Project Final Report
Deep Generative Modeling

Name:	Muhammad Ibrahim
Student ID:	8029808

(Graduate Student)

Residual Vector Quantization based VAE

Learning Outcome.

- Explain, Implement and Train Vector Quantized (VQ) & Residual Quantized (RQ) VAE
- Use VQ and RQ VAE for learning discrete representation
- Comparison of VQ and RQ

Problem Statement:

In traditional Variational Autoencoders (VAEs), the encoder maps input data to a latent distribution, from which a sample is drawn and passed to the decoder. However, when the decoder is too powerful, it can learn to ignore the latent representation. This leads to posterior collapse, where the latent space becomes meaningless. To address this, Vector Quantized Variational Autoencoder (VQ-VAE) [1] was introduced, which learns discrete latent representations through vector quantization, ensuring meaningful latent encodings and enabling the learning of a trainable prior. However, for high-resolution data, a fixed-size codebook leads to increased quantization error, while larger codebooks introduce significant computational overhead.

Background Theory

Variational Autoencoder (VAE):

Variational Autoencoders (VAEs) are a class of generative models that learn a probabilistic mapping between data and a latent space. In a standard VAE, an encoder network maps the input data to a latent distribution, and a sample is drawn from this distribution to pass through a decoder to reconstruct the original input. However, a well-known issue in VAEs is posterior collapse [2], which occurs when latent representations become too weak or noise that decoder starts ignoring it and produces some generic outputs that are crude representatives of all seen x.

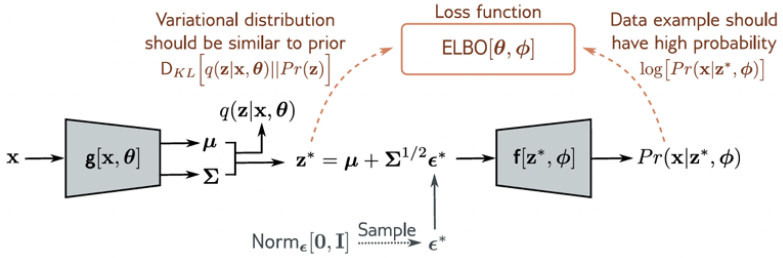

Fig. 1: VAE architecture

Vector Quantized VAE (VQ VAE):

To overcome this limitation, the VQ-VAE was introduced. Instead of using continuous latent variables, VQ-VAE employs discrete latent representations through a technique called vector quantization (VQ). In this approach, the encoder maps the input data to a latent space, which is then quantized using a learned codebook consisting of discrete embeddings. Each latent representation is assigned to the closest entry in the codebook using a nearest-neighbor condition [3] which calculates the distance between all the codebooks and closest codebook is assigned to that vector. Additionally, unlike standard VAEs, VQ-VAE learns a prior distribution over the latent codes, rather than assuming a fixed Gaussian prior. [1] uses Pixel CNN as autoregressive model to learn prior for sampling in VAE.

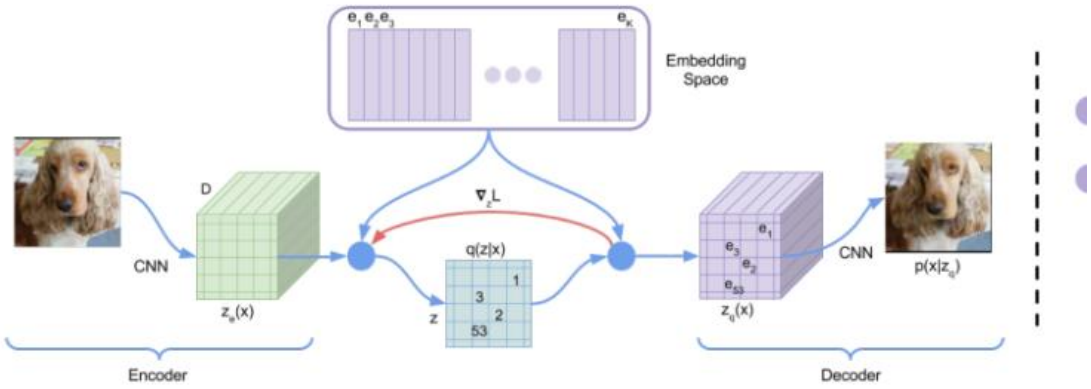

Fig. 2: VQ VAE architecture

Despite its advantages, VQ-VAE has a significant limitation when encoding high-resolution images. As we increase the codebook size to reduce quantization error computational complexity increases significantly. To overcome this problem [4] introduced RQ VAE, which uses residual quantization (RQ) instead of VQ.

Residual Quantization (RQ):

RQ refines the latent representation through multiple quantization steps, reducing the quantization error progressively. Instead of assigning the input to a single codebook entry, RQ encodes the input as a sum of multiple quantized vectors. Each step in the quantization process captures the residual information that was not accounted for in the previous step. This hierarchical quantization allows for a more accurate representation with fewer codebook entries, reducing both error and computational overhead.

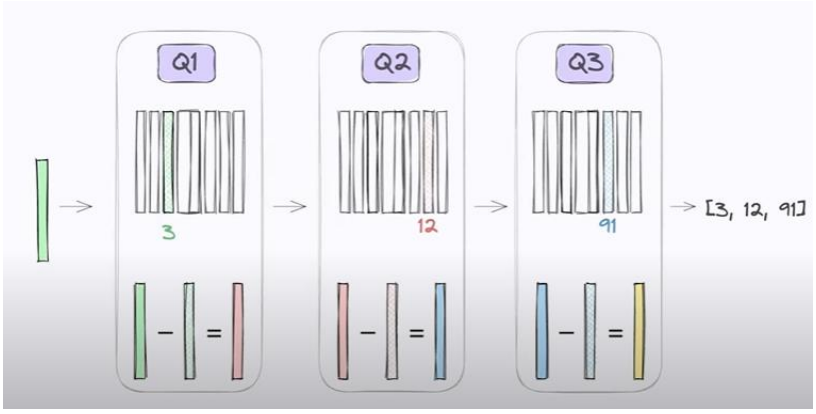

Fig. 3: RQ Procedure

RQ VAE uses this residual quantization in latent dimensions to learn discrete representation. However, the model trained will be deterministic. To generate samples probabilistically [4] uses RQ transformer as autoregressive model to learn prior. So RQ VAE with transformers includes the following steps:

1. The input image is passed to the encoder to generate compressed image in latent dimension. Then the output of encoder is passed through multiple quantization layers, each iteratively refining the latent representation by selecting codebook entries and reducing the quantization error at each stage.
2. The learned discrete codes are then modeled using a transformer-based autoregressive model to learn prior, which enables efficient image generation with high fidelity. Then the samples generated by autoregressive model is passed to the decoder to generate samples

Exponential Moving Average:

In [1] it uses model the learn the codebooks. However sometimes it becomes unstable which is shown in later part of the report. In recent research it uses exponential moving average to update the codebook 6

$$e^{(t)} = \gamma\, e^{(t-1)} + (1 - \gamma) \cdot \hat{e}^{(t)}$$

Where $\hat{e}^{(t)}$ is the current embedding vector, $e^{(t-1)}$ is the previous EMA value and $e^{(t)}$ is the updated EMA codebook embedding.

Methodology:

For VAE, simple encode and decoder was used with two convolutional layers

```
def get_encoder(latent_dim=16):
    encoder_inputs = keras.Input(shape=(28, 28, 1))
    x = layers.Conv2D(32, 3, activation="relu", strides=2, padding="same")(
        encoder_inputs
    )
    x = layers.Conv2D(64, 3, activation="relu", strides=2, padding="same")(x)
    encoder_outputs = layers.Conv2D(latent_dim, 1, padding="same")(x)
    return keras.Model(encoder_inputs, encoder_outputs, name="encoder")


def get_decoder(latent_dim=16):
    latent_inputs = keras.Input(shape=get_encoder(latent_dim).output.shape[1:])
    x = layers.Conv2DTranspose(64, 3, activation="relu", strides=2, padding="same")(
        latent_inputs
    )
    x = layers.Conv2DTranspose(32, 3, activation="relu", strides=2, padding="same")(x)
    decoder_outputs = layers.Conv2DTranspose(1, 3, padding="same")(x)
    return keras.Model(latent_inputs, decoder_outputs, name="decoder")
```

Fig. 4: Encoder & decoder models

The methodology of VQ-VAE and RVQ-VAE is similar to VAE. The loss term of VAE includes KL divergence term. However, in VQ VAE and RQ VAE, it uses commitment and codebook loss.

$$\mathcal{L}_{VAE} = E_{q(z|x)}[\log p\,(x|z)] - D_{\mathrm{KL}}[q(z|x) \parallel p(z)]$$

The first terms tell us about the reconstruction loss and the second term is KL divergence term in which we try to make the posterior distribution as close as possible to the prior of latent variable.

In VQ-VAE we use the following loss terms

$$\mathcal{L}_{VQ-VAE} = \left[\log p\left(x|z_e(x)\right)\right] + ||\text{sg}[z_e(x)] - e||_2^2 + \lambda||z_e(x) - sg(e)||_2^2$$

In the above equation first term represents the reconstruction loss similar to VAE, the second term represents the codebook loss which penalizes each code vector e based on its square distance from its associated encoding vector z_e. It uses stop gradient operator so only codebook is optimized. It's a sparse loss that optimizes the codebooks which are chosen during the quantization. To avoid undesirable fluctuations in codebook during the training it uses commitment loss. It avoids fluctuations and arbitrary growth of the encoding. It penalizes the encoding if they stray far from the selected codebooks.

In RVQ we do it in multiple steps and the residual error is passed through the next stage as shown in the following figure

$$L_{\{RQ-VAE\}} = log\ p(x\ |\ z_e(x)) + \sum_{i=1}^{n}\left(||\ sg\left(r^{(i)}\right) - q^{(i)}||_2^2 + \lambda||\ r^{(i)} - sg\left(q^{(i)}\right)||_2^2\right)$$

Where n represents the number of quantizer layers and r represents the quantization error, which is initially the output of the encoder and is used for training the codebook through residual quantization, as shown in Fig. 3. The quantized outputs from each quantization layer are combined and is passed to RQ transformer for sample generation which is then passed to the decoder for image reconstruction.

The following code shows the implementation of RQ. For each quantizer, it computes the encoding indices of the residual, performs one-hot encoding, and applies matrix multiplication to obtain the quantized value. The code then calculates two types of losses: the codebook loss and the commitment loss. The total loss is a weighted combination of these losses, which is added to the model's loss for optimization.

```
for i in range(self.num_quantizers):

    encoding_indices = self.get_code_indices(residual, self.embeddings[i])
    encodings = tf.one_hot(encoding_indices, self.num_embeddings)
    quantized = tf.matmul(encodings, self.embeddings[i], transpose_b=True)


    # Compute loss
    codebook_loss = tf.reduce_mean((tf.stop_gradient(residual) - quantized) ** 2)
    commitment_loss = tf.reduce_mean((residual - tf.stop_gradient(quantized)) ** 2)
    total_commitment_loss += commitment_loss
    total_codebook_loss += codebook_loss

    residual -= quantized
    quantized_output += quantized


total_loss = self.beta * total_commitment_loss + total_codebook_loss
self.add_loss(total_loss)
```

Fig. 5: RQ implementation

In my case, I trained RQ VAE separately without the autoregressive model. Once the deterministic model is trained. To make it probabilistic, RQ transformer was trained to learn the prior distribution of quantized codes. As prior is learned, we can generate samples from RQ transformer and then pass those quantized codes into the decoder to generate images.

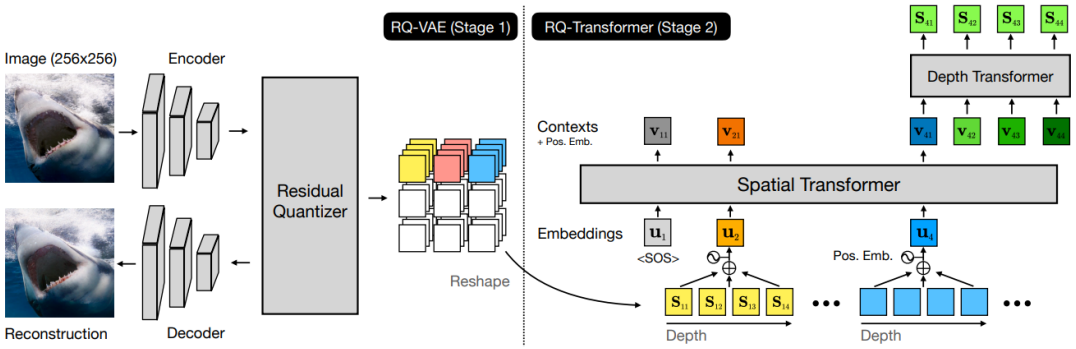

Fig. 6: RQ VAE architecture with RQ transformer [4]

I used RQ transformer as given in [5]. However few parameters were changed with respect to my model

```
from rq_transformer import RQTransformer

rq_transformer = RQTransformer(
    num_tokens = quantizer.num_embeddings,  #Number of embdeddings in each quantizer
    dim = 512,
    max_spatial_seq_len = 49,              # matches 7x7 = 49 patches
    depth_seq_len = 2,                     # number of quantizers
    spatial_layers = 6,                    # number of transformer layers for spatial modeling
    depth_layers = 4,                      # number of transformer layers for depth modeling
    dim_head = 64,
    heads = 8
)
```

Fig. 7: RQ VAE architecture with RQ transformer [4]

Results and Discussion:

For the analysis of both models, I used MNIST dataset. The input dimension of the encoder is set to 28,28,1. While the latent dimension is set to 7,7,4. The size of vector in the codebook is 4 or mentioned otherwise. While the number of codebooks in each quantizer is set to 256 or mentioned otherwise. All experiments are done for deterministic models as training of RQ transformer was difficult to train properly. The training of both RQ VAE and RQ transformer is done separately.

In fig. 8, we can see that RQ VAE performed much better than VQ VAE for the same number of total codebooks. For VQ, I used 256 codebooks while in RQ, two quantizers were used with 128 codebook each.

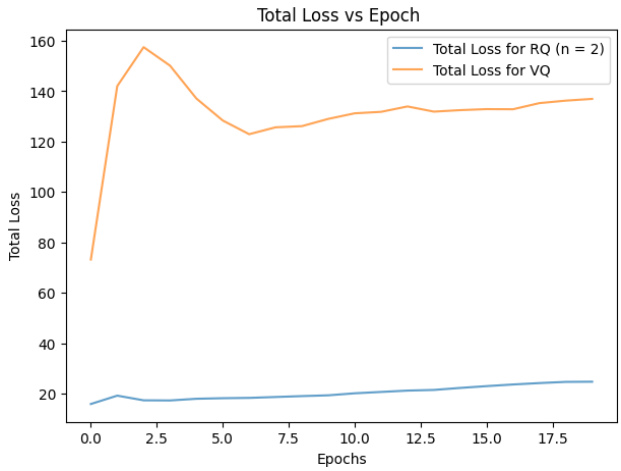

Fig. 8: Total loss (RQ vs VQ)

In fig. 9, we can see the quantized codebook improvement using multiple quantizer (Number of quantizer = 3). For loss we used L2 norm to see how quantizer is reducing the quantization error in each step.

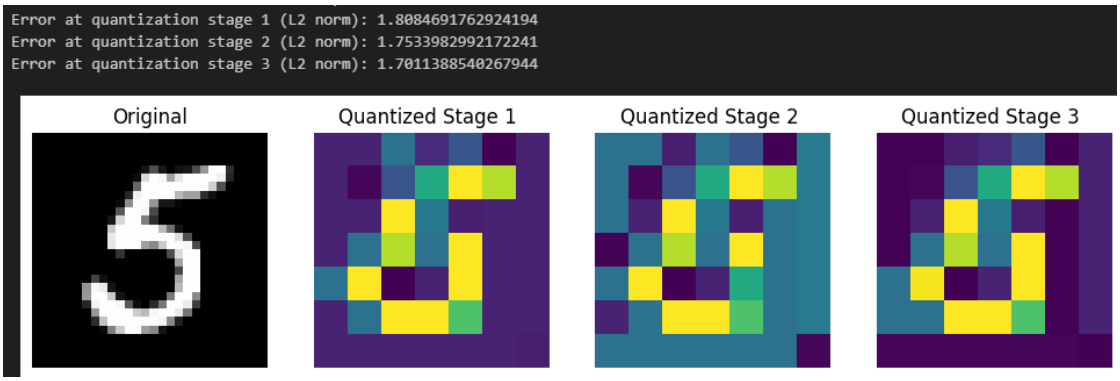

Fig. 9: Quantized values after each quantizer in latent dimension

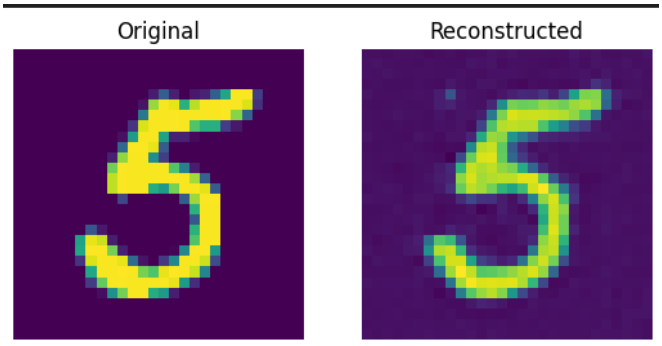

Fig. 10: Image reconstruction

I also analyzed the effect of using exponential moving average (EMA) and letting the model learn the codebook. In fig. 11, and fig. 12, we can see EMA performed better for the VQ case (Code book size = 256, number of quantizer = 1). I observed that for VQ VAE. The model sometimes became unstable due to the hard assignment not being differentiable. However, for RQ there was not a huge difference in losses due to use of multiple quantization layers.

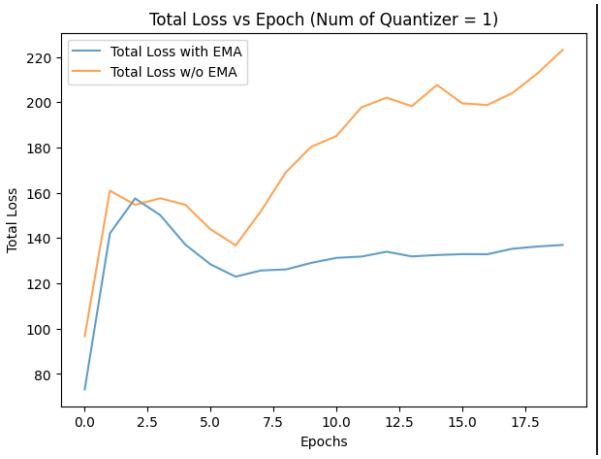

Fig 11: Total loss vs Epoch

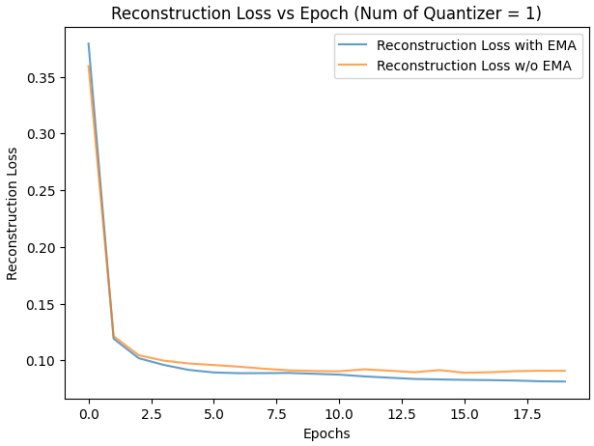

Fig 12: Reconstruction loss vs Epoch

Due to limited complexity RQ transformer is trained on only 2000 images and 200 epoch. The training curve can be seen in fig. 13.

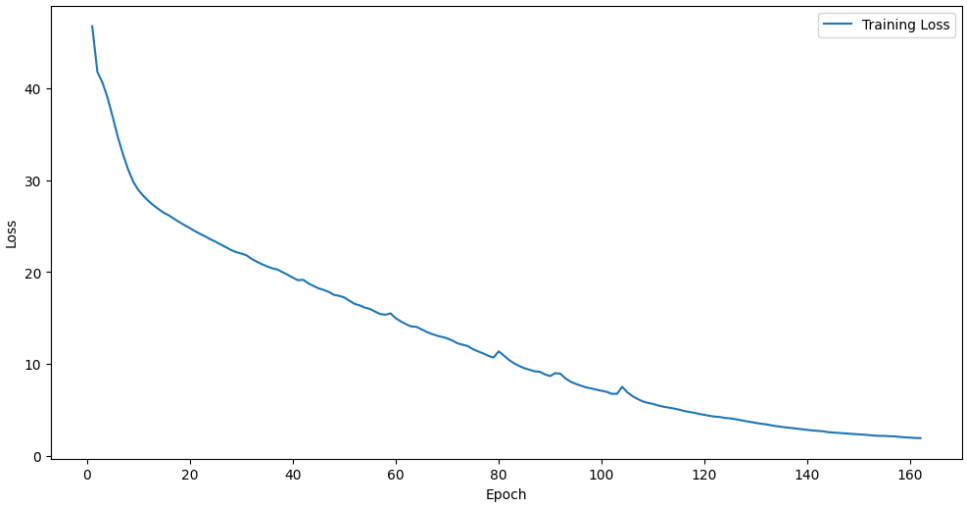

Fig 13: Total loss vs Epoch for RQ transformer

Using this autoregressive model a few samples were generated. It doesn't look good as transformer model and takes a lot of time to train and I had limited resources.

Fig. 14: Generated images using RQ transformer

Summary:

In this project, we showed that RQ-VAE performs much better than VQ-VAE while keeping the complexity the same. We also explored how Exponential Moving Average (EMA) helps fix instability problems in VQ-VAE, making the training process more stable. Additionally, we looked into the benefits of learning discrete representations instead of continuous ones, which is commonly used in audio compression. The main goal was to demonstrate how RQ-VAE addresses the quantization error problem in VQ-VAE, especially when using many codebooks. We also explored how autoregressive models can be used to generate high-quality samples. Overall, the project aims to improve the performance of Variational Autoencoders (VAEs) for generating high-resolution images by solving issues like posterior collapse and provide a solid comparison between VQ and RQ.

The initial code is taken from [6] and updated to get the desired implementation.

References:

[1] Van Den Oord, A. and Vinyals, O., 2017. Neural discrete representation learning. *Advances in neural information processing systems*, *30*.

[2] Lucas, J., Tucker, G., Grosse, R.B. and Norouzi, M., 2019. Don't blame the elbo! a linear vae perspective on posterior collapse. *Advances in Neural Information Processing Systems*, *32*.

[3] https://en.wikipedia.org/wiki/Nearest_neighbor_search#:~:text=Nearest%20neighbor%20search%20(NNS)%2C,the%20larger%20the%20function%20values.

[4] Lee, D., Kim, C., Kim, S., Cho, M. and Han, W.S., 2022. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 11523-11532).

[5] https://github.com/lucidrains/RQ-Transformer?tab=readme-ov-file

[6] https://keras.io/examples/generative/vq_vae/