Avantari AutoEncoder Task

The problem statement was to identify the images of similar type from the given dataset.

I make use of the Pytorch library for building the model. Basically since we are building a custom model optimizing is much more preferable in Pytorch and more it gives better performance as the processing of images requires high computational power and this has better training duration. To compute complex chained derivatives, TensorFlow uses a Define and Run paradigm, whereas PyTorch uses a more ingenuous Define by Run paradigm. This Define by Run paradigm also has many advantages other than just creating cleaner and simpler code. Debugging also becomes extremely easy and all of the tools that you currently use to debug Python code can be used with PyTorch as well.

The model that I have taken into consideration is the Variational Autoencoder model. It is basically a type of generative model that has an autoencoder architecture. It consists of an Encoder-Decoder system. The encoder takes the input samples and converts this information into some vector form. These vectors are then taken by the Decoder and it expands them out to reconstruct the input sample. What we are doing wrt. the problem statement is basically Image retrieval, hence why we are using VAE here. Variational autoencoders give the proper internal features that an human also can not recognize thereby helping in better image retrieval.

Steps Followed:

- I make use of Google Colab for performing the said task at hand.
- Once having uploaded the dataset and mounted the files, we start with importing all the necessary packages.
- The images are unzipped and a function is defined for preprocessing and reading the images. For the sake of reducing the computation I have the resized the images into 255*255 format and then these are stored in rgb format into an array.
- I have written a .py file which imports the ‘Dataset’ package from torch.utils.data and have created an Imageset class. The function of this class is to take the input array of images and them convert or transform them into tensors.
- The images are therefore transformed into tensors of batch size of 7 in my case.\
- The ‘DataLoader’ is then used which iterate through the data. The shape after transformation is in the form of say, [e,g (3,255,255)] while actually it should be the other way around and hence to set this properly, the ‘shuffle’ parameter is set to False.

Building the Model and Training

- We write the structure for the VAE within ‘VAE.py’ file in our case. The encoder is a sequential model where we are applying a ‘conv 2D’ network which takes in the parameters of the image channels, number of layers, kernel_size and the stride.
- The decoder is a Converse Transpose 2D network that has a similar structure and the layers of the encoder model except in the final layer it makes use of the sigmoid function.
- The loss function is also defined which takes in the reconstructed image, original image , mean of distribution , log variable(similar to std. deviation) and beta value as the parameters. We are calculating the ‘binary_cross_entropy’ , ‘KL-Divergence’ and also their sum to determine the loss.

- The model is then specified by the ‘vae.VAE’ where the parameters are image_dim, image_channel, z_dim (bottleneck latent representations) and also the device, in our case cuda cpu or gpu.
- The optimizer used is adam with a learning rate of 0.001
- Training is dione on the model for 100 epochs with a batch size of 32.
- Finally in order for prediction we convert the output back to cuda image or to its tensor form. Thus the reconstructed and the original image are compared and concatenated by torch.cat method.
- The result is then displayed by using IPython library where the Dataloader is used for iterating within each batch sizes, which there are totally 7 of.