Generate Anime Faces using Variational Autoencoders (VAEs)

Variational Autoencoders (VAEs) are a type of generative model particularly suited for image editing through concept vectors. They are an evolution of autoencoders, which are designed to compress input data into a lower-dimensional latent space and then reconstruct it. The unique feature of VAEs is that they regularize the distribution of their encodings during training. This ensures that their latent space is structured in a manner that facilitates the generation of new samples.

As a specialist in deep learning, you are tasked with crafting a **Python** program leveraging the **TensorFlow library**. Your goal is to construct a **Variational Autoencoders (VAEs) model** in line with the subsequent criteria:

- **Load and Split Dataset**
    - **Load Dataset**
      In this section, you are required to:
        - **Assign** the dataset path to the previously declared PATH variable.
        - **Load** the image from the given dataset into an array using RGB as the color mode.
        - **Randomly** shuffle the dataset.
    - **Split Dataset**
      You are tasked to **split** the given dataset into 3 main parts which are training, validation, and testing sets with the proportion of 80/10/10.
- **Data Preprocessing**
  You are tasked with **reshaping** the given dataset, which includes the training, validation, and testing sets, by converting the data type to `float32`. Following this, you will need to **normalize** the dataset, comprising the training, validation, and testing sets, by dividing the values of all image pixels by `255`.

- **Variational Autoencoders Architecture (Model Architecture Visualization, Initialization, and Configuration)**
    - **Model Visualization**

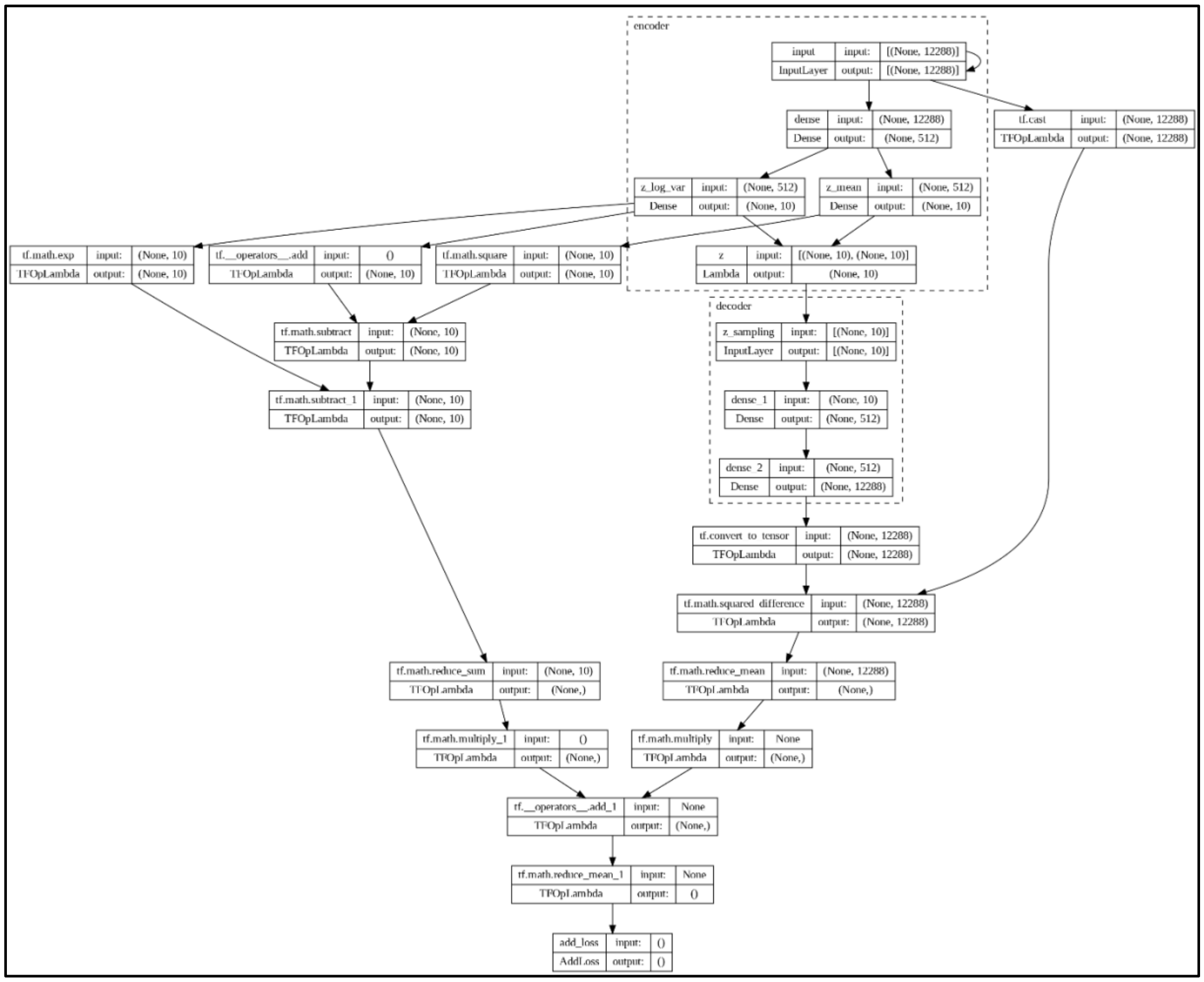

encoder
input | input: [(None, 12288)]
InputLayer | output: [(None, 12288)]
dense | input: (None, 12288)
Dense | output: (None, 512)
tf.cast | input: (None, 12288)
TFOpLambda | output: (None, 12288)
z_log_var | input: (None, 512)
Dense | output: (None, 10)
z_mean | input: (None, 512)
Dense | output: (None, 10)
tf.math.exp | input: (None, 10)
TFOpLambda | output: (None, 10)
tf.__operators__.add | input: ()
TFOpLambda | output: (None, 10)
tf.math.square | input: (None, 10)
TFOpLambda | output: (None, 10)
z | input: [(None, 10), (None, 10)]
Lambda | output: (None, 10)
decoder
tf.math.subtract | input: (None, 10)
TFOpLambda | output: (None, 10)
z_sampling | input: [(None, 10)]
InputLayer | output: [(None, 10)]
tf.math.subtract_1 | input: (None, 10)
TFOpLambda | output: (None, 10)
dense_1 | input: (None, 10)
Dense | output: (None, 512)
dense_2 | input: (None, 512)
Dense | output: (None, 12288)
tf.convert_to_tensor | input: (None, 12288)
TFOpLambda | output: (None, 12288)
tf.math.squared_difference | input: (None, 12288)
TFOpLambda | output: (None, 12288)
tf.math.reduce_sum | input: (None, 10)
TFOpLambda | output: (None,)
tf.math.reduce_mean | input: (None, 12288)
TFOpLambda | output: (None,)
tf.math.multiply_1 | input: ()
TFOpLambda | output: (None,)
tf.math.multiply | input: None
TFOpLambda | output: (None,)
tf.__operators__.add_1 | input: None
TFOpLambda | output: (None,)
tf.math.reduce_mean_1 | input: None
TFOpLambda | output: ()
add_loss | input: ()
AddLoss | output: ()

Figure 1. Model Visualization

- **Model Initialization and Configuration – Encoder**

  Input Layer: This takes the data (like an image) as input.

  Hidden Layers: One or more layers that transform the input data. These can be fully connected layers, convolutional layers, or any other type of layer suitable for the data type.

  Output Layer: Produces two vectors for each input data point: a mean vector (μ) and a log variance vector $(\log(\sigma^2))$.

These vectors are used to parameterize the Gaussian distribution from which we'll sample the latent variable 'z'.

- **Model Initialization and Configuration – Decoder**

  This is a lower-dimensional space where we encode our data. A point 'z' in this space is sampled from the Gaussian distribution parameterized by μ and $\log(\sigma^2)$ from the encoder. The **reparameterization trick** is used here to make this sampling process differentiable. Instead of directly sampling from the distribution, we sample from a standard normal distribution and then scale and shift using μ and σ. The **formula** for the **reparameterization trick** is $z = \mu + \sigma \odot \epsilon$, where μ is the mean vector produced by the encoder, σ is the standard deviation, and ϵ is a random noise sampled from a standard normal distribution.

- **Model Initialization and Configuration – Decoder**

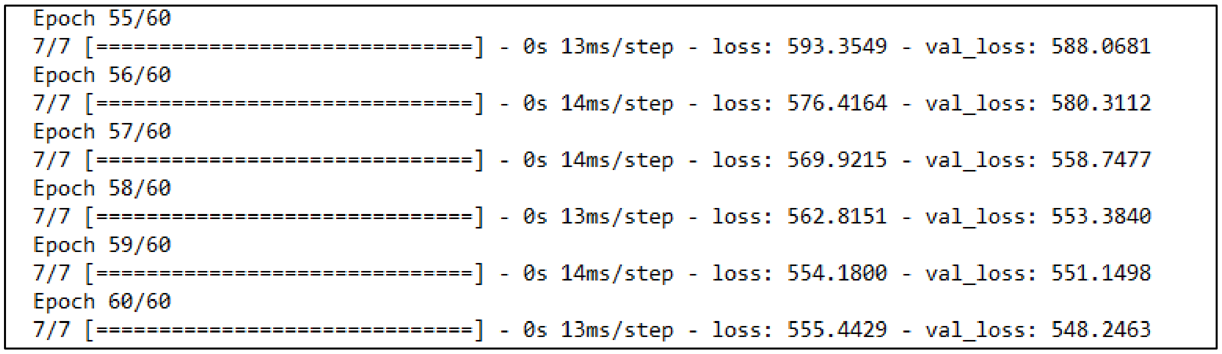
  Input Layer: Takes the sampled 'z' from the latent space.

  Hidden Layers: One or more layers that transform the latent variable back towards the original data's dimensionality.

  Output Layer: Produces the reconstructed data, ideally as close to the original input data as possible.

- **Model Training**

  You are tasked with **training the model** that you previously constructed. **Utilize** the Adam optimizer and **select** the appropriate loss function based on the specifics of your case. **Monitor** the performance using **reconstruction loss** and **KL Divergence** as the metric. Ensure that the training process spans a minimum of 60 epochs. You have the flexibility to set the batch size and learning rate for the optimizer according to your judgment.



```
Epoch 55/60
7/7 [==============================] - 0s 13ms/step - loss: 593.3549 - val_loss: 588.0681
Epoch 56/60
7/7 [==============================] - 0s 14ms/step - loss: 576.4164 - val_loss: 580.3112
Epoch 57/60
7/7 [==============================] - 0s 14ms/step - loss: 569.9215 - val_loss: 558.7477
Epoch 58/60
7/7 [==============================] - 0s 13ms/step - loss: 562.8151 - val_loss: 553.3840
Epoch 59/60
7/7 [==============================] - 0s 14ms/step - loss: 554.1800 - val_loss: 551.1498
Epoch 60/60
7/7 [==============================] - 0s 13ms/step - loss: 555.4429 - val_loss: 548.2463
```

Figure 2. Model Training

- **Predicting Data Using Created Model and Model Evaluation**

  You are tasked with **evaluating** the testing sets using the model that you previously constructed. **Display** both the **original and the reconstructed images**, **showcasing 10 images each**.

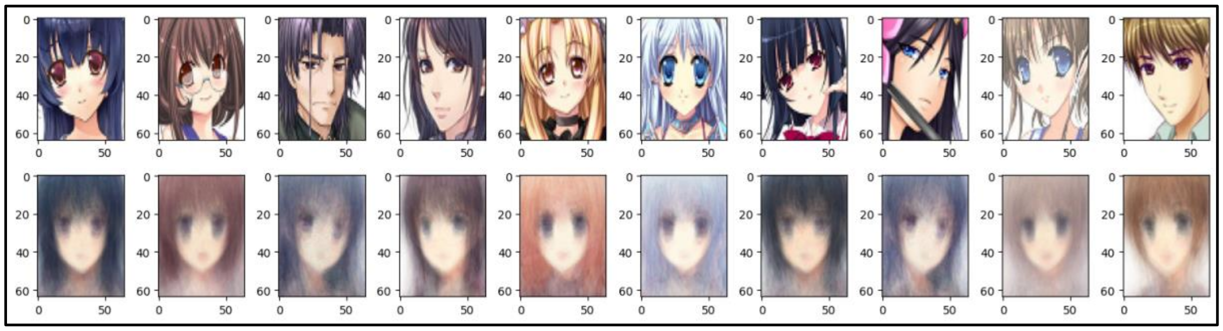


Figure 3. Original Images and Reconstructed Images