imageFolder = 'directory';

datasetFolder = fullfile(imageFolder);

imds = imageDatastore(imageFolder,IncludeSubfolders=true);

augmenter = imageDataAugmenter(RandXReflection=true);

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

filterSize = 5;

numFilters = 64;

numLatentInputs = 100;

projectionSize = [4 4 512];

layersGenerator = [

featureInputLayer(numLatentInputs)

projectAndReshapeLayer(projectionSize)

transposedConv2dLayer(filterSize,4\*numFilters)

batchNormalizationLayer

reluLayer

transposedConv2dLayer(filterSize,2\*numFilters,Stride=2,Cropping="same")

batchNormalizationLayer

reluLayer

transposedConv2dLayer(filterSize,numFilters,Stride=2,Cropping="same")

batchNormalizationLayer

reluLayer

transposedConv2dLayer(filterSize,3,Stride=2,Cropping="same")

tanhLayer];

netG = dlnetwork(layersGenerator);

dropoutProb = 0.5;

numFilters = 64;

scale = 0.2;

inputSize = [64 64 3];

filterSize = 5;

layersDiscriminator = [

imageInputLayer(inputSize,Normalization="none")

dropoutLayer(dropoutProb)

convolution2dLayer(filterSize,numFilters,Stride=2,Padding="same")

leakyReluLayer(scale)

convolution2dLayer(filterSize,2\*numFilters,Stride=2,Padding="same")

batchNormalizationLayer

leakyReluLayer(scale)

convolution2dLayer(filterSize,4\*numFilters,Stride=2,Padding="same")

batchNormalizationLayer

leakyReluLayer(scale)

convolution2dLayer(filterSize,8\*numFilters,Stride=2,Padding="same")

batchNormalizationLayer

leakyReluLayer(scale)

convolution2dLayer(4,1)

sigmoidLayer];

netD = dlnetwork(layersDiscriminator);

numEpochs = 500;

miniBatchSize = 128;

learnRate = 0.0002;

gradientDecayFactor = 0.5;

squaredGradientDecayFactor = 0.999;

flipProb = 0.35;

validationFrequency = 100;

augimds.MiniBatchSize = miniBatchSize;

mbq = minibatchqueue(augimds, ...

MiniBatchSize=miniBatchSize, ...

PartialMiniBatch="discard", ...

MiniBatchFcn=@preprocessMiniBatch, ...

MiniBatchFormat="SSCB");

trailingAvgG = [];

trailingAvgSqG = [];

trailingAvg = [];

trailingAvgSqD = [];

numValidationImages = 25;

ZValidation = randn(numLatentInputs,numValidationImages,"single");

ZValidation = dlarray(ZValidation,"CB");

if canUseGPU

ZValidation = gpuArray(ZValidation);

end

f = figure;

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

imageAxes = subplot(1,2,1);

scoreAxes = subplot(1,2,2);

C = colororder;

lineScoreG = animatedline(scoreAxes,Color=C(1,:));

lineScoreD = animatedline(scoreAxes,Color=C(2,:));

legend("Generator","Discriminator");

ylim([0 1])

xlabel("Iteration","FontSize",32)

ylabel("Score","FontSize",32)

set(gca,'FontSize',30)

grid off

iteration = 0;

start = tic;

% Loop over epochs.

for epoch = 1:numEpochs

% Reset and shuffle datastore.

shuffle(mbq);

% Loop over mini-batches.

while hasdata(mbq)

iteration = iteration + 1;

% Read mini-batch of data.

X = next(mbq);

% Generate latent inputs for the generator network. Convert to

% dlarray and specify the format "CB" (channel, batch). If a GPU is

% available, then convert latent inputs to gpuArray.

Z = randn(numLatentInputs,miniBatchSize,"single");

Z = dlarray(Z,"CB");

if canUseGPU

Z = gpuArray(Z);

end

% Evaluate the gradients of the loss with respect to the learnable

% parameters, the generator state, and the network scores using

% dlfeval and the modelLoss function.

[~,~,gradientsG,gradientsD,stateG,scoreG,scoreD] = ...

dlfeval(@modelLoss,netG,netD,X,Z,flipProb);

netG.State = stateG;

% Update the discriminator network parameters.

[netD,trailingAvg,trailingAvgSqD] = adamupdate(netD, gradientsD, ...

trailingAvg, trailingAvgSqD, iteration, ...

learnRate, gradientDecayFactor, squaredGradientDecayFactor);

% Update the generator network parameters.

[netG,trailingAvgG,trailingAvgSqG] = adamupdate(netG, gradientsG, ...

trailingAvgG, trailingAvgSqG, 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.

XGeneratedValidation = predict(netG,ZValidation);

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

I = imtile(extractdata(XGeneratedValidation));

I = rescale(I);

% Display the images.

subplot(1,2,1);

image(imageAxes,I)

subplot(1,2,2)

ylim([0.00 1.00])

xticklabels([]);

yticklabels([]);

title("Generated Images");

end

% Update the scores plot.

subplot(1,2,2)

scoreG = double(extractdata(scoreG));

addpoints(lineScoreG,iteration,scoreG);

scoreD = double(extractdata(scoreD));

addpoints(lineScoreD,iteration,scoreD);

% 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

save('filename.mat', 'netG', 'numLatentInputs','netD')

numObservations = 25;

ZNew = randn(numLatentInputs,numObservations,"single");

ZNew = dlarray(ZNew,"CB");

if canUseGPU

ZNew = gpuArray(ZNew);

end

XGeneratedNew = predict(netG,ZNew);

I = imtile(extractdata(XGeneratedNew));

I = rescale(I);

figure

image(I)

axis off

title("Generated Images")

function [lossG,lossD,gradientsG,gradientsD,stateG,scoreG,scoreD] = ...

modelLoss(netG,netD,X,Z,flipProb)

% Calculate the predictions for real data with the discriminator network.

YReal = forward(netD,X);

% Calculate the predictions for generated data with the discriminator

% network.

[XGenerated,stateG] = forward(netG,Z);

YGenerated = forward(netD,XGenerated);

% Calculate the score of the discriminator.

scoreD = (mean(YReal) + mean(1-YGenerated)) / 2;

% Calculate the score of the generator.

scoreG = mean(YGenerated);

% Randomly flip the labels of the real images.

numObservations = size(YReal,4);

idx = rand(1,numObservations) < flipProb;

YReal(:,:,:,idx) = 1 - YReal(:,:,:,idx);

% Calculate the GAN loss.

[lossG, lossD] = ganLoss(YReal,YGenerated);

% For each network, calculate the gradients with respect to the loss.

gradientsG = dlgradient(lossG,netG.Learnables,RetainData=true);

gradientsD = dlgradient(lossD,netD.Learnables);

end

function [lossG,lossD] = ganLoss(YReal,YGenerated)

% Calculate the loss for the discriminator network.

lossD = -mean(log(YReal)) - mean(log(1-YGenerated));

% Calculate the loss for the generator network.

lossG = -mean(log(YGenerated));

end

function X = preprocessMiniBatch(data)

% Concatenate mini-batch

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

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

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

End

(*Train Generative Adversarial Network (GAN) - MATLAB & Simulink*, n.d.)

References

*Train Generative Adversarial Network (GAN)—MATLAB & Simulink*. (n.d.). Retrieved 24 October 2024, from https://www.mathworks.com/help/deeplearning/ug/train-generative-adversarial-network.html