clc;

clear;

matlabpath='C:\Users\Acer\Desktop\deep'

data=fullfile(matlabpath,'F\_dataset\_299 size')

imds=imageDatastore(data,'IncludeSubfolders',true,'Labelsource','foldernames');

count=imds.countEachLabel;

augmenter = imageDataAugmenter('RandXReflection',true);

augimds = augmentedImageDatastore([64 64],imds,'DataAugmentation',augmenter);

filterSize = 5;

numFilters = 64;

numLatentInputs = 100;

projectionSize = [4 4 512];

layersGenerator = [

imageInputLayer([1 1 numLatentInputs],'Normalization','none','Name','in')

projectAndReshapeLayer(projectionSize,numLatentInputs,'proj');

transposedConv2dLayer(filterSize,4\*numFilters,'Name','tconv1')

batchNormalizationLayer('Name','bnorm1')

reluLayer('Name','relu1')

transposedConv2dLayer(filterSize,2\*numFilters,'Stride',2,'Cropping','same','Name','tconv2')

batchNormalizationLayer('Name','bnorm2')

reluLayer('Name','relu2')

transposedConv2dLayer(filterSize,numFilters,'Stride',2,'Cropping','same','Name','tconv3')

batchNormalizationLayer('Name','bnorm3')

reluLayer('Name','relu3')

transposedConv2dLayer(filterSize,3,'Stride',2,'Cropping','same','Name','tconv4')

tanhLayer('Name','tanh')];

lgraphGenerator = layerGraph(layersGenerator);

dlnetGenerator = dlnetwork(lgraphGenerator);

dropoutProb = 0.5;

numFilters = 64;

scale = 0.2;

inputSize = [64 64 3];

filterSize = 5;

layersDiscriminator = [

imageInputLayer(inputSize,'Normalization','none','Name','in')

dropoutLayer(0.5,'Name','dropout')

convolution2dLayer(filterSize,numFilters,'Stride',2,'Padding','same','Name','conv1')

leakyReluLayer(scale,'Name','lrelu1')

convolution2dLayer(filterSize,2\*numFilters,'Stride',2,'Padding','same','Name','conv2')

batchNormalizationLayer('Name','bn2')

leakyReluLayer(scale,'Name','lrelu2')

convolution2dLayer(filterSize,4\*numFilters,'Stride',2,'Padding','same','Name','conv3')

batchNormalizationLayer('Name','bn3')

leakyReluLayer(scale,'Name','lrelu3')

convolution2dLayer(filterSize,8\*numFilters,'Stride',2,'Padding','same','Name','conv4')

batchNormalizationLayer('Name','bn4')

leakyReluLayer(scale,'Name','lrelu4')

convolution2dLayer(4,1,'Name','conv5')];

lgraphDiscriminator = layerGraph(layersDiscriminator);

dlnetDiscriminator = dlnetwork(lgraphDiscriminator);

numEpochs = 100;

miniBatchSize = 15;

augimds.MiniBatchSize = miniBatchSize;

learnRate = 0.0001;

gradientDecayFactor = 0.5;

squaredGradientDecayFactor = 0.999;

executionEnvironment = "auto";

flipFactor = 0.3;

validationFrequency = 100;

trailingAvgGenerator = [];

trailingAvgSqGenerator = [];

trailingAvgDiscriminator = [];

trailingAvgSqDiscriminator = [];

numValidationImages = 25;

ZValidation = randn(1,1,numLatentInputs,numValidationImages,'single');

dlZValidation = dlarray(ZValidation,'SSCB');

if (executionEnvironment == "auto" && canUseGPU) || executionEnvironment == "gpu"

dlZValidation = gpuArray(dlZValidation);

end

f = figure;

f.Position(3) = 2\*f.Position(3);

imageAxes = subplot(1,2,1);

scoreAxes = subplot(1,2,2);

lineScoreGenerator = animatedline(scoreAxes,'Color',[0 0.447 0.741]);

lineScoreDiscriminator = animatedline(scoreAxes, 'Color', [0.85 0.325 0.098]);

legend('Generator','Discriminator');

ylim([0 1])

xlabel("Iteration")

ylabel("Score")

grid on

iteration = 0;

start = tic;

% Loop over epochs.

for epoch = 1:numEpochs

% Reset and shuffle datastore.

reset(augimds);

augimds = shuffle(augimds);

% Loop over mini-batches.

while hasdata(augimds)

iteration = iteration + 1;

% Read mini-batch of data.

data = read(augimds);

% Ignore last partial mini-batch of epoch.

if size(data,1) < miniBatchSize

continue

end

% Concatenate mini-batch of data and generate latent inputs for the

% generator network.

X = cat(4,data{:,1}{:});

X = single(X);

Z = randn(1,1,numLatentInputs,size(X,4),'single');

% Rescale the images in the range [-1 1].

X = rescale(X,-1,1,'InputMin',0,'InputMax',255);

% Convert mini-batch of data to dlarray and specify the dimension labels

% 'SSCB' (spatial, spatial, channel, batch).

dlX = dlarray(X, 'SSCB');

dlZ = dlarray(Z, 'SSCB');

% If training on a GPU, then convert data to gpuArray.

if (executionEnvironment == "auto" && canUseGPU) || executionEnvironment == "gpu"

dlX = gpuArray(dlX);

dlZ = gpuArray(dlZ);

end

% Evaluate the model gradients and the generator state using

% dlfeval and the modelGradients function listed at the end of the

% example.

[gradientsGenerator, gradientsDiscriminator, stateGenerator, scoreGenerator, scoreDiscriminator] = ...

dlfeval(@modelGradients, dlnetGenerator, dlnetDiscriminator, dlX, dlZ, flipFactor);

dlnetGenerator.State = stateGenerator;

% Update the discriminator network parameters.

[dlnetDiscriminator,trailingAvgDiscriminator,trailingAvgSqDiscriminator] = ...

adamupdate(dlnetDiscriminator, gradientsDiscriminator, ...

trailingAvgDiscriminator, trailingAvgSqDiscriminator, iteration, ...

learnRate, gradientDecayFactor, squaredGradientDecayFactor);

% Update the generator network parameters.

[dlnetGenerator,trailingAvgGenerator,trailingAvgSqGenerator] = ...

adamupdate(dlnetGenerator, gradientsGenerator, ...

trailingAvgGenerator, trailingAvgSqGenerator, iteration, ...

learnRate, gradientDecayFactor, squaredGradientDecayFactor);

% Every validationFrequency iterations, display batch of generated images using the

% held-out generator input

if mod(iteration,validationFrequency) == 0 || iteration == 1

% Generate images using the held-out generator input.

dlXGeneratedValidation = predict(dlnetGenerator,dlZValidation);

% Tile and rescale the images in the range [0 1].

I = imtile(extractdata(dlXGeneratedValidation));

I = rescale(I);

% Display the images.

subplot(1,2,1);

image(imageAxes,I)

xticklabels([]);

yticklabels([]);

title("Generated Images");

end

% Update the scores plot

subplot(1,2,2)

addpoints(lineScoreGenerator,iteration,...

double(gather(extractdata(scoreGenerator))));

addpoints(lineScoreDiscriminator,iteration,...

double(gather(extractdata(scoreDiscriminator))));

% Update the title with training progress information.

D = duration(0,0,toc(start),'Format','hh:mm:ss');

title(...

"Epoch: " + epoch + ", " + ...

"Iteration: " + iteration + ", " + ...

"Elapsed: " + string(D))

drawnow

end

end