## Variational Autoencoder

While traditional autoencoders can be used as compressors or denoisers in many applications, we cannot generate new images from them. This is because the learnt latent space has no structure and no constraint to follow a specific probability distribution. So, Variational Autoencoder (VAE) was introduced to allow us generate new images from the latent space, by making the neural network tries to learn an isotropic normal distribution for each image in the latent space instead of learning just an encoding, this is done by making the network learn a mean and a log variance for each image and by introducing a constraint which is the Kullback–Leibler divergence (KL divergence) on this distribution to make it close to a normal distribution. Therefore, to generate a new image, we just sample from a normal distribution and pass this sample point to the VAE decoder.

In an autoencoder, each image is mapped directly to one point in the latent space. In a variational autoencoder, each image is instead mapped to a multivariate normal distribution around a point in the latent space, as shown in Figure 3-10.

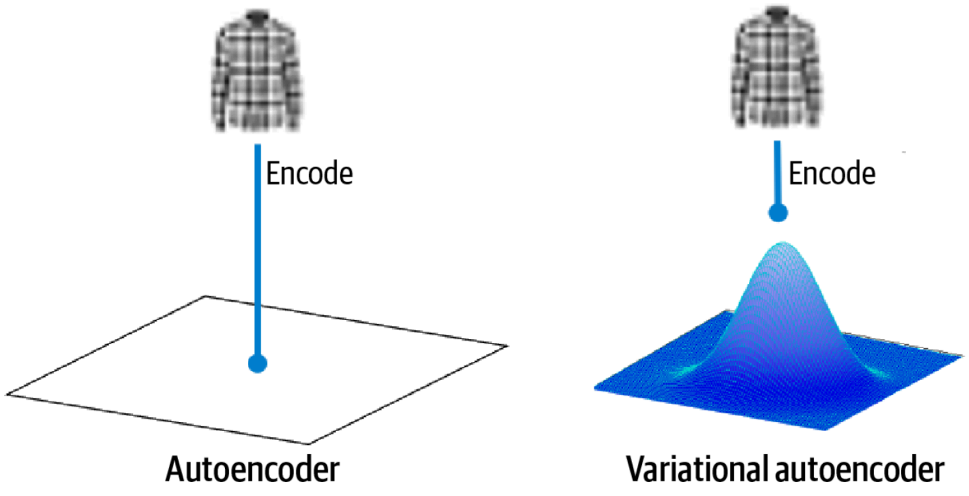


*Figure 3-10. The difference between the encoders in an autoencoder and a variational autoencoder*

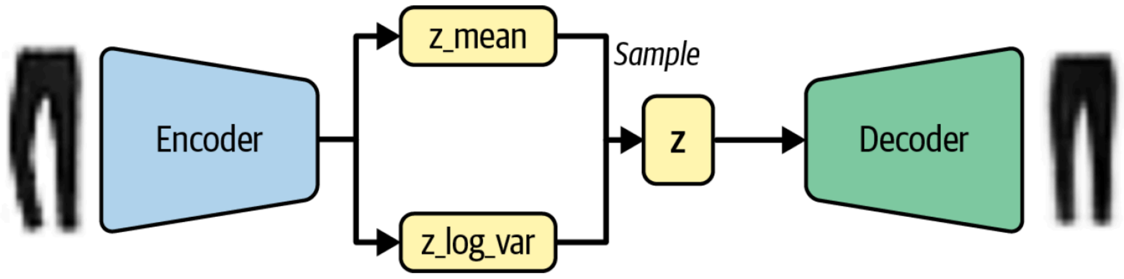


*Figure 3-12. VAE architecture diagram*

## Hyperparameters

**Batch size**

Tried vanella VAE with 128, 256, and 1024 batch sizes for one epoch and observed the loss, it was better with 128 batch size. And then added conceptual loss with VGG19, and as it consumed more VRAM, I had to lower the batch size to 64.

**Learning rate**

Tried many numbers for one epoch and chose 0.05 with step scheduler with steps size 1 and gamma 0.1.

**Number of epochs**

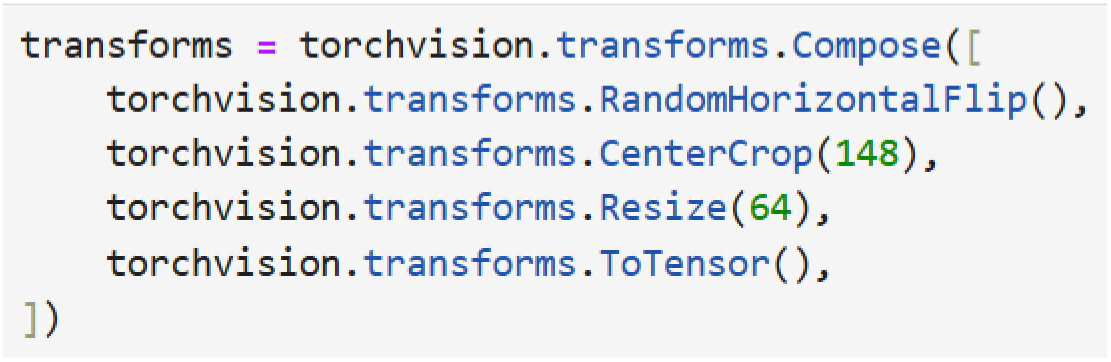
Tried many numbers at the beginning and the loss was stabilized after 6 epochs so I chose 10.

## Data Preprocessing

Used CelebA database which consists of 202,599 number of face images and 40 binary attributes annotations per image. These attributes will be used to manipulate the latent space, for example, will use the “smiling” attribute to get all images with smile and calculate the smiling vector the latent space and use it to add or subtract a smile from the image.

**Transformers**

Original resolution is 178x218, so I took a center crop of 148 to crop only around the faces and discard the background, and then resized them to 64x64.
148 was the standard preprocessing from open sources implementations and 64 to be able to try many experiments. Using 128x128 resolution will give better results but it also will take a long time for each epoch.
And used RandomHorizontalFlip as an augmentation.



```
transforms = torchvision.transforms.Compose([
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.CenterCrop(148),
    torchvision.transforms.Resize(64),
    torchvision.transforms.ToTensor(),
])
```

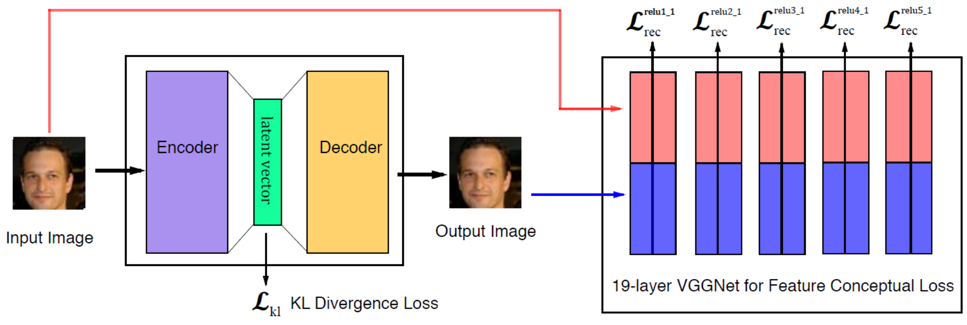
## Model architecture

I've used the traditional VAE architecture which consists of an encoder with five hidden layers with the following channel sizes: [32, 64, 128, 256, 512] to transform the input image from 3x64x64 to 512x2x2 feature maps. And then flattened the feature map to 2048 and used two fully connected layers to produce the mean and log variance for the learnt distribution with latent dimension of 128 neurons. And then used a decoder with transposed convolution to bring back the image to the original shape 3x64x64.
Additionally, I used a pretrained VGG19 network to calculate some feature maps for the original image and the constructed images, to calculate a perception loss.

## Loss function

1- Reconstruction loss between the original image and the reconstructed image
2- KL-divergence loss to let the encoding in the latent space follow a normal distribution which makes the space continuous and can be used to sample and generate new images.
3- Conceptual loss of five feature maps for the original image and the reconstructed one, resulted from passing the images to a pretrained VGG19 network.

This architecture and loss follows this paper: "Deep Feature Consistent Variational Autoencoder", where the authors used a traditional VAE network with the conceptual loss to measure the similarity between the input and generated images instead of pixel-by-pixel loss.



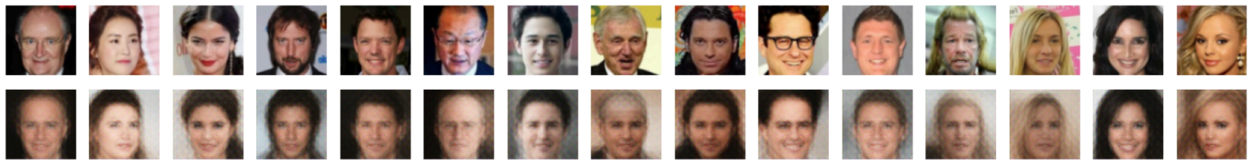
$\mathcal{L}_{rec}^{relu1\_1}$
$\mathcal{L}_{rec}^{relu2\_1}$
$\mathcal{L}_{rec}^{relu3\_1}$
$\mathcal{L}_{rec}^{relu4\_1}$
$\mathcal{L}_{rec}^{relu5\_1}$
Encoder
latent vector
Decoder
Input Image
Output Image
$\mathcal{L}_{kl}$ KL Divergence Loss
19-layer VGGNet for Feature Conceptual Loss

Figure 1. Model Overview. The left is a deep CNN-based Variational Autoencoder, and the right is a pretrained deep CNN used to compute feature perceptual loss.

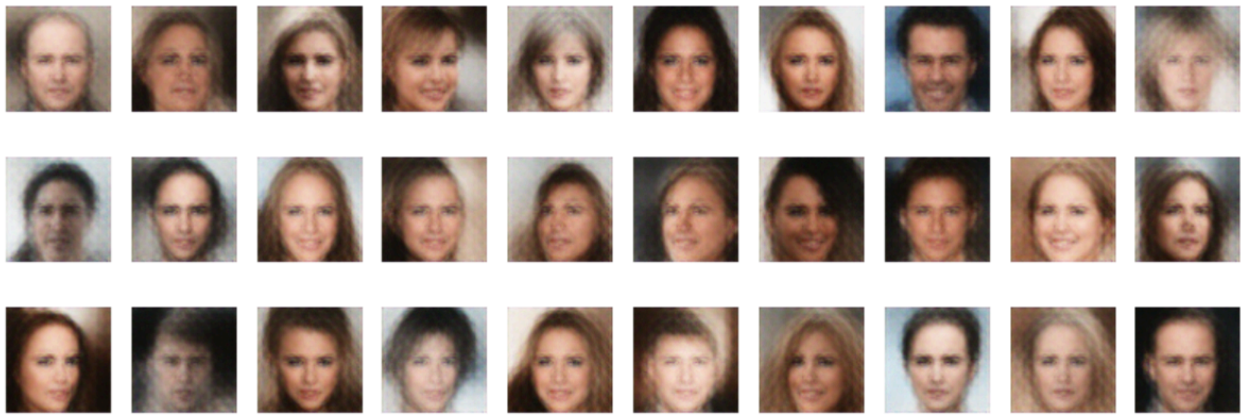
```
for epoch in range(num_epochs):
    model.train()

    for batch_idx, (x, _) in enumerate(train_loader):
        x = x.to(device)
        encoded, recons, x_features, recons_features, mu, log_var = model(x)
        recons_loss = loss_fn(recons, x)

        feature_loss = 0.0

        for (r, i) in zip(recons_features, x_features):
            feature_loss += loss_fn(r, i)

        kld_loss = torch.mean(-0.5 * torch.sum(1 + log_var - mu ** 2 - log_var.exp(), dim = 1), dim = 0)

        loss = 0.5 * rec_weight * (recons_loss + feature_loss) + kl_weight * kld_loss

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

## Result

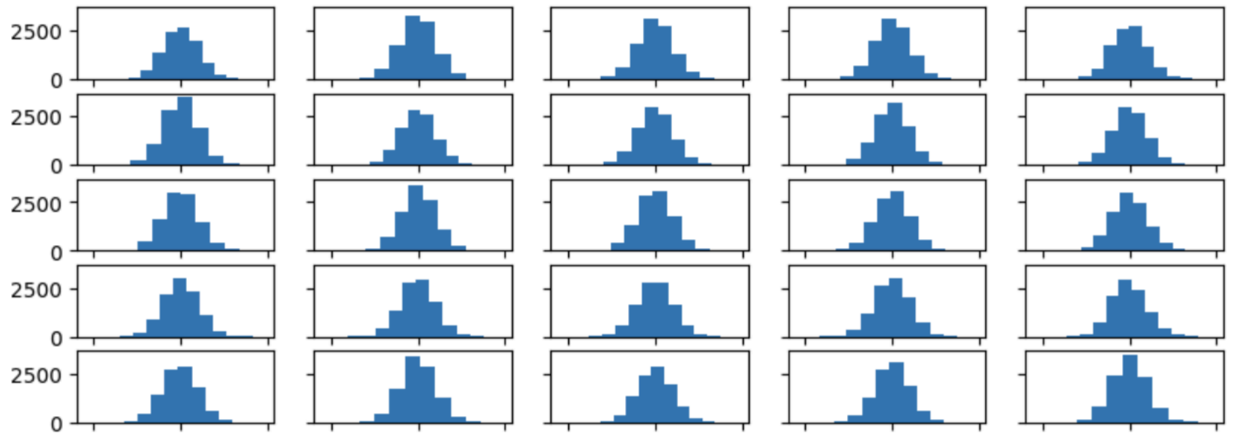
### Reconstructed images



### Generated images

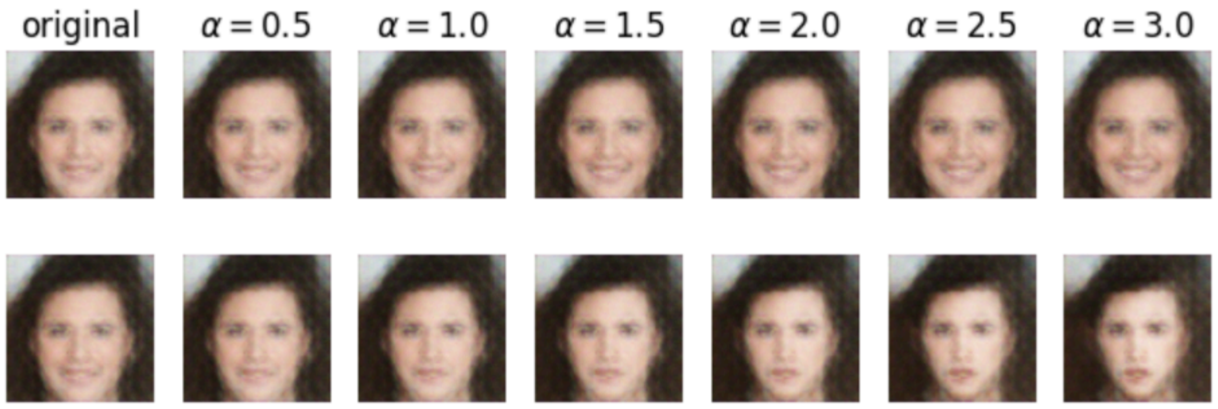


**Encoding distribution**

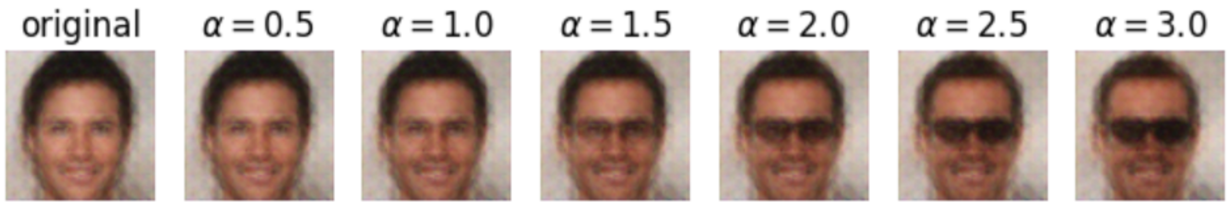


**Manipulating the latent space**

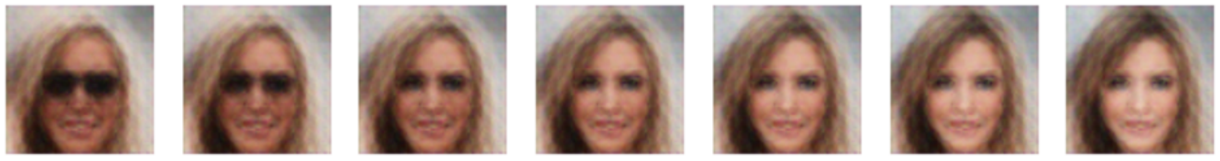
Adding and removing a smile



Adding eyeglasses



Removing eyeglasses



## Resources

[1] Generated Deep Learning book.
https://www.oreilly.com/library/view/generative-deep-learning/9781098134174/
[2] Deep Feature Consistent Variational Autoencoder Paper
https://arxiv.org/abs/1610.00291
[3] PyTorch-VAE repo
https://github.com/AntixK/PyTorch-VAE