```
%  load dataset
[XTrain,YTrain,anglesTrain] = digitTrain4DArrayData;

% Hyperparameters
num_epochs = 100;
minibatchsize = 256;

learnRate = 0.0002;
gradientDecayFactor = 0.5;
squaredGradientDecayFactor = 0.999;

%Define Generator
layers = [
    featureInputLayer(128,"Name","sequence")
    fullyConnectedLayer(12544,"Name","fc")
    batchNormalizationLayer("Name","batchnorm")
    leakyReluLayer(0.01,"Name","leakyrelu")

    functionLayer(@(x) dlarray(reshape(x, 7,7,256, []), "SSCB"), ...
    Formattable=true, Acceleratable=true)

    transposedConv2dLayer([5 5],128,"Name","transposed-conv", ...
    "BiasLearnRateFactor",0,"Cropping","same")
    batchNormalizationLayer("Name","batchnorm_1")
    leakyReluLayer(0.01,"Name","leakyrelu_1")
    transposedConv2dLayer([5 5],64,"Name","transposed-conv_1", ...
    "BiasLearnRateFactor",0,"Cropping","same","Stride",[2 2])
    batchNormalizationLayer("Name","batchnorm_2")
    leakyReluLayer(0.01,"Name","leakyrelu_2")
    transposedConv2dLayer([5 5],1,"Name","transposed-conv_2", ...
    "BiasLearnRateFactor",0,"Cropping","same","Stride",[2 2])
    sigmoidLayer()
    ];
generator = dlnetwork(layers)

% define Discriminator
disc_layers = [
    imageInputLayer([28 28 1],"Name","imageinput", ...
    "Normalization","none")

    convolution2dLayer([5 5],4,"Name","conv", ...
    "Padding","same","Stride",[2 2])
    leakyReluLayer(0.01,"Name","leakyrelu")
    batchNormalizationLayer()
    dropoutLayer(0.5,"Name","dropout")

    convolution2dLayer([5 5],8,"Name","conv_1", ...
    "Padding","same","Stride",[2 2])
    leakyReluLayer(0.01,"Name","leakyrelu_1")
    batchNormalizationLayer()
```

```
    dropoutLayer(0.5,"Name","dropout_1")

    flattenLayer("Name","flatten")

    fullyConnectedLayer(1)
    sigmoidLayer("Name","sigmoid")];
discriminator = dlnetwork(disc_layers)

latent_vector = dlarray(randn([128, 1], "single"), "CB");
pred = predict(generator, latent_vector)*255;
image(extractdata(pred))
title("Untrained model Output")

x = XTrain(:, :, 1);
image(x*255)
title("Sample Out of the dataset")

XTrain_datastore = arrayDatastore(XTrain, ...
    "IterationDimension",4)

mbq = minibatchqueue(XTrain_datastore, ...
    minibatchsize=minibatchsize, ...
    PartialMiniBatch="discard", ...
    MiniBatchFormat="SSBC");

% parameters for training
iteration = 0;
averageGradDisc = [];
averageSqGradDisc = [];

averageGradGen = [];
averageSqGradGen = [];

val_latent_vector = dlarray(randn([128, 16]), "CB");

if canUseGPU
    val_latent_vector = gpuArray(val_latent_vector);
end

%bar = waitbar(0,"training...")

for epoch = 1:num_epochs
    shuffle(mbq);

    epochloss_d = zeros([minibatchsize,1]);
    epochloss_g = zeros([minibatchsize,1]);

    while mbq.hasdata
        iteration = iteration + 1;
        data = next(mbq);
```

```
        % generate training Latent vector
        latent_vactor = dlarray(randn([128,minibatchsize]), "CB");
        if canUseGPU
            latent_vactor = gpuArray(latent_vactor);
        end
        fake_images = forward(generator, latent_vactor);

        % calulate gradients
        [lossD,gradientsD] = dlfeval(@modelLossD, discriminator, ...
            data, fake_images);
        %apply discriminator gradients
        [discriminator, averageGradDisc, averageSqGradDisc] = adamupdate(...
            discriminator, ...
            gradientsD, averageGradDisc, ...
            averageSqGradDisc, iteration, ...
            learnRate, gradientDecayFactor, ...
            squaredGradientDecayFactor);

        latent_vactor = dlarray(randn([128,minibatchsize]), "CB");
        % generate training Latent vector
        if canUseGPU
            latent_vactor = gpuArray(latent_vactor);
        end

        [fake_images, stateG] = forward(generator, latent_vactor);
        % calculate Generator gradients
        [lossG,gradientsG] = dlfeval(@modelLossG, generator, ...
            discriminator, fake_images);
        % apply generator gradients
        [generator, averageGradGen, averageSqGradGen] = adamupdate( ...
            generator, ...
            gradientsG, averageGradGen, ...
            averageSqGradGen, iteration, ...
            learnRate, gradientDecayFactor, ...
            squaredGradientDecayFactor);

        generator.State = stateG;
        % save loss
        epochloss_d(epoch) = lossD;
        epochloss_g(epoch) = lossG;
    end
    %generate validation images and print loss
    val_images = extractdata(forward(generator, val_latent_vector)*255);
    fprintf("Discriminator: %0.5f; Generator: %0.5f\n", mean(epochloss_d), ...
        mean(epochloss_g))
    f = figure;
    for i = 1:16
        subplot(4, 4, i)
        image(val_images(:,:,i))
```

```
    end
    %waitbar(epoch/num_epochs, bar, "training...")
end
```

```
function [loss, grad] = modelLossD(net, images, generated_images)
    y_real = forward(net, images);
    loss_real = log(y_real);

    y_fake = forward(net, generated_images);
    loss_fake = log(1-y_fake);

    loss = - mean(loss_real) - mean(loss_fake);
    grad = dlgradient(loss, net.Learnables, RetainData=true);
end

function [loss, grad] = modelLossG(net, discriminator, generated_images)
    y_fake = forward(discriminator, generated_images);
    loss = - mean(log(y_fake));
    grad = dlgradient(loss, net.Learnables);
end
```