Transfer Learning using AlexNet and Dog Breed Classifier Dataset

You will need to have modified dataset.

- Download the "Dog Breed Identification" dataset from (https://www.kaggle.com/c/dog-breed-identification/data)
- Run "ModifyDataset.mlx" script to modify the dataset into directory wise categories

Import Alex Net

```
net = alexnet;

% You can also grab AlexNet from add-on explorer in the home tab, or file exchange online
```

Access Layers

Visualize the layers of AlexNet, what do we see about this architecture? What do we have to change for this to work for our new data?

```
layers = net.Layers;

layers
```

```
layers =
  25×1 Layer array with layers:

     1   'data'     Image Input                  227×227×3 images with 'zerocenter' normalization
     2   'conv1'    Convolution                  96 11×11×3 convolutions with stride [4  4] and padding [0  0  0  0
     3   'relu1'    ReLU                         ReLU
     4   'norm1'    Cross Channel Normalization  cross channel normalization with 5 channels per element
     5   'pool1'    Max Pooling                  3×3 max pooling with stride [2  2] and padding [0  0  0  0]
     6   'conv2'    Grouped Convolution          2 groups of 128 5×5×48 convolutions with stride [1  1] and padding
     7   'relu2'    ReLU                         ReLU
     8   'norm2'    Cross Channel Normalization  cross channel normalization with 5 channels per element
     9   'pool2'    Max Pooling                  3×3 max pooling with stride [2  2] and padding [0  0  0  0]
    10   'conv3'    Convolution                  384 3×3×256 convolutions with stride [1  1] and padding [1  1  1
    11   'relu3'    ReLU                         ReLU
    12   'conv4'    Grouped Convolution          2 groups of 192 3×3×192 convolutions with stride [1  1] and paddi
    13   'relu4'    ReLU                         ReLU
    14   'conv5'    Grouped Convolution          2 groups of 128 3×3×192 convolutions with stride [1  1] and paddi
    15   'relu5'    ReLU                         ReLU
    16   'pool5'    Max Pooling                  3×3 max pooling with stride [2  2] and padding [0  0  0  0]
    17   'fc6'      Fully Connected              4096 fully connected layer
    18   'relu6'    ReLU                         ReLU
    19   'drop6'    Dropout                      50% dropout
    20   'fc7'      Fully Connected              4096 fully connected layer
    21   'relu7'    ReLU                         ReLU
    22   'drop7'    Dropout                      50% dropout
    23   'fc8'      Fully Connected              1000 fully connected layer
    24   'prob'     Softmax                      softmax
    25   'output'   Classification Output        crossentropyex with 'tench' and 999 other classes
```

```
% notice the 1000 in the last fully connected layer. This is for the 1000 categories AlexNet kn
```

Train

Set up training data

```
rootFolder = 'train';

LabelData = readtable('.\labels.csv', 'Format', '%C%C');
BreedLabels = string(transpose(table2cell(unique(LabelData(:,'breed')))));

BreedCount = numel(BreedLabels)
```

```
BreedCount = 120
```

```
imds = imageDatastore(fullfile(rootFolder, BreedLabels), 'LabelSource', 'foldernames');

%imds = splitEachLabel(imds, 500, 'randomize') % we only need 500 images per class
imds.ReadFcn = @readFunctionTrain;
```

Take layers from Alex Net, then add our own

```
layers = layers(1:end-3);

layers(end+1) = fullyConnectedLayer(64, 'Name', 'special_2');
layers(end+1) = reluLayer;
layers(end+1) = fullyConnectedLayer(BreedCount, 'Name', 'fc8_2 ');
layers(end+1) = softmaxLayer;
layers(end+1) = classificationLayer()
```

```
layers =
  27×1 Layer array with layers:

     1   'data'        Image Input                   227×227×3 images with 'zerocenter' normalization
     2   'conv1'       Convolution                   96 11×11×3 convolutions with stride [4  4] and padding [0  0  0
     3   'relu1'       ReLU                          ReLU
     4   'norm1'       Cross Channel Normalization   cross channel normalization with 5 channels per element
     5   'pool1'       Max Pooling                   3×3 max pooling with stride [2  2] and padding [0  0  0  0]
     6   'conv2'       Grouped Convolution           2 groups of 128 5×5×48 convolutions with stride [1  1] and pado
     7   'relu2'       ReLU                          ReLU
     8   'norm2'       Cross Channel Normalization   cross channel normalization with 5 channels per element
     9   'pool2'       Max Pooling                   3×3 max pooling with stride [2  2] and padding [0  0  0  0]
    10   'conv3'       Convolution                   384 3×3×256 convolutions with stride [1  1] and padding [1  1
    11   'relu3'       ReLU                          ReLU
    12   'conv4'       Grouped Convolution           2 groups of 192 3×3×192 convolutions with stride [1  1] and pad
    13   'relu4'       ReLU                          ReLU
    14   'conv5'       Grouped Convolution           2 groups of 128 3×3×192 convolutions with stride [1  1] and pad
    15   'relu5'       ReLU                          ReLU
    16   'pool5'       Max Pooling                   3×3 max pooling with stride [2  2] and padding [0  0  0  0]
    17   'fc6'         Fully Connected               4096 fully connected layer
    18   'relu6'       ReLU                          ReLU
    19   'drop6'       Dropout                       50% dropout
    20   'fc7'         Fully Connected               4096 fully connected layer
    21   'relu7'       ReLU                          ReLU
    22   'drop7'       Dropout                       50% dropout
    23   'special_2'   Fully Connected               64 fully connected layer
    24   ''            ReLU                          ReLU
```

```
    25   'fc8_2 '      Fully Connected              120 fully connected layer
    26   ''            Softmax                      softmax
    27   ''            Classification Output        crossentropyex
```

Fine-tune learning rates [advanced]

```
layers(end-2).WeightLearnRateFactor = 10;
layers(end-2).WeightL2Factor = 1;
layers(end-2).BiasLearnRateFactor = 20;
layers(end-2).BiasL2Factor = 0;
```

Other training options

```
opts = trainingOptions('sgdm', ...
    'LearnRateSchedule', 'none',...
    'InitialLearnRate', .0001,...
    'MaxEpochs', 20, ...
    'MiniBatchSize', 128);
```

Test GPU before running?

```
gpuDevice()
```

```
ans =
  CUDADevice with properties:

                      Name: 'GeForce GTX 1660 Ti'
                     Index: 1
         ComputeCapability: '7.5'
            SupportsDouble: 1
             DriverVersion: 11.1000
            ToolkitVersion: 11
        MaxThreadsPerBlock: 1024
          MaxShmemPerBlock: 49152
        MaxThreadBlockSize: [1024 1024 64]
               MaxGridSize: [2.1475e+09 65535 65535]
                 SIMDWidth: 32
               TotalMemory: 6.4425e+09
           AvailableMemory: 5.2528e+09
      MultiprocessorCount: 24
             ClockRateKHz: 1590000
              ComputeMode: 'Default'
     GPUOverlapsTransfers: 1
   KernelExecutionTimeout: 1
         CanMapHostMemory: 1
          DeviceSupported: 1
          DeviceAvailable: 1
           DeviceSelected: 1
```

```
gpuDevice(1);
```

Train!

Please note this may take a few minutes or longer depending on hardware. Training with a GPU is strongly encouraged.

```
convnet = trainNetwork(imds, layers, opts);
```

```
Training on single GPU.
Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:02 |        0.78% |       6.3083 |      1.0000e-04 |
|       1 |          50 |       00:00:55 |        0.78% |       4.8987 |      1.0000e-04 |
|       2 |         100 |       00:01:48 |        0.78% |       4.8134 |      1.0000e-04 |
|       2 |         150 |       00:02:42 |        0.78% |       4.8625 |      1.0000e-04 |
|       3 |         200 |       00:03:35 |        1.56% |       4.7766 |      1.0000e-04 |
|       4 |         250 |       00:04:29 |        2.34% |       4.6281 |      1.0000e-04 |
|       4 |         300 |       00:05:22 |        7.03% |       4.6533 |      1.0000e-04 |
|       5 |         350 |       00:06:15 |        7.03% |       4.4769 |      1.0000e-04 |
|       6 |         400 |       00:07:09 |        4.69% |       4.5620 |      1.0000e-04 |
|       6 |         450 |       00:08:03 |        7.03% |       4.3130 |      1.0000e-04 |
|       7 |         500 |       00:08:56 |       14.84% |       4.2020 |      1.0000e-04 |
|       7 |         550 |       00:09:51 |        9.38% |       4.1246 |      1.0000e-04 |
|       8 |         600 |       00:10:44 |       13.28% |       3.8384 |      1.0000e-04 |
|       9 |         650 |       00:11:38 |       22.66% |       3.7281 |      1.0000e-04 |
|       9 |         700 |       00:12:32 |       25.78% |       3.3433 |      1.0000e-04 |
|      10 |         750 |       00:13:25 |       28.12% |       3.4208 |      1.0000e-04 |
|      11 |         800 |       00:14:19 |       29.69% |       3.0885 |      1.0000e-04 |
|      11 |         850 |       00:15:14 |       32.03% |       2.8115 |      1.0000e-04 |
|      12 |         900 |       00:16:09 |       32.03% |       2.7894 |      1.0000e-04 |
|      13 |         950 |       00:17:04 |       29.69% |       2.6385 |      1.0000e-04 |
|      13 |        1000 |       00:17:58 |       40.62% |       2.1943 |      1.0000e-04 |
|      14 |        1050 |       00:18:52 |       43.75% |       2.2778 |      1.0000e-04 |
|      14 |        1100 |       00:19:47 |       40.62% |       2.3786 |      1.0000e-04 |
|      15 |        1150 |       00:20:41 |       34.38% |       2.2446 |      1.0000e-04 |
|      16 |        1200 |       00:21:35 |       37.50% |       2.3385 |      1.0000e-04 |
|      16 |        1250 |       00:22:29 |       44.53% |       2.0821 |      1.0000e-04 |
|      17 |        1300 |       00:23:22 |       39.06% |       2.1643 |      1.0000e-04 |
|      18 |        1350 |       00:24:16 |       46.09% |       2.0995 |      1.0000e-04 |
|      18 |        1400 |       00:25:10 |       47.66% |       2.0064 |      1.0000e-04 |
|      19 |        1450 |       00:26:05 |       46.09% |       1.7603 |      1.0000e-04 |
|      19 |        1500 |       00:27:01 |       51.56% |       1.9441 |      1.0000e-04 |
|      20 |        1550 |       00:27:55 |       43.75% |       2.3045 |      1.0000e-04 |
|      20 |        1580 |       00:28:28 |       51.56% |       1.6720 |      1.0000e-04 |
|=========================================================================================|
Training finished: Max epochs completed.
```

Test

Set up test data

```
rootFolder = 'test';
testDS = imageDatastore(fullfile(rootFolder, BreedLabels), 'LabelSource', 'foldernames');
testDS.ReadFcn = @readFunctionTrain;
```

Test classifer

```
[labels,err_test] = classify(convnet, testDS, 'MiniBatchSize', 64);
```

Determine overall accuracy

```
confMat = confusionmat(testDS.Labels, labels);
confMat = confMat./sum(confMat,2);
OverallAccuracy = mean(diag(confMat))
```

```
OverallAccuracy = 0.6869
```

```
BreedAcc = diag(confMat).';
int_confMat = int64(confMat .* 10000)
```

```
int_confMat = 120×120 int64 matrix
   7500      0      0      0      0      0      0      0      0    125      0 ···
      0   8707      0      0      0      0      0      0      0      0      0
      0      0   9302      0      0      0      0      0      0      0      0
      0      0      0   7570      0      0    187      0      0      0      0
      0      0    135      0   3919      0      0    135      0    270      0
      0      0      0      0      0   6154      0      0    128      0      0
      0      0      0    196      0      0   7059      0      0      0      0
      0      0      0      0      0      0      0   8364      0      0      0
      0      0      0      0      0    122      0    122   6341    976      0
      0      0      0      0      0      0      0      0    381   7905      0
      ⋮
```

```
confusionchart(int_confMat )
```