VAE model to reconstruct & generate dSprites and CelebA images

Paper followed: Auto-Encoding Variational Bayes (Authors: Kingma & Welling, arXiv:1312.6114v10)
Datasets Used: CelebA - https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html, dSprites - https://github.com/deepmind/dsprites-dataset
Framework Used: PyTorch Lightning (1.6.4), PyTorch (1.11.0), TorchVision (0.12.0)
Other Libraries: sklearn GMM, PCA

1. Code Details:

a) **dSprites_VAE**: Class that creates the VAE model for the dSprites dataset
  a. Encoder uses 5 layer convolutions to extract features which are then passed through two fully connected layers to produce the mean and the log variance parameters. Each convolution layer has PReLU as activation function and uses BatchNorm as regulariser.
  b. Decoder uses 4 layer deconvolutions to reconstruct the image. Each of them have PReLU as activation function and BatchNorm as regulariser. It additionally has a last layer of deconvolutions which uses HardSigmoid as activation and no BatchNorm layer after it as it produces the required image.
  c. The class takes as input latent_dim, which is the required dimension of the latent space, num_z, which is the number of samples to be drawn from the variational distribution and reduce_kl, which is a Boolean flag indicating whether KL loss should be included or not in the loss function. The num_z parameter is yet to be used.
  d. Reconstruction loss is calculated as the mean across samples of the L2 norm of the difference vector (difference between prediction and target).
  e. forward(): Gets the mean and logvar from the encoder. Samples values from the normal distribution. Uses the reparameterization trick to get latents. These are then passed to the decoder to generate the images.

b) **celeba_VAE**: Class that creates the VAE model for the dSprites dataset
  a. Similar architecture design as the dSprites_VAE with changes in the kernel size, strides and number of channels etc as the data is of different dimensions.

c) **celeba_VAE_MAE**: Exactly same as celeba_VAE class except it uses MAE loss (i.e., L1 norm) instead of MSE for reconstruction errors

d) **celeba_VAE_meanmse**: Exactly same as celeba_VAE class except the mean of the L2 norm of the difference vector is calculated across dimensions and across samples.

e) **Data Related Utilities**:
  a. create_dsprites_split: dSprites dataset comes as one big bundle of all images. The data was split into train, validation and test.
  b. get_dsprites_dataloaders: function which returns three dataloaders (train, validation and test) to work with PyTorch framework for dSprites dataset.
  c. get_celeba_dataloaders: function which returns three dataloaders for the CelebA dataset. We use TorchVision library for ease of interfacing with PyTorch.

f) **Training Utilities**: train_and_test: General function which allows running different configurations. It takes parameters such as batch size, max_epochs, which GPUs to use etc. Even takes the model_class and the model_kwargs to instantiate the model inside the function. It uses the Trainer class of PyTorch Lightning to train, validate and test the model on the given dataset and the model. It logs everything into a tensorboard (created inside the function), which can then be used to view loss curves.

g) **Plots and Analysis Utilities**: 4 scripts each for the two datasets are made to analyse the results

  a. disentanglement_results: View 100 images generated from the model in a 10 X 10 grid while varying the first 2 dimensions only
  b. see_some_generations: View 100 images generated randomly from the model in a 10 X 10 grid
  c. see_some_reconstructions: View 50 test images' reconstructions from the model in a 10 X 10 grid
  d. log_likelihood_estimate: Get the log likelihood estimate of all the points in the test dataset according to the method proposed in Appendix D of the VAE paper

h) All randomness is controlled by using seeds which ensures reproducibility.

2. Architecture Choices & Details:

a. PReLU was used as activation function because of the findings of this study: https://arxiv.org/abs/1502.01852v1. It is essentially LeakyReLU where the slope parameter is kept learnable. Kaiming normal initialisation was used for the layer preceding this activation function.
b. HardSigmoid was used as the activation function at the last layer for dSprites dataset, as the image lies in {0,1} space. As opposed to sigmoid, this function can output 0's and 1's. Xavier normal initialisation was used for the later preceding this activation function.
c. For deconvolution layers, the kernel size was a multiple of the stride to avoid checkerboard artifacts.
d. For choosing kernel size, stride values and number of channels, a general principle was followed. In the encoder part, kernel size, stride values and number of channels were increased from layer to the next. In the decoder part, exact opposite trend was followed. This was done while trying to keep the number of learnable parameters at a low value.
e. Each epoch in the model training was taking a lot of time. Due to computational restrictions, the max_epochs was set to 50. Hence, the results presented here should be taken with a grain of salt.

3. Experiments, Results & Inferences:

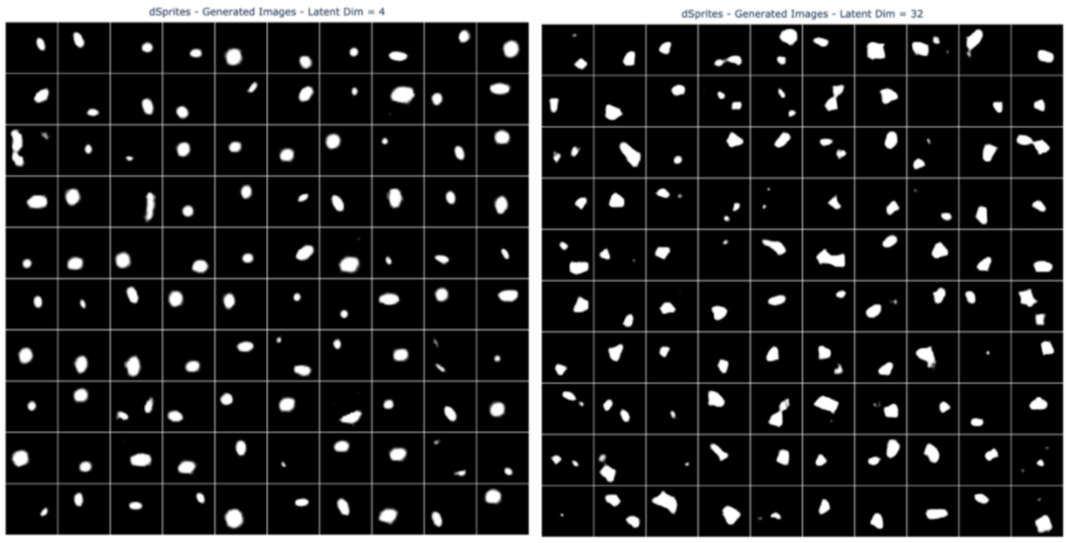
The model has many hyperparameters worthy of tuning. Hyperparameters such as number of CNN layers, kernel size etc which are related to CNNs were fixed for the assignment. We want to understand the effects of hyperparameters introduced in the VAE model. In that line of thought, two hyperparameters were explored: number of latent dimensions and effect of KL loss. The batch size was kept at 256 and so, the number of samples from the posterior was fixed at 1 (as per the advice in the VAE paper). Only some of the results are mentioned here to keep the report concise.

dSprites Results

Figure 1 shows the generation results for latent dimensions of 4 and 32. Inference: as the number of latent dimensions increase,

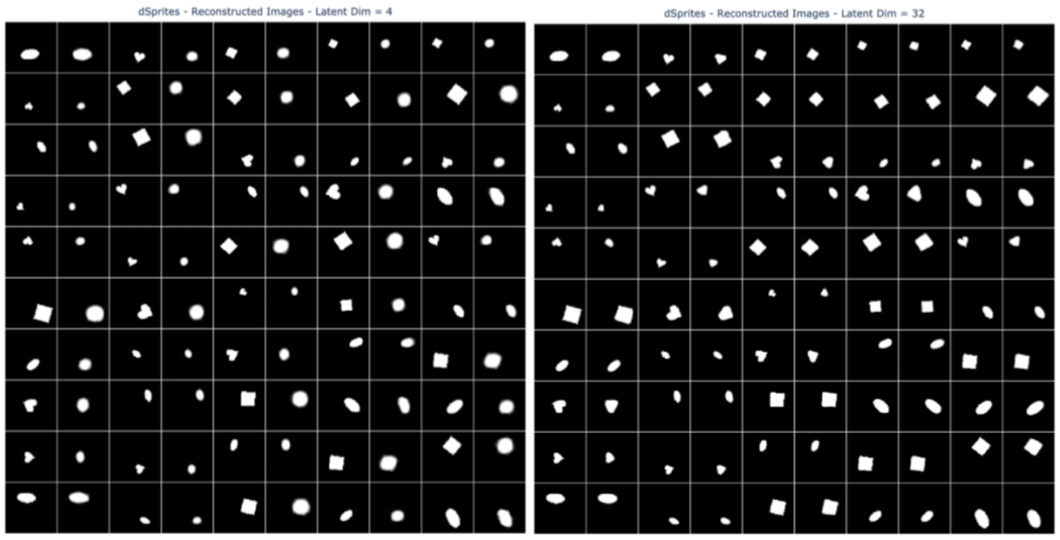
1. Model is able to generate complex shapes
2. The generations become fragmented – not a contiguous blob anymore

Figure 2 shows the reconstruction results for latent dimensions of 4 and 32. Inference: as the number of latent dimensions increase, model is able to better reconstruct. The model with 4 latent dimensions majorly produces elliptic blobs. One striking thing is that the location & size of the blob matches input image! This shows the model is clearly learning the right things. The model with 32 latent dimensions is much better at reconstructions. This improvement is gradual from 4 to 8 to 16 to 32. Basically, the model has more dimensions to learn and is able to capture more variations.


dSprites - Generated Images - Latent Dim = 4
dSprites - Generated Images - Latent Dim = 32

(a) (b)

Fig 1: 100 Randomly generated images in a 10 X 10 grid for dSprites dataset for two latent dimension variations (a) 4 (b) 32

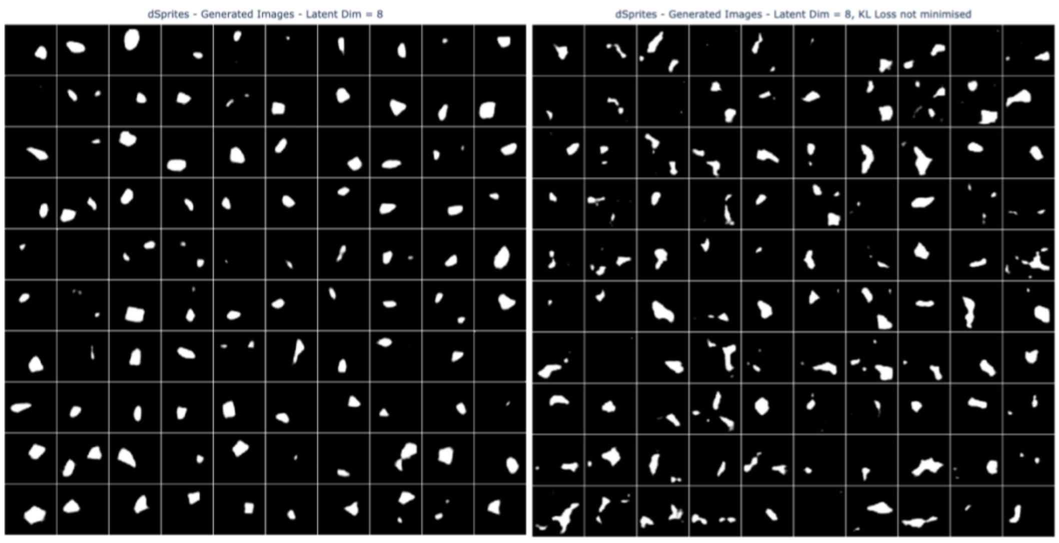

dSprites - Reconstructed Images - Latent Dim = 4
dSprites - Reconstructed Images - Latent Dim = 32

(a) (b)

Fig 2: 50 test image reconstructions for dSprites dataset for latent dimension variations (a) 4 (b) 32. Odd column number is the true image and the corresponding even number column has the reconstructed image. Same test images are used across all for fair comparison.

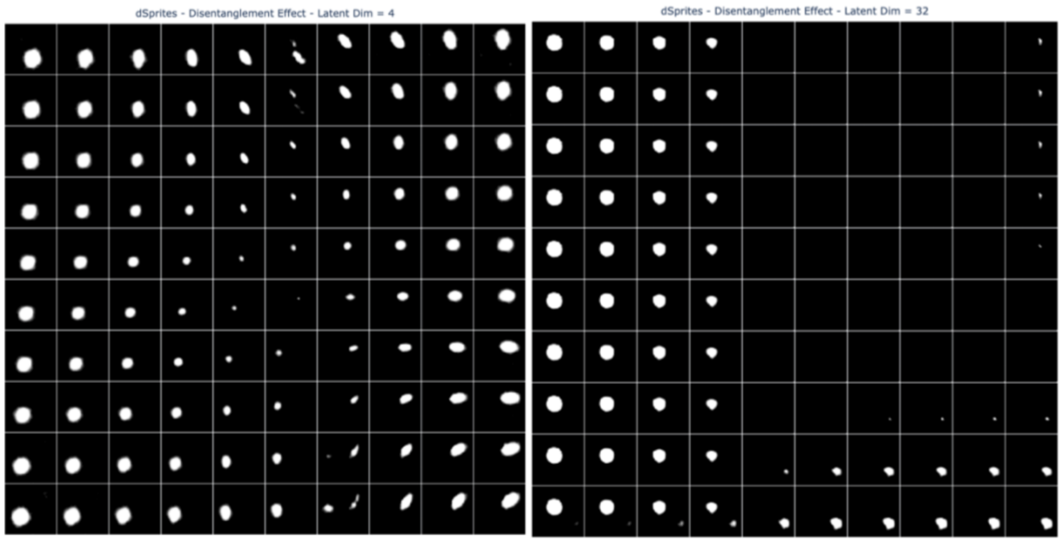
Figure 3 shows the generation results for latent dimensions of 8 with and without minimising KL Loss. Inference: When KL loss is not minimised, the generations are much more fragmented i.e., bad. This is in line with our expectations.

Figure 4 shows some generations when the input latent was varied systematically i.e., first 2 dimensions were varied. Model with 4 latent dimensions shows very clean disentanglement: The x-position of the blob varies from top to down. The y-position and the size of the blob varies from left to right. This is not very clear for the model with 32 latent dimensions. Inference: Very clean expected results on disentanglement. Right way will be to look at some disentanglement metric. Figure 5 shows the loss curves for the models.


dSprites - Generated Images - Latent Dim = 8
dSprites - Generated Images - Latent Dim = 8, KL Loss not minimised

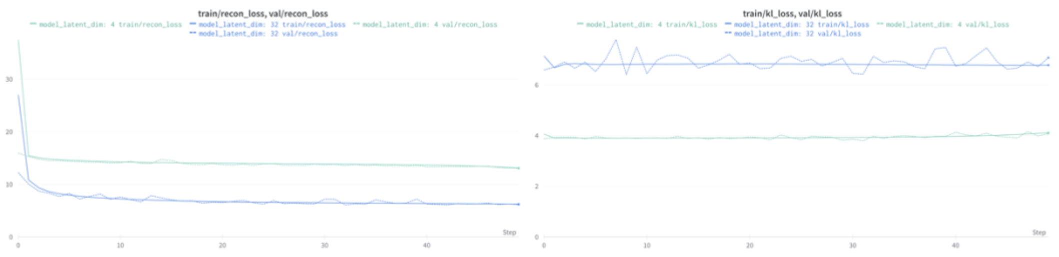
(a) (b)

Fig 3: 100 Randomly generated images in a 10 X 10 grid for dSprites dataset for latent dimension = 8 with (a) and without minimising KL Loss (b)


dSprites - Disentanglement Effect - Latent Dim = 4
dSprites - Disentanglement Effect - Latent Dim = 32

(a) (b)

Fig 4: 100 Generated images in a 10 X 10 grid for dSprites dataset for two latent dimension variations (a) 4 (b) 32 with systematic variations to the first 2 dimensions.



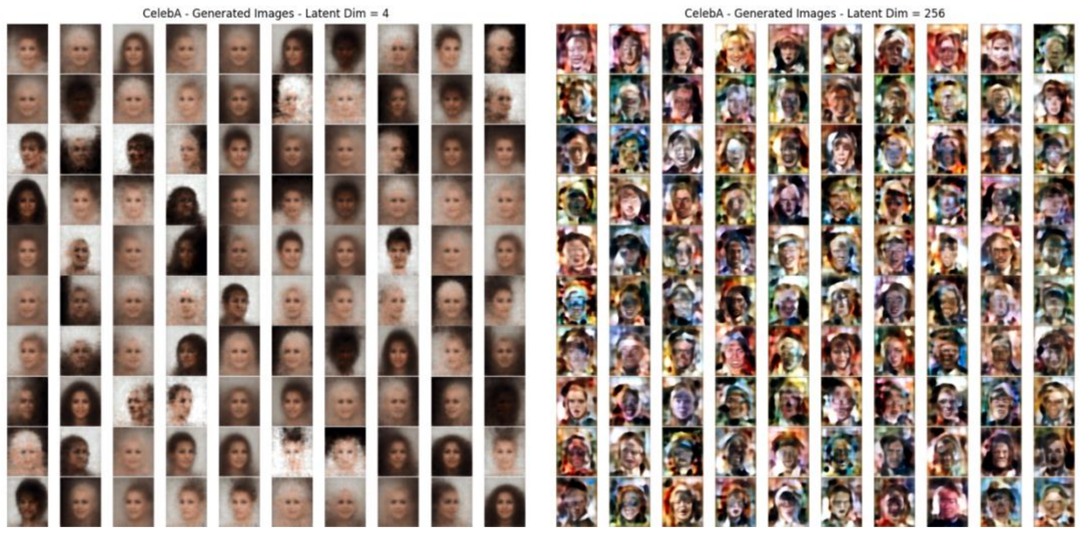
(a) (b)

Fig 5: Loss Curves for different dSprites model dimensions (a) Reconstruction loss (b) KL Loss

CelebA Results

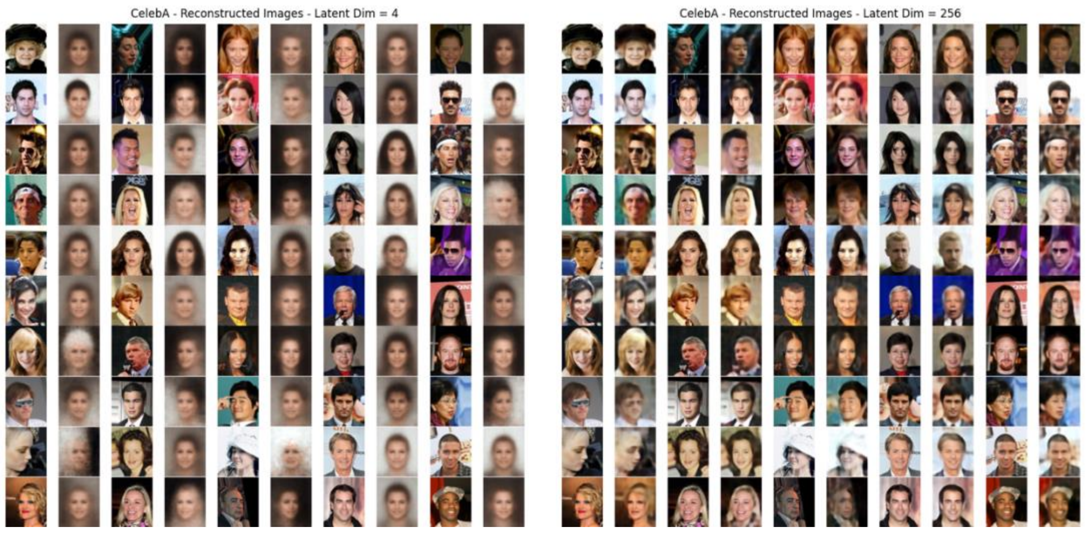
Figure 6 shows the generation results for latent dimensions of 4 and 256. Inference: as the number of latent dimensions increase, model is able to "more" realistic image.

Figure 7 shows the reconstruction results for latent dimensions of 4 and 256. Inference: as the number of latent dimensions increase, model is able to better reconstruct. The model with 4 latent dimensions majorly produces similar images. It is able to capture the face orientation and the background. The model with 256 latent dimensions is much better at reconstructions. This improvement is gradual from 4 to 32 to 64 to 128 to 256. Basically, the model has more dimensions to learn and is able to capture more variations.


CelebA - Generated Images - Latent Dim = 4
CelebA - Generated Images - Latent Dim = 256

(a) (b)

Fig 6: 100 Randomly generated images in a 10 X 10 grid for CelebA dataset for two latent dimension variations (a) 4 (b) 256

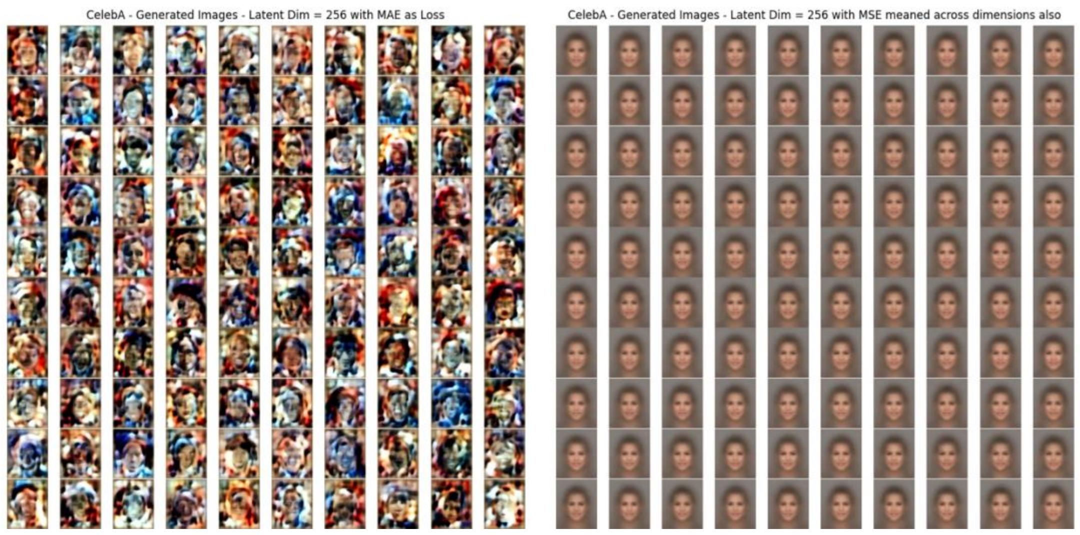

CelebA - Reconstructed Images - Latent Dim = 4
CelebA - Reconstructed Images - Latent Dim = 256

(a) (b)

Fig 7: 50 test image reconstructions for CelebA dataset for latent dimension variations (a) 4 (b) 256. Odd column number is the true image and the corresponding even number column has the reconstructed image. Same test images are used across all for fair comparison.

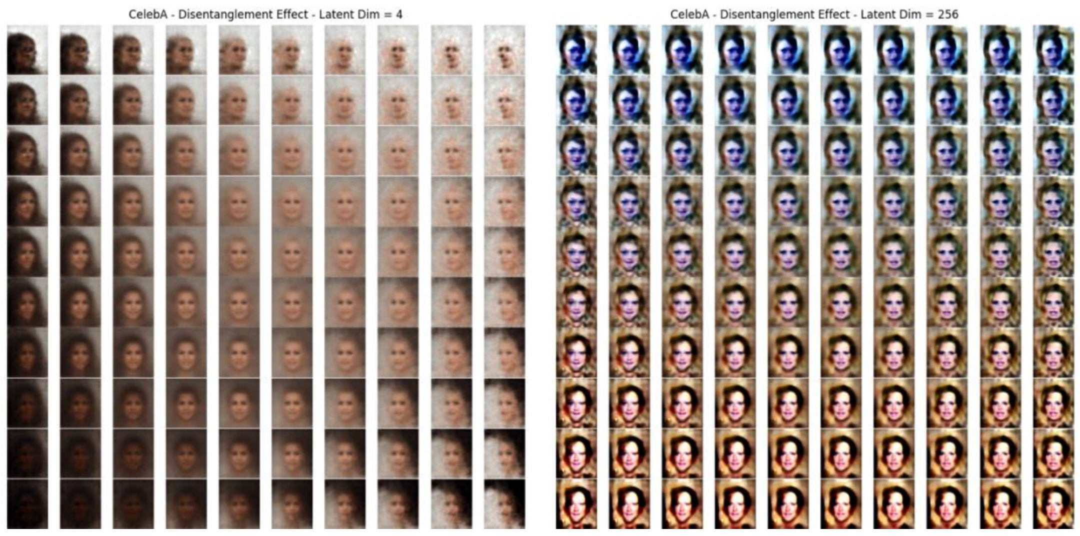
Figure 8 shows the generation results for latent dimensions of 256 with some loss variations. Inference: When MAE loss is used the reconstructions are more blurry, which was not expected. When MSE loss is meaned across dimensions and samples, its value becomes very low compared to KL loss and thus the model just minimises the KL loss causing mode collapse.

Figure 9 shows some generations when the input latent was varied systematically i.e., first 2 dimensions were varied. Both the models show very clean disentanglement. The model with 4 latent dimensions shows face orientation variation and background variation. The model with 32 latent dimensions shows variation in face colour and lips closed to open. Inference: Very clean expected results on disentanglement. Right way will be to look at some disentanglement metric. Figure 10 shows the loss curves for the models.


CelebA - Generated Images - Latent Dim = 256 with MAE as Loss
CelebA - Generated Images - Latent Dim = 256 with MSE meaned across dimensions also

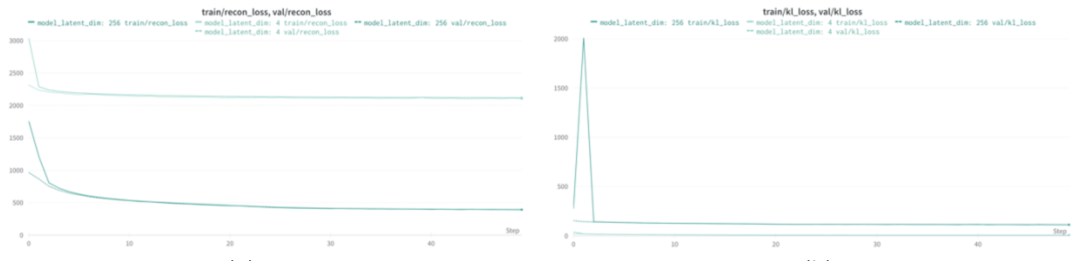
(a) (b)

Fig 8: 100 Randomly generated images in a 10 X 10 grid for CelebA dataset for latent dimension = 256 (a) with MAE as loss and (b) MSE meaned across dimensions and samples


CelebA - Disentanglement Effect - Latent Dim = 4
CelebA - Disentanglement Effect - Latent Dim = 256

(a) (b)

Fig 9: 100 Generated images in a 10 X 10 grid for CelebA dataset for two latent dimension variations (a) 4 (b) 256 with systematic variations to the first 2 dimensions.



(a) (b)

Fig 10: Loss Curves for different CelebA model dimensions (a) Reconstruction loss (b) KL Loss

4. Problems faced, solutions used:

a) While working on the CelebA dataset, the output of the logvar layer sometimes would give numbers like 102, which when passed through exp would result in overflow and infinity as the loss. This causes all the gradients to become nans and all subsequent runs to produce nans as outputs. For this reason, the logvar layer output was clamped between -16 and 16. This helped stabilise the training!

b) When calculating the marginal likelihood, high step-size for the gradient ascent caused the calculations to have a very large value. I chose a low step-size like 0.001. And even then sometimes the calculations gave inf. I thought it could be because we are dealing with very small values. So I used the log scale and then convert to likelihood, but to no avail. The final numbers were reported in log scale so as to allow for comparisons.

Conclusion & Future Work:

This assignment helped us understand the VAE model better, that too in conjunction with CNNs. More variations can be tried with different loss functions (like BCE) and more powerful CNN architectures like residual connections etc and explore their effects