%% Speech Command Recognition Using Deep Learning

% Load the pre-trained network.

load('commandNet.mat')

%%

% The network is trained to recognize the following speech commands:

%

% \* "yes"

% \* "no"

% \* "up"

% \* "down"

% \* "left"

% \* "right"

% \* "on"

% \* "off"

% \* "stop"

% \* "go"

%

%%

% Load a short speech signal where a person says "stop".

[x,fs] = audioread('stop\_command.flac');

%%

% Listen to the command.

sound(x,fs)

auditorySpect = helperExtractAuditoryFeatures(x,fs);

%%

% Classify the command based on its auditory spectrogram.

command = classify(trainedNet,auditorySpect)

%%

% Load the speech signal and listen to it.

x = audioread('play\_command.flac');

sound(x,fs)

%%

% Compute the auditory spectrogram.

auditorySpect = helperExtractAuditoryFeatures(x,fs);

%%

% Classify the signal.

command = classify(trainedNet,auditorySpect)

%%

% The network is trained to classify background noise as "background".

%%

% Create a one-second signal consisting of random noise.

x = 0.01 \* randn(16e3,1);

%%

% Compute the auditory spectrogram.

auditorySpect = helperExtractAuditoryFeatures(x,fs);

%%

% Classify the background noise.

command = classify(trainedNet,auditorySpect);

%% Detect Commands Using Streaming Audio from Microphone

% Test your pre-trained command detection network on streaming audio from

% your microphone. Try saying one of the commands, for example, \_yes\_,

% \_no\_, or \_stop\_. Then, try saying one of the unknown words such as

classificationRate = 20;

adr = audioDeviceReader('SampleRate',fs,'SamplesPerFrame',floor(fs/classificationRate));

%%

% Initialize a buffer for the audio. Extract the classification labels of

% the network. Initialize buffers of half a second for the labels and

audioBuffer = dsp.AsyncBuffer(fs);

labels = trainedNet.Layers(end).Classes;

YBuffer(1:classificationRate/2) = categorical("background");

probBuffer = zeros([numel(labels),classificationRate/2]);

countThreshold = ceil(classificationRate\*0.2);

probThreshold = 0.7;

%%

% Create a figure and detect commands as long as the created figure exists.

h = figure('Units','normalized','Position',[0.2 0.1 0.6 0.8]);

timeLimit = 20;

tic;

while ishandle(h) && toc < timeLimit

% Extract audio samples from the audio device and add the samples to

% the buffer.

x = adr();

write(audioBuffer,x);

y = read(audioBuffer,fs,fs-adr.SamplesPerFrame);

spec = helperExtractAuditoryFeatures(y,fs);

% Classify the current spectrogram, save the label to the label buffer,

[YPredicted,probs] = classify(trainedNet,spec,'ExecutionEnvironment','cpu');

YBuffer = [YBuffer(2:end),YPredicted];

probBuffer = [probBuffer(:,2:end),probs(:)];

% Plot the current waveform and spectrogram.

subplot(2,1,1)

plot(y)

axis tight

ylim([-1,1])

subplot(2,1,2)

pcolor(spec')

caxis([-4 2.6445])

shading flat

% Now do the actual command detection by performing a very simple

[YMode,count] = mode(YBuffer);

maxProb = max(probBuffer(labels == YMode,:));

subplot(2,1,1)

if YMode == "background" || count < countThreshold || maxProb < probThreshold

title(" ")

else

title(string(YMode),'FontSize',20)

end

drawnow

end

%%

% <<../streaming\_commands.png>>

%% Load Speech Commands Data Set

% Download and extract the data set [1].

url = 'https://storage.googleapis.com/download.tensorflow.org/data/speech\_commands\_v0.01.tar.gz';

downloadFolder = tempdir;

datasetFolder = fullfile(downloadFolder,'google\_speech');

if ~exist(datasetFolder,'dir')

disp('Downloading speech commands data set (1.5 GB)...')

untar(url,datasetFolder)

end

%%

% Create an <docid:audio\_ref#mw\_6315b106-9a7b-4a11-a7c6-322c073e343a

% audioDatastore> that points to the data set.

ads = audioDatastore(datasetFolder, ...

'IncludeSubfolders',true, ...

'FileExtensions','.wav', ...

'LabelSource','foldernames')

%% Choose Words to Recognize

commands = categorical(["yes","no","up","down","left","right","on","off","stop","go"]);

isCommand = ismember(ads.Labels,commands);

isUnknown = ~ismember(ads.Labels,[commands,"\_background\_noise\_"]);

includeFraction = 0.2;

mask = rand(numel(ads.Labels),1) < includeFraction;

isUnknown = isUnknown & mask;

ads.Labels(isUnknown) = categorical("unknown");

adsSubset = subset(ads,isCommand|isUnknown);

countEachLabel(adsSubset)

%% Split Data into Training, Validation, and Test Sets

% Read the list of validation files.

c = importdata(fullfile(datasetFolder,'validation\_list.txt'));

filesValidation = string(c);

%%

% Read the list of test files.

c = importdata(fullfile(datasetFolder,'testing\_list.txt'));

filesTest = string(c);

%%

% Determine which files in the datastore should go to validation set and

% which should go to test set.

files = adsSubset.Files;

sf = split(files,filesep);

isValidation = ismember(sf(:,end-1) + "/" + sf(:,end),filesValidation);

isTest = ismember(sf(:,end-1) + "/" + sf(:,end),filesTest);

adsValidation = subset(adsSubset,isValidation);

adsTrain = subset(adsSubset,~isValidation & ~isTest);

%%

% To train the network with the entire dataset and achieve the highest

% possible accuracy, set |reduceDataset| to |false|. To run this example

% quickly, set |reduceDataset| to |true|.

reduceDataset = false;

if reduceDataset

numUniqueLabels = numel(unique(adsTrain.Labels));

% Reduce the dataset by a factor of 20

adsTrain = splitEachLabel(adsTrain,round(numel(adsTrain.Files) / numUniqueLabels / 20));

adsValidation = splitEachLabel(adsValidation,round(numel(adsValidation.Files) / numUniqueLabels / 20));

end

%% Compute Auditory Spectrograms

% Create an <docid:audio\_ref#mw\_b56cd7dc-af31-4da4-a43e-b13debc30322

% audioFeatureExtractor> object to perform the feature extraction.

fs = 16e3; % Known sample rate of the data set.

segmentDuration = 1;

frameDuration = 0.025;

hopDuration = 0.010;

segmentSamples = round(segmentDuration\*fs);

frameSamples = round(frameDuration\*fs);

hopSamples = round(hopDuration\*fs);

overlapSamples = frameSamples - hopSamples;

FFTLength = 512;

numBands = 50;

afe = audioFeatureExtractor( ...

'SampleRate',fs, ...

'FFTLength',FFTLength, ...

'Window',hann(frameSamples,'periodic'), ...

'OverlapLength',overlapSamples, ...

'barkSpectrum',true);

setExtractorParams(afe,'barkSpectrum','NumBands',numBands);

%%

% Read a file from the dataset. Training a convolutional neural network

x = read(adsTrain);

numSamples = size(x,1);

numToPadFront = floor( (segmentSamples - numSamples)/2 );

numToPadBack = ceil( (segmentSamples - numSamples)/2 );

xPadded = [zeros(numToPadFront,1,'like',x);x;zeros(numToPadBack,1,'like',x)];

%%

% To extract audio features, call |extract|. The output is a Bark spectrum

% with time across rows.

features = extract(afe,xPadded);

[numHops,numFeatures] = size(features)

%%

% The |audioFeatureExtractor| normalizes auditory spectrograms by the

% window power so that measurements are independent of the type of window

unNorm = 2/(sum(afe.Window)^2);

%%

%

if ~isempty(ver('parallel')) && ~reduceDataset

pool = gcp;

numPar = numpartitions(adsTrain,pool);

else

numPar = 1;

end

%%

% For each partition, read from the datastore, zero-pad the signal, and

% then extract the features.

parfor ii = 1:numPar

subds = partition(adsTrain,numPar,ii);

XTrain = zeros(numHops,numBands,1,numel(subds.Files));

for idx = 1:numel(subds.Files)

x = read(subds);

xPadded = [zeros(floor((segmentSamples-size(x,1))/2),1);x;zeros(ceil((segmentSamples-size(x,1))/2),1)];

XTrain(:,:,:,idx) = extract(afe,xPadded);

end

XTrainC{ii} = XTrain;

end

%%

% Convert the output to a 4-dimensional array with auditory spectrograms

% along the fourth dimension.

XTrain = cat(4,XTrainC{:});

[numHops,numBands,numChannels,numSpec] = size(XTrain)

%%

% Scale the features by the window power and then take the log. To obtain

XTrain = XTrain/unNorm;

epsil = 1e-6;

XTrain = log10(XTrain + epsil);

%%

% Perform the feature extraction steps described above to the validation

% set.

if ~isempty(ver('parallel'))

pool = gcp;

numPar = numpartitions(adsValidation,pool);

else

numPar = 1;

end

parfor ii = 1:numPar

subds = partition(adsValidation,numPar,ii);

XValidation = zeros(numHops,numBands,1,numel(subds.Files));

for idx = 1:numel(subds.Files)

x = read(subds);

xPadded = [zeros(floor((segmentSamples-size(x,1))/2),1);x;zeros(ceil((segmentSamples-size(x,1))/2),1)];

XValidation(:,:,:,idx) = extract(afe,xPadded);

end

XValidationC{ii} = XValidation;

end

XValidation = cat(4,XValidationC{:});

XValidation = XValidation/unNorm;

XValidation = log10(XValidation + epsil);

%%

% Isolate the train and validation labels. Remove empty categories.

YTrain = removecats(adsTrain.Labels);

YValidation = removecats(adsValidation.Labels);

%% Visualize Data

% Plot the waveforms and auditory spectrograms of a few training samples.

% Play the corresponding audio clips.

specMin = min(XTrain,[],'all');

specMax = max(XTrain,[],'all');

idx = randperm(numel(adsTrain.Files),3);

figure('Units','normalized','Position',[0.2 0.2 0.6 0.6]);

for i = 1:3

[x,fs] = audioread(adsTrain.Files{idx(i)});

subplot(2,3,i)

plot(x)

axis tight

title(string(adsTrain.Labels(idx(i))))

subplot(2,3,i+3)

spect = (XTrain(:,:,1,idx(i))');

pcolor(spect)

caxis([specMin specMax])

shading flat

sound(x,fs)

pause(2)

end

%% Add Background Noise Data

% The network must be able not only to recognize different spoken words but

adsBkg = subset(ads,ads.Labels=="\_background\_noise\_");

numBkgClips = 4000;

if reduceDataset

numBkgClips = numBkgClips/20;

end

volumeRange = log10([1e-4,1]);

numBkgFiles = numel(adsBkg.Files);

numClipsPerFile = histcounts(1:numBkgClips,linspace(1,numBkgClips,numBkgFiles+1));

Xbkg = zeros(size(XTrain,1),size(XTrain,2),1,numBkgClips,'single');

bkgAll = readall(adsBkg);

ind = 1;

for count = 1:numBkgFiles

bkg = bkgAll{count};

idxStart = randi(numel(bkg)-fs,numClipsPerFile(count),1);

idxEnd = idxStart+fs-1;

gain = 10.^((volumeRange(2)-volumeRange(1))\*rand(numClipsPerFile(count),1) + volumeRange(1));

for j = 1:numClipsPerFile(count)

x = bkg(idxStart(j):idxEnd(j))\*gain(j);

x = max(min(x,1),-1);

Xbkg(:,:,:,ind) = extract(afe,x);

if mod(ind,1000)==0

disp("Processed " + string(ind) + " background clips out of " + string(numBkgClips))

end

ind = ind + 1;

end

end

Xbkg = Xbkg/unNorm;

Xbkg = log10(Xbkg + epsil);

%%

% Split the spectrograms of background noise between the training,

numTrainBkg = floor(0.85\*numBkgClips);

numValidationBkg = floor(0.15\*numBkgClips);

XTrain(:,:,:,end+1:end+numTrainBkg) = Xbkg(:,:,:,1:numTrainBkg);

YTrain(end+1:end+numTrainBkg) = "background";

XValidation(:,:,:,end+1:end+numValidationBkg) = Xbkg(:,:,:,numTrainBkg+1:end);

YValidation(end+1:end+numValidationBkg) = "background";

%%

% Plot the distribution of the different class labels in the training and

% validation sets.

figure('Units','normalized','Position',[0.2 0.2 0.5 0.5])

subplot(2,1,1)

histogram(YTrain)

title("Training Label Distribution")

subplot(2,1,2)

histogram(YValidation)

title("Validation Label Distribution")

%% Define Neural Network Architecture

% Create a simple network architecture as an array of layers. Use

classWeights = 1./countcats(YTrain);

classWeights = classWeights'/mean(classWeights);

numClasses = numel(categories(YTrain));

timePoolSize = ceil(numHops/8);

dropoutProb = 0.2;

numF = 12;

layers = [

imageInputLayer([numHops numBands])

convolution2dLayer(3,numF,'Padding','same')

batchNormalizationLayer

reluLayer

maxPooling2dLayer(3,'Stride',2,'Padding','same')

convolution2dLayer(3,2\*numF,'Padding','same')

batchNormalizationLayer

reluLayer

maxPooling2dLayer(3,'Stride',2,'Padding','same')

convolution2dLayer(3,4\*numF,'Padding','same')

batchNormalizationLayer

reluLayer

maxPooling2dLayer(3,'Stride',2,'Padding','same')

convolution2dLayer(3,4\*numF,'Padding','same')

batchNormalizationLayer

reluLayer

convolution2dLayer(3,4\*numF,'Padding','same')

batchNormalizationLayer

reluLayer

maxPooling2dLayer([timePoolSize,1])

dropoutLayer(dropoutProb)

fullyConnectedLayer(numClasses)

softmaxLayer

weightedClassificationLayer(classWeights)];

%% Train Network

miniBatchSize = 128;

validationFrequency = floor(numel(YTrain)/miniBatchSize);

options = trainingOptions('adam', ...

'InitialLearnRate',3e-4, ...

'MaxEpochs',25, ...

'MiniBatchSize',miniBatchSize, ...

'Shuffle','every-epoch', ...

'Plots','training-progress', ...

'Verbose',false, ...

'ValidationData',{XValidation,YValidation}, ...

'ValidationFrequency',validationFrequency, ...

'LearnRateSchedule','piecewise', ...

'LearnRateDropFactor',0.1, ...

'LearnRateDropPeriod',20);

%%

% Train the network. If you do not have a GPU, then training the network

% can take time.

trainedNet = trainNetwork(XTrain,YTrain,layers,options);

%% Evaluate Trained Network

if reduceDataset

load('commandNet.mat','trainedNet');

end

YValPred = classify(trainedNet,XValidation);

validationError = mean(YValPred ~= YValidation);

YTrainPred = classify(trainedNet,XTrain);

trainError = mean(YTrainPred ~= YTrain);

disp("Training error: " + trainError\*100 + "%")

disp("Validation error: " + validationError\*100 + "%")

%%

% Plot the confusion matrix. Display the precision and recall for each

figure('Units','normalized','Position',[0.2 0.2 0.5 0.5]);

cm = confusionchart(YValidation,YValPred);

cm.Title = 'Confusion Matrix for Validation Data';

cm.ColumnSummary = 'column-normalized';

cm.RowSummary = 'row-normalized';

sortClasses(cm, [commands,"unknown","background"])

%%

% When working on applications with constrained hardware resources such as

info = whos('trainedNet');

disp("Network size: " + info.bytes/1024 + " kB")

for i = 1:100

x = randn([numHops,numBands]);

tic

[YPredicted,probs] = classify(trainedNet,x,"ExecutionEnvironment",'cpu');

time(i) = toc;

end

disp("Single-image prediction time on CPU: " + mean(time(11:end))\*1000 + " ms")