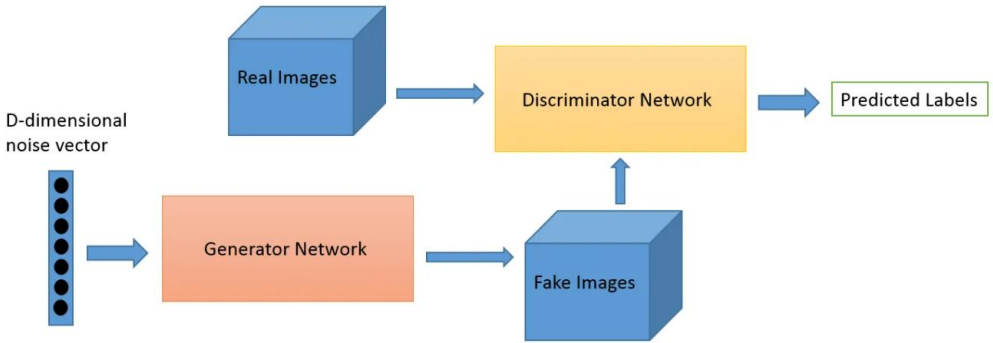
GAN


Real Images
Discriminator Network
Predicted Labels
D-dimensional
noise vector
Generator Network
Fake Images

关于生成对抗网络

生成对抗网络(GANs) 是一种包含两个网络的深度神经网络结构，将一个网络与另一个网络相互对立(因此称为“对抗‘).

在2014年， GANs由Goodfellow 和Yoshua Bengio等研究者提出来，提及GANs， Yann LeCun 称对抗训练为“过去10年机器学习领域最有趣的idea”

GANs 的潜力巨大，因为它们能去学习模仿任何数据分布，因此，GANs能被教导在任何领域创造类似于我们的世界，比如图像、音乐、演讲、散文。在某种意义上，GANs是机器人艺术家，它们的输出令人印象深刻。

生成算法 VS 判别算法

为了理解GANs， 需要知道生成算法是如何工作的，为此，可以拿判别算法与之进行对比。判别算法尝试去区分输入的数据：给它们数据实例的特征，它们将预测这些数据所属的标签或者类别。

比如说，给定一张图片，判别算法能够判别这张图片属于哪个特定的分类。图片分类就是标签，从这张图片中提取的卷积特征图就是特征。当以数学来表述这个问题，标签被称为y，特征被称为x 。 公式 p(y|x) 的意思是“给定x，y发生的概率”。

生成算法 VS 判别算法

判别算法是将特征映射为标签，它们只关心这种相关性。生成方法所做的事情恰恰是相反的，生成并非是由给定特定的特征去预测标签，而是尝试由给定的标签去预测特征。

生成算法在努力回答这样一个问题：假定这张图片的分类是轮船，那么这张图片应该是什么样的？ 判别模型关心y和x的关系，但生成模型关心的是“怎样得到x”。换句话说，判别模型是去学习类之间的界限，生成模型需要对某一类的分布进行建模。

生成算法的任务

具体来说，假设我们现在有一些来自未知分布$P_r(x)$的可观测样本$x^1,x^2,...,x^n$，我们想做以下两件事情：

- 学习一个参数化的模型$P_\theta(x)$来近似未知分布$P_r(x)$；
- 基于学习模型$P_\theta(x)$生成一些样本，使得生成样本和真实样本尽可能接近。

在高维空间中，直接建模$P_r(x)$比较困难，通常通过引入隐变量z来简化模型，如此密度估计问题就转化为估计变量(x,z)的两个局部条件概率$P_\theta(z)$和$P_\theta(x|z)$。得到上述两个局部条件概率后，生成数据的过程如下：

- 根据$P_\theta(z)$进行采样，得到样本z；
- 根据$P_\theta(x|z)$进行采样，得到样本x。

GAN思路

生成对抗网络GAN借助神经网络，简单粗暴地解决了上述问题，既避免了密度估计，又降低了采样的难度。具体为：

- 从一个简单分布P_x（例如标准正态分布）中采样得到z；
- 利用神经网络G: z -> x ，即使得$G(z)\sim P_r(x)$。

GAN并不显式建模$P_r(x)$，而是建模其生成过程G(z)（令G(z)服从数据分布$P_r(x)$），因此GAN属于隐式密度模型。G(z)成为生成网络，一个完整的GAN还需要一个判别网络：尽量准确地判断一个样本是来自真实数据还是来自生成网络，生成网络的目的则是尽可能地欺骗判别网络，使其无法区分样本的来源。这两个网络的目标相互对抗。

判别网络

判别网络D(x;ϕ)的目的是区分一个样本x是来自真实分布$P_r(x)$还是生成模型$P_\theta(x)$，因此它其实就是一个二分类器，训练二分类器最常用的损失函数为交叉熵，令判别网络D(x;ϕ)的输出表示x来自真实数据分布的概率：

$$\min_{\phi} -(\mathbb{E}_x[y\log D(x;\phi)+(1-y)\log(1-D(x;\phi))])$$

假设P(x)由分布$P_r(x)$和$P_\theta(x)$等比例混合而成，即$P(x)=1/2(P_r(x)+P_\theta(x))$，则有

$$\begin{aligned}
\mathbb{E}_x[y\log D(x;\phi)+(1-y)\log(1-D(x;\phi))] &= \int p(x)[y\log D(x;\phi)+(1-y)\log(1-D(x;\phi))]dx \\
&= \int \frac{1}{2}(p_r(x)+p_\theta(x))[y\log D(x;\phi)+(1-y)\log(1-D(x;\phi))]dx \\
&= \int \frac{1}{2}p_r(x)\log D(x;\phi)dx + \int \frac{1}{2}p_\theta(x)\log(1-D(x;\phi))dx \\
&= \frac{1}{2}\mathbb{E}_{x\sim p_r(x)}[\log D(x;\phi)] + \frac{1}{2}\mathbb{E}_{x\sim p_\theta(x)}[\log(1-D(x;\phi))]
\end{aligned}$$

生成网络

生成网络的目的是让判别网络将自己生成的样本判别为真实样本，即对于自己生成的样本，判别网络的输出（样本来自真实分布的概率）越大越好：

$$\max_{\theta} \mathbb{E}_{z\sim p(z)}\left[\log D(G(z;\theta),\phi)\right]$$
$$= \min_{\theta} \mathbb{E}_{z\sim p(z)}\left[1-\log D(G(z;\theta),\phi)\right]$$

把两个网络结合起来看，即把两个公式统一起来，那么生成对抗网络的目标函数其实是一个最小最大化游戏：

$$\min_{\theta}\max_{\phi}(\mathbb{E}_{\mathrm{x}\sim \mathrm{p_r(x)}}\left[\log D(x;\phi)\right] + \mathbb{E}_{\mathrm{x}\sim \mathrm{p}_\theta\mathrm{(x)}}\left[\log(1-D(x;\phi))\right])$$
$$= \min_{\theta}\max_{\phi}(\mathbb{E}_{\mathrm{x}\sim \mathrm{p_r(x)}}\left[\log D(x;\phi)\right] + \mathbb{E}_{\mathrm{z}\sim \mathrm{p(z)}}\left[\log(1-D(G(z;\theta);\phi))\right])$$

训练

$$\min_{\theta}\max_{\phi}(\mathbb{E}_{x\sim p_r(x)}[\log D(x;\phi)]+\mathbb{E}_{x\sim p_\theta(x)}[\log(1-D(x;\phi))])$$
$$=\min_{\theta}\max_{\phi}(\mathbb{E}_{x\sim p_r(x)}[\log D(x;\phi)]+\mathbb{E}_{z\sim p(z)}[\log(1-D(G(z;\theta);\phi))])$$

GAN的训练过程：

- 先看内层的最大化部分，固定生成网络，找出当前最优的判别网络；
- 再看外层的最小化部分，固定判别网络，找出当前最优的生成网络。

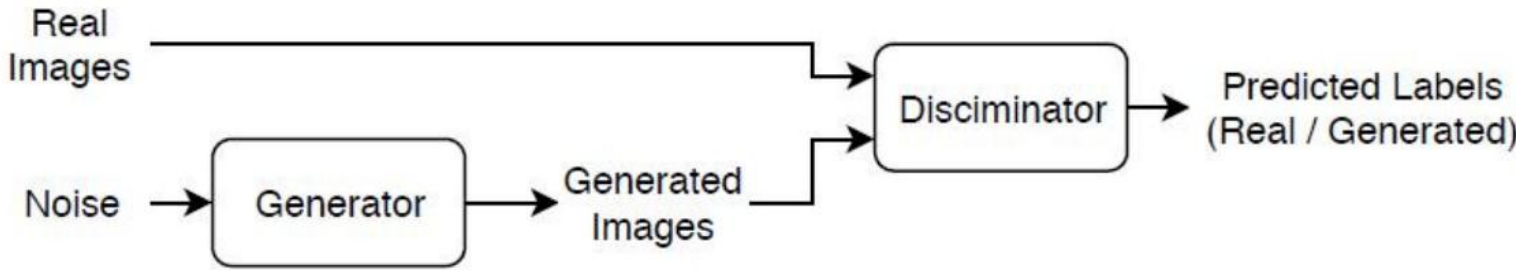
在实际训练中，通常需要平衡好两个网络的能力，每次迭代判别网络的能力应该比生成网络强一些，但又不能强太多，因此通常在一次迭代中，判别网络更新K次，生成网络才更新一次。

Basic GAN with Matlab

- Generative Adversarial Network (GAN)
- GAN example
- Conditional Generative Adversarial Network (cGAN)
- cGAN example

About Generative Adversarial Network (GAN)

- A generative adversarial network (GAN) is a type of deep learning network that can generate data with similar characteristics as the input real data. A GAN consists of two networks that train together:
  - Generator — Given a vector of random values (latent inputs) as input, this network generates data with the same structure as the training data.
  - Discriminator — Given batches of data containing observations from both the training data, and generated data from the generator, this network attempts to classify the observations as "real" or "generated".



Train Generative Adversarial Network (GAN)

- To train a GAN, train both networks simultaneously to maximize the performance of both:
  - Train the generator to generate data that "fools" the discriminator.
  - Train the discriminator to distinguish between real and generated data.
- To optimize the performance of the generator, maximize the loss of the discriminator when given generated data. That is, the objective of the generator is to generate data that the discriminator classifies as "real".

Train Generative Adversarial Network (GAN)

- To optimize the performance of the discriminator, minimize the loss of the discriminator when given batches of both real and generated data. That is, the objective of the discriminator is to not be "fooled" by the generator.
- Ideally, these strategies result in a generator that generates convincingly realistic data and a discriminator that has learned strong feature representations that are characteristic of the training data.

Train Generative Adversarial Network (GAN) for Image Generation

- Download and extract the Flowers data set.

```
url = 'http://download.tensorflow.org/example_images/flower_photos.tgz';
downloadFolder = tempdir;
    filename = fullfile(downloadFolder,'flower_dataset.tgz');

    imageFolder = fullfile(downloadFolder,'flower_photos');
    if ~exist(imageFolder,'dir')
       disp('Downloading Flowers data set (218 MB)...') websave(filename,url);
       untar(filename,downloadFolder)
    end
```

Load Training Data

- Create an image datastore containing the photos of the flowers.

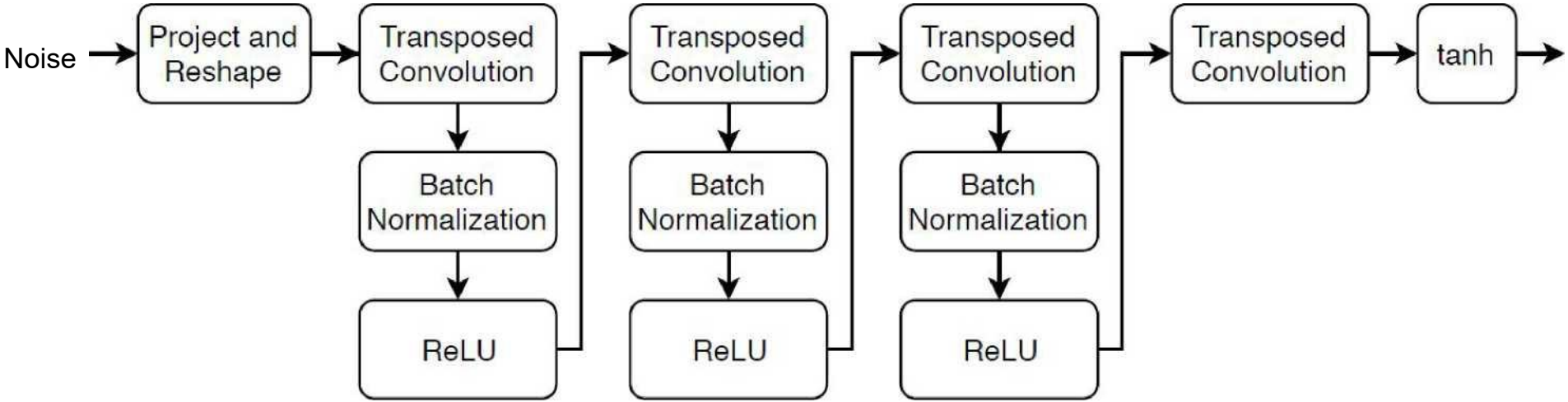
```
datasetFolder = fullfile(imageFolder);
imds = imageDatastore(datasetFolder, 'IncludeSubfolders',true);
```

- Augment the data to include random horizontal flipping and resize the images to have size 64-by-64.

```
augmenter = imageDataAugmenter('RandXReflection',true);
augimds = augmentedImageDatastore([64 64],imds,'DataAugmentation',augmenter);
```

Define Generator Network

- Define the following network architecture, which generates images from 1-by-1-by-100 arrays of random values:



Define Generator Network

- This network: Converts the 1-by-1-by-100 arrays of noise to 4-by-4-by-512 arrays using a project and reshape layer. Upscales the resulting arrays to 64-by-64-by-3 arrays using a series of transposed convolution layers with batch normalization and ReLU layers.
- Define this network architecture as a layer graph and specify the following network properties. For the transposed convolution layers, specify 5-by-5 filters with a decreasing number of filters for each layer, a stride of 2 and cropping of the output on each edge. For the final transposed convolution layer, specify three 5-by-5 filters corresponding to the three RGB channels of the generated images, and the output size of the previous layer. At the end of the network, include a tanh layer.

Define Generator Network

- To project and reshape the noise input, use the custom layer projectAndReshapeLayer, attached to this example as a supporting file. The projectAndReshapeLayer layer upscales the input using a fully connected operation and reshapes the output to the specified size.

```
filterSize = 5;
numFilters = 64;
numLatentInputs = 100;
projectionSize = [4 4 512];
layersGenerator =[
  imageInputLayer([1 1 numLatentInputs],'Normalization','none','Name','in')
  projectAndReshapeLayer(projectionSize,numLatentInputs,'proj'); % [4, 4, 512]
  transposedConv2dLayer(filterSize,4*numFilters,'Name','tconv1')
  batchNormalizationLayer('Name','bnorm1')
  reluLayer('Name','relu1') % [8, 8, 256]
  transposedConv2dLayer(filterSize,2*numFilters,'Stride',2,'Cropping','same','Name','tconv2')
  batchNormalizationLayer('Name','bnorm2')
  reluLayer('Name','relu2') % [16, 16, 128]
  transposedConv2dLayer(filterSize,numFilters,'Stride',2,'Cropping','same','Name','tconv3')
  batchNormalizationLayer('Name','bnorm3')
  reluLayer('Name','relu3') % [32, 32, 64]
  transposedConv2dLayer(filterSize,3,'Stride',2,'Cropping','same','Name','tconv4')
  tanhLayer('Name','tanh')]; % [64, 64, 3]
lgraphGenerator = layerGraph(layersGenerator);
```

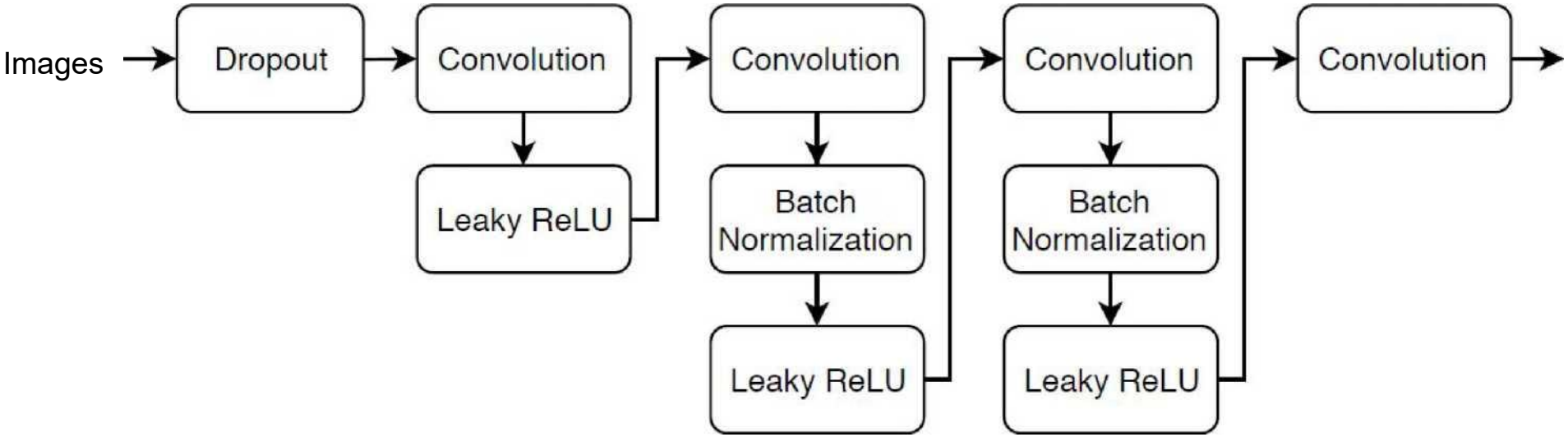
Define Generator Network

- To train the network with a custom training loop and enable automatic differentiation, convert the layer graph to a dlnetwork object.

```
dlnetGenerator = dlnetwork(lgraphGenerator);
```

Define Discriminator Network

- Define the following network, which classifies real and generated 64-by-64 images.



Define Discriminator Network

- Create a network that takes 64-by-64-by-3 images and returns a scalar prediction score using a series of convolution layers with batch normalization and leaky ReLU layers. Add noise to the input images using dropout.
  - For the dropout layer, specify a dropout probability of 0.5.
  - For the convolution layers, specify 5-by-5 filters with an increasing number of filters for each layer. Also specify a stride of 2 and a padding of the output.
  - For the leaky ReLU layers, specify a scale of 0.2.
  - For the final layer, specify a convolutional layer with one 4-by-4 filter.

```
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];
```

Define Discriminator Network

- To train the network with a custom training loop and enable automatic differentiation, convert the layer graph to a dlnetwork object.

```
dlnetDiscriminator = dlnetwork(lgraphDiscriminator);
```

Define Model Gradients, Loss Functions and Scores

- Create the function modelGradients, which takes as input the generator and discriminator networks, a mini-batch of input data, an array of random values and the flip factor, and returns the gradients of the loss with respect to the learnable parameters in the networks and the scores of the two networks.

Model Gradients Function

- The function modelGradients takes as input the generator and discriminator dlnetwork objects dlnetGenerator and dlnetDiscriminator, a mini-batch of input data dlX, an array of random values dlZ and the percentage of real labels to flip flipFactor, and returns the gradients of the loss with respect to the learnable parameters in the networks, the generator state, and the scores of the two networks. Because the discriminator output is not in the range [0,1], modelGradients applies the sigmoid function to convert it into probabilities.

```
function [gradientsGenerator, gradientsDiscriminator, stateGenerator, scoreGenerat
scoreDiscriminator]=...
        modelGradients(dlnetGenerator, dlnetDiscriminator, dlX, dlZ, flipFactor)
    % Calculate the predictions for real data with the discriminator network.
    dlYPred = forward(dlnetDiscriminator, dlX);
% Calculate the predictions for generated data with the discriminator network.
[dlXGenerated,stateGenerator] = forward(dlnetGenerator,dlZ);
dlYPredGenerated = forward(dlnetDiscriminator, dlXGenerated);
% Convert the discriminator outputs to probabilities.
probGenerated = sigmoid(dlYPredGenerated);
probReal = sigmoid(dlYPred);
    % Calculate the score of the discriminator.
scoreDiscriminator = ((mean(probReal)+mean(1-probGenerated))/2);
% Calculate the score of the generator.
    scoreGenerator = mean(probGenerated);
```

```
% Randomly flip a fraction of the labels of the real images.
numObservations = size(probReal,4);
idx = randperm(numObservations,floor(flipFactor * numObservations));
% Flip the labels
probReal(:,:,:,idx) = 1-probReal(:,:,:,idx);
% Calculate the GAN loss.

[lossGenerator, lossDiscriminator] = ganLoss(probReal,probGenerated);
% For each network, calculate the gradients with respect to the loss.
gradientsGenerator = dlgradient(lossGenerator,
dlnetGenerator.Learnables,'RetainData',true);
gradientsDiscriminator = dlgradient(lossDiscriminator, dlnetDiscriminator.Learnables);
end
```

Specify Training Options

- Train with a mini-batch size of 128 for 500 epochs. For larger datasets, you might not need to train for as many epochs.

```
numEpochs = 500;
miniBatchSize = 128;
```

- Specify the options for Adam optimization. For both networks, specify
  - A learning rate of 0.0002
  - A gradient decay factor of 0.5
  - A squared gradient decay factor of 0.999

```
learnRate = 0.0002;

gradientDecayFactor = 0.5;
squaredGradientDecayFactor = 0.999;
```

Specify Training Options

- If the discriminator learns to discriminate between real and generated images too quickly, then the generator may fail to train. To better balance the learning of the discriminator and the generator, add noise to the real data by randomly flipping the labels.
- Specify to flip 30% of the real labels. This means that 15% of the total number of labels are flipped during training. Note that this does not impair the generator as all the generated images are still labelled correctly.

```
flipFactor = 0.3;
```

- Display the generated validation images every 100 iterations.

```
validationFrequency = 100;
```

Train Model

- Use minibatchqueue to process and manage the mini-batches of images. For each mini-batch:
  - Use the custom mini-batch preprocessing function preprocessMiniBatch to rescale the images in the range [-1,1].
  - Discard any partial mini-batches with less than 128 observations.
  - Format the image data with the dimension labels 'SSCB' (spatial, spatial, channel, batch). By default, the minibatchqueue object converts the data to dlarray objects with underlying type single.

Train Model

```
augimds.MiniBatchSize = miniBatchSize;

executionEnvironment = "auto";

mbq = minibatchqueue(augimds,...
    'MiniBatchSize',miniBatchSize,...
    'PartialMiniBatch','discard',...
    'MiniBatchFcn', @preprocessMiniBatch,...
    MiniBatchFormat, SSCB,...
    OutputEnvironment,executionEnvironment);
```

Train Model

- Train the model using a custom training loop. Loop over the training data and update the network parameters at each iteration. To monitor the training progress, display a batch of generated images using a held-out array of random values to input into the generator as well as a plot of the scores.
- Initialize the parameters for Adam.

```
trailingAvgGenerator =[];
trailingAvgSqGenerator =[];
trailingAvgDiscriminator =[];
trailingAvgSqDiscriminator =[];
```

Train Model

- To monitor training progress, display a batch of generated images using a held-out batch of fixed arrays of random values fed into the generator and plot the network scores.
- Create an array of held-out random values.

```
numValidationImages = 25;
ZValidation = randn(1,1,numLatentInputs,numValidationImages,'single');
```

- Convert the data to dlarray objects and specify the dimension labels 'SSCB' (spatial, spatial, channel, batch).

```
dlZValidation = dlarray(ZValidation,'SSCB');
```

Train Model

- For GPU training, convert the data to gpuArray objects.

```
if (executionEnvironment == "auto" && canUseGPU) 11 executionEnvironment == "gpu"
   dlZValidation = gpuArray(dlZValidation);
end
```

- Initialize the training progress plots. Create a figure and resize it to have twice the width.

```
f = figure;
f.Position(3) = 2*f.Position(3);
```

- Create a subplot for the generated images and the network scores.

```
imageAxes = subplot(1,2,1);
scoreAxes = subplot(1,2,2);
```

Train Model

- Initialize the animated lines for the scores plot.

```
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("lteration") ylabel("Score") grid on
```

Train Model

- Train the GAN. For each epoch, shuffle the datastore and loop over mini-batches of data.
- For each mini-batch:
  - Evaluate the model gradients using dlfeval and the modelGradients function.
  - Update the network parameters using the adamupdate function.
  - Plot the scores of the two networks.
  - After every validationFrequency iterations, display a batch of generated images for a fixed held-out generator input.

- Training can take some time to run.

```
iteration = 0;   start = tic;
for epoch = 1:numEpochs   % Loop over epochs.
    shuffle(mbq);   % Reset and shuffle datastore.
    while hasdata(mbq)   % Loop over mini-batches.
      iteration = iteration + 1;
      X = next(mbq);   % Read mini-batch of data.
      % Generate latent inputs for the generator network. Convert to
      % dlarray and specify the format "CB" (channel, batch).
      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;
```

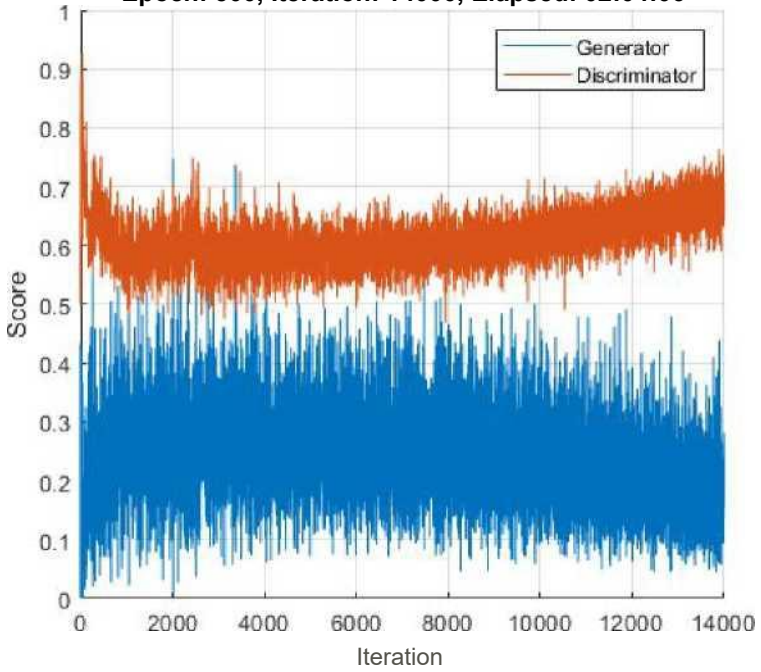
```
[netD,trailingAvg,trailingAvgSqD] = adamupdate(netD, gradientsD, ...
    trailingAvg, trailingAvgSqD, iteration, ...
    learnRate, gradientDecayFactor, squaredGradientDecayFactor);
[netG,trailingAvgG,trailingAvgSqG] = adamupdate(netG, gradientsG, ...
    trailingAvgG, trailingAvgSqG, iteration, ...
    learnRate, gradientDecayFactor, squaredGradientDecayFactor);
% Every validationFrequency iterations, display current images
if mod(iteration,validationFrequency) == 0 || iteration == 1
    XGeneratedValidation = predict(netG,ZValidation);
    % Tile and rescale the images in the range [0 1].
    I = imtile(extractdata(XGeneratedValidation));
    I = rescale(I);
    subplot(1,2,1);   % Display the images.
    image(imageAxes,I)
    xticklabels([]);
    yticklabels([]);
    title("Generated Images");
end
```

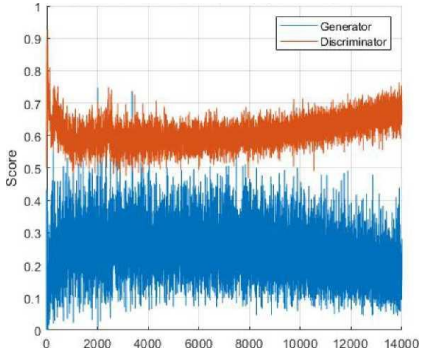
```
        % Update the scores plot.
        subplot(1,2,2)
        scoreG = double(extractdata(scoreG));
        addpoints(lineScoreG,iteration,scoreG);

        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
```

Train Model

Generated Images

Epoch: 500, Iteration: 14000, Elapsed: 02:04:06





- The discriminator has learned a strong feature representation that identifies real images among generated images. In turn, the generator has learned a similarly strong feature representation that allows it to generate realistic looking data.
- The training plot shows the scores of the generator and discriminator networks.

Generate New Images

- To generate new images, use the predict function on the generator with a dlarray object containing a batch of 1-by-1-by-100 arrays of random values. To display the images together, use the imtile function and rescale the images using the rescale function.
- Create a dlarray object containing a batch of 25 1-by-1-by-100 arrays of random values to input into the generator network.

```
ZNew = randn(1,1,numLatentInputs,25,'single');
dlZNew = dlarray(ZNew,'SSCB');
```

Generate New Images

- To generate images using the GPU, also convert the data to gpuArray objects.

```
if (executionEnvironment == "auto" && canUseGPU) | |
executionEnvironment == "gpu"
    dlZNew = gpuArray(dlZNew);
end
```

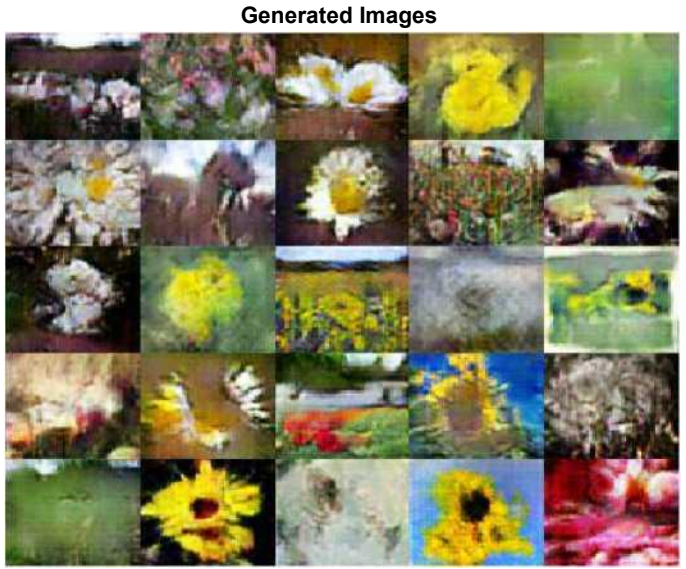
- Generate new images using the predict function with the generator and the input data.

```
dlXGeneratedNew = predict(dlnetGenerator,dlZNew);
```

Generate New Images

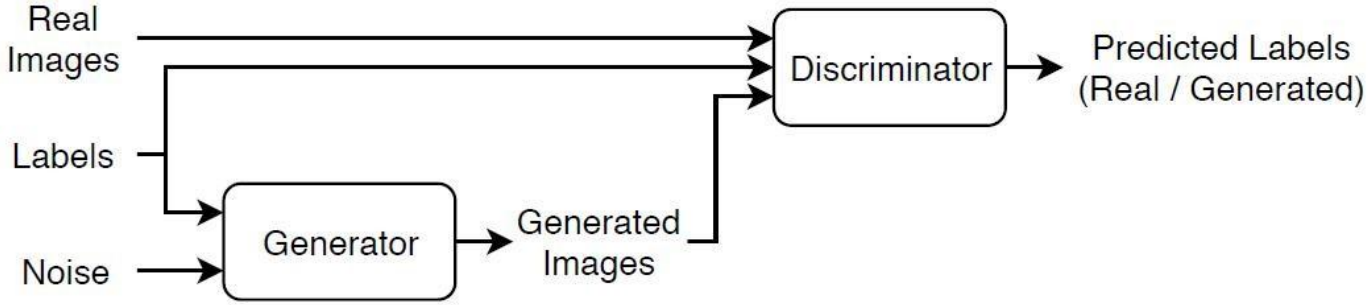
- Display the images.

```
I = imtile(extractdata(dlXGeneratedNew));
I = rescale(I);
figure
image(I) axis
off

title("Generated Images")
```



conditional generative adversarial network (CGAN)

- A conditional generative adversarial network (CGAN) is a type of GAN that also takes advantage of labels during the training process.
  - Generator — Given a label and random array as input, this network generates data with the same structure as the training data observations corresponding to the same label.
  - Discriminator — Given batches of labeled data containing observations from both the training data and generated data from the generator, this network attempts to classify the observations as "real" or "generated".



conditional generative adversarial network (CGAN)

- To train a conditional GAN, train both networks simultaneously to maximize the performance of both:
  - Train the generator to generate data that "fools" the discriminator.
  - Train the discriminator to distinguish between real and generated data.
- To maximize the performance of the generator, maximize the loss of the discriminator when given generated labeled data. That is, the objective of the generator is to generate labeled data that the discriminator classifies as "real".
- To maximize the performance of the discriminator, minimize the loss of the discriminator when given batches of both real and generated labeled data. That is, the objective of the discriminator is to not be "fooled" by the generator.
- Ideally, these strategies result in a generator that generates convincingly realistic data that corresponds to the input labels and a discriminator that has learned strong feature representations that are characteristic of the training data for each label.

conditional generative adversarial network (CGAN)

- Download and extract the Flowers data set.

```
url = 'http://download.tensorflow.org/example_images/flower_photos.tgz';
downloadFolder = tempdir;
filename = fullfile(downloadFolder,'flower_dataset.tgz');

imageFolder = fullfile(downloadFolder,'flower_photos');
if ~exist(imageFolder,'dir')
   disp('Downloading Flowers data set (218 MB)...')
   websave(filename,url);
   untar(filename,downloadFolder)
end
```

conditional generative adversarial network (CGAN)

- Create an image datastore containing the photos of the flowers.

```
datasetFolder = fullfile(imageFolder);
imds = imageDatastore(datasetFolder,IncludeSubfolders=true,LabelSource="foldernames");
```

- View the number of classes.

```
classes = categories(imds.Labels);
numClasses = numel(classes)
numClasses = 5
```

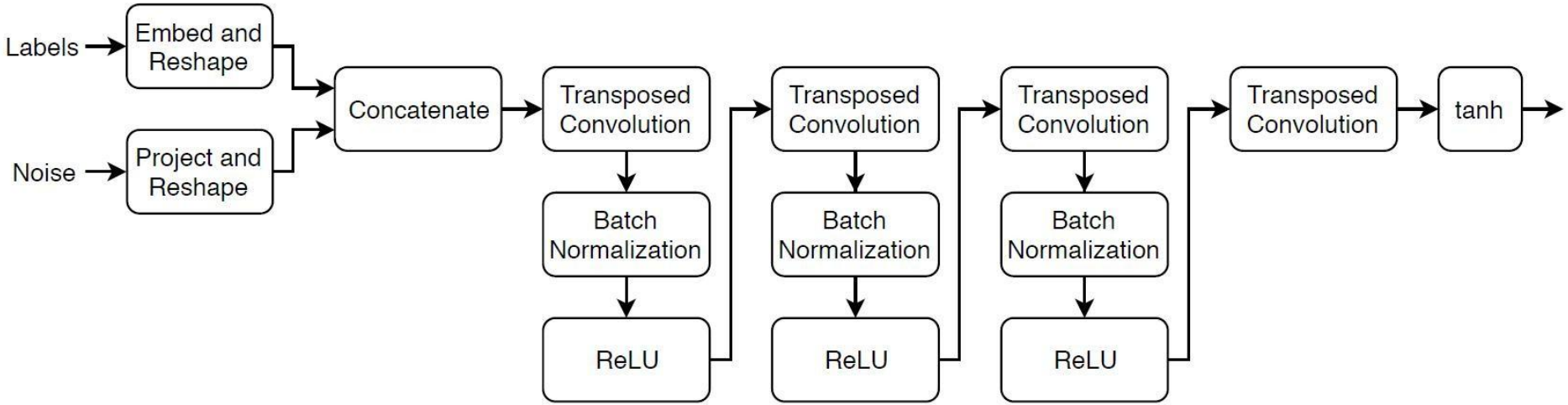
conditional generative adversarial network (CGAN)

- Augment the data to include random horizontal flipping and resize the images to have size 64-by-64.

```
augmenter = imageDataAugmenter(RandXReflection=true);
augimds = augmentedImageDatastore([64 64],imds,DataAugmentation=augmenter);
```

Define Generator Network

- Define the following two-input network, which generates images given random vectors of size 100 and corresponding labels.



Define Generator Network

This network:

- Converts the random vectors of size 100 to 4-by-4-by-1024 arrays using a fully connected layer followed by a reshape operation.
- Converts the categorical labels to embedding vectors and reshapes them to a 4-by-4 array.
- Concatenates the resulting images from the two inputs along the channel dimension. The output is a 4-by-4-by-1025 array.
- Upscales the resulting arrays to 64-by-64-by-3 arrays using a series of transposed convolution layers with batch normalization and ReLU layers.

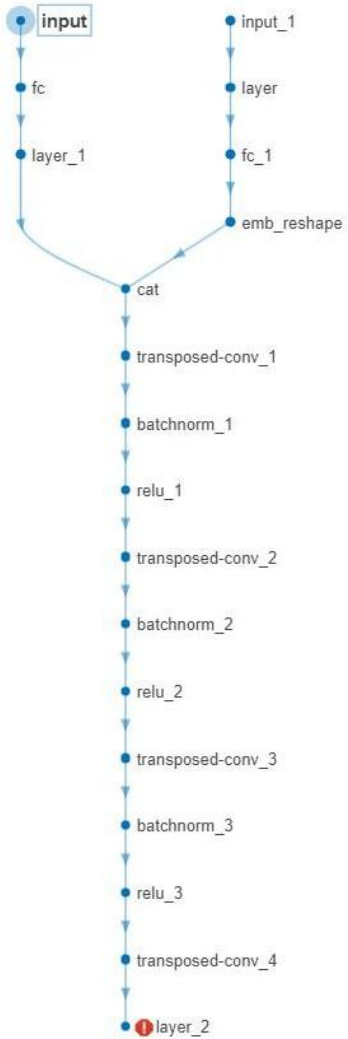
Define Generator Network

Define this network architecture as a layer graph and specify the following network properties.

- For the categorical inputs, use an embedding dimension of 50.
- For the transposed convolution layers, specify 5-by-5 filters with a decreasing number of filters for each layer, a stride of 2, and "same" cropping of the output.
- For the final transposed convolution layer, specify a three 5-by-5 filter corresponding to the three RGB channels of the generated images.
- At the end of the network, include a tanh layer.

```
numLatentInputs = 100;
embeddingDimension = 50;
numFilters = 64;

filterSize = 5;
projectionSize = [4 4 1024];

layersGenerator =
   [  featureInputLayer(numLatentInputs)
   fullyConnectedLayer(prod(projectionSize))
   functionLayer(@(X) feature2image(X,projectionSize),Formattable=true)
   concatenationLayer(3,2,Name="cat");
   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];
```

```
lgraphGenerator = layerGraph(layersGenerator);

layers =
   [
   featureInputLayer(1)
   embeddingLayer(embeddingDimension,numClasses)
   fullyConnectedLayer(prod(projectionSize(1:2)))
   functionLayer(@(X) feature2image(X,
                              [projectionSize(1:2) 1]),
                              Formattable=true,
                              Name="emb_reshape")];
lgraphGenerator = addLayers(lgraphGenerator,layers);
lgraphGenerator = connectLayers(lgraphGenerator,
                                    "emb_reshape","cat/in2");
```



	Name	Type	Activations	Learnables
1	input 100 features	Feature Input	100	-
2	input_1 1 features	Feature Input	1	-
3	fc 16384 fully connected layer	Fully Connected	16384	Weights 16384×100 Bias 16384×1
4	layer Embedding layer with dimension 50	embeddingLayer	50	Weights 50×5
5	fc_1 16 fully connected layer	Fully Connected	16	Weights 16×50 Bias 16×1
6	layer_1 @(X)feature2image(X,projectionSize)	Function	4×4×1024	-
7	emb_reshape @(X)feature2image(X,[projectionSize(1:2),1])	Function	4×4×1	-
8	cat Concatenation of 2 inputs along dimension 3	Concatenation	4×4×1025	-
9	transposed-conv_1 256 5×5 transposed convolutions with stride [1 1] and cropping [0 0 0 0]	Transposed Convolution	8×8×256	Weigh… 5×5×256×10… Bias 1×1×256
10	batchnorm_1 Batch normalization	Batch Normalization	8×8×256	Offset 1×1×256 Scale 1×1×256
11	relu_1 ReLU	ReLU	8×8×256	-
12	transposed-conv_2 128 5×5 transposed convolutions with stride [2 2] and cropping 'same'	Transposed Convolution	16×16×128	Weights 5×5×128×256 Bias 1×1×128
13	batchnorm_2 Batch normalization	Batch Normalization	16×16×128	Offset 1×1×128 Scale 1×1×128
14	relu_2 ReLU	ReLU	16×16×128	-
15	transposed-conv_3 64 5×5 transposed convolutions with stride [2 2] and cropping 'same'	Transposed Convolution	32×32×64	Weights 5×5×64×128 Bias 1×1×64
16	batchnorm_3 Batch normalization	Batch Normalization	32×32×64	Offset 1×1×64 Scale 1×1×64
17	relu_3 ReLU	ReLU	32×32×64	-
18	transposed-conv_4 3 5×5 transposed convolutions with stride [2 2] and cropping 'same'	Transposed Convolution	64×64×3	Weights 5×5×3×64 Bias 1×1×3
19	layer_2	Tanh	64×64×3	-

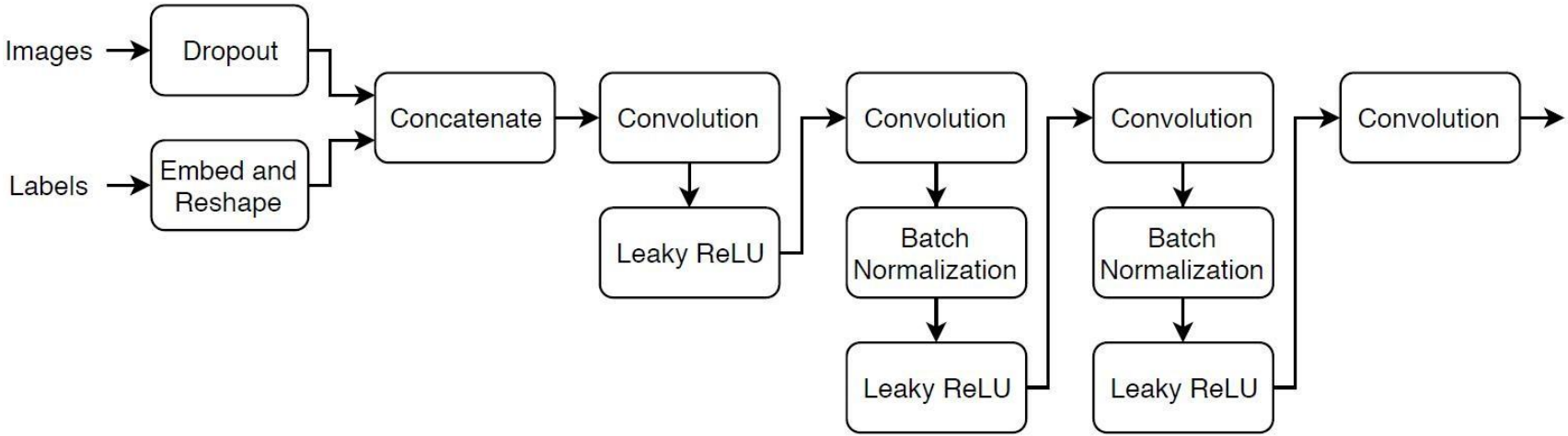
Define Generator Network

To train the network with a custom training loop and enable automatic differentiation, convert the layer graph to a dlnetwork object.

```
dlnetGenerator = dlnetwork(lgraphGenerator)
dlnetGenerator =
  dlnetwork with properties:

         Layers: [19×1 nnet.cnn.layer.Layer]
    Connections: [18×2 table]
    Learnables: [19×3 table]
          State: [6×3 table]
     InputNames: {'input'  'input_1'}
    OutputNames: {'layer_2'}
    Initialized: 1
```

Define Discriminator Network

Define the following two-input network, which classifies real and generated 64-by-64 images given a set of images and the corresponding labels.



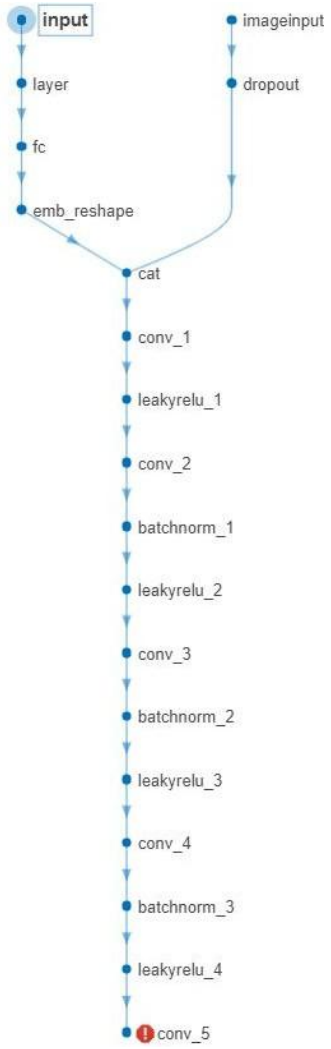
Define Discriminator Network

Create a network that takes as input 64-by-64-by-1 images and the corresponding labels and outputs a scalar prediction score using a series of convolution layers with batch normalization and leaky ReLU layers. Add noise to the input images using dropout.

- For the dropout layer, specify a dropout probability of 0.75.
- For the convolution layers, specify 5-by-5 filters with an increasing number of filters for each layer. Also specify a stride of 2 and padding of the output on each edge.
- For the leaky ReLU layers, specify a scale of 0.2.
- For the final layer, specify a convolution layer with one 4-by-4 filter.

```
dropoutProb = 0.75;
numFilters = 64;
scale = 0.2;

inputSize = [64 64 3];
filterSize = 5;

layersDiscriminator =
   [
   imageInputLayer(inputSize,Normalization="none")
   dropoutLayer(dropoutProb)
   concatenationLayer(3,2,Name="cat")
   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)
```

```
lgraphDiscriminator = layerGraph(layersDiscriminator);

layers =
   [
   featureInputLayer(1)
   embeddingLayer(embeddingDimension,numClasses)
   fullyConnectedLayer(prod(inputSize(1:2)))
   functionLayer(@(X) feature2image(X,
                                    [inputSize(1:2) ]),
                                    Formattable=true,
                                    Name="emb_reshape")];
lgraphDiscriminator = addLayers(lgraphDiscriminator,layers);
lgraphDiscriminator = connectLayers(lgraphDiscriminator,
                                    "emb_reshape","cat/in2");
```



	Name	Type	Activations	Learnables
1	input 1 features	Feature Input	1	-
2	layer Embedding layer with dimension 50	embeddingLayer	50	Weights 50×5
3	imageinput 64×64×3 images	Image Input	64×64×3	-
4	dropout 75% dropout	Dropout	64×64×3	-
5	fc 4096 fully connected layer	Fully Connected	4096	Weights 4096×50 Bias 4096×1
6	emb_reshape @(X)feature2image(X,[inputSize(1:2),1])	Function	64×64×1	-
7	cat Concatenation of 2 inputs along dimension 3	Concatenation	64×64×4	-
8	conv_1 64 5×5 convolutions with stride [2 2] and padding 'same'	Convolution	32×32×64	Weights 5×5×4×64 Bias 1×1×64
9	leakyrelu_1 Leaky ReLU with scale 0.2	Leaky ReLU	32×32×64	-
10	conv_2 128 5×5 convolutions with stride [2 2] and padding 'same'	Convolution	16×16×128	Weights 5×5×64×128 Bias 1×1×128
11	batchnorm_1 Batch normalization	Batch Normalization	16×16×128	Offset 1×1×128 Scale 1×1×128
12	leakyrelu_2 Leaky ReLU with scale 0.2	Leaky ReLU	16×16×128	-
13	conv_3 256 5×5 convolutions with stride [2 2] and padding 'same'	Convolution	8×8×256	Weights 5×5×128×256 Bias 1×1×256
14	batchnorm_2 Batch normalization	Batch Normalization	8×8×256	Offset 1×1×256 Scale 1×1×256
15	leakyrelu_3 Leaky ReLU with scale 0.2	Leaky ReLU	8×8×256	-
16	conv_4 512 5×5 convolutions with stride [2 2] and padding 'same'	Convolution	4×4×512	Weights 5×5×256×512 Bias 1×1×512
17	batchnorm_3 Batch normalization	Batch Normalization	4×4×512	Offset 1×1×512 Scale 1×1×512
18	leakyrelu_4 Leaky ReLU with scale 0.2	Leaky ReLU	4×4×512	-
19	conv_5	Convolution	1×1×1	Weights 4×4×512

Define Discriminator Network

To train the network with a custom training loop and enable automatic differentiation, convert the layer graph to a dlnetwork object.

```
dlnetDiscriminator = dlnetwork(lgraphDiscriminator)
dlnetDiscriminator =
  dlnetwork with properties:

         Layers: [19×1 nnet.cnn.layer.Layer]
    Connections: [18×2 table]
    Learnables: [19×3 table]
          State: [6×3 table]
     InputNames: {'imageinput'  'input'}
    OutputNames: {'conv_5'}
    Initialized: 1
```

Define Model Gradients and Loss Functions

Create the function modelGradients, which takes as input the generator and discriminator networks, a mini-batch of input data, and an array of random values, and returns the gradients of the loss with respect to the learnable parameters in the networks and an array of generated images.

Specify Training Options

Train with a mini-batch size of 128 for 500 epochs.

numEpochs = 500;

miniBatchSize = 128;

Specify the options for Adam optimization. For both networks, use:

- A learning rate of 0.0002
- A gradient decay factor of 0.5
- A squared gradient decay factor of 0.999

learnRate = 0.0002;

gradientDecayFactor = 0.5;

squaredGradientDecayFactor = 0.999;

Specify Training Options

Update the training progress plots every 100 iterations.

validationFrequency = 100;

If the discriminator learns to discriminate between real and generated images too quickly, then the generator can fail to train. To better balance the learning of the discriminator and the generator, randomly flip the labels of a proportion of the real images. Specify a flip factor of 0.5.

flipFactor = 0.5;

Train Model

Train the model using a custom training loop. Loop over the training data and update the network parameters at each iteration. To monitor the training progress, display a batch of generated images using a held-out array of random values to input into the generator and the network scores.

Use minibatchqueue to process and manage the mini-batches of images during training. For each mini-batch:

- Use the custom mini-batch preprocessing function preprocessMiniBatch to rescale the images in the range [-1,1].
- Discard any partial mini-batches with less than 128 observations.
- Format the image data with the dimension labels "SSCB" (spatial, spatial, channel, batch).
- Format the label data with the dimension labels "BC" (batch, channel).
- Train on a GPU if one is available. When the OutputEnvironment option of minibatchqueue is "auto", minibatchqueue converts each output to a gpuArray if a GPU is available.

Train Model

The minibatchqueue object, by default, converts the data to dlarray objects with underlying type single.

```
augimds.MiniBatchSize = miniBatchSize;
executionEnvironment = "auto";

mbq = minibatchqueue(augimds, ...
    MiniBatchSize=miniBatchSize, ...
    PartialMiniBatch="discard", ...
    MiniBatchFcn=@preprocessData, ...
    MiniBatchFormat=["SSCB" "BC"], ...
    OutputEnvironment=executionEnvironment);
```

Train Model

Initialize the parameters for the Adam optimizer.

velocityDiscriminator = [];

trailingAvgGenerator = [];

trailingAvgSqGenerator = [];

trailingAvgDiscriminator = [];

trailingAvgSqDiscriminator = [];

Train Model

Initialize the plot of the training progress. Create a figure and resize it to have twice the width.

```
f = figure;
f.Position(3) = 2*f.Position(3);
```

Create subplots of the generated images and of the scores plot.

```
imageAxes = subplot(1,2,1);
scoreAxes = subplot(1,2,2);
```

Train Model

Initialize animated lines for the scores plot.

```
lineScoreGenerator = animatedline(scoreAxes,Color=[0 0.447 0.741]);
lineScoreDiscriminator = animatedline(scoreAxes,Color=[0.85 0.325 0.098]);
```

Customize the appearance of the plots.

```
legend("Generator","Discriminator");
ylim([0 1])
xlabel("Iteration")
ylabel("Score")
grid on
```

Train Model

To monitor training progress, create a held-out batch of 25 random vectors and a corresponding set of labels 1 through 5 (corresponding to the classes) repeated five times.

```
numValidationImagesPerClass = 5;
ZValidation = randn(numLatentInputs,numValidationImagesPerClass*numClasses,"single");
TValidation = single(repmat(1:numClasses,[1 numValidationImagesPerClass]));
```

Convert the data to dlarray objects and specify the dimension labels "CB" (channel, batch).

```
dlZValidation = dlarray(ZValidation,"CB");
dlTValidation = dlarray(TValidation,"CB");
```

Train Model

Train the conditional GAN. For each epoch, shuffle the data and loop over mini-batches of data.

For each mini-batch:

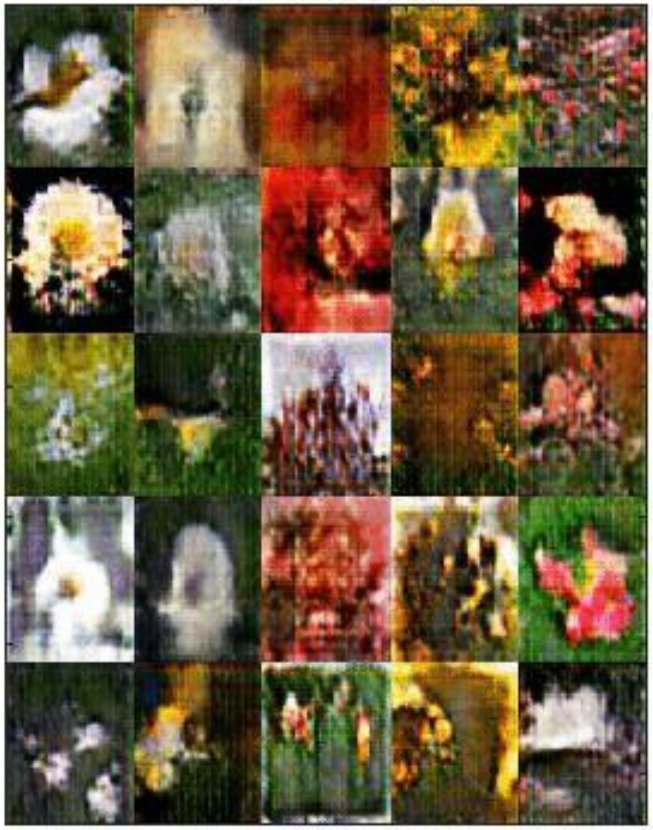
- Evaluate the model gradients using dlfeval and the modelGradients function.
- Update the network parameters using the adamupdate function.
- Plot the scores of the two networks.
- After every validationFrequency iterations, display a batch of generated images for a fixed held-out generator input.

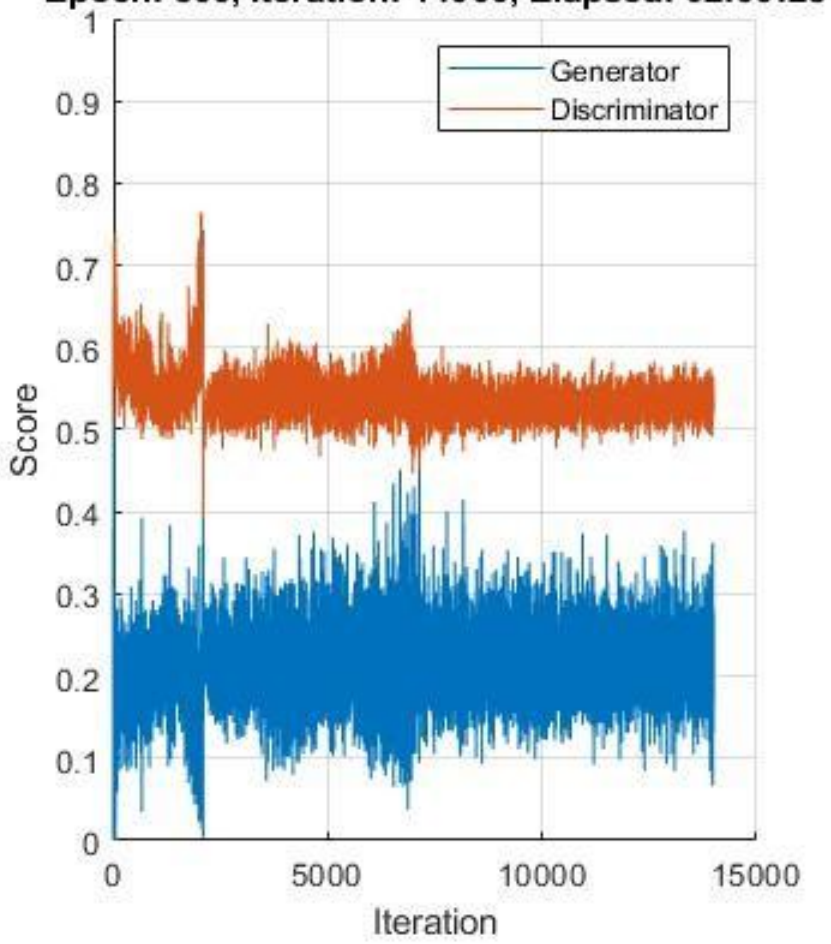
Training can take some time to run.

```
iteration = 0;
start = tic;

% Loop over epochs.
for epoch = 1:numEpochs

    % Reset and shuffle data.
    shuffle(mbq);

    % Loop over mini-batches.
    while hasdata(mbq)
        iteration = iteration + 1;

        % Read mini-batch of data.
        [dlX,dlT] = next(mbq);
        Z = randn(numLatentInputs,miniBatchSize,"single");
        dlZ = dlarray(Z,"CB");

        [gradientsGenerator, gradientsDiscriminator, stateGenerator, scoreGenerator, scoreDiscriminator] = ...
            dlfeval(@modelGradients, dlnetGenerator, dlnetDiscriminator, dlX, dlT, dlZ, flipFactor);
        dlnetGenerator.State = stateGenerator;
```

```
[dlnetDiscriminator,trailingAvgDiscriminator,trailingAvgSqDiscriminator] = ...
    adamupdate(dlnetDiscriminator, gradientsDiscriminator, ...
    trailingAvgDiscriminator, trailingAvgSqDiscriminator, iteration, ...  learnRate,
    gradientDecayFactor, squaredGradientDecayFactor);

[dlnetGenerator,trailingAvgGenerator,trailingAvgSqGenerator] = ...
    adamupdate(dlnetGenerator, gradientsGenerator, ...
    trailingAvgGenerator, trailingAvgSqGenerator, iteration, ...
    learnRate, gradientDecayFactor, squaredGradientDecayFactor);

% Every validationFrequency iterations, display batch of generated images.
if mod(iteration,validationFrequency) == 0 || iteration == 1
    dlXGeneratedValidation = predict(dlnetGenerator,dlZValidation,dlTValidation);  I
    = imtile(extractdata(dlXGeneratedValidation), ...
        GridSize=[numValidationImagesPerClass numClasses]);
    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.
        D = duration(0,0,toc(start),Format="hh:mm:ss");
        title(...
            "Epoch: " + epoch + ", " + ...
            "Iteration: " + iteration + ", " + ...
            "Elapsed: " + string(D))

        drawnow
    end
end
```


Generated Images


Epoch: 500, Iteration: 14000, Elapsed: 02:00:28
Generator
Discriminator
Score
Iteration
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5000
10000
15000

Generate New Images

To generate new images of a particular class, use the predict function on the generator with a dlarray object containing a batch of random vectors and an array of labels corresponding to the desired classes. Convert the data to dlarray objects and specify the dimension labels "CB" (channel, batch). For GPU prediction, convert the data to gpuArray objects. To display the images together, use the imtile function and rescale the images using the rescale function.

Create an array of 36 vectors of random values corresponding to the first class.

```
numObservationsNew = 36;

idxClass = 1;
Z = randn(numLatentInputs,numObservationsNew,"single");
T = repmat(single(idxClass),[1 numObservationsNew]);
```

Generate New Images

Convert the data to dlarray objects with the dimension labels "SSCB" (spatial, spatial, channels, batch).

```
dlZ = dlarray(Z,"CB");
dlT = dlarray(T,"CB");
```

To generate images using the GPU, also convert the data to gpuArray objects.

```
if (executionEnvironment == "auto" && canUseGPU) || executionEnvironment == "gpu"
    dlZ = gpuArray(dlZ);
    dlT = gpuArray(dlT);
end
```

Generate New Images

Generate images using the predict function with the generator network.

```
dlXGenerated = predict(dlnetGenerator,dlZ,dlT);
```

Display the generated images in a plot.

```
figure
I = imtile(extractdata(dlXGenerated));
I = rescale(I);
imshow(I)
title("Class: " + classes(idxClass))
```

Model Gradients Function

The function modelGradients takes as input the generator and discriminator dlnetwork objects dlnetGenerator and dlnetDiscriminator, a mini-batch of input data dlX, the corresponding labels dlT, and an array of random values dlZ, and returns the gradients of the loss with respect to the learnable parameters in the networks, the generator state, and the network scores.

If the discriminator learns to discriminate between real and generated images too quickly, then the generator can fail to train. To better balance the learning of the discriminator and the generator, randomly flip the labels of a proportion of the real images.

```
function [gradientsGenerator, gradientsDiscriminator, stateGenerator, scoreGenerator, scoreDiscriminator] = ...
    modelGradients(dlnetGenerator, dlnetDiscriminator, dlX, dlT, dlZ, flipFactor)

% Calculate the predictions for real data with the discriminator network.
dlYPred = forward(dlnetDiscriminator, dlX, dlT);

% Calculate the predictions for generated data with the discriminator network.
[dlXGenerated,stateGenerator] = forward(dlnetGenerator, dlZ, dlT);
dlYPredGenerated = forward(dlnetDiscriminator, dlXGenerated, dlT);

% Calculate probabilities.
probGenerated = sigmoid(dlYPredGenerated);
probReal = sigmoid(dlYPred);

% Calculate the generator and discriminator scores.
scoreGenerator = mean(probGenerated);
scoreDiscriminator = (mean(probReal) + mean(1-probGenerated)) / 2;

% Flip labels.
numObservations = size(dlYPred,4);
idx = randperm(numObservations,floor(flipFactor * numObservations));
probReal(:,:,:,idx) = 1 - probReal(:,:,:,idx);
```

```
% Calculate the GAN loss.
[lossGenerator, lossDiscriminator] = ganLoss(probReal, probGenerated);

% For each network, calculate the gradients with respect to the loss.
gradientsGenerator = dlgradient(lossGenerator, dlnetGenerator.Learnables,RetainData=true);
gradientsDiscriminator = dlgradient(lossDiscriminator, dlnetDiscriminator.Learnables);

end
```

GAN Loss Function

```
function [lossGenerator, lossDiscriminator] = ganLoss(scoresReal,scoresGenerated)

% Calculate losses for the discriminator network.
lossGenerated = -mean(log(1 - scoresGenerated));
lossReal = -mean(log(scoresReal));

% Combine the losses for the discriminator network.
lossDiscriminator = lossReal + lossGenerated;

% Calculate the loss for the generator network.
lossGenerator = -mean(log(scoresGenerated));

end
```

Mini-Batch Preprocessing Function

The preprocessMiniBatch function preprocesses the data using the following steps:

- Extract the image and label data from the input cell arrays and concatenate them into numeric arrays.
- Rescale the images to be in the range [-1,1].

```
function [X,T] = preprocessData(XCell,TCell)

% Extract image data from cell and concatenate
X = cat(4,XCell{:});

% Extract label data from cell and concatenate
T = cat(1,TCell{:});

% Rescale the images in the range [-1 1].
X = rescale(X,-1,1,InputMin=0,InputMax=255);
end
```

GAN examples

%Train Generative Adversarial Network (GAN)

openExample('nnet/TrainGenerativeAdversarialNetworkGANExample')

%Train Conditional Generative Adversarial Network (CGAN)

openExample('nnet/TrainConditionalGenerativeAdversarialNetworkCGANExample')