Appendix

A. Experimental Details for FactorVAE and β-VAE

We use a Convolutional Neural Network for the encoder, a Deconvolutional Neural Network for the decoder and a Multi-Layer Perceptron (MLP) with for the discriminator in FactorVAE for experiments on all data sets. We use [0,1] normalised data as targets for the mean of a Bernoulli distribution, using negative cross-entropy for $\log p(x|z)$ and Adam optimiser (Kingma & Ba, 2015) with learning rate $10^{-4}, \beta_1 = 0.9, \beta_2 = 0.999$ for the VAE updates, as in Higgins et al. (2016). We also use Adam for the discriminator updates with $\beta_1 = 0.5, \beta_2 = 0.9$ and a learning rate tuned from $\{10^{-4}, 10^{-5}\}$. We use 10^{-4} for 2D Shapes and 3D Faces, and 10^{-5} for 3D Shapes, 3D Chairs and CelebA. The encoder outputs parameters for the mean and log-variance of Gaussian $q(z|x)$, and the decoder outputs logits for each entry of the image. We use the same encoder/decoder architecture for β-VAE and FactorVAE, shown in Tables 1, 2, and 3. We use the same 6 layer MLP discriminator with 1000 hidden units per layer and leaky ReLU (lReLU) nonlinearity, that outputs 2 logits in all FactorVAE experiments. We noticed that smaller discriminator architectures work fine, but noticed small improvements up to 6 hidden layers and 1000 hidden units per layer. Note that scaling the discriminator learning rate is not equivalent to scaling γ, since γ does not affect the discriminator loss. See Algorithm 2 for details of FactorVAE updates. We train for 3×10^5 iterations on 2D Shapes, 5×10^5 iterations on 3D Shapes, and 10^6 iterations on Chairs, 3D Faces and CelebA. We use a batch size of 64 for all data sets.

Table 1. Encoder and Decoder architecture for 2D Shapes data.

Encoder	Decoder
Input 64×64 binary image	Input $\in \mathbb{R}^{10}$
4×4 conv. 32 ReLU. stride 2	FC. 128 ReLU.
4×4 conv. 32 ReLU. stride 2	FC. $4 \times 4 \times 64$ ReLU.
4×4 conv. 64 ReLU. stride 2	4×4 upconv. 64 ReLU. stride 2
4×4 conv. 64 ReLU. stride 2	4×4 upconv. 32 ReLU. stride 2
FC. 128. FC. 2×10.	4×4 upconv. 32 ReLU. stride 2
	4×4 upconv. 1. stride 2

Table 2. Encoder and Decoder architecture for 3D Shapes, CelebA, Chairs data.

Encoder	Decoder
Input $64 \times 64 \times 3$ RGB image	Input $\in \mathbb{R}^6$ (3D Shapes) $\mathbb{R}^{10}$ (CelebA, Chairs)
4×4 conv. 32 ReLU. stride 2	FC. 256 ReLU.
4×4 conv. 32 ReLU. stride 2	FC. $4 \times 4 \times 64$ ReLU.
4×4 conv. 64 ReLU. stride 2	4×4 upconv. 64 ReLU. stride 2
4×4 conv. 64 ReLU. stride 2	4×4 upconv. 32 ReLU. stride 2
FC. 256. FC. 2×10.	4×4 upconv. 32 ReLU. stride 2
	4×4 upconv. 3. stride 2

Table 3. Encoder and Decoder architecture for 3D Faces data.

Encoder	Decoder
Input 64×64 greyscale image	Input $\in \mathbb{R}^{10}$
4×4 conv. 32 ReLU. stride 2	FC. 256 ReLU.
4×4 conv. 32 ReLU. stride 2	FC. $4 \times 4 \times 64$ ReLU.
4×4 conv. 64 ReLU. stride 2	4×4 upconv. 64 ReLU. stride 2
4×4 conv. 64 ReLU. stride 2	4×4 upconv. 32 ReLU. stride 2
FC. 256. FC. 2×10.	4×4 upconv. 32 ReLU. stride 2
	4×4 upconv. 1. stride 2

B. Details for the Disentanglement Metrics

We performed a sensitivity analysis of each metric with respect to its hyperparameters (c.f. Figure 2). In Figures 16, we show that the metric in Higgins et al. (2016) is very sensitive to number of iterations of the Adagrad (Duchi et al., 2011) optimiser with learning rate 0.01 (used in Higgins et al. (2016)), and constantly improves with more iterations. This suggests that one might want to use less noisy multi-class logistic regression solvers than gradient-descent based methods. The number of data points used to evaluate the metric after optimisation did not seem to help reduce variance beyond 800. So in our experiments, we use $L = 200$ and 10000 iterations, with a batch size of 10 per iteration of training the linear classifier, and use a batch of size 800 to evaluate the metric at the end of training. Each evaluation of this metric took around 30 minutes on a single GPU, hence we could not afford to train for more iterations.

For our disentanglement metric, we first prune out all latent dimensions that have collapsed to the prior ($q(z_j|x) = p(z_j)$). Then we just use the surviving dimensions for the majority vote. From the sensitivity analysis our metric in Figure 17, we observe that our metric is much less sensitive to hyperparameters than the metric in Higgins et al. (2016). We use $L = 100$ and take the majority vote classifier from 800 votes. This only takes a few seconds on a single GPU. The majority vote classifier C works as follows: suppose we are given data $(a_i, b_i)_{i=1}^M, a_i \in \{1, \dots, D\}, b_i \in \{1, \dots, K\}$ (so $M = 500$). Then for $j \in \{1, \dots, D\}$, let $V_{jk} = \sum_{i=1}^M \mathbb{I}(a_i = j, b_i = k)$. Then the majority vote classifier is defined to be $C(j) = \arg\max_k V_{jk}$.

Note that D, the dimensionality of the latents, does not affect the metric; for a classifier that chooses at random, the accuracy is $1/K$, independent of D.

For discrete latent variables, we use Gini's definition of

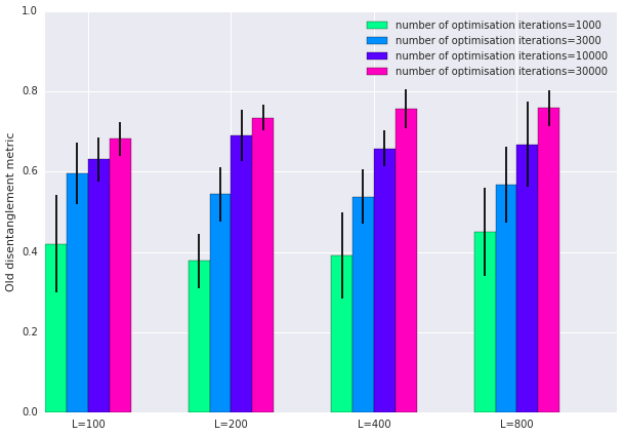

Figure 16. Mean and standard deviation of metric in Higgins et al. (2016) across 10 random seeds for varying L and number of Adagrad optimiser iterations (batch size 10). The number of points used for evaluation after optimisation is fixed to 800. These were all evaluated on a fixed, randomly chosen β-VAE model that was trained to convergence on the 2D Shapes data.

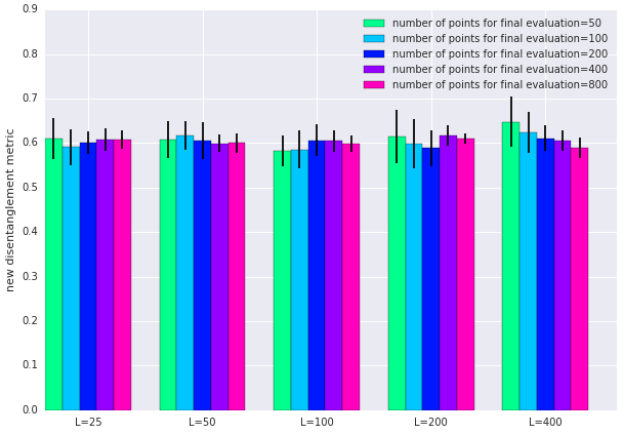

Figure 17. Mean and standard deviation of our metric across 10 random seeds for varying L and number of points used for evaluation. These were all evaluated on a fixed, randomly chosen β-VAE model that was trained to convergence on the 2D Shapes data.

empirical variance:

$$\widehat{Var}(x) = \frac{1}{2N(N-1)} \sum_{i,j=1}^{N} d(x_i, x_j) \tag{4}$$

for $x = [x_1, \dots, x_N] \in \mathbb{R}^N$, $d(x_i, x_j) = 1$ if $x_i \neq x_j$ and 0 if $x_i = x_j$. Note that this is equal to empirical variance for continuous variables when $d(x_i, x_j) = (x_i - x_j)^2$.

C. KL Decomposition

The KL term in the VAE objective decomposes as follows (Makhzani & Frey, 2017):

Lemma 1. $\mathbb{E}_{p_{data}(x)}[KL(q(z|x)||p(z))] = I_q(x;z) + KL(q(z)||p(z))$ *where* $q(x,z) = p_{data}(x)q(z|x)$.

Proof.

$$\begin{aligned}
&\mathbb{E}_{p_{data}(x)}[KL(q(z|x)||p(z))] \\
&= \mathbb{E}_{p_{data}(x)}\mathbb{E}_{q(z|x)}\left[\log \frac{q(z|x)}{p(z)}\right] \\
&= \mathbb{E}_{p_{data}(x)}\mathbb{E}_{q(z|x)}\left[\log \frac{q(z|x)}{q(z)}\frac{q(z)}{p(z)}\right] \\
&= \mathbb{E}_{p_{data}(x)}\mathbb{E}_{q(z|x)}\left[\log \frac{q(z|x)}{q(z)} + \log\frac{q(z)}{p(z)}\right] \\
&= \mathbb{E}_{p_{data}(x)}[KL(q(z|x)||q(z))] + \mathbb{E}_{q(x,z)}\left[\log \frac{q(z)}{p(z)}\right] \\
&= I_q(x;z) + \mathbb{E}_{q(z)}\left[\log \frac{q(z)}{p(z)}\right] \\
&= I_q(x;z) + KL(q(z)||p(z))
\end{aligned}$$

□

Remark. *Note that this decomposition is equivalent to that in Hoffman & Johnson (2016), written as follows:* $\mathbb{E}_{p_{data}(x)}[KL(q(z|x)||p(z))] = I_r(i;z) + KL(q(z)||p(z))$ *where* $r(i,z) = \frac{1}{N}q(z|x^{(i)})$, *hence* $r(z|i) = q(z|x^{(i)})$, $r(z) = \frac{1}{N}\sum_{i=1}^{N} q(z|x^{(i)}) = q(z)$.

Proof.

$$\begin{aligned}
I_r(i;z) &= \mathbb{E}_{r(i)}[KL(r(z|i)||r(z))] \\
&= \frac{1}{N}\sum_{i=1}^{N} KL(q(z|x^{(i)})||q(z)) \\
&= \mathbb{E}_{p_{data}(x)}[KL(q(z|x)||q(z))] \\
&= I_q(x;z)
\end{aligned}$$

□

D. Using a Batch Estimate of $q(z)$ for Estimating TC

We have also tried using a batch estimate for the density $q(z)$, thus optimising this estimate of the TC directly instead of having a discriminator and using the density ratio trick. In other words, we tried $q(z) \approx \hat{q}(z) = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} q(z|x^{(i)})$, and using the estimate:

$$\begin{aligned}
KL(q(z)||\prod_j q(z_j)) &= \mathbb{E}_{q(z)}\left[\log \frac{q(z)}{\prod_j q(z_j)}\right] \\
&\approx \mathbb{E}_{q(z)}\left[\log \frac{\hat{q}(z)}{\prod_j \hat{q}(z_j)}\right]
\end{aligned} \tag{5}$$

Note that:

$$\mathbb{E}_{q(z)}\left[\log \frac{\hat{q}(z)}{\prod_j \hat{q}(z_j)}\right] \approx \frac{1}{H}\sum_{h=1}^{H}\left[\log \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\prod_{j=1}^{D} q(z_j^{(h)}|x^{(i)}) - \log \prod_{j=1}^{D}\frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} q(z_j^{(h)}|x^{(i)})\right] \tag{6}$$

for $z^{(h)} \overset{iid}{\sim} q(z)$. However while experimenting on 2D Shapes, we observed that the value of $\log q(z^{(h)})$ becomes very small (negative with high absolute value) for latent dimension $d \geq 2$ during training, because $\hat{q}(z)$ is not a good enough approximation to $q(z)$ unless $\mathcal{B}$ is very big. As training progresses for the VAE, the variance of Gaussians $q(z|x^{(i)})$ becomes smaller and smaller, so they do not overlap too much in higher dimensions. Hence we get $z^{(h)} \sim q(z)$ that land on the tails of $\hat{q}(z) = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} q(z|x^{(i)})$, giving worryingly small values of $\log \hat{q}(z^{(h)})$. On the other hand $\prod_j \hat{q}(z_j^{(h)})$, a mixture of $|\mathcal{B}|^d$ Gaussians hence of much higher entropy, gives much more stable values of $\log \prod_j \hat{q}(z_j^{(h)})$. From Figure 18, we can see that even with $\mathcal{B}$ as big as 10,000, we get negative values for the estimate of TC, which is a KL divergence and hence should be non-negative, hence this method of using a batch estimate for $q(z)$ does not work. A fix is to use samples from $\hat{q}(z)$ instead of $q(z)$, but this seemed to give a similar reconstruction-disentanglement trade-off to β-VAE. Very recently, work from (Chen et al., 2018) has shown that disentangling can be improved by using samples from $\hat{q}(z)$.

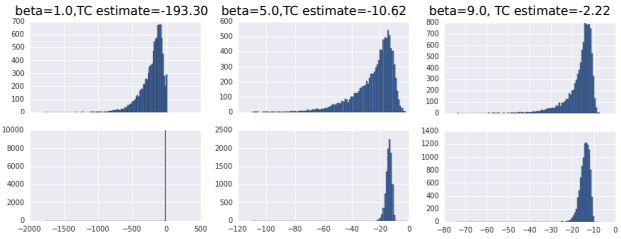

Figure 18. Histogram of $\log \hat{q}(z^{(h)})$ (top) and $\prod_{j=1}^{d} \hat{q}(z_j^{(h)})$ (bottom) for $z^{(h)} \overset{iid}{\sim} q(z)$ with $|\mathcal{B}| = 10000$, $d = 10$. The columns correspond to values of $\beta = 1, 5, 9$ for training β-VAE. In the title of each histogram, there is an estimate of TC based on the samples of $z^{(h)}$.

E. Log Marginal Likelihood and Samples

We give the log marginal likelihood of each of the best performing β-VAE and FactorVAE models (in terms of disentanglement) for both the 2D Shapes and 3D Shapes data sets along with samples from the generative model. Since the log marginal likelihood is intractable, we report the *Importance-Weighted Autoencoder* (IWAE) bound with 5000 particles, in line with standard practice in the generative modelling literature.

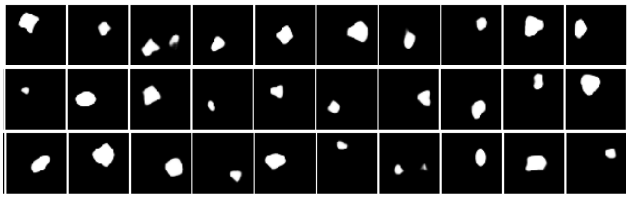

Figure 19. Randomly chosen samples from the best performing (in terms of disentanglement) β-VAE generative model ($\beta = 4$).

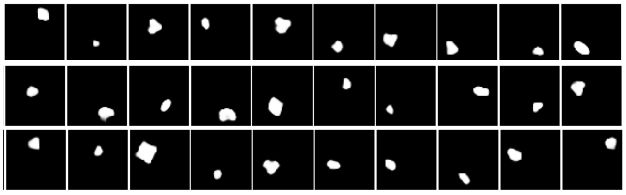

Figure 20. Randomly chosen samples from the best performing (in terms of disentanglement) FactorVAE generative model ($\gamma = 35$).

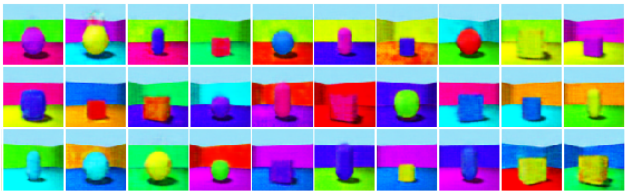

Figure 21. Randomly chosen samples from the best performing (in terms of disentanglement) β-VAE generative model ($\beta = 32$).

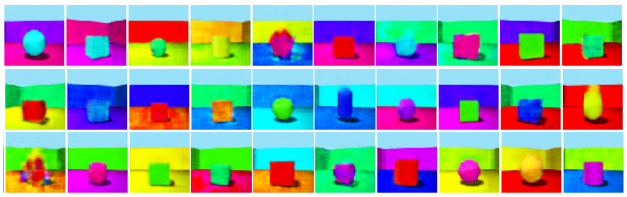

Figure 22. Randomly chosen samples from the best performing (in terms of disentanglement) FactorVAE generative model ($\gamma = 6$).

In Figures 19 and 20, the samples for FactorVAE are arguably more representative of the data set than those of β-VAE. For example β-VAE has occasional samples with two separate shapes in the same image (Figure 19). The log marginal likelihood for the best performing β-VAE ($\beta = 4$) is -46.1, whereas for FactorVAE it is -51.9 ($\gamma = 35$) (a randomly chosen VAE run gives -43.3). So on 2D Shapes, FactorVAE gives better samples but worse log marginal likelihood.

In Figures 21 and 22, the samples for β-VAE appear more coherent than those for FactorVAE. However the log marginal likelihood for β-VAE ($\beta = 32$) is -3534, whereas for FactorVAE it is -3520 ($\gamma = 6$) (a randomly chosen VAE run gives -3517). So on 3D Shapes, FactorVAE gives worse

samples but better log marginal likelihood.

In general, if one seeks to learn a generative model with a disentangled latent space, it would make sense to choose the model with the lowest value of β or γ among those with similarly high disentanglement performance.

F. Losses and Experiments for other related Methods

The Adversarial Autoencoder (AAE) (Makhzani et al., 2015) uses the following objective

$$\frac{1}{N}\sum_{i=1}^{N}\Big[\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)]\Big] - KL(q(z)||p(z)), \quad (7)$$

utilising the density ratio trick to estimate the KL term.

Information Dropout (Achille & Soatto, 2018) uses the objective

$$\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)] - \beta KL(q(z|x^{(i)})||q(z)). \quad (8)$$

The following objective is also considered in the paper but is dismissed as intractable:

$$\frac{1}{N}\sum_{i=1}^{N}\Big[\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)] - \beta KL(q(z|x^{(i)})||q(z))\Big] - \gamma KL(q(z)||\prod_{j=1}^{d} q(z_j)) \quad (9)$$

Note that it is similar to the FactorVAE objective (which has $\beta = 1$), but with $p(z)$ in the first KL term replaced with $q(z)$.

DIP-VAE (Kumar et al., 2018) uses the VAE objective with an additional penalty on how much the covariance of $q(z)$ deviates from the identity matrix, either using the law of total covariance $Cov_{q(z)}[z] = \mathbb{E}_{p_{data}(x)}Cov_{q(z|x)}[z] + Cov_{p_{data}(x)}(\mathbb{E}_{q(z|x)}[z])$ (DIP-VAE I):

$$\frac{1}{N}\sum_{i=1}^{N}\Big[\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)] - KL(q(z|x^{(i)})||p(z))\Big] - \lambda_{od}\sum_{i\neq j}[Cov_{p_{data}(x)}[\mu(x)]]_{ij}^2 - \lambda_d\sum_i([Cov_{p_{data}(x)}[\mu(x)]]_{ii}-1)^2 \quad (10)$$

where $\mu(x) = mean(q(z|x))$, or directly (DIP-VAE II):

$$\frac{1}{N}\sum_{i=1}^{N}\Big[\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)] - KL(q(z|x^{(i)})||p(z))\Big] - \lambda_{od}\sum_{i\neq j}[Cov_{q(z)}[z]]_{ij}^2 - \lambda_d\sum_i([Cov_{q(z)}[z]]_{ii}-1)^2 \quad (11)$$

One could argue that during training of FactorVAE, $\prod_j q(z_j)$ will be similar to $p(z)$, assuming the prior is factorial, due to the $KL(q(z|x)||p(z))$ term in the objective. Hence we also investigate a modified FactorVAE objective that replaces $\prod_j q(z_j)$ with $p(z)$:

$$\frac{1}{N}\sum_{i=1}^{N}\Big[\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)] - KL(q(z|x^{(i)})||p(z))\Big] - \gamma KL(q(z)||p(z)) \quad (12)$$

However as shown in Figure 40 of Appendix I, the histograms of samples from the marginals are clearly quite different from the the prior for FactorVAE.

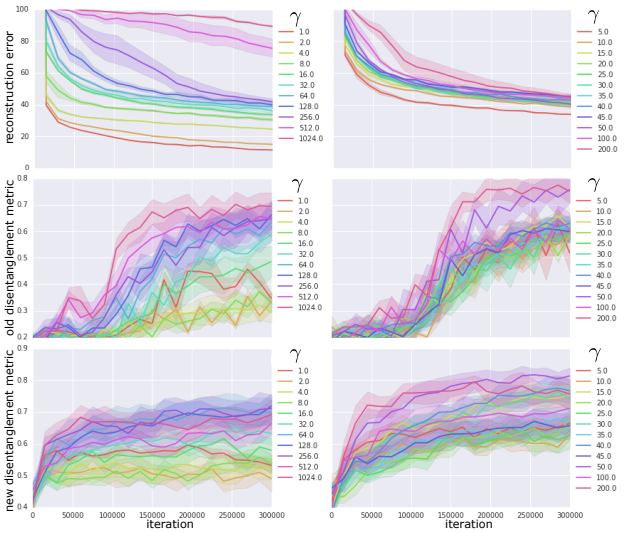

Figure 23. Same as Figure 4 but for AAE (left) and a variant of FactorVAE (Eqn. (12).

Moreover we show experimental results for AAE (adding a γ coefficient in front of the $KL(q(z)||p(z))$ term of the objective and tuning it) and the variant of FactorVAE (Eqn. (12)) on the 2D Shapes data. From Figure 23, we see that the disentanglement performance for both are somewhat lower than that for FactorVAE. This difference could be explained as a benefit of directly encouraging $q(z)$ to be factorised (FactorVAE) instead of encouraging it to approach an arbitrarily chosen factorised prior $p(z) = \mathcal{N}(0, I)$ (AAE, Eqn. (12)). Information Dropout and DIP-VAE did

not have enough experimental details in the paper nor publicly available code to have their results reproduced and compared against.

G. InfoGAN and InfoWGAN-GP

We give an overview of InfoGAN (Chen et al., 2016) and InfoWGAN-GP, its counterpart using Wasserstein distance and gradient penalty. InfoGAN uses latents $z = (c, \epsilon)$ where c models semantically meaningful codes and ϵ models incompressible noise. The generative model is defined by a generator G with the process: $c \sim p(c), \epsilon \sim p(\epsilon), z = (c, \epsilon), x = G(z)$. i.e. $p(z) = p(c)p(\epsilon)$. GANs are defined as a minimax game on some objective $V(D, G)$, where D is either a discriminator (e.g. for the original GAN (Goodfellow et al., 2014)) that outputs log probabilities for binary classification, or a critic (e.g. for Wasserstein-GAN (Arjovsky et al., 2017)) that outputs a real-valued scalar. InfoGAN defines an extra encoding distribution $Q(c|x)$ that is used to define an extra penalty:

$$L(G, Q) = \mathbb{E}_{p(c)}\mathbb{E}_{p(\epsilon)}[\log Q(c|G(c, \epsilon))] \tag{13}$$

that is added to the GAN objective. Hence InfoGAN is the following minimax game on the parameters of neural nets D, G, Q:

$$\min_{G,Q}\max_{D} V_I(D, G, Q) = \min_{G,Q}\max_{D} V(D, G) - \lambda L(G, Q) \tag{14}$$

L can be interpreted as a variational lower bound to $I(c; G(c, \epsilon))$, with equality at $Q = \arg\min_Q V_I(D, G, Q)$. i.e. L encourages the codes to be more informative about the image. From the definition of L, it can also be seen as the reconstruction error of codes in the latent space. The original InfoGAN defines:

$$V(D, G) = \mathbb{E}_{p_{data}(x)}[D(x)] - \mathbb{E}_{p(z)}[D(G(z))] \tag{15}$$

same as the original GAN objective where D outputs log probabilities. However as we'll show in Appendix H this has known instability issues in training. So it is natural to try replacing this with the more stable WGAN-GP (Gulrajani et al., 2017) objective:

$$\begin{aligned} V(D, G) =& \mathbb{E}_{p_{data}(x)}[D(x)] - \mathbb{E}_{p(z)}[D(G(z))] \\ &+ \eta(||\nabla_x D(x)|_{x=\hat{x}}||_2 - 1)^2 \end{aligned} \tag{16}$$

for $\hat{x} = \pi x_r + (1 - \pi)x_f$ with $\pi \sim U[0, 1]$, $x_r \sim p_{data}(x)$, $z_f \sim p(z)$, $x_f = \texttt{stop\_gradient}(G(z_f))$ and with a new $\hat{x}$ for each iteration of optimisation. Thus we obtain InfoWGAN-GP.

H. Empirical Study of InfoGAN and InfoWGAN-GP

To begin with, we implemented InfoGAN and InfoWGAN-GP on MNIST using the hyperparameters given in Chen et al. (2016) to better understand its behaviour, using 1 categorical code with 10 categories, 2 continuous codes, and 62 noise variables. We use priors $p(c_j) = U[-1, 1]$ for the continuous codes, $p(c_j) = \frac{1}{J}$ for categorical codes with J categories, and $p(\epsilon_j) = \mathcal{N}(0, 1)$ for the noise variables. For 2D Shapes data we use 1 categorical codes with 3 categories ($J = 3$), 4 continuous codes, and 5 noise variables. The number of noise variables did not seem to have a noticeable effect on the experiment results. We use the Adam optimiser (Kingma & Ba, 2015) with $\beta_1 = 0.5, \beta_2 = 0.999$, and learning rate 10^{-3} for the generator updates and 10^{-4} for the discriminator updates. The detailed Discriminator/Encoder/Generator architecture are given in Tables 4 and 5. The architecture for InfoWGAN-GP is the same as InfoGAN, except that we use no Batch Normalisation (batchnorm) (Ioffe & Szegedy, 2015) for the convolutions in the discriminator, and replace batchnorm with Layer Normalisation (Ba et al., 2016) in the fully connected layer that follows the convolutions as recommended in (Gulrajani et al., 2017). We use gradient penalty coefficient $\eta = 10$, again as recommended.

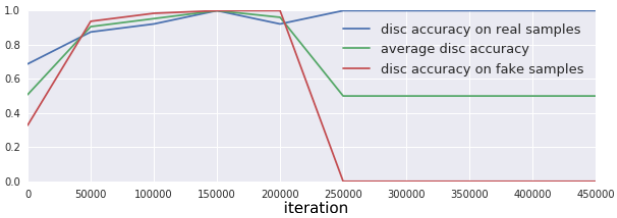

Figure 24. Discriminator accuracy of InfoGAN on MNIST throughout training.

We firstly observe that for all runs, we eventually get a degenerate discriminator that predicts all inputs to be real, as in Figure 24. This is the well-known instability issue of the original GAN. We have tried using a smaller learning rate for the discriminator, and although this delays the degenerate behaviour it does not prevent it. Hence early stopping seems crucial, and all results shown below are from well before the degenerate behaviour occurs.

Chen et al. (2016) claim that the categorical code learns digit class (discrete factor of variation) and that the continuous codes learn azimuth and width, but when plotting latent traversals for each run, we observed that this is inconsistent. We show five randomly chosen runs in Figure 25. The digit class changes in the continuous code traversals and there are overlapping digits in the categorical code traversal. Similar results hold for InfoWGAN-GP in Figure 36.

We also tried visualising the reconstructions: given an image, we push the image through the encoder to obtain latent codes c, fix this c and vary the noise ϵ to generate multiple reconstructions for the same image. This is to check the extent to which the noise ϵ can affect the generation. We can see in Figure 26 that digit class often changes when varying

Table 4. InfoGAN architecture for MNIST data. 2 continuous codes, 1 categorical code with 10 categories, 62 noise variables.

discriminator D / **encoder** Q	**generator** G
Input 28×28 greyscale image	Input $\in \mathbb{R}^{74}$
4×4 conv. 64 lReLU. stride 2	FC. 1024 ReLU. batchnorm
4×4 conv. 128 lReLU. stride 2. batchnorm	FC. $7 \times 7 \times 128$ ReLU. batchnorm
FC. 1024 lReLU. batchnorm	4×4 upconv. 64 ReLU. stride 2. batchnorm
FC. 1. output layer for D	4×4 upconv. 1 Sigmoid. stride 2
FC. 128 lReLU. batchnorm. FC $2 \times 2 + 1 \times 10$	

Table 5. InfoGAN architecture for 2D Shapes data. 4 continuous codes, 1 categorical code with 3 categories, 5 noise variables.

discriminator D / **encoder** Q	**generator** G
Input 64×64 binary image	Input $\in \mathbb{R}^{12}$
4×4 conv. 32 lReLU. stride 2	FC. 128 ReLU. batchnorm
4×4 conv. 32 lReLU. stride 2. batchnorm	FC. $4 \times 4 \times 64$ ReLU. batchnorm
4×4 conv. 64 lReLU. stride 2. batchnorm	4×4 upconv. 64 lReLU. stride 2. batchnorm
4×4 conv. 64 lReLU. stride 2. batchnorm	4×4 upconv. 32 lReLU. stride 2. batchnorm
FC. 128 lReLU. batchnorm	4×4 upconv. 32 lReLU. stride 2. batchnorm
FC. 1. output layer for D	4×4 upconv. 1 Sigmoid. stride 2
FC. 128 lReLU. batchnorm. FC $4 \times 2 + 1 \times 3$ for Q	

Table 6. Bigger InfoGAN architecture for 2D Shapes data. 4 continuous codes, 1 categorical code with 3 categories, 128 noise variables.

discriminator D / **encoder** Q	**generator** G
Input 64×64 binary image	Input $\in \mathbb{R}^{136}$
4×4 conv. 64 lReLU. stride 2	FC. 1024 ReLU. batchnorm
4×4 conv. 128 lReLU. stride 2. batchnorm	FC. $8 \times 8 \times 256$ ReLU. batchnorm
4×4 conv. 256 lReLU. stride 2. batchnorm	4×4 upconv. 256 lReLU. stride 1. batchnorm
4×4 conv. 256 lReLU. stride 1. batchnorm	4×4 upconv. 256 lReLU. stride 1. batchnorm
4×4 conv. 256 lReLU. stride 1. batchnorm	4×4 upconv. 128 lReLU. stride 2. batchnorm
FC. 1024 lReLU. batchnorm	4×4 upconv. 64 lReLU. stride 2. batchnorm
FC. 1. output layer for D	4×4 upconv. 1 Sigmoid. stride 2
FC. 128 lReLU. batchnorm. FC $4 \times 2 + 1 \times 3$ for Q	

ϵ, so the model struggles to cleanly separate semantically meaningful information and incompressible noise.

Furthermore, we investigated the sensitivity of the model to the number of latent codes. We show latent traversals using three continuous codes instead of two in Figure 27. It is evident that the model tries to put more digit class information into the continuous traversals. So the number of codes is an important hyperparameter to tune, whereas VAE methods are less sensitive to the choice of number of codes since they can prune out unnecessary latents by collapsing $q(z_j|x)$ to the prior $p(z_j)$.

We also tried varying the number of categories for the categorical code. Using 2 categories, we see from Figure 28 that the model tries to put much more information about digit class into the continuous latents, as expected. Moreover from Figure 30, we can see that the noise variables also have more information about the digit class. However, when we use 20 categories, we see that the model still puts

Figure 25. Latent traversals for InfoGAN on MNIST across the two continuous codes (first two rows) and the categorical code (last row) for 5 different random seeds.

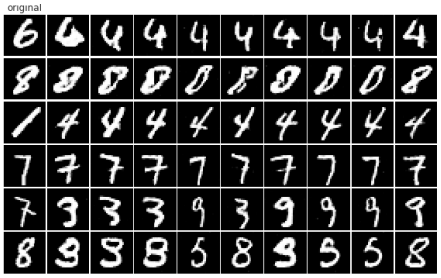

Figure 26. Reconstructions for InfoGAN on MNIST. First column: original image. Remaining columns: reconstructions varying the noise latent ϵ.

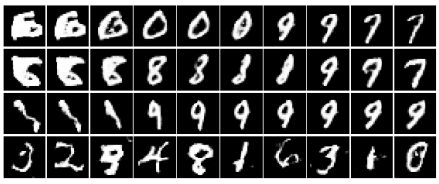

Figure 27. Latent traversals for InfoGAN on MNIST across the three continuous codes (first three rows) and the categorical code (last row).

information about the digit class in the continuous latents. However from Figure 31 we see that the noise variables contain less semantically meaningful information.

Using InfoWGAN-GP solved the degeneracy issue and makes training more stable (see Figure 33), but we observed that the other problems persisted (see e.g. Figure 36).

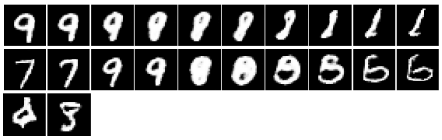

Figure 28. Latent traversals for InfoGAN on MNIST across the two continuous codes (first two rows) and the categorical code (last row) using 2 categories

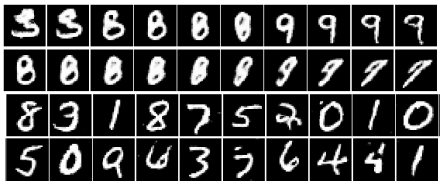

Figure 29. Latent traversals for InfoGAN on MNIST across the two continuous codes (first two rows) and the categorical code (last two rows) using 20 categories

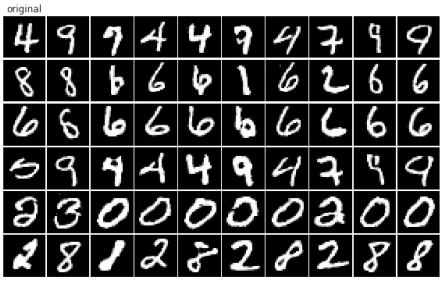

Figure 30. Same as Figure 26 but the categorical code having 2 categories.

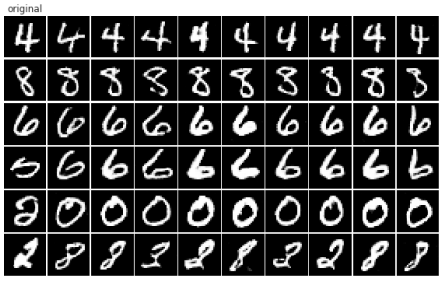

Figure 31. Same as Figure 30 but with 20 categories.

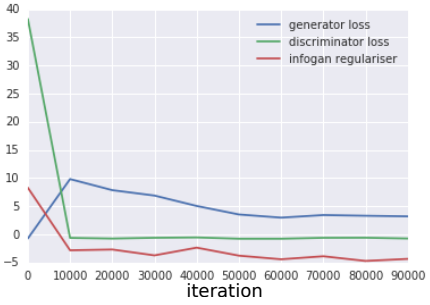

Figure 32. The generator loss $-\mathbb{E}_{p(z)}[D(G(z))]$, discriminator loss $\mathbb{E}_{p_{data}(x)}[D(x)] - \mathbb{E}_{p(z)}[D(G(z))]$ and the InfoGAN regulariser term $-L$ for InfoWGAN-GP on MNIST with $\lambda = 1$

For 2D Shapes, we have also tried using a bigger architecture for InfoWGAN-GP that is used for a data set of similar dimensions (Chairs data set) in Chen et al. (2016). See Table 6. However as can be seen in Figure 34 this did not improve disentanglement scores, yet the latent traversals look slightly more realistic (Figure 35).

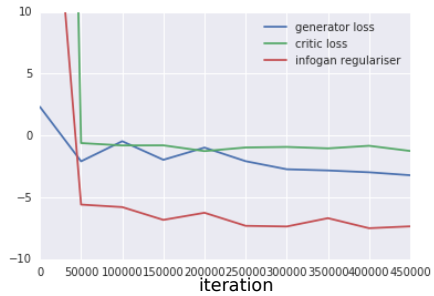

Figure 33. Same as Figure 32 but for 2D Shapes.

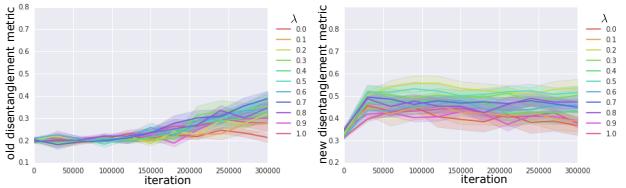

Figure 34. Disentanglement scores for InfoWGAN-GP on 2D Shapes with bigger architecture (Table 6) for 10 random seeds per hyperparameter setting. Left: Metric in Higgins et al. (2016). Right: Our metric.

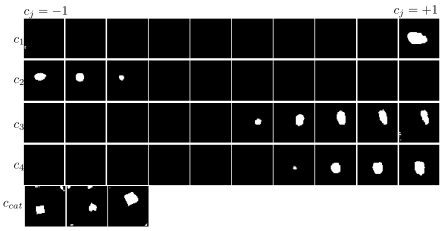

Figure 35. Latent traversals for InfoWGAN-GP on 2D Shapes across the four continuous codes (first four rows) and the categorical code (last row) with bigger architecture (Table 6) for run with best disentanglement score ($\lambda = 0.6$).

In summary, InfoWGAN-GP can help prevent the instabilities in training faced by InfoGAN, but it does not help overcome the following weaknesses compared to VAE-based methods: 1) Disentangling performance is sensitive to the number of code latents. 2) More often than not, the noise variables contain semantically meaningful information. 3) The model does not always generalise well to all across the domain of $p(z)$.

I. Further Experimental Results

From Figure 37, we see that higher values of γ in FactorVAE leads to a lower discriminator accuracy. This is as expected, since a higher γ encourages $q(z)$ and $\prod_j q(z_j)$ to be closer together, hence a lower accuracy for the discriminator to successfully classify samples from the two distributions.

We also show histograms of $q(z_j)$ for each j in β-VAE

Figure 36. Same as Figure 25 but for InfoWGAN-GP.

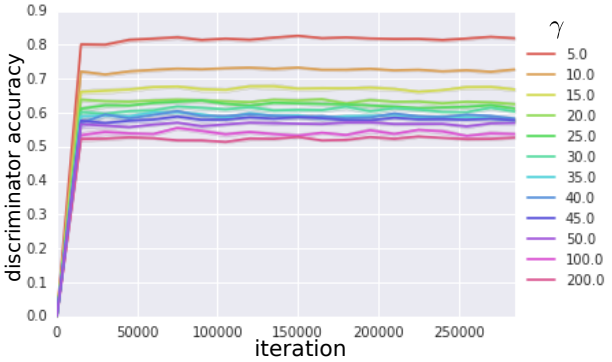

Figure 37. Plot of discriminator accuracy of FactorVAE on 2D Shapes data across iterations over 5 random seeds.

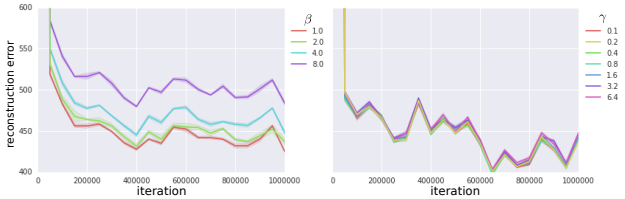

Figure 38. Same as Figure 12 but for 3D Chairs.

and FactorVAE for different values of β and γ at the end of training on 2D Shapes in Figure 40. We can see that the marginals of FactorVAE are quite different from the prior, which could be a reason that the variant of FactorVAE using the objective given by Eqn. (12) leads to different results to FactorVAE. For FactorVAE, the model is able to focus on factorising $q(z)$ instead of pushing it towards some

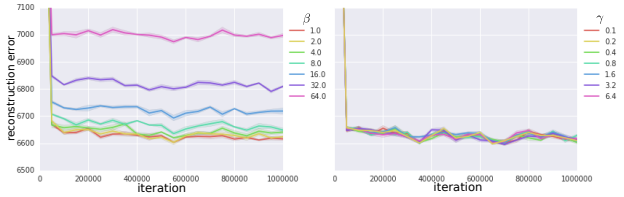

Figure 39. Same as Figure 12 but for CelebA.

arbitrarily specified prior $p(z)$.

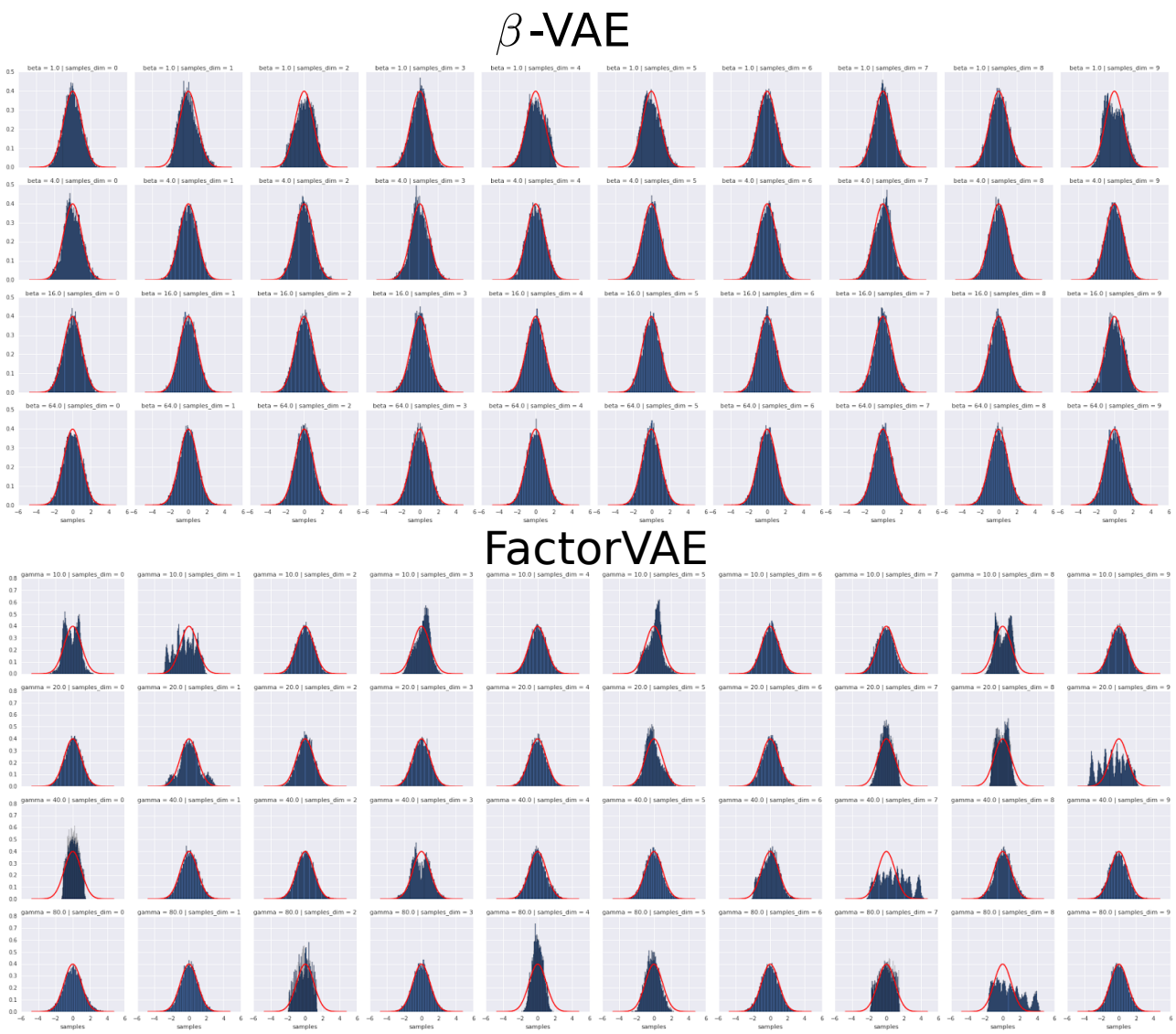

Figure 40. Histograms of $q(z_j)$ for each j (columns) for β-VAE and FactorVAE at the end of training on 2D Shapes, with the pdf of Gaussian $\mathcal{N}(0, 1)$ overlaid in red. The rows correspond to different values of β $(1, 4, 16, 64)$ and γ $(10, 20, 40, 80)$ respectively.