Anomaly Detection Using MNIST Hand Written Digits With VAE's

Group-3

Maskani Naveen Yadav - 20201CEI0025

Katipally Yashwanth Reddy - 20201CEI0039

Peram Mahendra Reddy - 20201CEI0112

Indla MadhanKumar - 20201CEI0136

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, Model
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load and preprocess MNIST dataset
(x_train, _), _ = mnist.load_data()
x_train = x_train.astype('float32') / 255.0
x_train = np.expand_dims(x_train, -1)
```

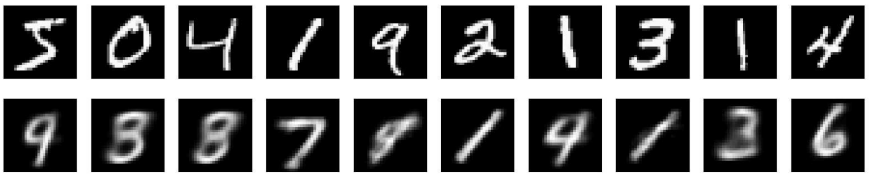
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
# Define the Variational Autoencoder (VAE) model
class VAE(Model):
    def __init__(self, latent_dim):
        super(VAE, self).__init__()
        self.latent_dim = latent_dim

        # Encoder
        self.encoder = tf.keras.Sequential([
            layers.Input(shape=(28, 28, 1)),
            layers.Flatten(),
            layers.Dense(256, activation='relu'),
            layers.Dense(128, activation='relu'),
            layers.Dense(latent_dim * 2)  # Outputting mean and log variance
        ])

        # Decoder
        self.decoder = tf.keras.Sequential([
            layers.Input(shape=(latent_dim,)),
            layers.Dense(128, activation='relu'),
            layers.Dense(256, activation='relu'),
            layers.Dense(28 * 28, activation='sigmoid'),
            layers.Reshape((28, 28, 1))
        ])

    def reparameterize(self, mean, logvar):
        eps = tf.random.normal(shape=tf.shape(mean))
        return eps * tf.exp(logvar * 0.5) + mean

    def call(self, x):
        # Encoding
        latent_params = self.encoder(x)
        mean, logvar = latent_params[:, :self.latent_dim], latent_params[:, self.latent_dim:]
        z = self.reparameterize(mean, logvar)

        # Decoding
        reconstructed_x = self.decoder(z)

        return reconstructed_x, mean, logvar
```

```
# Define the loss function for VAE
def vae_loss(recon_x, x, mean, logvar):
    # Reconstruction loss (binary cross entropy)
    reconstruction_loss = tf.keras.losses.binary_crossentropy(x, recon_x)
    reconstruction_loss = tf.reduce_sum(reconstruction_loss, axis=(1, 2))

    # KL divergence loss
    kl_divergence_loss = -0.5 * tf.reduce_sum(1 + logvar - tf.square(mean) - tf.exp(logvar), axis=1)

    return tf.reduce_mean(reconstruction_loss + kl_divergence_loss)


# Initialize parameters
latent_dim = 2  # Dimension of latent space
lr = 1e-3  # Learning rate
epochs = 20  # Number of training epochs
batch_size = 128

# Create VAE model
vae = VAE(latent_dim)
optimizer = tf.keras.optimizers.Adam(lr)


# Training loop
for epoch in range(epochs):
    total_loss = 0
    num_batches = x_train.shape[0] // batch_size
    for i in range(num_batches):
        start_idx = i * batch_size
        end_idx = (i + 1) * batch_size
        batch_x = x_train[start_idx:end_idx]

        with tf.GradientTape() as tape:
            recon_batch, mean, logvar = vae(batch_x)
            loss = vae_loss(recon_batch, batch_x, mean, logvar)
        gradients = tape.gradient(loss, vae.trainable_variables)
        optimizer.apply_gradients(zip(gradients, vae.trainable_variables))

        total_loss += loss

    print(f"Epoch {epoch+1}, Loss: {total_loss / num_batches}")
```

```
Epoch 1, Loss: 196.12351989746094
Epoch 2, Loss: 170.01319885253906
Epoch 3, Loss: 165.23135375976562
Epoch 4, Loss: 161.5552978515625
Epoch 5, Loss: 158.65994262695312
Epoch 6, Loss: 156.51344299316406
Epoch 7, Loss: 154.7811279296875
Epoch 8, Loss: 153.07965087890625
Epoch 9, Loss: 151.554443359375
Epoch 10, Loss: 150.358642578125
Epoch 11, Loss: 149.4595184326172
Epoch 12, Loss: 148.47784423828125
Epoch 13, Loss: 147.72445678710938
Epoch 14, Loss: 146.98626708984375
Epoch 15, Loss: 146.39459228515625
Epoch 16, Loss: 145.9656982421875
Epoch 17, Loss: 145.46951293945312
Epoch 18, Loss: 144.8653564453125
Epoch 19, Loss: 144.5868377685547
Epoch 20, Loss: 144.12124633789062
```

```
# Visualize reconstructed digits
n = 10  # Number of digits to visualize
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original images
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_train[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Reconstructed images
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(recon_batch[i].numpy().reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



```
# Anomaly detection
test_data = np.random.randn(1, 28, 28, 1).astype('float32')  # Generate random digit (anomaly)
reconstructed_test_data, mean, logvar = vae(test_data)
loss = vae_loss(reconstructed_test_data, test_data, mean, logvar)
print("Reconstruction loss for anomaly:", loss.numpy())
```

```
Reconstruction loss for anomaly: 150.83191
```

```
# Visualize reconstructed test data
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Test Data')
plt.imshow(test_data.squeeze(), cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Reconstructed Test Data')
plt.imshow(reconstructed_test_data.numpy().squeeze(), cmap='gray')
plt.axis('off')
plt.show()
```