Emoji Generation With VQ-VAE

Generative AI Project Report

Prateek Rath IMT2022017 Shreyas S IMT2022078
Team ID 6

Motivation & Background

Generative models aim to create new, high-quality, and diverse images, but many struggle to balance reconstruction fidelity with creativity. To address this, this project explores a two-stage generative framework using **Vector Quantized Variational Autoencoders (VQ-VAE and VQ-VAE2)**.

Unlike standard VAEs that learn continuous latent spaces, VQ-VAE learns a **discrete codebook**—a set of visual building blocks that capture essential patterns and structures within images. In **Part A**, the model is trained to encode input images into these discrete representations and reconstruct them, learning the fundamental components of the dataset.

However, VQ-VAE (and VQ-VAE2) alone only performs reconstruction. To enable **generation**, a separate **prior model** such as PixelCNN or PixelSNAIL is trained to model the distribution of these discrete codes. This prior captures how different codebook elements co-occur, allowing the system to sample new, coherent code combinations.

VQ-VAE2 extends this by introducing a **hierarchical latent structure**—a top and bottom codebook—where the higher level captures global image features, and the lower level refines details. This hierarchical design improves reconstruction quality and representation of richness, forming a stronger foundation for high-quality image generation.

Methodology (dataset, architecture, training setup)

We've used both vqvae and vqvae2 in the project. Each section either has separate subsections for both parts or just information corresponding to the relevant part used.

Dataset

We resized the images to either 32 x 32 or 64 x 64 pixels. A 32 x 32 image input could lead to lower computational requirements and faster training with lesser parameters. Deep learning models are extremely sensitive to the **scale of input values**. If we feed them pixel intensities in

the range [0, 255], gradients and activations can blow up or vanish because the inputs are too large. We split the data 80-20 into train and validation sets and created dataloaders for the sets.

Data Preprocessing:

VQVAE

1) **Resizing:** Images are resized to 32x32 pixels.
2) **Tensor Conversion:** Images are converted to PyTorch tensors for GPU computation. The call to ToTensor makes pixels have values in the range [0, 1] for each channel instead of [0, 255].

VQVAE2

1) **Resizing**: Images are resized to 64x64 pixels.
2) **Tensor Conversion**: Images are converted to PyTorch tensors for GPU computation. It makes the pixels have values in the range [0,1] for each channel instead of [0,255].
3) Normalize the image from [0,1] to [-1,1] for each of the channels.

Architecture

VQVAE

Part	Description
Encoder	Compresses input images into a lower-dimensional latent representation using convolutional layers and residual blocks.
Vector Quantizer	Discretizes these latent features by mapping them to the nearest embeddings in a learned codebook, enforcing discrete latent representations with a commitment loss.
Decoder	Reconstructs images from quantized latents using transposed convolutions and residual blocks.

VQVAE2

Part

Encoder (Bottom & Top Encoders)	The model has two hierarchical encoders: a **bottom encoder** that downsamples the input image by a factor of 4 and extracts mid-level features, and a **top encoder** that further compresses these features by another factor of 2 to capture high-level semantics. Both encoders use convolutional and residual blocks to model spatial dependencies efficiently.
Top-Level Quantizer (`quantize_t`)	Converts the top encoder's continuous latent features into discrete codes using a learned **embedding codebook**. Each latent vector is replaced by the nearest embedding, enforcing a discrete latent space and stabilizing training via an exponential moving average (EMA) update of embeddings.
Top Decoder (`dec_t`)	Reconstructs a coarse representation from the top-level quantized embeddings. This upsampled feature map provides high-level context to guide the bottom-level quantization process.
Bottom-Level Quantizer (`quantize_b`)	Takes both the bottom encoder's features and the upsampled top decoder output, concatenates them, and then quantizes the combined representation into discrete embeddings using another learned codebook. This enables fine-detail reconstruction conditioned on the higher-level context.
Upsampler (`upsample_t`)	Expands the spatial resolution of the top quantized embeddings (from the top quantizer) before fusing them with the bottom quantized features for decoding.
Decoder (Final Decoder)	Combines the upsampled top quantized embeddings and the bottom quantized embeddings to reconstruct the full-resolution image. It uses transposed convolutions and residual blocks to synthesize high-quality reconstructions from the discrete latent representations.
Commitment Loss (`diff`)	Measures the mean squared difference between input features and quantized embeddings, ensuring that the encoder commits to specific codebook entries while preventing codebook collapse. This loss is added to the reconstruction loss during training.

Training Setup

VQVAE

We trained **VQ-VAE** model on the input emoji images using **pytorch** with the **Adam optimizer** (learning rate = 1e-3). The loss combined **reconstruction loss** (MSE) and **vector quantization commitment loss** weighted by β = 0.25. We continued until validation loss stopped decreasing.

We trained the **PixelCNN prior** on the latent grids (indices 0-CODEBOOK_SIZE-1) using **cross-entropy loss** and **Adam** (a lower learning rate of = 2e-4). It learned the spatial

distribution of latent codes to generate new samples that, when decoded by the VQ-VAE, produced novel emoji-style images.

VQVAE2

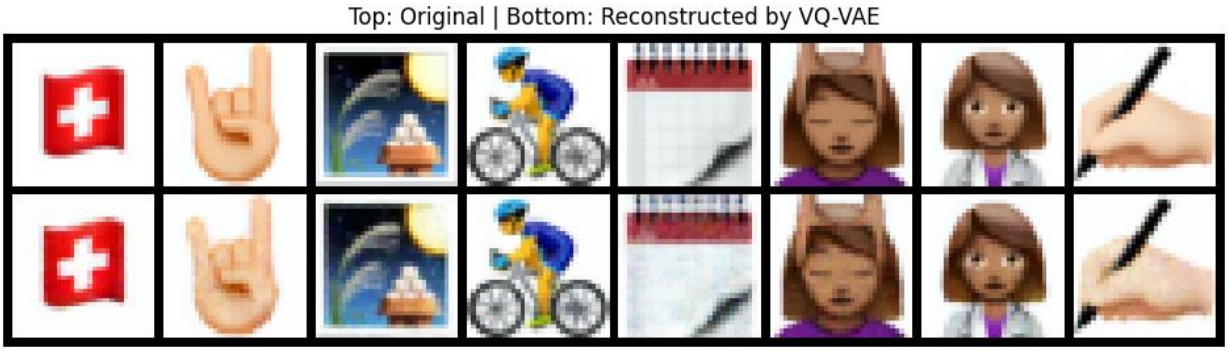
The training setup initializes the VQ-VAE2 model on GPU (if available) and optimizes it using the **Adam optimizer** with a learning rate of 2×10^{-4}. During each epoch, the model reconstructs input images and computes the total loss as the sum of reconstruction and quantization losses. The optimizer updates the model weights to minimize this loss. Training runs for 200 epochs, printing the average loss for train and validation and displaying reconstructed images after each epoch to monitor performance.

We trained a PIXELCNN prior and a PIXELSNAIL prior (attention based) using cross-entropy loss and **Adam optimizer** with a learning rate of 2×10^{-4}.

Experiments and Results

VQVAE

Reconstruction:



Original Image and Reconstructed Image Comparison

While training, we tried **two different loss functions**. The original loss, which is the sum of the mse, codebook and commitment losses (no VGG perception loss), resulted in mse values (low) but poor ssim and psnr values (low), but the reconstruction results on the validation data seem to be fine. The table below shows the case where the VGG perception loss and ssim were also added as part of the training process. It yields better metrics.

Metric	Value(Train)	Value(Validation)

MMSE	0.001631	0.001806
PSNR	0.956241	0.953946
SSIM	29.37 dB	29.24 dB

Generation:

We have trained the VQ-VAE to produce compact emoji features (the codebook) and reconstruct images. However, it cannot generate *new* images on its own. It has no understanding of *how* to arrange the codebook vectors to form a plausible emoji. Sampling random values from the latent space is irrational as those latent codes could correspond to something other than emojis. To solve this, we trained a second model, a **generative prior**, on the discrete latent codes.

For this project, we implemented a **PixelCNN prior**, an autoregressive model, to learn how to sample highly probable samples that correspond to emojis from the latent space.

Training the Prior:

The goal of the PixelCNN is to model the complex probability distribution of the latent codes. In simple terms, we want to teach it: "Given all the codes that come *before* this position, what is the most likely code for *this* position?"

1. **Input Embedding Layer**
   The model first passes the integer-encoded codes (0 to num_embeddings - 1) through an nn.Embedding layer.
   This transforms each code into a dense vector, resulting in a feature map of shape **(B × D × H × W)**, where D = input_channels.
2. **Initial MaskedConv2d ('A') Layer**
   The first convolution is a **7×7 MaskedConv2d** of type 'A'.
   It ensures that the model cannot see the current or future pixels, strictly adhering to the autoregressive setup.
   A **BatchNorm2d** layer follows to stabilize training.
3. **Residual PixelCNN Blocks**
   Next comes a stack of **num_layers (e.g., 20) residual blocks**, each implemented as a PixelCNNBlock.
   Each block contains:
   a. A ReLU activation
   b. A **3×3 MaskedConv2d ('B')** layer (allowing access to the current pixel)
   c. A **BatchNorm2d** layer
   The block output is added back to its input (residual connection), improving gradient flow and training stability.

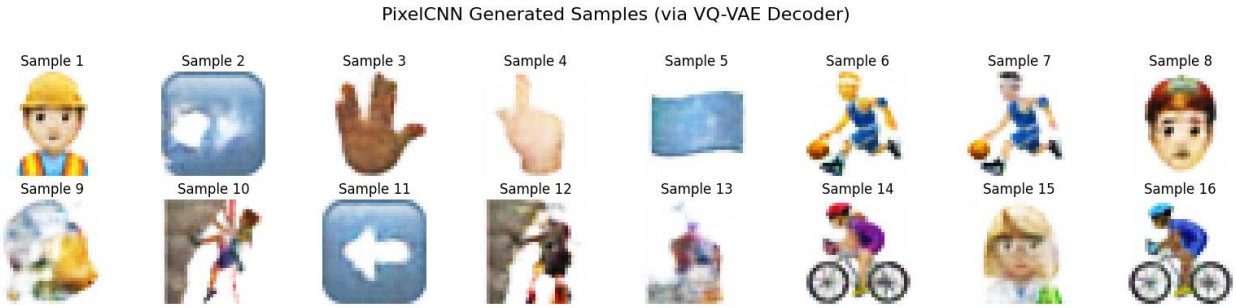
4. **Final Projection Layers**
   After the residual stack, a few **1×1 MaskedConv2d ('B')** layers are applied to project the hidden features back into the embedding space:
   a. The first 1×1 MaskedConv2d expands or mixes feature channels.
   b. The final 1×1 MaskedConv2d outputs **num_embeddings logits** for each spatial position — one score per possible codebook entry.

Generation Using Autoregressive Sampling:

Once PixelCNN learns how to sample, we can use it to generate novel emoji code maps. This is done via an iterative, pixel-wise, autoregressive sampling process.

1. **Start:** We begin by creating an empty `8x8` grid filled with zeros.
2. **Predict (0, 0):** This empty grid is fed into the PixelCNN. The model, having learned what a "starting pixel" looks like, outputs 512 probabilities for the code at position `(0, 0)`.
3. **Sample (0, 0):** We sample from this probability distribution (using `torch.multinomial`) to select a single code (e.g., code `111`). This code is then placed into our grid at `(0, 0)`.
4. **Predict (0, 1):** The *updated* grid (now containing one code) is fed back into the PixelCNN. The model now predicts the probabilities for position `(0, 1)`, conditioned on the code at `(0, 0)`.
5. **Sample (0, 1):** We sample again (e.g., getting code `62`) and place it at `(0, 1)`.
6. **Repeat:** This sequence of "predict, sample, update" operations is repeated 64 times, sequentially filling the entire `8x8` grid. Each new code is predicted based on all the codes that were sampled before it.
7. **Decode:** The final `8x8` grid is a completely synthetic latent map. We feed this grid into our trained VQ-VAE's **decoder**, which translates it from the code "language" back into a visual emoji image.

Visuals And Metrics:



16 Generated Samples

Quantitative Evaluation:

To objectively measure the quality of our generated distribution against the real data, we employed the **Fréchet Inception Distance (FID)** score. The FID score measures the statistical similarity between two sets of images (a certain number of generated samples vs. the entire real training dataset) based on their high-level features from an InceptionV3 network. A lower score signifies that the two distributions are more similar.

Our model achieved an **FID score of around 176.** This might seem to be poor but the images generated are decent in quality. We can identify the emojis, even though they are a little blurry and the edges aren't sharp.

This competitive score confirms our qualitative observations. It indicates that the PixelCNN prior, in combination with the VQ-VAE's sharp decoder, is capable of producing a diverse set of novel emojis that are statistically close to the distribution of real emojis. This successfully fulfills the primary goal of Part B of the project.

Creative Extensions:

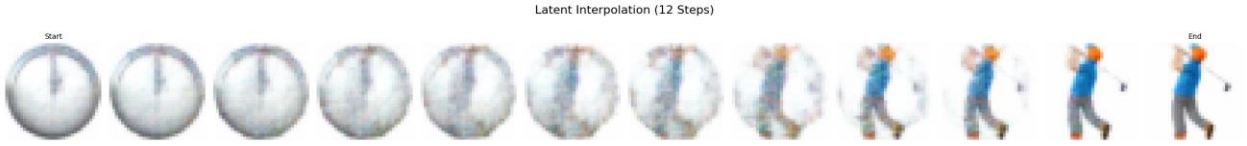
Latent Interpolation

This extension was implemented to investigate the *structure* and *smoothness* of the VQ-VAE's learned latent space. A well-trained model should not have a "chaotic" or "gappy" space; rather, the path between any two known data points (emojis) should also contain plausible, in-between variations. A smooth transition would prove that the model has learnt a semantically meaningful "representation" of emoji features."

To test this, we selected two distinct emojis (A and B) from the test set.

1. Both images were first passed through the VQ-VAE's encoder to get their `8x8` grids of *discrete code indices*.
2. We then used these indices to look up the corresponding *continuous embedding vectors* from the VQ-VAE's codebook. This gave us two `8x8` grids of high-dimensional vectors, EA and EB.
3. The interpolation was performed *in this continuous embedding space*. We generated 12 intermediate grids, $E\alpha$, by linearly blending the vectors: $E\alpha=(1-\alpha)\cdot EA+\alpha\cdot EB$, where α was spaced from 0 to 1.
4. Each of these 12 blended vector-grids was passed *directly to the VQ-VAE's decoder* to be rendered into an image.

The results visualized show a smooth transition. This confirms that our VQ-VAE has learned a decent representation of the latent space. We go from the clock to a golf shot followed through.



Latent Interpolation between the two Emojis

Image Inpainting

This extension was implemented to evaluate the VQ-VAE's ability to reason for missing visual information and to reconstruct coherent images from incomplete inputs. A well-trained VQ-VAE should not merely "copy" what it sees but rather use its learned latent representations to infer and plausibly fill in the masked regions based on context.

To test this, we applied **8×8 random square masks** on **32×32 emoji images** from the test set. Each masked image was then passed through the trained VQ-VAE model.

During this process, the model had to predict the contents of the missing 8×8 region purely from the surrounding visual cues. The reconstructed outputs were compared to the original unmasked images to assess visual consistency.

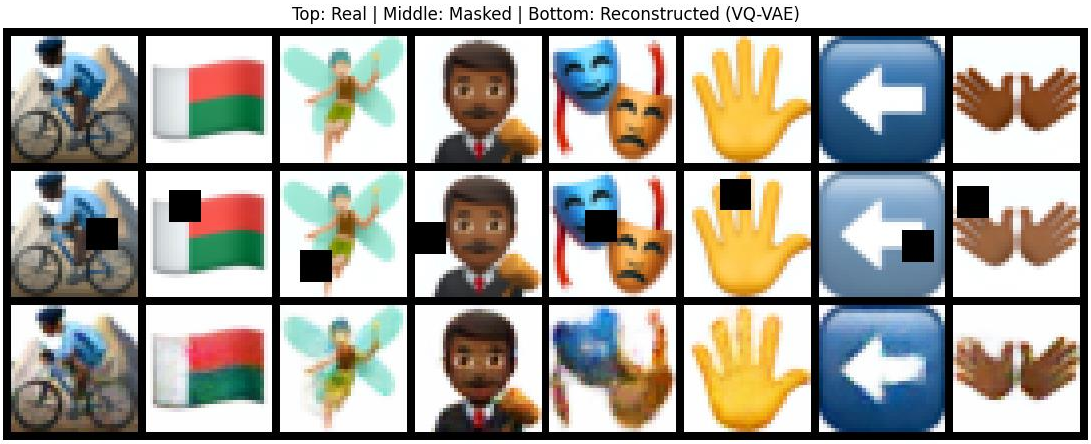


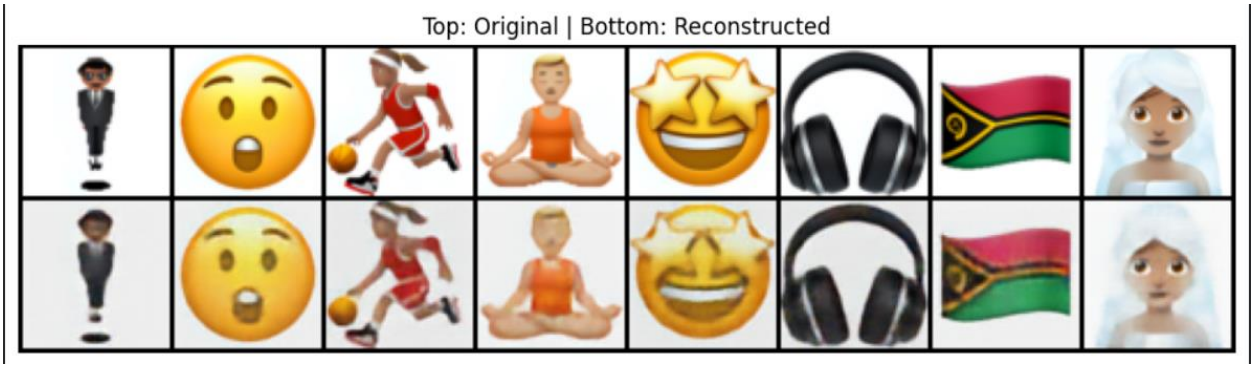
Image Inpainting

The metrics below are for the inpainting task. They are of course slightly worse than the normal reconstruction metrics as it is harder to reconstruct when lesser information is present.

Metric	Value(Train)	Value(Validation)
MMSE	0.003699	0.003760
SSIM	0.908436	0.911058
PSNR	25.42	25.32

VQVAE2

Reconstruction:



Metric	Value (Val)
MSE	0.00761
SSIM	0.8972
PSNR	22.8269

The training was done for 200 epochs in the epochs. The training loss was decreasing constantly; the validation was initially decreasing fast but was decreasing slowly by the end of training.

Generation:

PixelSnail

1. **`WNConv2d`** – A convolutional layer with weight normalization applied for stable and efficient training. It helps control the scale of weights and improves gradient flow.
2. **`CausalConv2d`** – A convolution layer that enforces causality by masking future pixels, ensuring each pixel only depends on previous ones.
3. **`GatedResBlock`** – A residual block with gated convolutions that regulate information flow and improve model expressiveness, optionally using conditional inputs.

4. **`CausalAttention`** – Implements masked self-attention so each position attends only to previous ones, capturing long-range dependencies while maintaining autoregressive order.
5. **`PixelBlock`** – A stack of gated residual blocks (and optional attention) that jointly process local and global features for better generation quality.
6. **`CondResNet`** – A small ResNet that encodes conditional inputs (e.g., top-level VQ-VAE codes) into features used to condition PixelSNAIL's generation.
7. **`PixelSNAIL`** – The main model combining causal convolutions, gated residual blocks, and attention mechanisms to autoregressively generate discrete image codes.

We took the codebook entries from both top and bottom encoders and trained for both top and bottom with 2 pixelsnails. Same for pixelcnn.

FID: 341

PixelCNN

FID: 416.82

Even though the VQ-VAE2 achieved good reconstruction results, the generation of quality using PixelCNN or PixelSNAIL was poor mainly due to insufficient data and limited diversity in the 2.4k emoji dataset. These autoregressive models require large, diverse datasets to learn the complex spatial dependencies between codebook tokens. With such a small dataset, they tend to overfit, producing blurry or repetitive outputs instead of coherent new images. Additionally, emojis have low texture variation, making it harder for the model to generalize beyond memorized patterns, further degrading generation quality.

Creative Extensions:

Super-Resolution using VQ-VAE2

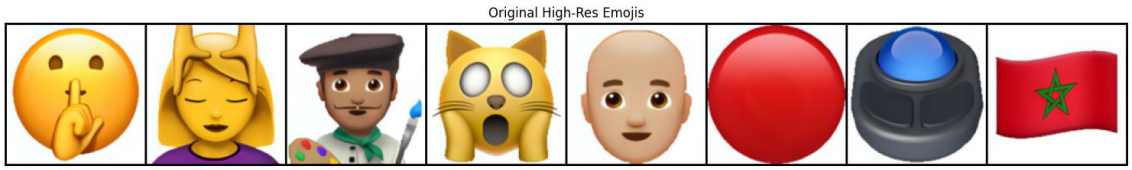
Objective:
The goal of this project was to perform image super-resolution on emoji images, upscaling them from 64×64 to 128×128 pixels using a Vector Quantized Variational Autoencoder (VQ-VAE2).

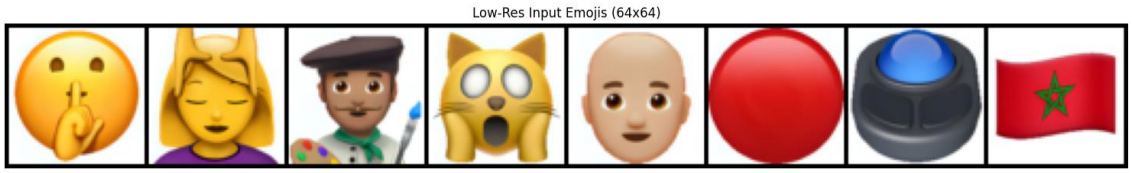
Approach:
A pre-trained VQ-VAE2 was employed to encode low-resolution images into discrete latent codes. These codes capture key image features and were used to train a new **SuperResolutionDecoder**, designed to reconstruct high-resolution images from the encoded representations while keeping the VQ-VAE2 encoder and codebooks frozen.

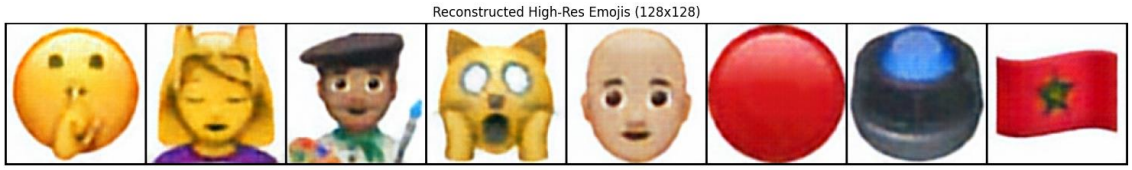
Methodology:
A paired dataset of 128×128 images and their 64×64 sampled versions was prepared and split into training, validation, and test sets. The SuperResolutionDecoder received codebook vectors from the frozen encoder and was trained using an Adam optimizer to minimize reconstruction loss (MSE) between the generated and original high-resolution images. Only the decoder parameters were updated during training.

Evaluation:
After training, the decoder's performance was tested using metrics such as Mean Squared Error (MSE), Structural Similarity Index (SSIM), and Peak Signal-to-Noise Ratio (PSNR). Visual comparisons of low-resolution inputs predicted outputs, and ground-truth images were used to assess perceptual quality.

Conclusion:
By leveraging the discrete latent representations of a pre-trained VQ-VAE2 and training a specialized decoder, the system effectively upscaled low-resolution emoji images to higher resolutions, demonstrating strong reconstruction quality both quantitatively and visually.







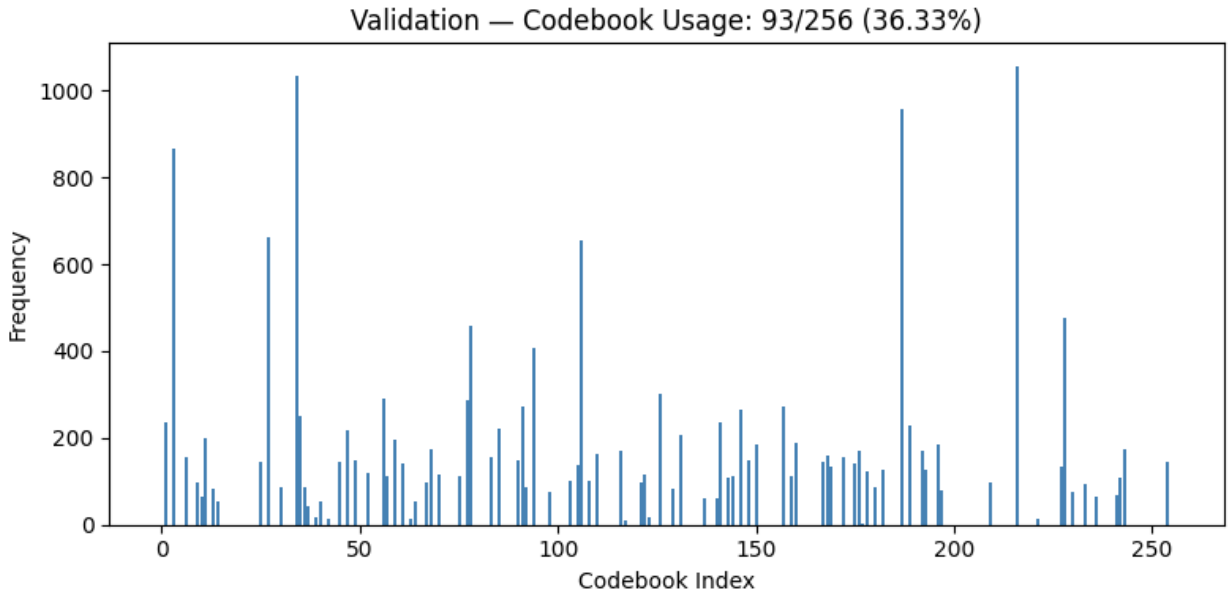
Metric	Value (Validation)
MSE	0.0172
SSIM	0.7974
PSNR	23.6471

Discussion

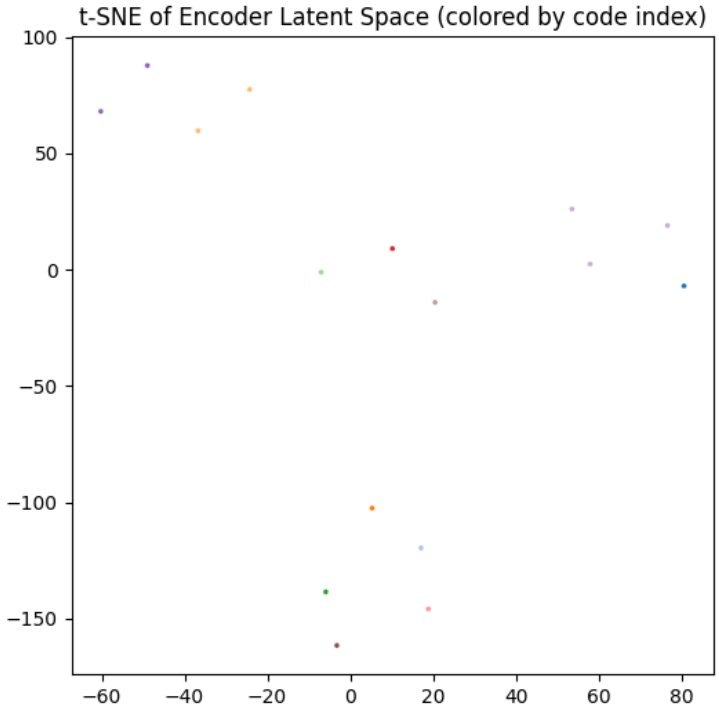
VQVAE:

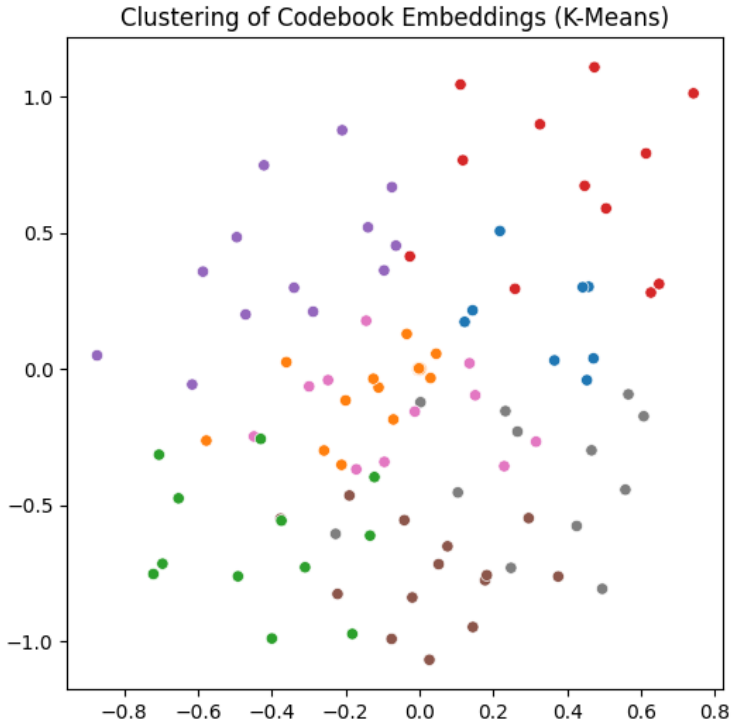
The best hyperparameters are as follows:

hyperparameter	value	explanation
BATCH_SIZE	64	To get better gradient measures
IMG_SIZE	32	Easier task
CODEBOOK_SIZE	256	To avoid codebook collapse and ensure sufficient usage
HIDDEN_DIMENSION	256	Hidden layer dimension in the vqvae and pixelcnn
EMBEDDING_DIMENSION	256	To learn rich representations
BETA	0.25	To give enough flexibility to codebook vectors to adapt



Our codebook usage is between 30% and 70%, which is fine.
Below are the t-SNE and k-means plots





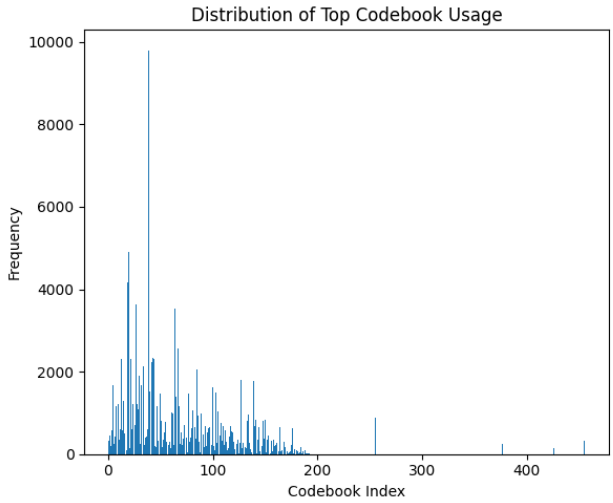
To some extent we see that the t-sne plot is scattered well, and the k-means plot shows distinct means. We can see that the different colors in the plot are localized. The means do overlap to some extent, but in general we can see that they are quite separate, which is a good sign.

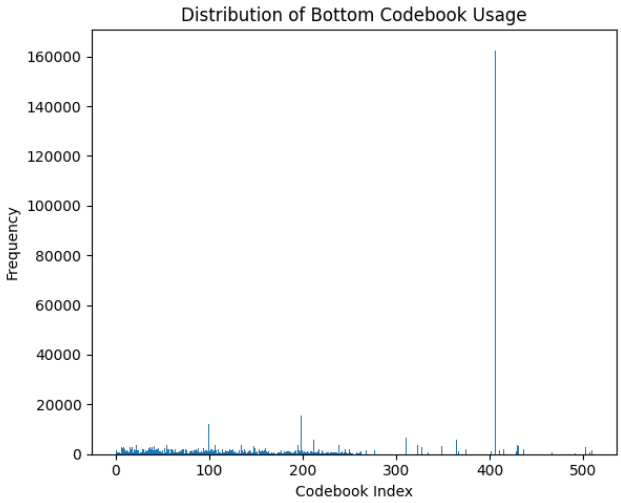
While training, we used gradient clipping along with exponential moving averages for the codebook to ensure better results.

VQVAE2:

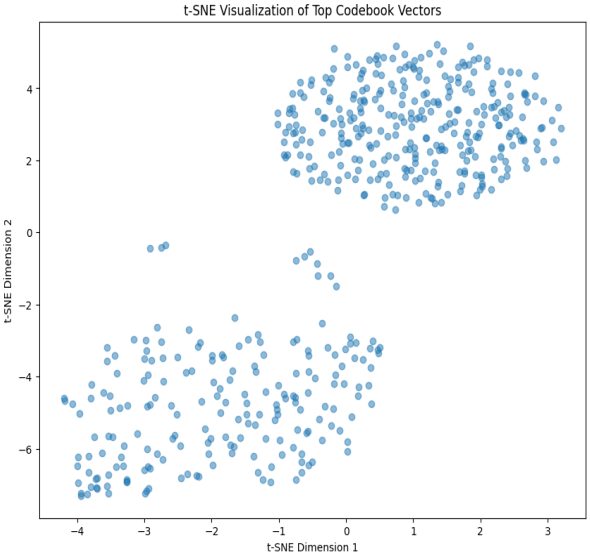
The best hyperparameters for VQVAE2 for reconstruction of 64x64 images

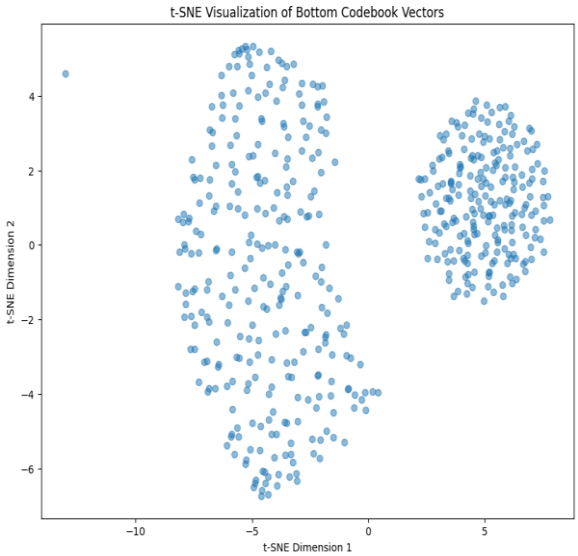
Hyperparameter	Value
IMG_SIZE	64
BATCH_SIZE	32
EMBED_DIM	256
NUM_EMBEDDINGS	512





Percentage of used top codebook vectors: 38.67%
Percentage of used bottom codebook vectors: 56.64%
So, for the top 198 out of 512 codebook vectors were used.
And for the bottom 290 out of 512 codebook vectors were used.





t-SNE Visualization of Codebook Vectors

To analyze the learned latent representations, t-SNE was applied to the codebook embeddings from both the top and bottom levels of the VQ-VAE2 model.

Top Codebook Vectors:
The t-SNE visualization of the top-level embeddings shows two clearly separated clusters. This indicates that the encoder has learned to group high-level semantic features into distinct categories. In the context of emoji images, these clusters likely represent broad visual or

conceptual differences (e.g., faces vs. objects). The top-level quantization thus captures global structural or categorical information.

Bottom Codebook Vectors:
The t-SNE plot of the bottom-level embeddings also reveals two major clusters, but with more internal variation and spread within each cluster. This suggests that the bottom codebook captures finer, local-level details such as texture, color, or small shape variations that complement the global features learned at the top level. Together, these representations allow the model to reconstruct detailed and coherent images.

Although the VQ-VAE2 model achieved good reconstructions, the generations from both PixelCNN and PixelSNAIL were poor because the 2.4k emoji dataset was too small to capture sufficient latent diversity. The VQ-VAE2 used only a limited portion of its codebook, leading the autoregressive models to learn a narrow and repetitive distribution of latent codes. Consequently, during sampling, they failed to produce coherent or diverse outputs, resulting in generating unrealistic or distorted images.

Limitations

Reconstruction

Reconstructions look good and have great metrics, but the vqvae/vqvae2 way of doing things has **inherent issues**. The model's latent space and codebook may not represent the additional fine details, leading to loss of sharpness. That's why we see blurry results or lack of sharp edges in images.

Generation

VQVAE

1. A key limitation of using PixelCNN with VQ-VAE for generation is that it models pixel dependencies sequentially, predicting one pixel (or code) at a time.
2. This makes **generation slow** and often leads to **blurry or repetitive samples** when the dataset is small or lacks diversity.
3. PixelCNN also struggles to capture long-range spatial dependencies, so while it can model local textures well, it may fail to produce globally coherent or diverse structures, limiting the overall quality of generated images.

VQVAE2

While VQ-VAE2 achieved good reconstruction quality, its generation performance was poor compared to VQ-VAE. This happened because VQ-VAE2's hierarchical structure (top and

bottom codebooks) allows it to memorize local and global image details effectively during reconstruction but makes it harder for the autoregressive prior (PixelCNN or PixelSNAIL) to learn the complex joint distribution across both levels with limited data. As a result, the model reconstructs training images well but fails to generate diverse and coherent new samples, especially when trained on a small and less varied dataset.

Improvements

1. The quality of generation could be improved by training VQ-VAE2 on a l**arger and more diverse dataset**, which would help the model learn richer latent representations and improve codebook utilization.
2. **Simplifying the VQ-VAE2 architecture** or introducing regularization techniques such as dropout could also prevent overfitting and promote better generalization.
3. For VQ-VAE, careful **hyperparameter tuning** (e.g., codebook size, learning rate, or commitment loss weight) could enhance latent space representation and improve generative quality. Additionally, replacing the PixelCNN prior with a more powerful **diffusion-based prior** could yield more coherent and higher-quality generations.