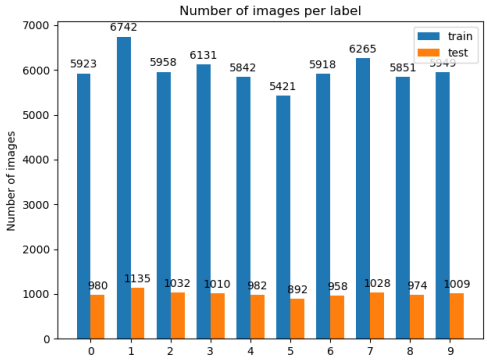


Figure 8: MNIST dataset

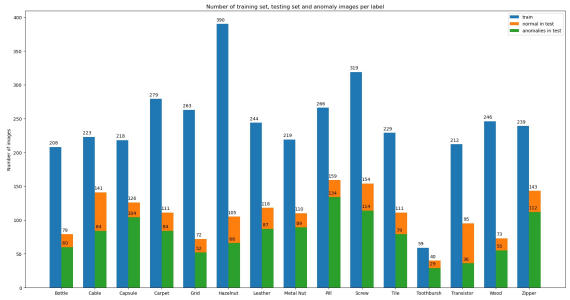


Figure 9: MVTec-AD dataset

Figure 10: The training, test (and anomaly for MVTec-AD) distribution of the labels for the MVTec-AD and MNIST dataset

Appendix

A Dataset information

The detailed labels distribution for some of the anomaly detection datasets is described in figure 10.

B Details of Anomaly localisation experiments

B.1 Architecture

For the models used on MNIST and UCSD pedestrians, no explicit model architecture was specified. However, the codebase from the authors did provide a simple VAE architecture to be used on MNIST, which provided decent qualitative results. We adapted this model to fit the resized UCSD pedestrian images, by adding an extra convolutional layer. Initially, adding two additional convolutional layers to the existing two convolutional layers seemed logical, as to help reduce the number of parameters. This resulted in poor performance relative to the results reported in the paper, with the attention maps highlighting large regions of the pedestrian paths, instead of localizing its attention on smaller regions. Instead, the better architecture seemed to be 3 total convolutional layers for the encoder, rather than 4. Although the exact model used on the UCSD dataset is unclear, we could infer that parts of the model were correct, as the first 3 convolutional layers are supposed to output shapes of 50x50, 25x25 and 12x12, which was the case after the addition of the third convolutional layer. The model used on the MVTec-AD dataset was said to be a ResNet-18 based VAE. Even though the ResNet-18 architecture is a well defined model, this is usually the case for classification problems, whereas in the case of attention generation, we require a ResNet-18 based encoder and decoder model. To build the ResNet-18 Encoder/Decoder, we used a ResNet-18 VAE implementation from PyTorch Lightning Bolts, which replaces the last two layers in the encoder to map to our latent space dimension of 32. For the decoder, it is the same model, but using transpose convolutions and upsampling layers instead.

C Details of Disentanglement experiments

C.1 Architecture

The Variational Autoencoder architecture used in all the experiments is depicted in table 5. Additionally, the discriminator is a 6 layers Multilayer Perceptron with leaky ReLU (lReLU) activation that outputs 2 values. The lReLU is applied between every two FC layers with a negative slope of 0.2.

C.2 About training time

As mentioned previously, there are several motives that led to reduce the training computation time. Similarly to (Kim and Mnih, 2019) after 150000 iterations, the variations in the training losses are smaller and the results tend to stabilize (see figure 11). This might be related to the fact with the batch size of 64 images, this training schedule consisted

MNIST		UCSD	
Encoder	Tensor Shape	Encoder	Tensor Shape
Input	28×28 binary image	Input	100×100 binary image
4×4 conv. ReLU $64, 2, 1$	$14 \times 14 \times 64$	4×4 conv. ReLU $64, 2, 1$	$50 \times 50 \times 64$
4×4 conv. ReLU $128, 2, 1$	$7 \times 7 \times 128$	4×4 conv. ReLU $128, 2, 1$	$25 \times 25 \times 128$
Flatten	6272	5×5 conv. ReLU $128, 2, 1$	$12 \times 12 \times 128$
Linear	1024	4×4 conv. ReLU $128, 2, 1$	$6 \times 6 \times 128$
ReLU	1024	Flatten	4608
Linear	32	Linear	1024
		ReLU	1024
		Linear	32
Decoder	Tensor Shape	Decoder	Tensor Shape
Input	latent variables $\in \mathbb{R}^z$	Input	latent variables $\in \mathbb{R}^z$
linear	1024	linear	1024
ReLU	1024	ReLU	1024
linear	6272	linear	4608
ReLU	6272	ReLU	4608
Reshape	$7 \times 7 \times 128$	Reshape	$6 \times 6 \times 128$
ReLU	$7 \times 7 \times 128$	ReLU	$6 \times 6 \times 128$
4×4 upconv. ReLU $64, 2, 1$	$14 \times 14 \times 64$	4×4 upconv. ReLU $128, 2, 1$	$12 \times 12 \times 128$
ReLU	$14 \times 14 \times 64$	5×5 upconv. ReLU $128, 2, 1$	$25 \times 25 \times 128$
4×4 upconv. ReLU $1, 2, 1$	$28 \times 28 \times 1$	4×4 upconv. ReLU $64, 2, 1$	$50 \times 50 \times 64$
Sigmoid	$28 \times 28 \times 1$	4×4 upconv. $1, 2, 1$	$100 \times 100 \times 1$
		Sigmoid	$100 \times 100 \times 1$

Table 3: Architecture details of the encoder and decoder used for the MNIST experiments on the left and for the UCSD experiments on the right. The notation per layer is - kernel size, type of 2D convolution, activation function, output shape, stride, padding. Moreover, upconv represents the transpose convolution.

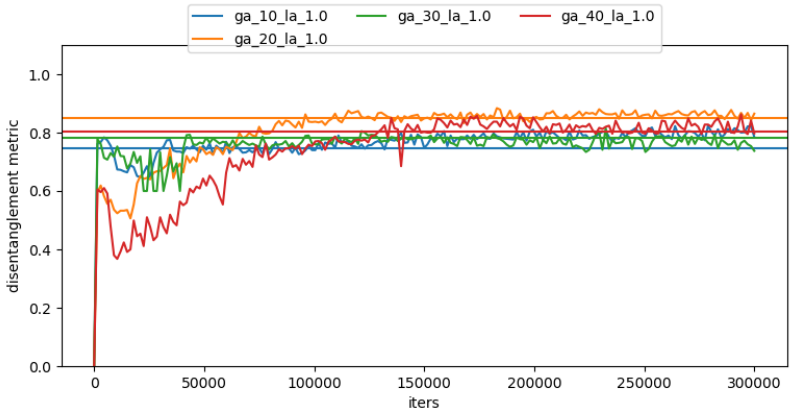


Figure 11: Disentanglement metric over iterations of the AD-FactorVAE with $\lambda = 1.0$. The horizontal lines mark the disentanglement score at iteration 150000 and it is observed that the fluctuations in the score are minimal until 300000 iterations.

of 13 epochs over the 737280 images which is a considerable training schedule given the dimensions of the dataset. Additionally, for better results, given the computational resources available, we decided to prioritize training with more than one random seed over training longer. Hence we decided to train the models with 150000 iterations.

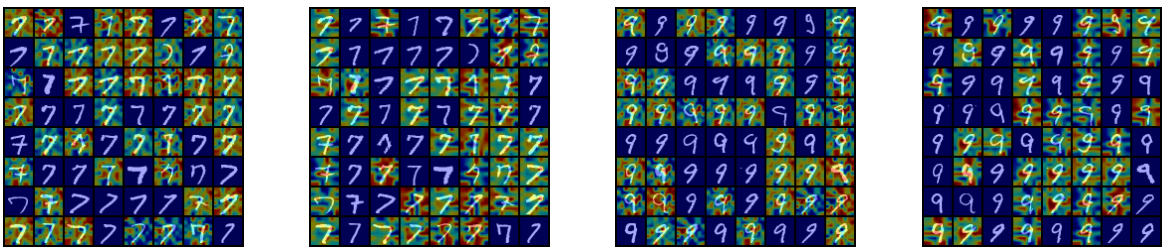
D Results for Stacked Restricted Boltzmann Machine extension

	Encoder	Tensor Shape	Decoder	Tensor Shape
Layers	Input	100×100 binary image	Input	latent variables $\in \mathbb{R}^z$
Layer 0	7×7 conv. ReLU $64, 2, 3$	$128 \times \times 128 \times 64$	Linear	8192
	BatchNorm	$128 \times \times 128 \times 64$	Interpolate	$8 \times 8 \times 512$
	ReLU	$128 \times \times 128 \times 64$		
	3×3 MaxPool $64, 2, 1$	$64 \times 64 \times 64$		
Layer 1, Encoder block 0	3×3 conv. $64, 1, 1$	$64 \times 64 \times 64$	3×3 conv. $512, 1, 1$	$8 \times 8 \times 512$
	BatchNorm	$64 \times 64 \times 64$	BatchNorm	$8 \times 8 \times 512$
	ReLU	$64 \times 64 \times 64$	Interpolate	$8 \times 8 \times 512$
	3×3 conv. $64, 1, 1$	$64 \times 64 \times 64$	3×3 conv. $256, 1, 1$	$16 \times 16 \times 256$
	Batchnorm	$64 \times 64 \times 64$	Batchnorm	$16 \times 16 \times 256$
Layer 1, Encoder block 1	3×3 conv. $64, 1, 1$	$64 \times 64 \times 64$	3×3 conv. $256, 1, 1$	$16 \times 16 \times 256$
	BatchNorm	$64 \times 64 \times 64$	BatchNorm	$16 \times 16 \times 256$
	ReLU	$64 \times 64 \times 64$	ReLU	$16 \times 16 \times 256$
	3×3 conv. $64, 1, 1$	$64 \times 64 \times 64$	3×3 conv. $256, 1, 1$	$16 \times 16 \times 256$
	Batchnorm	$64 \times 64 \times 64$	BatchNorm	$16 \times 16 \times 256$
Layer 2, Encoder 0	3×3 conv. $128, 2, 1$	$32 \times 32 \times 128$	3×3 conv. $256, 1, 1$	$16 \times 16 \times 256$
	BatchNorm	$32 \times 32 \times 128$	BatchNorm	$16 \times 16 \times 256$
	ReLU	$32 \times 32 \times 128$	ReLU	$16 \times 16 \times 256$
	3×3 conv. $128, 1, 1$	$32 \times 32 \times 128$	Interpolate	$16 \times 16 \times 256$
	Batchnorm	$32 \times 32 \times 128$	3×3 conv. $128, 1, 1$	$32 \times 32 \times 128$
Downsample	1×1 conv. $128, 2, 0$	$32 \times 32 \times 128$	BatchNorm	$16 \times 16 \times 128$
	BatchNorm	$32 \times 32 \times 128$	Interpolate	$16 \times 16 \times 128$
			3×3 conv. $128, 1, 1$	$32 \times 32 \times 128$
Layer 2, Encoder Block 1	3×3 conv. $128, 2, 1$	$32 \times 32 \times 128$	3×3 conv. $128, 1, 1$	$32 \times 32 \times 128$
	BatchNorm	$32 \times 32 \times 128$	BatchNorm	$32 \times 32 \times 128$
	ReLU	$32 \times 32 \times 128$	ReLU	$32 \times 32 \times 128$
	3×3 conv. $128, 1, 1$	$32 \times 32 \times 128$	Interpolate	$32 \times 32 \times 128$
	Batchnorm	$32 \times 32 \times 128$	3×3 conv. $128, 1, 1$	$32 \times 32 \times 128$
			BatchNorm	$32 \times 32 \times 128$
			Interpolate	$32 \times 32 \times 128$
			3×3 conv. $128, 1, 1$	$32 \times 32 \times 128$
Layer 3, Encoder Block 0	3×3 conv. $256, 2, 1$	$16 \times 16 \times 256$	3×3 conv. $128, 1, 1$	$32 \times 32 \times 128$
	BatchNorm	$16 \times 16 \times 256$	BatchNorm	$32 \times 32 \times 128$
	ReLU	$16 \times 16 \times 256$	ReLU	$32 \times 32 \times 128$
	3×3 conv. $256, 1, 1$	$32 \times 32 \times 128$	Interpolate	$32 \times 32 \times 128$
	Batchnorm	$16 \times 16 \times 256$	3×3 conv. $64, 1, 1$	$64 \times 64 \times 64$
Downsample	1×1 conv. $256, 2, 0$	$16 \times 16 \times 256$	BatchNorm	$64 \times 64 \times 64$
	BatchNorm	$16 \times 16 \times 256$		
Layer 3, Encoder Block 1	3×3 conv. $256, 2, 1$	$16 \times 16 \times 256$	3×3 conv. $64, 1, 1$	$64 \times 64 \times 64$
	BatchNorm	$16 \times 16 \times 256$	BatchNorm	$64 \times 64 \times 64$
	ReLU	$16 \times 16 \times 256$	ReLU	$64 \times 64 \times 64$
	3×3 conv. $256, 1, 1$	$16 \times 16 \times 256$	3×3 conv. $64, 1, 1$	$64 \times 64 \times 64$
	Batchnorm	$16 \times 16 \times 256$	BatchNorm	$64 \times 64 \times 64$
Layer 4, Encoder 0	3×3 conv. $512, 2, 1$	$8 \times 8 \times 512$	3×3 conv. $64, 1, 1$	$32 \times 32 \times 64$
	BatchNorm	$8 \times 8 \times 512$	BatchNorm	$32 \times 32 \times 64$
	ReLU	$8 \times 8 \times 512$	ReLU	$32 \times 32 \times 64$
	3×3 conv. $512, 1, 1$	$8 \times 8 \times 512$	Interpolate	$32 \times 32 \times 64$
	Batchnorm	$8 \times 8 \times 512$	3×3 conv. $64, 1, 1$	$32 \times 32 \times 64$
Downsample	1×1 conv. $512, 2, 0$	$8 \times 8 \times 512$	BatchNorm	$32 \times 32 \times 64$
	BatchNorm	$8 \times 8 \times 512$	Interpolate	$32 \times 32 \times 64$
			3×3 conv. $64, 1, 1$	$64 \times 64 \times 64$
			BatchNorm	$64 \times 64 \times 64$
Layer 4, Encoder 1	3×3 conv. $512, 1, 1$	$8 \times 8 \times 512$	3×3 conv. $64, 1, 1$	$64 \times 64 \times 64$
	BatchNorm	$8 \times 8 \times 512$	BatchNorm	$64 \times 64 \times 64$
	ReLU	$8 \times 8 \times 512$	ReLU	$64 \times 64 \times 64$
	3×3 conv. $512, 1, 1$	$8 \times 8 \times 512$	3×3 conv. $64, 1, 1$	$64 \times 64 \times 64$
	Batchnorm	$8 \times 8 \times 512$	BatchNorm	$128 \times 128 \times 64$

Table 4: Architecture details of the encoder and decoder used on the MVTec-AD experiments. The notation per layer is - kernel size, type of 2D convolution, activation function, output shape, stride, padding.

Encoder	Tensor Shape
Input	64×64 binary image
4×4 conv. ReLU $32, 2, 1$	$32 \times 32 \times 32$
4×4 conv. ReLU $32, 2, 1$	$16 \times 16 \times 32$
4×4 conv. ReLU $64, 2, 1$	$8 \times 8 \times 64$
4×4 conv. ReLU $64, 2, 1$	$4 \times 4 \times 64$
1×1 conv. ReLU $128, 1, 0$	$1 \times 1 \times 128$
1×1 conv. ReLU $2 * z, 1, 0$	$2 * z$
Decoder	**Tensor Shape**
Input	latent variables $\in \mathbb{R}^z$
1×1 conv. ReLU $128, 1, 0$	$1 \times 1 \times 128$
4×4 upconv. ReLU $64, 1, 0$	$4 \times 4 \times 64$
4×4 upconv. ReLU $64, 2, 1$	$8 \times 8 \times 64$
4×4 upconv. ReLU $32, 2, 1$	$16 \times 16 \times 32$
4×4 upconv. ReLU $32, 2, 1$	$32 \times 32 \times 32$
4×4 upconv. $1, 2, 1$	$64 \times 64 \times 1$

Table 5: Architecture details of the encoder and decoder used on the dsprites experiments. The notation per layer is - kernel size, type of 2D convolution, activation function, output shape, stride, padding. Moreover, upconv represents the transpose convolution.



(a) Trained on digit 1, evaluated on digit 7, with gradients and activations from layer 6

(b) Trained on digit 1, evaluated on digit 7, with gradients and activations from layer 7

(c) Trained on digit 1, evaluated on digit 9, with gradients and activations from layer 6

(d) Trained on digit 1, evaluated on digit 9, with gradients and activations from layer 7

Figure 12: 8 layered Stacked RBM results trained on MNIST digits 1, for different layers and digits