGR iteratively solves a linear system while annealing its scale parameter c, which has the effect of gradually introducing nonconvexity. gFGR enables an alternative strategy in which we directly manipulate convexity by annealing α instead of c. This "shape-annealing gFGR" follows the same procedure as [39]: 64 iterations in which a parameter is annealed every 4 iterations. Instead of annealing c, we set it to its terminal value and instead anneal α over the following values:

$$2, 1, 1/2, 1/4, 0, -1/4, -1/2, -1, -2, -4, -8, -16, -32$$

Table 3 shows results for the 3D point cloud registration task of [39] (Table 1 in that paper), which shows that annealing shape produces moderately improved performance over FGR for high-noise inputs, and behaves equivalently in low-noise inputs. This suggests that performing graduated non-convexity by directly adjusting a shape parameter that controls non-convexity — a procedure that is enabled by our general loss – is preferable to indirectly controlling non-convexity by annealing a scale parameter.

Another generalization is to continue using the c-annealing strategy of [39], but treat α as a hyperparameter and tune it independently for each noise level in this task. In Figure 5 we set α to a wide range of values and report errors for each setting, using the same evaluation of [39]. We see that for high-noise inputs, more negative values of α are preferable, but for low-noise inputs values closer to 0 are optimal. We report the lowest-error entry for each

Dataset	AC-W	N-Cuts [32]	LDMGI [36]	PIC [37]	RCC-DR [31]	RCC [31]	gRCC*	Rel. Impr.
YaleB	0.767	0.928	0.945	0.941	0.974	0.975	**0.975**	0.4%
COIL-100	0.853	0.871	0.888	**0.965**	0.957	0.957	0.962	11.6%
MNIST	0.679	-	0.761	-	0.828	0.893	**0.901**	7.9%
YTF	0.801	0.752	0.518	0.676	0.874	0.836	**0.888**	31.9%
Pendigits	0.728	0.813	0.775	0.467	0.854	0.848	**0.871**	15.1%
Mice Protein	0.525	0.536	0.527	0.394	0.638	0.649	**0.650**	0.2%
Reuters	0.471	0.545	0.523	0.057	0.553	0.556	**0.561**	1.1%
Shuttle	0.291	0.000	**0.591**	-	0.513	0.488	0.493	0.9%
RCV1	0.364	0.140	0.382	0.015	**0.442**	0.138	0.338	23.2%

Table 4. Results on the clustering task of [31] where we compare their "RCC" algorithm to our "gRCC*" generalization in terms of AMI on several datasets. We also report the AMI increase of "gRCC*" with respect to "RCC". Baseline numbers are taken from [31].

noise level as "gFGR*" in Table 3 where we see a significant reduction in error, thereby demonstrating the improvement can be achieved from treating robustness as a hyperparameter.

3.4. Robust Continuous Clustering

In [31] robust losses are used for unsupervised clustering, by minimizing:

$$\sum_i \|\mathbf{x}_i - \mathbf{u}_i\|_2^2 + \lambda \sum_{(p,q)\in\mathcal{E}} w_{p,q}\rho\left(\|\mathbf{u}_p - \mathbf{u}_q\|_2\right) \quad (23)$$

where $\{\mathbf{x}_i\}$ is a set of input datapoints, $\{\mathbf{u}_i\}$ is a set of "representatives" (cluster centers), and $\mathcal{E}$ is a mutual k-nearest neighbors (m-kNN) graph. In [31], $\rho(\cdot)$ is Geman-McClure loss, which means that our loss can be used to generalize this algorithm. Using the RCC code provided by the authors (and keeping all hyperparameters fixed to their default values) we replace Geman-McClure loss with our general loss and then sweep over values of α, as was done in Section 3.3. In Figure 6 we show the adjusted mutual information (AMI, the metric used by [31]) of the resulting clustering for each value of α on the datasets used in [31], and in Table 4 we report the AMI for the best-performing value of α for each dataset as "gRCC*". On some datasets performance is insensitive to α, but on others adjusting α improves performance by as much as 32%. This improvement demonstrates the gains that can be achieved by introducing robustness as a hyperparameter and tuning it accordingly.

4. Conclusion

We have presented a two-parameter loss function that generalizes many existing one-parameter robust loss functions: the Cauchy/Lorentzian, Geman-McClure, Welsch/Leclerc, generalized Charbonnier, Charbonnier/pseudo-Huber/L1-L2, and L2 loss functions. By reducing a family of discrete single-parameter losses into a single function with two continuous parameters,

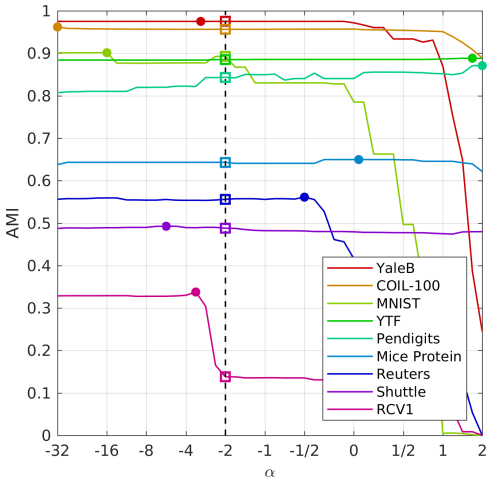

Figure 6. Performance (higher is better) of our gRCC algorithm on the clustering task of [31], for different values of our shape parameter α, with the highest-accuracy point indicated by a dot. Because the baseline RCC algorithm is equivalent to gRCC with $\alpha = -2$, we highlight that α value with a black dashed line and a square.

our loss function enables the convenient exploration and comparison of different robust penalties. This allows us to generalize and improve algorithms designed around the minimization of some fixed robust loss function, which we have demonstrated for registration and clustering. When used as a negative log-likelihood, this loss gives a general probability distribution that includes normal and Cauchy distributions as special cases. This distribution lets us train of neural networks whose loss has an adaptive degree of robustness for each output dimension, and we see that this allows training to automatically determine how much robustness should be imposed by the loss function without any manual parameter tuning. When this adaptive loss is paired with image representations in which heavy-tailed behavior occurs, such as wavelets, this adaptive training approach allows us to improve the performance of variational autoencoders for image synthesis and of neural networks for unsupervised monocular depth estimation.

References

[1] Nasir Ahmed, T_ Natarajan, and Kamisetty R Rao. Discrete cosine transform. *IEEE Transactions on Computers*, 1974.

[2] Michael J Black and Paul Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *CVIU*, 1996.

[3] Michael J. Black and Anand Rangarajan. On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *IJCV*, 1996.

[4] Andrew Blake and Andrew Zisserman. *Visual Reconstruction*. MIT Press, 1987.

[5] Richard H. Byrd and David A. Pyne. Convergence of the iteratively reweighted least-squares algorithm for robust re-

gression. Technical report, Dept. of Mathematical Sciences, The Johns Hopkins University, 1979.

[6] Pierre Charbonnier, Laure Blanc-Feraud, Gilles Aubert, and Michel Barlaud. Two deterministic half-quadratic regularization algorithms for computed imaging. *ICIP*, 1994.

[7] Qifeng Chen and Vladlen Koltun. Fast mrf optimization with application to depth reconstruction. *CVPR*, 2014.

[8] Albert Cohen, Ingrid Daubechies, and J-C Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on pure and applied mathematics*, 1992.

[9] Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. *NIPS*, 2016.

[10] David J. Field. Relations between the statistics of natural images and the response properties of cortical cells. *JOSA A*, 1987.

[11] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world's imagery. *CVPR*, 2016.

[12] Ravi Garg, BG Vijay Kumar, Gustavo Carneiro, and Ian Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue. *ECCV*, 2016.

[13] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. *CVPR*, 2012.

[14] Stuart Geman and Donald E. McClure. Bayesian image analysis: An application to single photon emission tomography. *Proceedings of the American Statistical Association*, 1985.

[15] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency. *CVPR*, 2017.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *NIPS*, 2014.

[17] Frank R. Hampel, Elvezio M. Ronchetti, Peter J. Rousseeuw, and Werner A. Stahel. *Robust Statistics: The Approach Based on Influence Functions*. Wiley, 1986.

[18] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman and Hall/CRC, 2015.

[19] Peter J. Huber. Robust estimation of a location parameter. *Annals of Mathematical Statistics*, 1964.

[20] Peter J. Huber. *Robust Statistics*. Wiley, 1981.

[21] John E. Dennis Jr. and Roy E. Welsch. Techniques for nonlinear least squares and robust regression. *Communications in Statistics-simulation and Computation*, 1978.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[23] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *ICLR*, 2014.

[24] Philipp Krähenbühl and Vladlen Koltun. Efficient nonlocal regularization for optical flow. *ECCV*, 2012.

[25] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *ICML*, 2016.

[26] Yvan G Leclerc. Constructing simple stable descriptions for image partitioning. *IJCV*, 1989.

[27] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. *ICCV*, 2015.

[28] Stéphane Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *TPAMI*, 1989.

[29] Saralees Nadarajah. A generalized normal distribution. *Journal of Applied Statistics*, 2005.

[30] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *ICML*, 2014.

[31] Sohil Atul Shah and Vladlen Koltun. Robust continuous clustering. *PNAS*, 2017.

[32] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *TPAMI*, 2000.

[33] M Th Subbotin. On the law of frequency of error. *Matematicheskii Sbornik*, 1923.

[34] Deqing Sun, Stefan Roth, and Michael J. Black. Secrets of optical flow estimation and their principles. *CVPR*, 2010.

[35] Rein van den Boomgaard and Joost van de Weijer. On the equivalence of local-mode finding, robust estimation and mean-shift analysis as used in early vision tasks. *ICPR*, 2002.

[36] Yi Yang, Dong Xu, Feiping Nie, Shuicheng Yan, and Yueting Zhuang. Image clustering using local discriminant models and global integration. *TIP*, 2010.

[37] Wei Zhang, Deli Zhao, and Xiaogang Wang. Agglomerative clustering via maximum incremental path integral. *Pattern Recognition*, 2013.

[38] Zhengyou Zhang. Parameter estimation techniques: A tutorial with application to conic fitting, 1995.

[39] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Fast global registration. *ECCV*, 2016.

[40] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised learning of depth and ego-motion from video. *CVPR*, 2017.

A. Alternative Forms

Using the equivalence between robust loss minimization and outlier processes established by Black and Rangarajan [3], we can derive our loss's Ψ-function:

$$\Psi(z,\alpha) = \begin{cases} -\log(z) + z - 1 & \text{if } \alpha = 0 \\ z\log(z) - z + 1 & \text{if } \alpha = -\infty \\ \frac{|2-\alpha|}{\alpha}\left(\left(1-\frac{\alpha}{2}\right) z^{\frac{\alpha}{(\alpha-2)}} + \frac{\alpha z}{2} - 1\right) & \text{if } \alpha < 2 \end{cases}$$

$$\rho(x,\alpha,c) = \min_{0\le z\le 1}\left(\frac{1}{2}\left({}^{x}/_{c}\right)^2 z + \Psi(z,\alpha)\right) \tag{24}$$

$\Psi(z,\alpha)$ is not defined when $\alpha \ge 2$ because for those values the loss is no longer robust, and so is not well described as a process that rejects outliers. This outlier process is used in our registration and clustering experiments.

We can also derive our loss's weight function to be used during iteratively reweighted least squares [5, 17]:

$$\frac{1}{x}\frac{\partial \rho}{\partial x}(x,\alpha,c) = \begin{cases} \frac{1}{c^2} & \text{if } \alpha = 2 \\ \frac{2}{x^2+2c^2} & \text{if } \alpha = 0 \\ \frac{1}{c^2}\exp\left(-\frac{1}{2}\left({}^{x}/_{c}\right)^2\right) & \text{if } \alpha = -\infty \\ \frac{1}{c^2}\left(\frac{\left({}^{x}/_{c}\right)^2}{|2-\alpha|} + 1\right)^{\left({}^{\alpha}/_{2}-1\right)} & \text{otherwise} \end{cases} \tag{25}$$

Curiously, these IRLS weights resemble a non-normalized form of Student's t-distribution. These weights are not used in any of our experiments, but they are an intuitive way to demonstrate how reducing α attenuates the effect of outliers. A visualization of our loss's Ψ-functions and weight functions for different values of α can be seen in Figure 7.

B. Practical Implementation

The special cases in the definition of $\rho(\cdot,\alpha,c)$ that are required because of the removable singularities of $f(\cdot,\alpha,c)$ at $\alpha = 0$ and $\alpha = 2$ can make implementing our loss somewhat inconvenient. Additionally, $f(\cdot,\alpha,c)$ is numerically unstable near these singularities, due to divisions by small values. Additionally, many deep learning frameworks handle special cases inefficiently by evaluating all cases despite only one case being returned. To circumvent these issues we can slightly modify our loss (and its gradient and Ψ-function) to guard against singularities and make implementation easier:

$$\rho(x,\alpha,c) = \frac{b}{d}\left(\left(\frac{\left({}^{x}/_{c}\right)^2}{b} + 1\right)^{\left({}^{d}/_{2}\right)} - 1\right)$$

$$\frac{\partial \rho}{\partial x}(x,\alpha,c) = \frac{x}{c^2}\left(\frac{\left({}^{x}/_{c}\right)^2}{b} + 1\right)^{\left({}^{d}/_{2}-1\right)}$$

$$\Psi(z,\alpha) = \begin{cases} \frac{b}{d}\left(\left(1-\frac{d}{2}\right) z^{\frac{d}{(d-2)}} + \frac{dz}{2} - 1\right) & \text{if } \alpha < 2 \\ 0 & \text{if } \alpha = 2 \end{cases}$$

$$b = |2-\alpha| + \epsilon \qquad d = \begin{cases} \alpha + \epsilon & \text{if } \alpha \ge 0 \\ \alpha - \epsilon & \text{if } \alpha < 0 \end{cases}$$

Where ϵ is some small value (we use 10^{-5}).

As a reference, here is TensorFlow code implementing this pratical form of our loss function:

```
def loss(x, a, c, e=1e-5):
  b = tf.abs(2.-a) + e
  d = tf.where(tf.greater_equal(a, 0.), a+e, a-e)
  return b/d*(tf.pow(tf.square(x/c)/b+1., 0.5*d)-1.)
```

And here is TensorFlow code for computing the negative log-likelihood of our general distribution, for $\alpha \in [0, 2]$:

```
def logZ1(a):
  ps = [1.49130350, 1.38998350, 1.32393250,
        1.26937670, 1.21922380, 1.16928990,
        1.11524570, 1.04887590, 0.91893853]
  ms = [-1.4264522e-01, -7.7125795e-02, -5.9283373e-02,
        -5.2147767e-02, -5.0225594e-02, -5.2624177e-02,
        -6.1122095e-02, -8.4174540e-02, -2.5111488e-01]
  x = 8. * (tf.log(alpha + 2.) / tf.log(2.) - 1.)
  i0 = tf.cast(tf.clip_by_value(
      x, 0., tf.cast(len(ps) - 2, tf.float32)), tf.int32)
  p0 = tf.gather(ps, i0)
  p1 = tf.gather(ps, i0 + 1)
  m0 = tf.gather(ms, i0)
  m1 = tf.gather(ms, i0 + 1)
  t = x - tf.cast(i0, tf.float32)
  h01 = t * t * (-2. * t + 3.)
  h00 = 1. - h01
  h11 = t * t * (t - 1.)
  h10 = h11 + t * (1. - t)
  return tf.where(t < 0., ms[0] * t + ps[0],
          tf.where(t > 1., ms[-1] * (t - 1.) + ps[-1],
            p0 * h00 + p1 * h01 + m0 * h10 + m1 * h11))

def nll(x, a, c, e=1e-5):
  return loss(x, a, c, e) + tf.log(c) + logZ1(a)
```

Much of this code is spent approximating the log partition function $\log(Z(\alpha))$ with cubic hermite spline interpolation. This is necessary because $Z(\alpha)$ is difficult to evaluate efficiently for any real number, and especially difficult to differentiate with respect to α. We therefore approximate $\log(Z(\alpha))$ with a cubic hermite spline: We transform α with a nonlinearity $\log_2(\alpha+2)-1$, and approximate $\log(Z(\alpha))$ in that transformed space using a spline with knots on $[0, 24]$ with spline knots spaced apart by $1/8$.

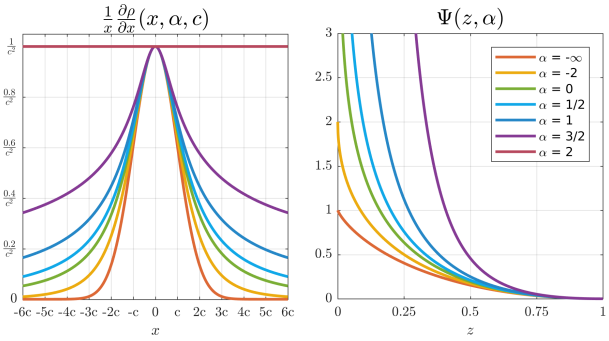

Figure 7. Our general loss's IRLS weight function (left) and Ψ-function (right) for different values of the shape parameter α.

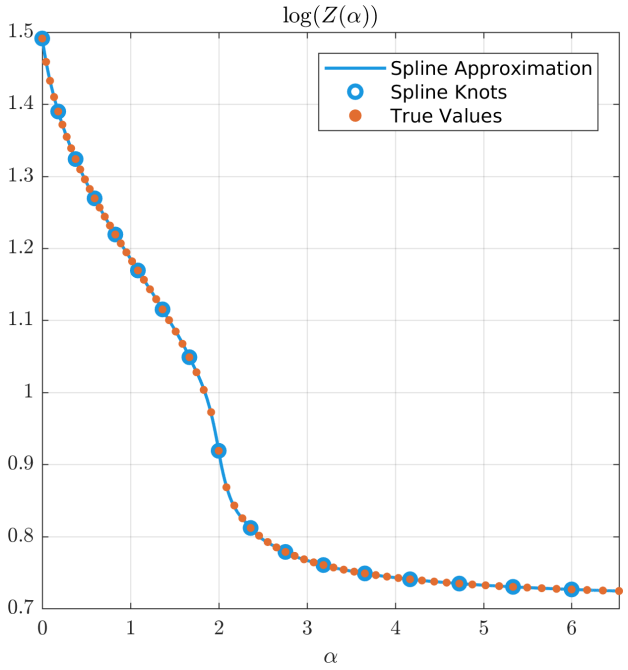

Figure 8. Because our distribution's log partition function $\log(Z(\alpha))$ is difficult to evaluate for arbitrary inputs, we approximate it using cubic hermite spline interpolation.

This nonlinearity was chosen so as to let us model all non-negative values using a fixed length spline, and such that $\alpha = 0$ and $\alpha = 2$ lie exactly at spline knots thereby yielding near-exact approximations for those important special cases. The tangents for each knot of the spline are set to minimize the squared error between the spline and the true log partition function. For all values of α this approximation has an RMSE of less than 0.0002, and a maximum error of less than 0.002, which appears to be sufficient for our purposes. The spline knot values and tangents for $\alpha \in [0, 2]$ can be seen in the `ps` and `ms` variables respectively in the above reference code, and our spline approximation relative to the true partition function for small values of α can be seen in Figure 8.

C. Motivation and Derivation

Our loss function is derived from the "generalized Charbonnier" loss [34], which itself builds upon the Charbonnier loss function [6]. To better motivate the construction of our loss function, and to clarify its relationship to prior work, here we work through how our loss function was constructed.

Generalized Charbonnier loss can be defined as:

$$d\left(x, \alpha, c\right) = \left(x^2 + c^2\right)^{\alpha/2} \tag{26}$$

Here we use a slightly different parametrization from [34] and use $\alpha/2$ as the exponent instead of just α. This makes the generalized Charbonnier somewhat easier to reason about with respect to standard loss functions: $d\left(x, 2, c\right)$ resembles L2 loss, $d\left(x, 1, c\right)$ resembles L1 loss, etc.

We can reparametrize generalized Charbonnier loss as:

$$d\left(x, \alpha, c\right) = c^\alpha \left(\left(x/c\right)^2 + 1\right)^{\alpha/2} \tag{27}$$

We omit the c^α multiplier to this loss, which gives us:

$$g\left(x, \alpha, c\right) = \left(\left(x/c\right)^2 + 1\right)^{\alpha/2} \tag{28}$$

Omitting that scale factor gives us a loss that is scale invariant with respect to c:

$$\forall_{k>0} \quad g(kx, \alpha, kc) = g(x, \alpha, c) \tag{29}$$

This lets us view the c "padding" variable as a "scale" parameter, similar to other common robust loss functions. Additionally, only after dropping this scale factor does setting α to a negative value yield a family of meaningful robust loss functions, such as Geman-McClure loss.

But this loss function still has several unintuitive properties: the loss is non-zero when $x = 0$ (assuming a non-zero values of c), and the curvature of the quadratic "bowl" near $x = 0$ varies as a function of c and α. We therefore construct a shifted and scaled version of Equation 28 that does not have these properties:

$$\frac{g\left(x, \alpha, c\right) - g\left(0, \alpha, c\right)}{c^2 g''\left(0, \alpha, c\right)} = \frac{1}{\alpha}\left(\left(\left(x/c\right)^2 + 1\right)^{\alpha/2} - 1\right) \tag{30}$$

This loss generalizes L2, Cauchy, and Geman-McClure loss, but it has the unfortunate side-effect of flattening out to 0 when $\alpha \ll 0$, thereby prohibiting many annealing strategies. This can be addressed by modifying the $1/\alpha$ scaling to approach 1 instead of 0 when $\alpha \ll 0$ by introducing another scaling that cancels out the division by α. To preserve the scale-invariance of Equation 29, this scaling also needs to be applied to the $\left(x/c\right)^2$ term in the loss. This scaling also needs to maintain the monotonicity of our loss with respect to α so as to make annealing possible. There are several scalings that satisfy this property, so we select one that is efficient to evaluation and which keeps our loss function smooth (ie, having derivatives of all orders everywhere) with respect to x, α, and c, which is $|2 - \alpha|$. This gives us our final loss function:

$$f\left(x, \alpha, c\right) = \frac{|2-\alpha|}{\alpha}\left(\left(\frac{\left(x/c\right)^2}{|2-\alpha|} + 1\right)^{(\alpha/2)} - 1\right) \tag{31}$$

This choice of $|2 - \alpha|$ as a scaling satisfies our criteria, though it does introduce a removable singularity into our loss function at $\alpha = 2$ and reduces numerical stability near $\alpha = 2$.

D. Additional Properties

Here we enumerate additional properties of our loss function that were not used in our experiments, in the hopes that they may be of use to the community.

At the origin the IRLS weight of our loss is $\frac{1}{c^2}$:

$$\rho\left(0, \alpha, c\right) = 0, \qquad \frac{1}{x}\frac{\partial \rho}{\partial x}\left(0, \alpha, c\right) = \frac{1}{c^2} \tag{32}$$

The roots of the second derivative of $\rho\left(x, \alpha, c\right)$ are:

$$x = \pm c\sqrt{\frac{\alpha - 2}{\alpha - 1}} \tag{33}$$

This tells us at what value of x the loss begins to redescend. This point has a magnitude of c when $\alpha = -\infty$, and that magnitude increases as α increases. The root is undefined when $\alpha \geq 1$, as our loss is redescending iff $\alpha < 1$. Our loss is strictly convex iff $\alpha \geq 1$, non-convex iff $\alpha < 1$, and pseudoconvex for all values of α.

Our loss's Ψ-function increases monotonically with respect to α when $\alpha < 2$ for all values of z in $[0, 1]$:

$$\frac{\partial \Psi}{\partial \alpha}\left(z, \alpha\right) \geq 0 \quad \text{if } 0 \leq z \leq 1 \tag{34}$$

For all values of α, when $|x|$ is small with respect to c the loss is well-approximated by a quadratic bowl:

$$\rho\left(x, \alpha, c\right) \approx \frac{1}{2}\left({}^{x}/_{c}\right)^2 \quad \text{if } |x| < c \tag{35}$$

Because the second derivative of the loss is maximized at $x = 0$, this quadratic approximation tells us that the second derivative is bounded from above:

$$\frac{\partial^2 \rho}{\partial x^2}(x, \alpha, c) \leq \frac{1}{c^2} \tag{36}$$

This can be used to derive approximate Jacobi preconditioners for optimization problems that minimize this loss.

When α is negative the loss approaches a constant as $|x|$ approaches infinity, letting us bound the loss:

$$\forall_{x,c}\ \rho\left(x, \alpha, c\right) \leq \frac{\alpha - 2}{\alpha} \quad \text{if } \alpha < 0 \tag{37}$$

E. Wavelet Implementation

Two of our experiments impose our loss on images reparametrized with the Cohen-Daubechies-Feauveau (CDF) 9/7 wavelet decomposition [8]. The analysis filters used for these experiments are:

lowpass	highpass
0.852698679009	0.788485616406
0.377402855613	-0.418092273222
-0.110624404418	-0.040689417609
-0.023849465020	0.064538882629
0.037828455507	

Here the origin coefficient of the filter is listed first, and the rest of the filter is symmetric. The synthesis filters are defined as usual, by reversing the sign of alternating wavelet coefficients in the analysis filters. The lowpass filter sums to $\sqrt{2}$, which means that image intensities are doubled at each scale of the wavelet decomposition, and that the magnitude of an image is preserved in its wavelet decomposition. Boundary conditions are "reflecting", or half-sample symmetric.

F. Variational Autoencoders

Our VAE experiments were done using the code included in the TensorFlow Probability codebase at `http://github.com/tensorflow/probability/blob/master/tensorflow_probability/examples/vae.py`. This code was designed for binarized MNIST data, so adapting it to real-valued color images in CelebA [27] required the following changes:

- We changed the input and output image resolution from $(28, 28, 1)$ to $(64, 64, 3)$.
- We increased the number of training steps from 5000 to 50000, as CelebA is significantly larger than MNIST.
- We changed the CNN architecture from a 5-layer network with 5-tap and 7-tap filters with interleaved strides of 1 and 2 (which maps from a 28×28 image to a vector) to a 6-layer network consisting of all 5-tap filters with strides of 2 (which maps from a 64×64 input to a vector). The number of hidden units was unchanged, and the one extra layer we added at the end of our decoder (and beginning of our decoder) was given the same number of hidden units as the layer before it.
- In our "DCT + YUV" and "Wavelets + YUV" models, before imposing our model's posterior we apply an RGB-to-YUV transformation and then a per-channel DCT or wavelet transformation to the YUV images, and then invert these transformations to visualize each sampled image. In the "Pixels + RGB" model this transformation and its inverse are the identity function.
- As discussed in the paper, for each output coefficient (pixel value, DCT coefficient, or wavelet coefficient) we added a scale variable (σ when using normal distributions, c when using our general distributions) and a shape variable α (when using our general distribution).

We made as few changes to the reference code as possible so as to keep our model architecture as simple as possible, as our goal is not to produce state-of-the-art image synthesis results for some task, but is instead to simply demonstrate the value of our general distribution in isolation.

CelebA [27] images are processed by extracting a square 160×160 image region at the center of each 178×218 image and downsampling it to 64×64 by a factor of $2.5\times$ using TensorFlow's bilinear interpolation implementation. Pixel intensities are scaled to $[0, 1]$.

In the main paper we demonstrated that using our general distribution to independently model the robustness of each coefficient of our image representation worked better than assuming a Cauchy ($\alpha = 0$) or normal distribution ($\alpha = 2$) for all coefficients (as those two distributions represent the two extremes of our general distribution). To further demonstrate the value of *independently* modeling the robustness of each individual coefficient, we ran a more thorough experiment in which we densely sampled values for α in $[0, 2]$ that are used for *all* coefficients. In Figure 9 we visualize the validation set ELBO for each fixed value of α compared to an independently-adapted model. As we can see, though quality can be improved by selecting a value for α in between 0 and 2, no single global setting of the shape parameter matches the performance achieved by allowing each coefficient's shape parameter to automatically adapt itself to the training data.

Comparing likelihoods across our different image representations requires that the "Wavelets + YUV" and "DCT + YUV" representations be normalized to match the "Pixels + RGB" representation. We therefore construct the linear transformations used for the "Wavelets + YUV" and "DCT + YUV" spaces to have determinants of 1 as per the change of variable formula (that is, both transformations are in the "special linear group"). Our wavelet construction in Section E satisfies this criteria, and we use the orthonormal version of the DCT which also satisfies this criteria. However, the standard RGB to YUV conversion matrix does not have a determinant of 1, so we scale it by the inverse of the cube root of the standard conversion matrix, thereby forcing its determinant to be 1. The resulting matrix is:

$$\begin{bmatrix} 0.47249 & 0.92759 & 0.18015 \\ -0.23252 & -0.45648 & 0.68900 \\ 0.97180 & -0.81376 & -0.15804 \end{bmatrix}$$

Naturally, its inverse maps from YUV to RGB.

Because our model can adapt the shape and scale parameters of our general distribution to each output coefficient, after training we can inspect the shapes and scales that have emerged during training, and from them gain insight into how optimization has modeled our training data. In Figures 10 and 11 we visualize the shape and scale parameters for our "Pixels + RGB" and "Wavelets + YUV" VAEs respectively. Our "Pixels" model is easy to visualize as each output coefficient simply corresponds to a pixel in a channel, and our "Wavelets" model can be visualized by flattening each wavelet scale and orientation into an image (our DCT-based model is difficult to visualize in any intuitive way). In both models we observe that training has determined that these face images should be modeled using normal-like distributions near the eyes and mouth, presumably because these structures are consistent and repeatable on human faces, and Cauchy-like distributions on the background and in flat regions of skin, likely because our autoencoder lacks the capacity to model all of the necessary variation of skin and backgrounds of portrait photos. Though our "Pixels + RGB" model has estimated similar distributions for each color channel, our "Wavelets + YUV" model has estimated very different behavior for luma and chroma: more Cauchy-like behavior is expected in luma variation, especially at fine frequencies, while chroma variation is modeled as being closer to a normal distribution across all scales.

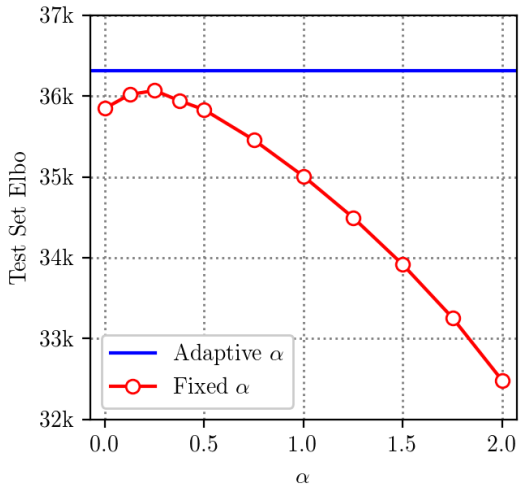

Figure 9. Here we compare the validation set ELBO of our adaptive "Wavelets + YUV" VAE model with the ELBO achieved when setting all wavelet coefficients to have the same fixed shape parameter α. We see that allowing our distribution to individually adapt its shape parameter to each coefficient outperforms any single fixed shape parameter.

See Figure 14 for additional samples from our models, and see Figure 15 for reconstructions from our models on validation-set images. As is common practice, the samples and reconstructions in those figures and in the paper are the means of the output distributions of the decoder, not samples from those distributions. That is, when sampling we draw samples from the latent encoded space and then decode them, but we do not draw samples in our output space. Samples drawn from these output distributions tend to look noisy and irregular across all distributions and image representations, but they provide a good intuition of how our general distribution behaves in each image representation, so in Figure 12 we present side-by-side visualizations of decoded means and samples.

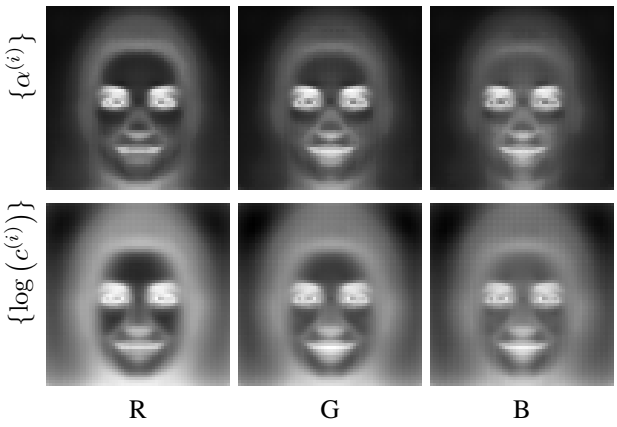

Figure 10. The final shape and scale parameters $\{\alpha^{(i)}\}$ and $\{c^{(i)}\}$ for our "Pixels + RGB" VAE after training has converged. We visualize α with black=0 and white=2 and $\log(c)$ with black=$\log(0.002)$ and white=$\log(0.02)$.

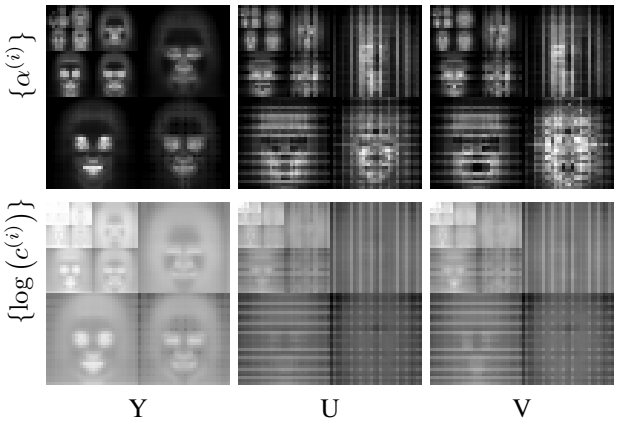

Figure 11. The final shape and scale parameters $\{\alpha^{(i)}\}$ and $\{c^{(i)}\}$ for our "Wavelets + YUV" VAE after training has converged. We visualize α with black=0 and white=2 and $\log(c)$ with black=$\log(0.00002)$ and white=$\log(0.2)$.

G. Unsupervised Monocular Depth Estimation

Our experiments use the code from `https://github.com/tinghuiz/SfMLearner`, which appears to correspond to the "Ours (w/o explainability)" model from Table 1 of [40]. The only changes we made to this code were: replacing its loss function with our own, reducing the number of training iterations from 200000 to 100000 (training converges faster when using our loss function) and disabling the smoothness term and multi-scale side predictions used by [40], as neither yielded much benefit when combined with our new loss function and they complicated experimentation by introducing hyperparameters. Because the reconstruction loss in [40] is the sum of the means of the losses imposed at each scale in a D-level pyramid of side predictions, we use a D level normalized wavelet decomposition (wherein images in $[0, 1]$ result in wavelet coefficients

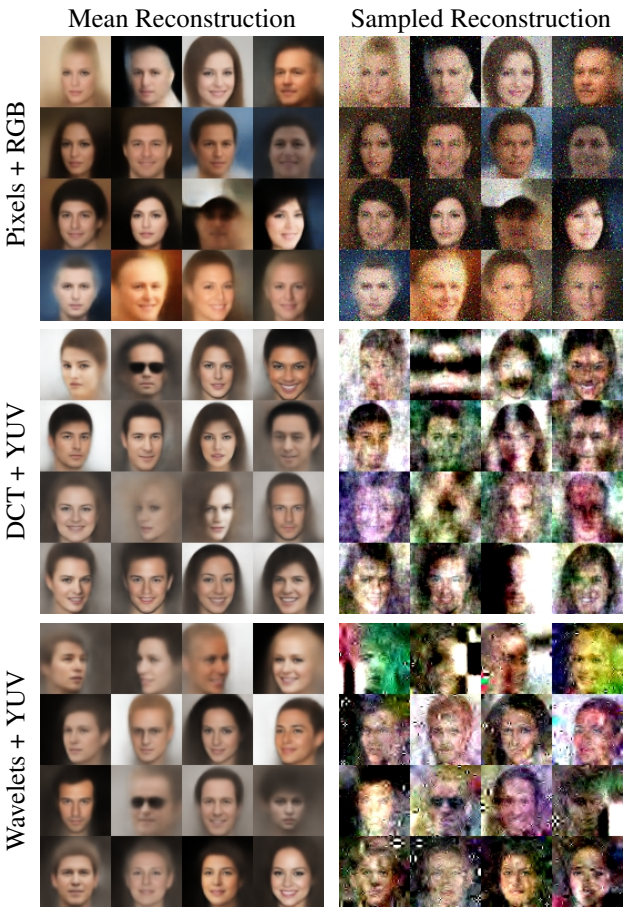

Figure 12. As is common practice, the VAE samples shown in this paper are samples from the latent space but not from the final conditional distribution. Here we contrast decoded means (left) and samples (right) from VAEs using our different output spaces, all using our general distribution.

in $[0, 1]$) and then scale each coefficient's loss by 2^d, where d is the coefficients level.

In Figure 13 we visualize the final shape parameters for each output coefficient that were converged upon during training. These results provide some insight into why our adaptive model produces improved results compared to the ablations of our model in which we use a single fixed or annealed value for α for all output coefficients. From the low α values in the luma channel we can infer that training has decided that luma variation often has outliers, and from the high α values in the chroma channel we can infer that chroma variation rarely has outliers. Horizontal luma variation (upper right) tends to have larger α values than vertical luma variation (lower left), perhaps because depth in this dataset is largely due to horizontal motion, and so horizontal gradients tend to provide more depth information than vertical gradients. Looking at the sides and the bottom of all scales and channels we see that the model ex-

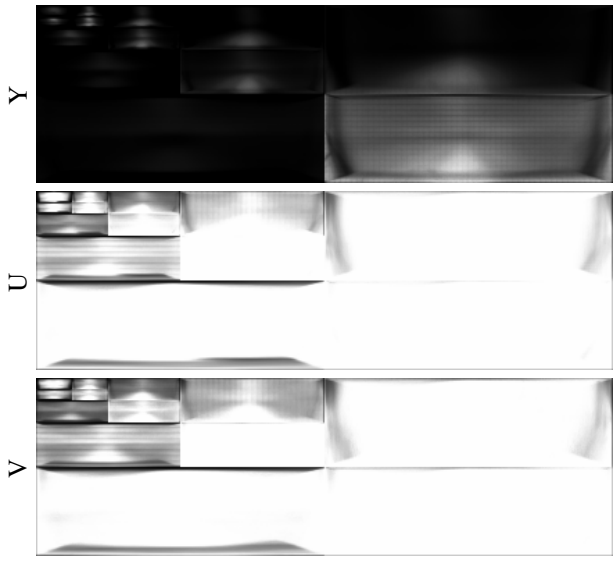

Figure 13. The final shape parameters α for our unsupervised monocular depth estimation model trained on KITTI data. The parameters are visualized in the same "YUV + Wavelet" output space as was used during training, where black is $\alpha = 0$ and white is $\alpha = 2$.

pects more outliers in these regions, which is likely due to boundary effects: these areas often contain consistent errors due to there not being a matching pixel in the alternate view, which in [40] causes the alternate view's interpolated pixel value to be 0.

In Figures 16 and 17 we present many more results from the test split of the KITTI dataset, in which we compare our "adaptive" model's output to the baseline model and the ground-truth depth. The improvement we see is substantial and consistent across a wide variety of scenes.

H. Fast Global Registration

Our registration results were produced using the code release corresponding to [39]. Because the numbers presented in [39] have low precision, we reproduced the performance of the baseline FGR algorithm using this code. This code included some evaluation details that were omitted from the paper that we determined through correspondence with the author: for each input, FGR is run 20 times with random initialization and the median error is reported. We use this procedure when reproducing the baseline performance of [39] and when evaluating our own models.

Figure 14. Random samples (more precisely, means of the output distributions decoded from random samples in our latent space) from our family of trained variational autoencoders, using the same format as in the paper.

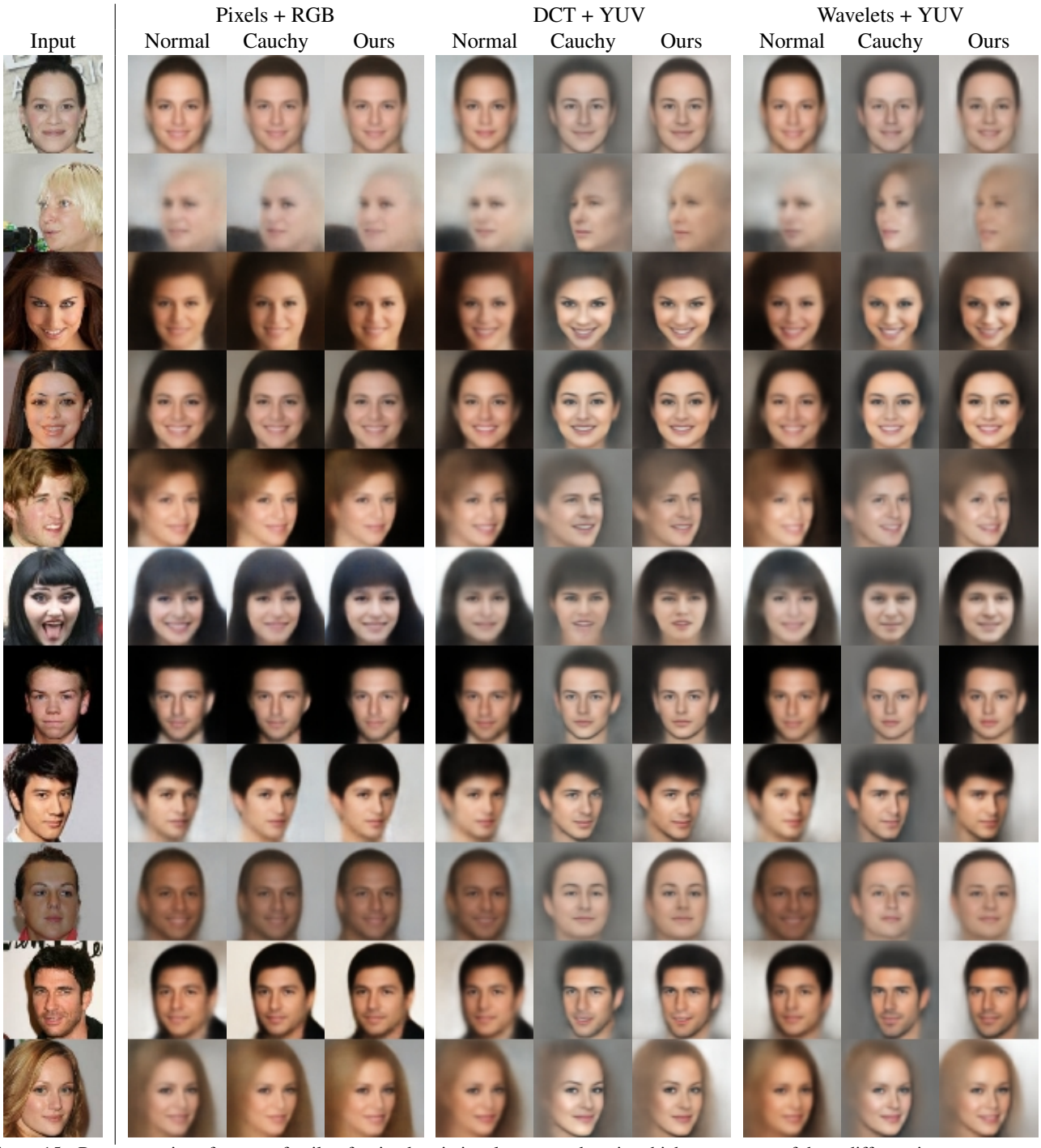

Figure 15. Reconstructions from our family of trained variational autoencoders, in which we use one of three different image representations for modeling images (triplets of columns) and and use either normal, Cauchy, or our general distributions for modeling the coefficients of each representation (sub-columns). The leftmost column shows the images which are used as input to each autoencoder. Reconstructions from models using general distributions tend to be sharper and more detailed than reconstructions from the corresponding model that uses normal distributions, particularly for the DCT or wavelet representations, though this difference is less pronounced than what is seen when comparing samples from these models. The DCT and wavelet models trained with Cauchy distributions systematically fail to preserve the background of the input image, as was noted when observing samples from these distributions.

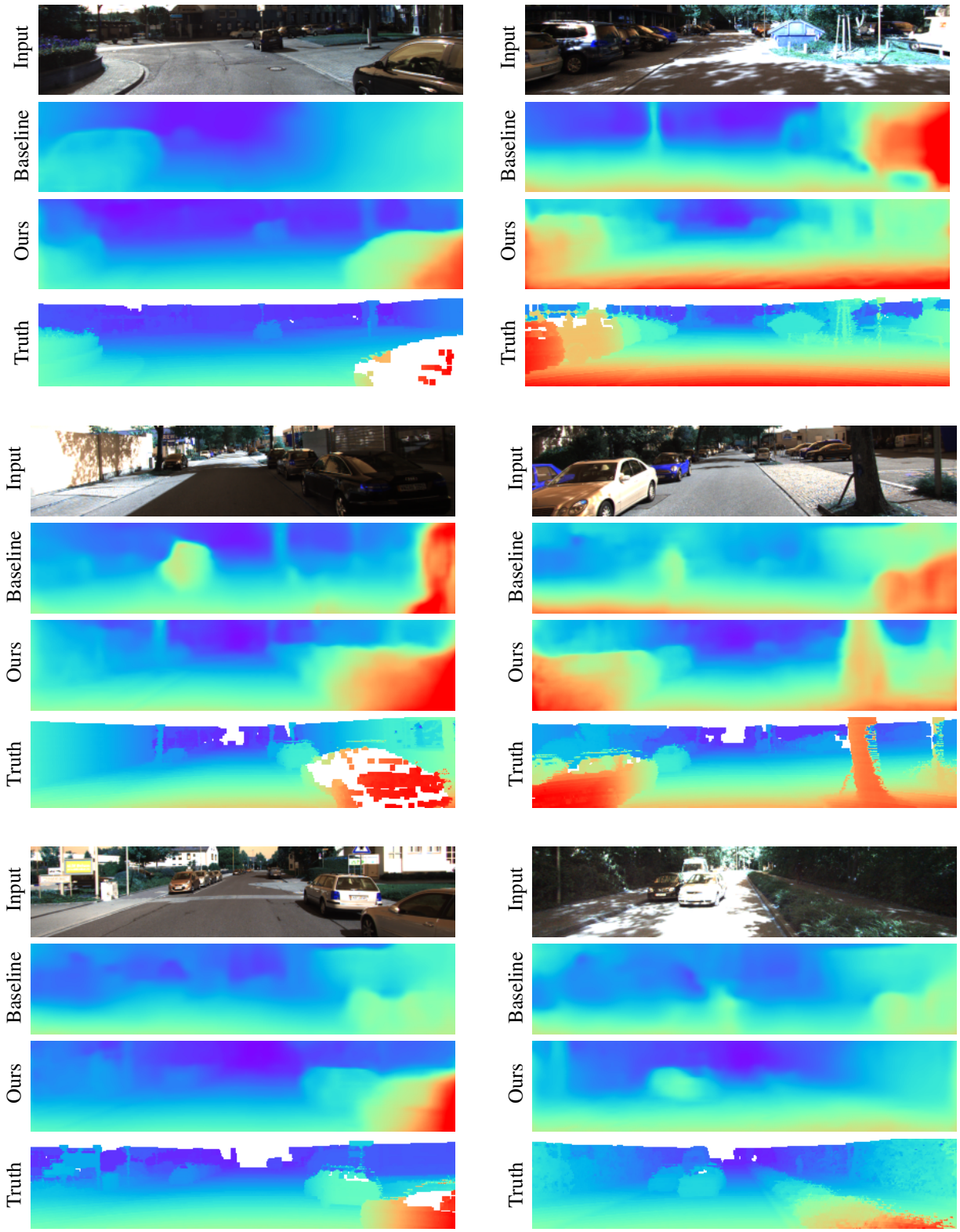

Figure 16. Monocular depth estimation results on the KITTI benchmark using the "Baseline" network of [40] and our own variant in which we replace the network's loss function with our own adaptive loss over wavelet coefficients. Changing only the loss function results in significantly improved depth estimates.

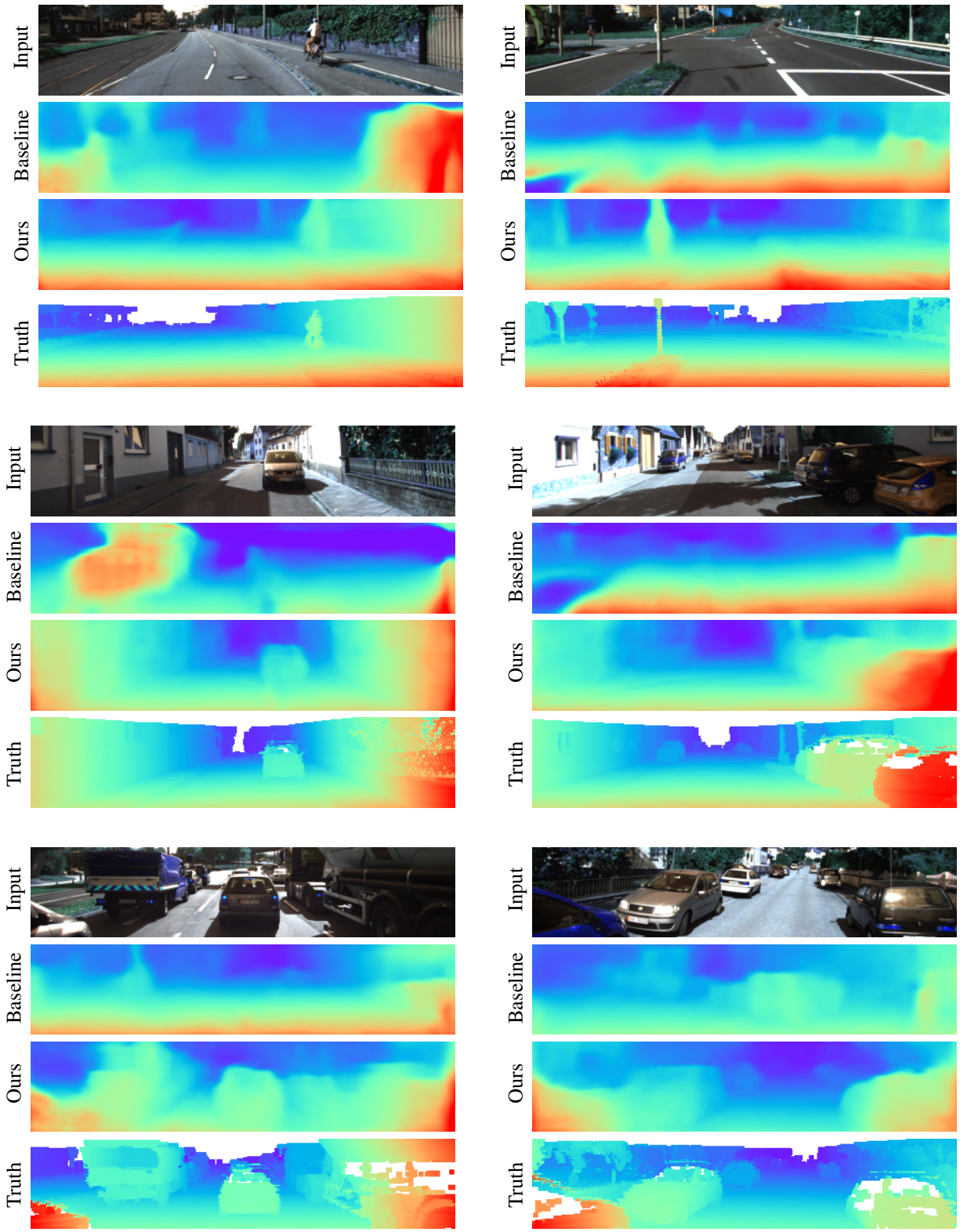

Figure 17. Additional monocular depth estimation results, in the same format as Figure 16.