# Building a simple Generative Adversarial Network (GAN) using TensorFlow

1. **Set Hyperparameters and Random Seed**
   The random seed is set for reproducibility. Hyperparameters like latent dimension, epochs, batch size, buffer size, and image shape (28x28 grayscale) are defined.

2. **Load and Preprocess the MNIST Dataset**
   The MNIST dataset is loaded and reshaped to match the input requirements of the GAN. The images are normalized to a range of [-1, 1] for better training stability, and the dataset is batched and shuffled.

3. **Define the Generator Model**
   The generator network is built using a series of dense layers, LeakyReLU activations, Batch Normalization layers, and a final dense layer that reshapes the output into the MNIST image shape. The output activation function is `tanh` to scale pixel values between -1 and 1.

4. **Define the Discriminator Model**
   The discriminator is built as a binary classifier. It takes the image as input, flattens it, and passes it through a few dense layers with LeakyReLU activations, and ends with a sigmoid activation to predict whether the image is real or fake.

5. **Loss Functions**

   Discriminator Loss: Binary cross-entropy loss is computed for both real and fake images. The goal of the discriminator is to classify real images as 1 and fake images as 0.

   Generator Loss: The generator's goal is to fool the discriminator, so its loss is calculated as binary cross-entropy with a target of 1 (since the generator wants to make fake images appear real).

6. **Initialize Models and Optimizers**
   The generator and discriminator are initialized using the defined models. Adam optimizers are used for both networks with a learning rate of `1e-4`.

7. **Training Step Function**
   **In each training step:**

   - Noise is generated as the input to the generator.
   - The generator creates fake images from the noise.
   - The discriminator evaluates both real and fake images.
   - Losses for both generator and discriminator are calculated, and gradients are updated using backpropagation.

8. **Generate and Save Images**
   At every 10th epoch, images are generated using a fixed input (`seed`) to visualize how the generator is learning over time. These images are saved as PNG files.

9. **Training Loop**
   The GAN is trained for the specified number of epochs (50 in this case). For each batch of images from the dataset, the `train_step` function is executed. After every 10 epochs, generated images are saved.

10. **Final Image Generation and Model Saving**
   After training, the generator produces final images, and the model is saved to a file (`generator_model.h5`). The final generated images are also displayed.