Name: Sujay Kumar Ingle (D23CSA003),
Manaswi Vichare (M23CSA013),
Pravin Kumar (M23CSA019)
Topic: Deep Learning

ASSIGNMENT 5

Colab Link: M23CSA013_D23CSA003_M23CSA019
Wandb Link: https://wandb.ai/m23csa013/Assignmet-5/runs/x72779a0?nw=nwuserm23csa013
Objective:
1. Train a Vector-Quantized Variational Autoencoder (VQ-VAE) on the skin lesion dataset to efficiently encode and decode high-dimensional image data while capturing meaningful latent representations.
2. Train an Auto-Regressive Model of your choice to generate new, realistic images based on the learned latent space representations.

I. Pre-process the input data by applying normalization and four different types of relevant data transformations of your choice:
The transformations used on the input data are:
i) Resize: Resize the image to a specified size (128x128).
ii) Normalize: Normalize the image with specified mean and standard deviation values.

Using the ColorJitter transformation we have adjusted:
i) Brightness: Adjusts the brightness of the image randomly. A positive value increases brightness, while a negative value decreases it.
ii) Contrast: Randomly adjusts the contrast of the image. It affects the difference in luminance or color that makes an object distinguishable.
iii) Saturation: Randomly adjusts the saturation of the image. Saturation refers to the intensity of colors in the image. Increasing saturation makes colors more vivid while decreasing saturation makes colors more muted.
iv) Hue: Randomly adjusts the hue of the image. Hue refers to the attribute of color that allows us to classify it as red, blue, yellow, etc. Adjusting hue changes the overall tone of the image by shifting the colors along the color spectrum.

A custom dataset class named SketchDataset is defined. It initializes with a data folder path and an optional transformation function. Upon instantiation, it loads the image filenames from the provided data folder. In the __getitem__ method, it loads an image based on the index provided, converts it to RGB mode using PIL, and applies the specified transformation if provided. Finally, it returns the transformed image.

We calculate the variance of pixel values across all images in the train_dataset. It collects all pixel values from the images, computes their variance, and stores the result in data_variance.

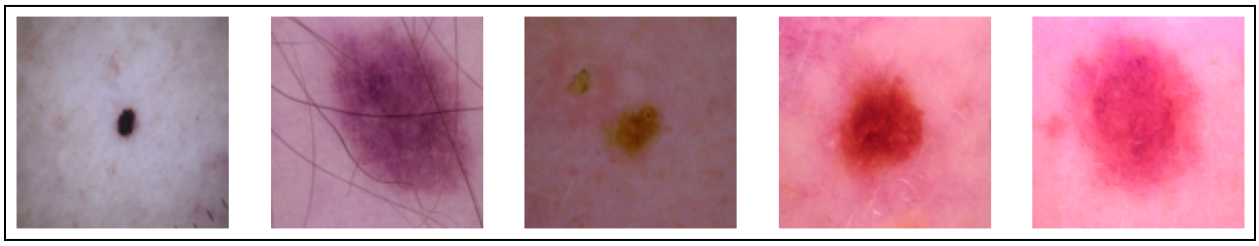


Fig 1(a): Visualizing the dataset

II. PHASE 1:

Design a VQ-VAE network using CNN encoder and decoder modules to learn a quantized latent space using a codebook:

1. Vector Quantizer:

The Vector Quantizer module compresses input vectors by finding their closest match in a codebook of embedding vectors. It minimizes the difference between input and quantized vectors using quantization and commitment losses to learn a concise representation.

Initially, input tensors are reshaped and flattened for efficient distance computation. Then, it calculates distances to find the closest embedding vector index for each input, encodes them into one-hot representations, and quantizes based on these encodings.

The module computes losses, updates the quantized tensor, and calculates perplexity to measure embedding diversity. Finally, it returns the quantized tensor, loss, perplexity, and encoding probabilities.

2. Encoder:

The Encoder module is designed for feature extraction in a neural network architecture. It consists of three convolutional layers, each followed by a Rectified Linear Unit (ReLU) activation function. The first two convolutional layers reduce the spatial dimensions of the input while increasing the number of channels progressively, and the third layer maintains spatial dimensions while preserving the number of channels. During the forward pass, input tensors undergo successive convolutions and activations, ultimately producing feature maps that capture hierarchical representations of the input data, which can be further utilized for downstream tasks such as classification or reconstruction.

3. Decoder:

The Decoder module is designed to reconstruct an image from learned features. It consists of two transposed convolutional layers followed by a regular convolutional layer. The first convolutional layer increases the number of channels while preserving spatial dimensions, and

the subsequent transpose convolutional layers gradually upsample the feature maps to the desired output size. During the forward pass, input tensors undergo convolutional and transpose convolutional operations, followed by ReLU activation functions where applicable, ultimately producing a reconstructed image with the desired number of channels and spatial dimensions.

4. VQ-VAE Architecture:

The Model module integrates an autoencoder architecture with a Vector Quantization Variational Autoencoder (VQ-VAE). It begins with an encoder that processes input images to extract features, followed by a 1x1 convolutional layer that prepares these features for quantization. The quantization is performed by the Vector Quantizer module, which compresses the feature maps into discrete latent representations. The decoder then takes these quantized representations and reconstructs the original images. During the forward pass, input images are encoded, pre-processed, quantized, and finally decoded, resulting in reconstructed images while optimizing the VQ-VAE's loss function and evaluating the model's performance using perplexity.

5. Optimizer and Parameters:

The parameters used are as follows:

i) Batch size = 128
ii) Number of hidden layers = 128
iii) Embedding dimension = 64
iv) Number of embeddings = 512
v) Commitment cost = 0.25
vi) Decay = 0.99
vii) Learning rate = 0.001

The optimizer used for training of VQ-VAE model is the Adam optimizer.

6. Training Loop:

The training loop iterates over a specified number of training updates, loading batches of data from the training loader. For each iteration, it performs a forward pass through the model to compute reconstruction error, VQ-VAE loss, and perplexity. These metrics are then used to calculate the total loss, which is used to update the model's parameters via backpropagation and optimization. During training, the reconstruction error and perplexity metrics are logged and periodically printed for monitoring.

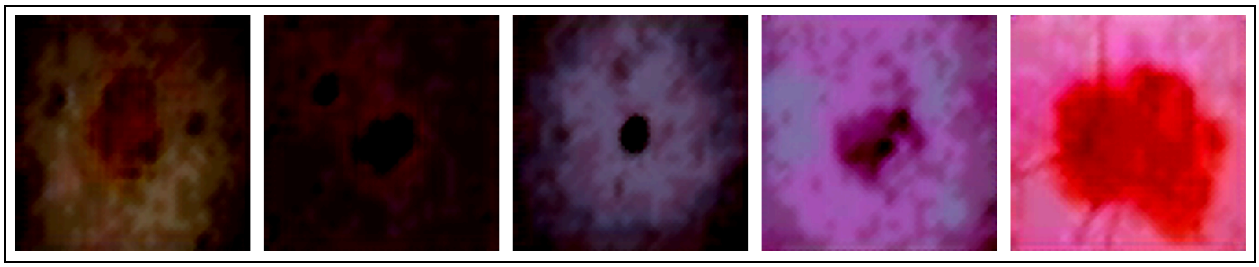


Fig 2(a): Reconstructed Images on Train data after 25000 epochs

Epochs	Reconstruction Loss	Perplexity
5000	0.021	120.782
10000	0.115	34.385
15000	0.083	53.812
20000	0.038	65.492
25000	0.023	77.078

Table 1(a): Observed Reconstruction Loss and Perplexity

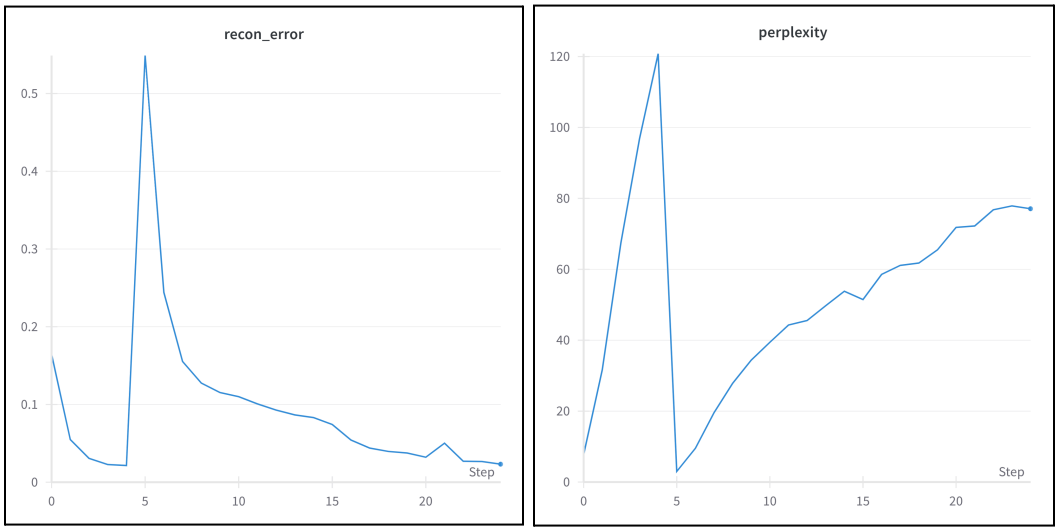


Fig 2(b): Reconstruction Loss and Perplexity Plots

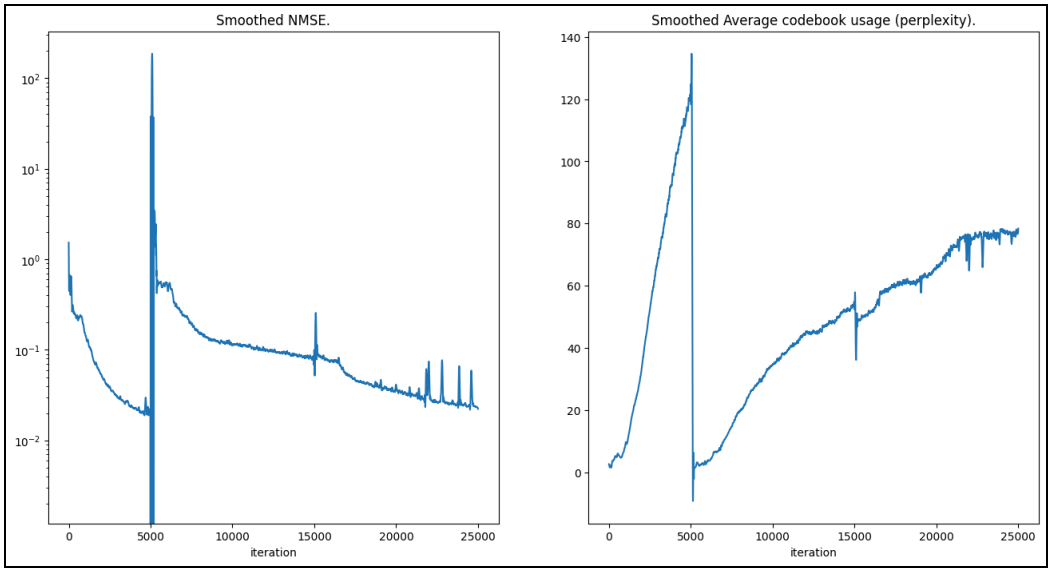
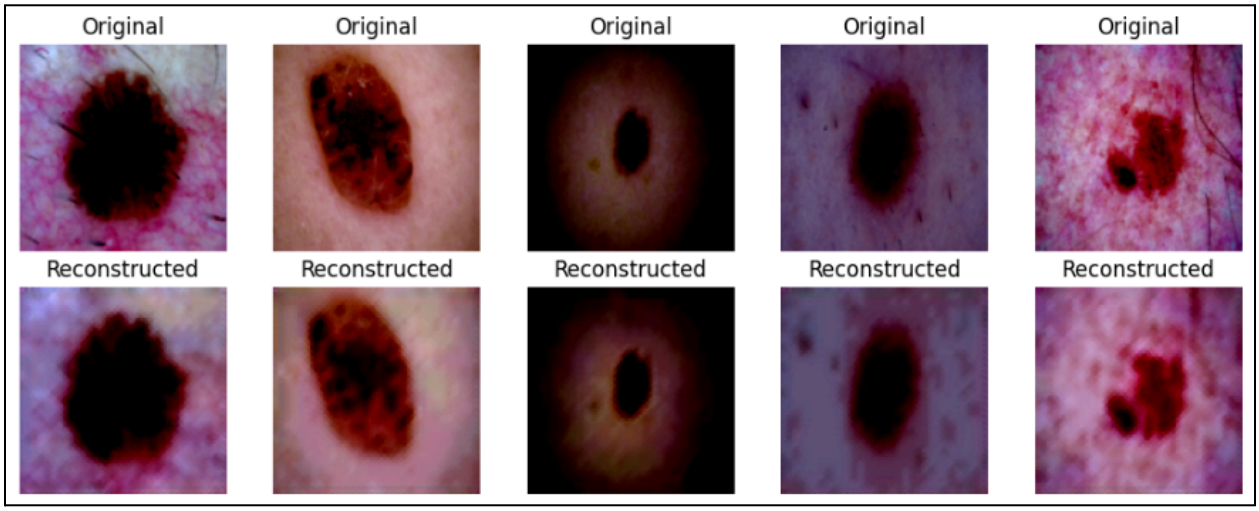


Fig 2(c): Smoothed NMSE and Smoothed Average codebook usage (Perplexity) Plots

7. <u>Testing Loop</u>:

The testing loop segment evaluates the model on a batch of test images. First, it retrieves a batch of test images from the test loader and forwards them through the model. The encoder extracts features from the images, which are then processed by a 1x1 convolutional layer _pre_vq_conv before being quantized by the Vector Quantizer module _vq_vae. The quantized representations are then passed to the decoder _decoder to reconstruct the images.



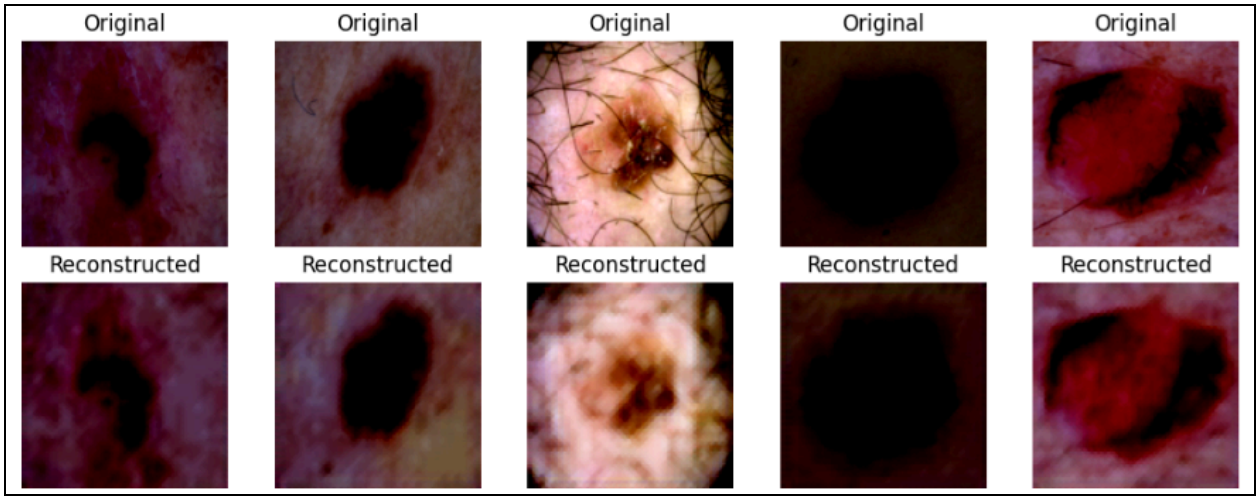


Fig 2(d): Reconstructed Images on Test data after 25000 epochs

8. <u>Analysis and Observations</u>:

Table 1(a) shows the observed reconstruction loss and perplexity values for different epochs during the training of the VQ-VAE model. As the number of epochs increases from 5000 to 25000, the reconstruction loss decreases from 0.021 to 0.023, indicating a better reconstruction of the input images by the model. However, the perplexity values do not follow a monotonic trend, suggesting that the diversity of the learned codebook embeddings may have fluctuated during training.

Fig 2(b) visualizes the reconstruction loss and perplexity plots over the training iterations. The reconstruction loss plot shows a sharp decrease initially, followed by a gradual decline as training progresses. The perplexity plot exhibits a peak around the initial training iterations, indicating that the codebook embeddings were diverse at the beginning, but as training continued, the perplexity decreased, suggesting a potential loss of diversity in the learned embeddings.

Fig 2(c) shows the smoothed NMSE (Normalized Mean Squared Error) and smoothed average codebook usage (perplexity) plots. The NMSE plot exhibits a sharp decline initially, followed by a flattening out, indicating that the model learned to reconstruct the input images effectively after a certain number of training iterations. The perplexity plot displays a peak early in training, followed by a gradual decrease and eventual stabilization, similar to the trend observed in Fig 2(b).

Fig 2(d) provides a visual comparison of the original and reconstructed skin lesion images by the trained VQ-VAE model. The reconstructed images appear to capture the essential features and patterns of the original images, demonstrating the model's ability to learn meaningful latent representations and reconstruct the input data effectively.

III. PHASE 2:

Design and train the autoregressive model (PixelCNN, RNN, LSTM, Transformer) to generate diverse realistic images using the codebook and the decoder trained during phase 1.

1. Loading the VQ-VAE model:

We load the trained VQ-VAE (Vector Quantized Variational Autoencoder) model in PyTorch. First, it initializes the model with specific hyperparameters such as the number of hidden units, embedding dimension, commitment cost, and decay. Then, it loads the trained weights of the model from a file. After loading the weights, it moves the model to a specified device, likely a GPU, to leverage its computational power. Finally, it sets the model to evaluation mode, which ensures proper behavior during inference.

2. Architecture of PixelCNN:

The Auto-Regression model used for this assignment is a neural network called PixelCNN, primarily used for image generation tasks. In the initialization part, the structure of the network is established.

It starts with a convolutional layer conv1 which extracts initial features from the input image. Then, a series of convolutional layers layers are added to capture increasingly complex patterns in the image. These layers enhance the network's ability to understand spatial dependencies within the image. Finally, a conv2 layer is employed to generate the output image, with each pixel's value being predicted based on the features learned by the preceding layers.

In the forward pass, the input image undergoes a sequence of operations through these layers, including convolution and activation functions like ReLU, to produce the final image prediction. This process enables the PixelCNN model to iteratively refine its predictions, generating images with increasingly realistic details.

3. Loss Function and Optimizer:

The Loss function used is Cross Entropy loss and the optimizer used is Adam

4. Training Loop:

This training loop iterates over a specified number of epochs, typically 50 in this case. Within each epoch, the model is set to evaluation mode (model.eval()) and the PixelCNN model is set to training mode (pixel_cnn_model.train()). Then, for each batch of data in the training loader, the loop iterates through.

For each batch:

1. The input data is passed through the VQ-VAE model (model), where it undergoes encoding, quantization, and decoding steps to reconstruct the input.

2. The reconstructed data is then passed through the PixelCNN model (pixel_cnn_model), which generates predictions for each pixel.
3. The loss between the predicted and original input data is computed using a predefined loss criterion (pix_criterion), typically a pixel-wise loss like Mean Squared Error (MSE).
4. The optimizer's gradients are reset (optimizer.zero_grad()), and the loss is backpropagated (loss.backward()).
5. The optimizer takes a step based on the gradients to update the model parameters (optimizer.step()).

No. of Epochs	Training Loss
10	1.3145
20	0.9108
30	0.7354
40	1.2422
50	0.998
70	0.8543
80	0.8579
90	0.6743
100	0.929

Table 2: Training Loss in PixelCNN

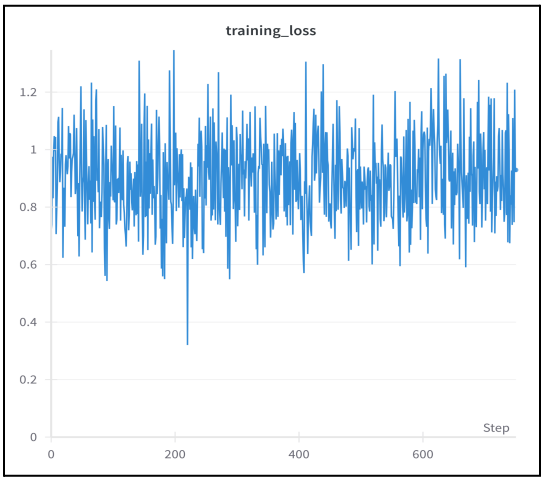


Fig 3(a): Training Loss in PixelCNN

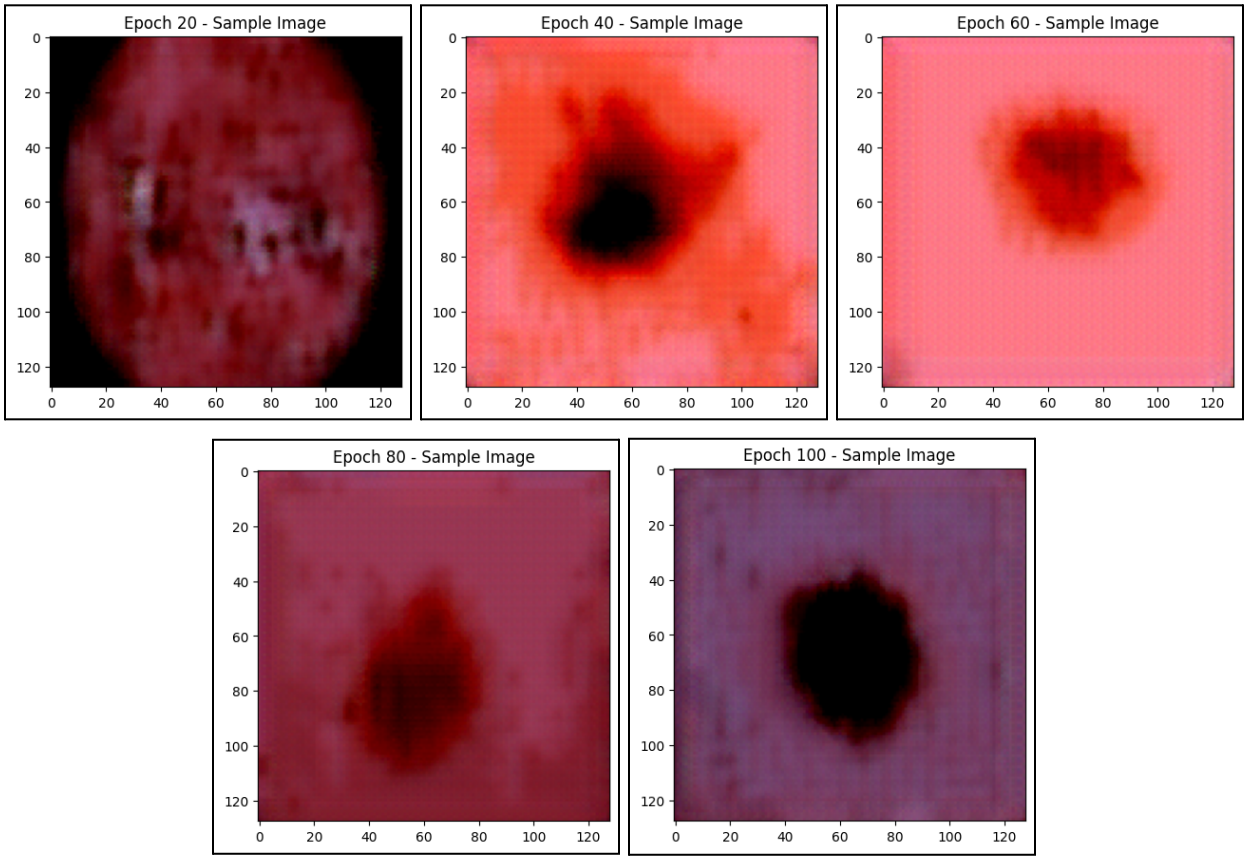


Fig 3(b): Images generated by training data

5. <u>Testing Loop:</u>

The Testing loop evaluates the performance of a model and a PixelCNN model on a test dataset. It iterates through batches of data from the test loader, computing the output of the VQ-VAE model on the input data, followed by quantization and reconstruction steps. The PixelCNN model then generates predictions based on the reconstructed data, and the loss between the predicted outputs and the original inputs is computed. These losses are accumulated and averaged to obtain the average test loss, indicating the overall performance of the models on unseen data. The models are set to evaluation mode to ensure proper behavior during testing, and gradient computation is disabled to optimize speed and memory usage.

The average test loss observed was 0.6543425619602203

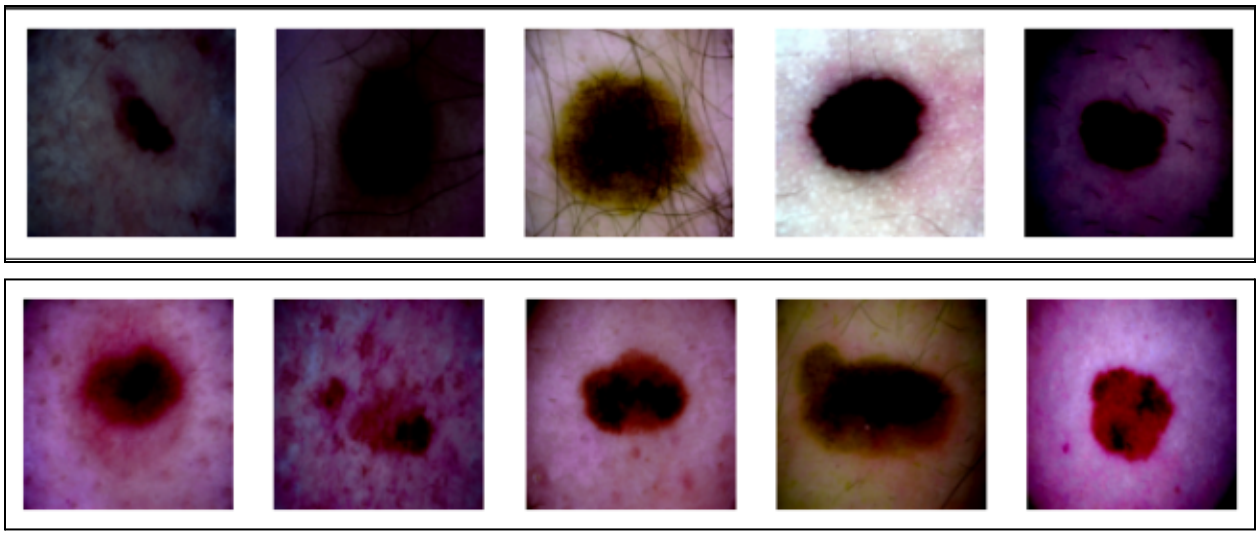


Fig 3(b): Images generated on test data

6. <u>Analysis and Observations</u>:

Table 2 shows the observed training loss for different epochs during the training of the PixelCNN model. As the number of epochs increases, the training loss fluctuates a lot, indicating a better reconstruction of the input images by the model.

Fig 3(a) and 3(b) provide the reconstructed skin lesion images by the trained PixelCNN model. The reconstructed images appear to capture the essential features and patterns of the original images, demonstrating the model's ability to learn meaningful latent representations and reconstruct the input data effectively. It is also able to recreate the same detailed images on test data as well which ensures proper training of the VQ-VAE model as well as the PixelCNN model.