Speech Command Recognition Using Deep Learning

Hafiz Harun Jamil

AD20216005

Pattern Recognition

Introduction

- This presentation shows you how to train a deep learning model that detects the presence of speech commands in audio.
- The example used in this presentation uses the Speech Commands Dataset [1] to train a convolutional neural network to recognize a given set of commands.
- To train a network from scratch, you must first download the data set, or one can load a pretrained network.
- This trained model
  - *Recognize Commands with a Pre-Trained Network*
  - *Detect Commands Using Streaming Audio from Microphone*

Detect Commands Using Streaming Audio from Microphone

- Pre-trained speech recognition network is used to identify speech commands.
- To load the pre-trained network load('commandNet.mat') is used.

```
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 "any of the above mentioned words".

```
[x,fs] = audioread('stop_command.flac');
```

Define Neural Network Architecture

- Create a simple network architecture as an array of layers.
- Use convolutional and batch normalization layers, and downsample the feature maps "spatially" (that is, in time and frequency) using max pooling layers.
- Add a final max pooling layer that pools the input feature map globally over time.
- This approximate time-translation invariance in the input spectrograms, allowing the network to perform the same classification independent of the exact position of the speech in time.
- Global pooling also significantly reduces the number of parameters in the final fully connected layer.
- To reduce the possibility of the network memorizing specific features of the training data, add a small amount of dropout to the input to the last fully connected layer.

Cont …

- The network is small, as it has only five convolutional layers with few filters.
- numF controls the number of filters in the convolutional layers.
- To increase the accuracy of the network, try increasing the network depth by adding identical blocks of convolutional, batch normalization, and ReLU layers.
- You can also try increasing the number of convolutional filters by increasing numF.
- Use a weighted cross entropy classification loss.
- weightedClassificationLayer(classWeights) creates a custom classification layer that calculates the cross entropy loss with observations weighted by classWeights.
- Specify the class weights in the same order as the classes appear in categories(YTrain).
- To give each class equal total weight in the loss, use class weights that are inversely proportional to the number of training examples in each class.
- When using the Adam optimizer to train the network, the training algorithm is independent of the overall normalization of the class weights.

```
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)];
```

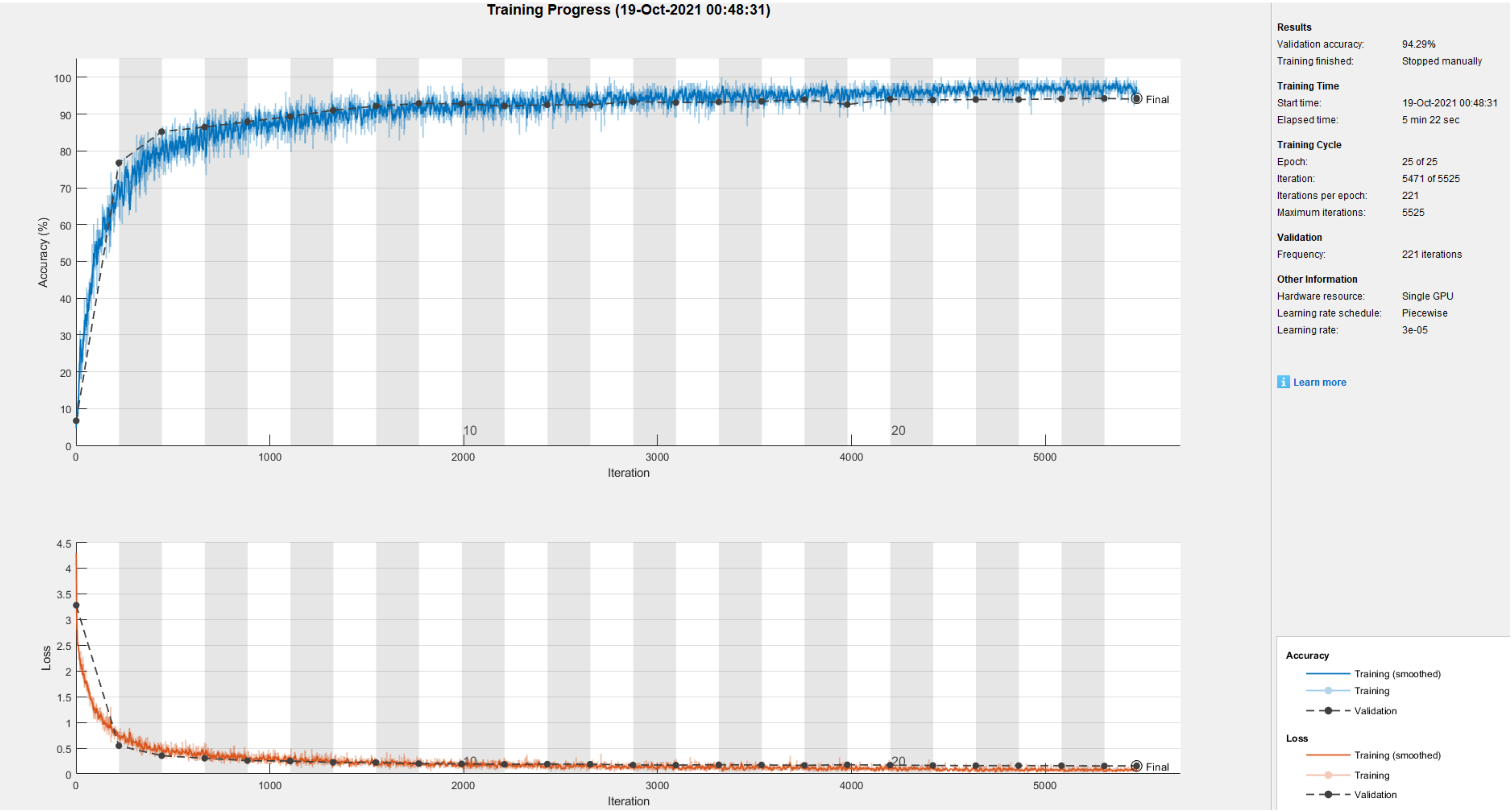
Train Network

- Specify the training options. Use the Adam optimizer with a mini-batch size of 128. Train for 25 epochs and reduce the learning rate by a factor of 10 after 20 epochs.

```
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 by using the command `trainedNet = trainNetwork(XTrain,YTrain,layers,options);`

Train Network



Evaluate Trained Network

- Calculate the final accuracy of the network on the training set (without data augmentation) and validation set.
- The network is very accurate on this data set. However, the training, validation, and test data all have similar distributions that do not necessarily reflect real-world environments.

Confusion Matrix for Validation Data

True Class \ Predicted Class	yes	no	up	down	left	right	on	off	stop	go	unknown	background		
yes	252	3		1			1				1	3	96.6%	3.4%
no		255		5	1					4	5		94.4%	5.6%
up		1	244				1	6			5	3	93.8%	6.2%
down		10		247			1			3	3		93.6%	6.4%
left	3	4			237	1				1		1	96.0%	4.0%
right		2	1		3	246					4		96.1%	3.9%
on			4				238	6			5	4	92.6%	7.4%
off			10				3	242			1		94.5%	5.5%
stop			2	1	1			2	234	1	3	2	95.1%	4.9%
go		13	3	3			1			233	3	4	89.6%	10.4%
unknown	2	8	6	11	2	10	15	4	5	18	789	1	90.6%	9.4%
background												600	100.0%	
	98.1%	86.1%	90.4%	92.2%	97.1%	95.7%	91.5%	93.1%	97.9%	89.6%	96.3%	97.1%		
	1.9%	13.9%	9.6%	7.8%	2.9%	4.3%	8.5%	6.9%	2.1%	10.4%	3.7%	2.9%		

True Class

Predicted Class

```
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 + "%")
```

Cont…

- When working on applications with constrained hardware resources such as mobile applications, consider the limitations on available memory and computational resources.
- Compute the total size of the network in kilobytes and test its prediction speed when using a CPU.
- The prediction time is the time for classifying a single input image.
- If you input multiple images to the network, these can be classified simultaneously, leading to shorter prediction times per image.
- When classifying streaming audio, however, the single-image prediction time is the most relevant.

```
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")
```

Network size: 286.7402 kB

Single-image prediction time on CPU: 2.495 ms

Pre-trained network takes auditory-based spectrograms

- Listen to the command by using this command.

```
sound(x,fs)
```

- The pre-trained network takes auditory-based spectrograms as inputs.
- First convert the speech waveform to an auditory-based spectrogram by using the function "extractAuditoryFeature".

```
auditorySpect = helperExtractAuditoryFeatures(x,fs);
```

- The next step is to classify the command based on its auditory spectrogram by using the command

```
command = classify(trainedNet,auditorySpect)
```

Unknown Auditory Spectrogram

- For the case of unknown auditory spectrogram, the network is trained to classify words not belonging to this set as "unknown".
- For example:
- Classify a word (“any") that was not included in the list (slide:3) of command to identify.
- Load the speech signal and listen to it by using command

```
x = audioread('play_command.flac');
sound(x,fs)
```

- Compute the auditory spectrogram

```
auditorySpect = helperExtractAuditoryFeatures(x,fs);
```

- Classify the signal.

```
command = classify(trainedNet,auditorySpect)
```

```
command =

  categorical

     unknown
```

Background Noise

- It is important to consider the noise factor in speech recognition, for that purpose the network is trained to classify background noise as "background".
- Create a one-second signal consisting of random noise by using command

```
x = pinknoise(16e3);
```

- Compute the auditory spectrogram

```
auditorySpect = helperExtractAuditoryFeatures(x,fs);
```

- Classify the background noise

```
command = classify(trainedNet,auditorySpect)
```

```
command = 
  categorical
     background
```

Detect Commands Using Streaming Audio from Microphone

- To test pre-trained command detection network on streaming audio from your microphone.
- Try saying one of the commands, from the list in slide 3.
- Then, try saying one of the unknown words such as *Marvin*, *Sheila*, *bed*, *house*, *cat*, *bird*, or any number from zero to nine.
- Specify the classification rate in Hz and create an audio device reader that can read audio from your microphone by using the command

```
classificationRate = 20;
adr = audioDeviceReader('SampleRate',fs,'SamplesPerFrame',floor(fs/classificationRate));
```

Buffer Initialize

- Initialize a buffer for the audio.
- Extract the classification labels of the network.
- Initialize buffers of half a second for the labels and classification probabilities of the streaming audio.
- Use these buffers to compare the classification results over a longer period of time and by that build 'agreement' over when a command is detected.
- Specify thresholds for the decision logic.

```
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;
```

Cont …

- Create a figure and detect commands as long as the created figure exists.
- To run the loop indefinitely, set timeLimit to Inf.
- To stop the live detection, simply close the figure.

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

timeLimit = 20;

tic
while ishandle(h) && toc < timeLimit
```

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

    spec = helperExtractAuditoryFeatures(y,fs);
```

```
    [YPredicted,probs] = classify(trainedNet,spec,'ExecutionEnvironment','cpu');
    YBuffer = [YBuffer(2:end),YPredicted];
    probBuffer = [probBuffer(:,2:end),probs(:)];
```

```
    subplot(2,1,1)
    plot(y)
    axis tight
    ylim([-1,1])

    subplot(2,1,2)
    pcolor(spec')
    caxis([-4 2.6445])
    shading flat
```

```
    [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
```

Load Speech Commands Data Set

- This example uses the Google Speech Commands Dataset [1].
- Download the dataset and untar the downloaded file.
- Set PathToDatabase to the location of the data.

.	2,055,161,8...	?	File folder	4/12/2018 2:45 ...
.\zero	75,377,540	?	File folder	7/28/2017 5:10 ...
.\stop	75,172,872	?	File folder	7/28/2017 5:10 ...
.\seven	75,165,002	?	File folder	7/28/2017 5:10 ...
.\six	75,133,166	?	File folder	7/28/2017 5:10 ...
.\yes	75,056,272	?	File folder	7/28/2017 5:10 ...
.\four	74,925,542	?	File folder	7/28/2017 5:10 ...
.\two	74,830,770	?	File folder	7/28/2017 5:10 ...
.\nine	74,806,158	?	File folder	7/28/2017 5:10 ...
.\right	74,754,036	?	File folder	7/28/2017 5:10 ...
.\no	74,722,134	?	File folder	7/28/2017 5:10 ...
.\down	74,611,672	?	File folder	7/28/2017 5:10 ...
.\five	74,569,366	?	File folder	7/28/2017 5:10 ...
.\one	74,559,656	?	File folder	7/28/2017 5:10 ...
.\up	74,510,102	?	File folder	7/28/2017 5:10 ...
.\off	74,506,438	?	File folder	7/28/2017 5:10 ...
.\left	74,506,080	?	File folder	7/28/2017 5:10 ...
.\go	74,496,148	?	File folder	7/28/2017 5:10 ...
.\on	74,472,528	?	File folder	7/28/2017 5:10 ...
.\three	74,423,116	?	File folder	7/28/2017 5:10 ...
.\eight	74,182,774	?	File folder	7/28/2017 5:10 ...
.\house	55,159,508	?	File folder	7/28/2017 5:10 ...
.\marvin	55,155,270	?	File folder	7/28/2017 5:10 ...
.\dog	54,982,048	?	File folder	7/28/2017 5:10 ...
.\happy	54,922,478	?	File folder	7/28/2017 5:10 ...
.\sheila	54,888,718	?	File folder	7/28/2017 5:10 ...
.\wow	54,836,460	?	File folder	7/28/2017 5:10 ...
.\tree	54,499,448	?	File folder	7/28/2017 5:10 ...
.\cat	54,476,952	?	File folder	7/28/2017 5:10 ...
.\bird	54,409,302	?	File folder	7/28/2017 5:10 ...
.\bed	53,877,116	?	File folder	7/28/2017 5:10 ...
._background_noise_	12,782,147	?	File folder	7/28/2017 5:11 ...

Create Training Datastore

- Create an audioDatastore (Audio Toolbox) that points to the training data set.

```
ads = audioDatastore(fullfile(dataFolder, 'train'), ...
    'IncludeSubfolders',true, ...
    'FileExtensions','.wav', ...
    'LabelSource','foldernames')
```

Choose Words to Recognize

- Specify the words that you want your model to recognize as commands.
- Label all words that are not commands as unknown.
- Labeling words that are not commands as unknown creates a group of words that approximates the distribution of all words other than the commands.
- The network uses this group to learn the difference between commands and all other words.
- To reduce the class imbalance between the known and unknown words and speed up processing, only include a fraction of the unknown words in the training set.

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

isCommand = ismember(ads.Labels,commands);
isUnknown = ~isCommand;

includeFraction = 0.2;
mask = rand(numel(ads.Labels),1) < includeFraction;
isUnknown = isUnknown & mask;
ads.Labels(isUnknown) = categorical("unknown");

adsTrain = subset(ads,isCommand|isUnknown);
countEachLabel(adsTrain)
```

```
ans =

  11×2 table

     Label     Count
    _______    _____

    down       1842
    go         1861
    left       1839
    no         1853
    off        1839
    on         1864
    right      1852
    stop       1885
    unknown    6483
    up         1843
    yes        1860
```

Create Validation Datastore

- Create an audioDatastore (Audio Toolbox) that points to the validation data set. Follow the same steps used to create the training datastore.

```
ads = audioDatastore(fullfile(dataFolder, 'validation'), ...
    'IncludeSubfolders',true, ...
    'FileExtensions','.wav', ...
    'LabelSource','foldernames')

isCommand = ismember(ads.Labels,commands);
isUnknown = ~isCommand;

includeFraction = 0.2;
mask = rand(numel(ads.Labels),1) < includeFraction;
isUnknown = isUnknown & mask;
ads.Labels(isUnknown) = categorical("unknown");

adsValidation = subset(ads,isCommand|isUnknown);
countEachLabel(adsValidation)
```

```
ads =

  audioDatastore with properties:

                       Files: {
                              ' ...\AppData\Local\Temp\google_speech\validation\bed\026290a7_nohash_0.wav';
                              ' ...\AppData\Local\Temp\google_speech\validation\bed\060cd039_nohash_0.wav';
                              ' ...\AppData\Local\Temp\google_speech\validation\bed\060cd039_nohash_1.wav'
                               ... and 6795 more
                              }
                     Folders: {
                              'C:\Users\jibrahim\AppData\Local\Temp\google_speech\validation'
                              }
                      Labels: [bed; bed; bed ... and 6795 more categorical]
    AlternateFileSystemRoots: {}
              OutputDataType: 'double'
      SupportedOutputFormats: ["wav"    "flac"    "ogg"    "mp4"    "m4a"]
         DefaultOutputFormat: "wav"


ans =

  11×2 table

    Label      Count
    _______    _____

    down        264
    go          260
    left        247
    no          270
    off         256
    on          257
    right       256
    stop        246
    unknown     850
    up          260
    yes         261
```

Compute Auditory Spectrograms

- To prepare the data for efficient training of a convolutional neural network, convert the speech waveforms to auditory-based spectrograms.
- Define the parameters of the feature extraction.
- segmentDuration is the duration of each speech clip (in seconds).
- frameDuration is the duration of each frame for spectrum calculation.
- hopDuration is the time step between each spectrum.
- numBands is the number of filters in the auditory spectrogram.

Cont…

- Create an audioFeatureExtractor (Audio Toolbox) 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,'WindowNormalization',false);
```

Note:

- Read a file from the dataset.
- Training a convolutional neural network requires input to be a consistent size.
- Some files in the data set are less than 1 second long.
- Apply zero-padding to the front and back of the audio signal so that it is of length segmentSamples.

```
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)];
```

Cont …

- To extract audio features, call extract. The output is a Bark spectrum with time across rows.

```
features = extract(afe,xPadded);
[numHops,numFeatures] = size(features)
```

- To speed up processing, you can distribute the feature extraction across multiple workers using parfor.
- First, determine the number of partitions for the dataset.
- If you do not have Parallel Computing Toolbox™, use a single partition.

```
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
```

Cont …

- 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)
```

- To obtain data with a smoother distribution, take the logarithm of the spectrograms using a small offset.

```
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 = log10(XValidation + epsil);
```

Cont ...

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

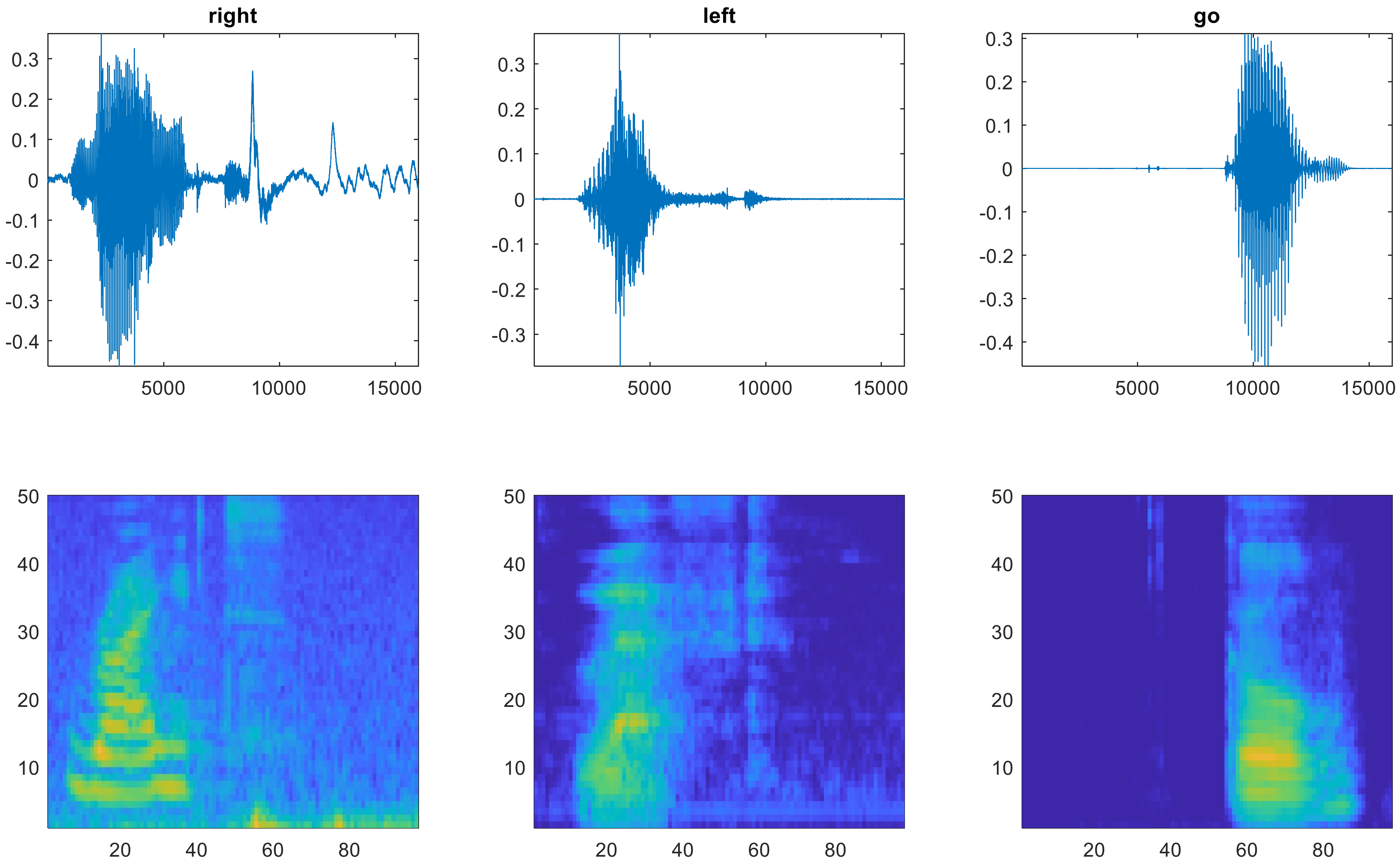
```
YTrain = removecats(adsTrain.Labels);
YValidation = removecats(adsValidation.Labels);
```

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
```

Cont...


right
left
go
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
5000
10000
15000
50
40
30
20
10
20
40
60
80

Add Background Noise Data

- The network must be able not only to recognize different spoken words but also to detect if the input contains silence or background noise.
- Use the audio files in the _background_ folder to create samples of one-second clips of background noise.
- Create an equal number of background clips from each background noise file. You can also create your own recordings of background noise and add them to the _background_ folder.
- Before calculating the spectrograms, the function rescales each audio clip with a factor sampled from a log-uniform distribution in the range given by volumeRange.

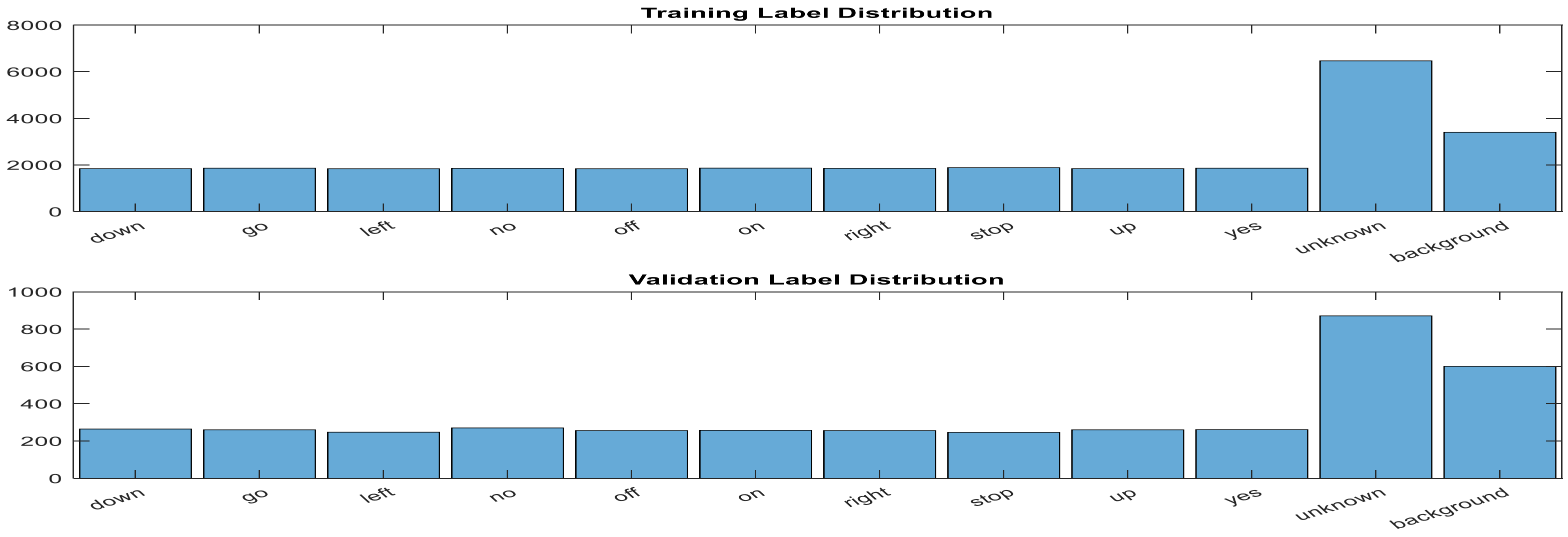
```
adsBkg = audioDatastore(fullfile(dataFolder, 'background'))
numBkgClips = 4000;
if reduceDataset
    numBkgClips = numBkgClips/20;
end
volumeRange = log10([1e-4,1]);

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 = log10(Xbkg + epsil);
```

Cont…

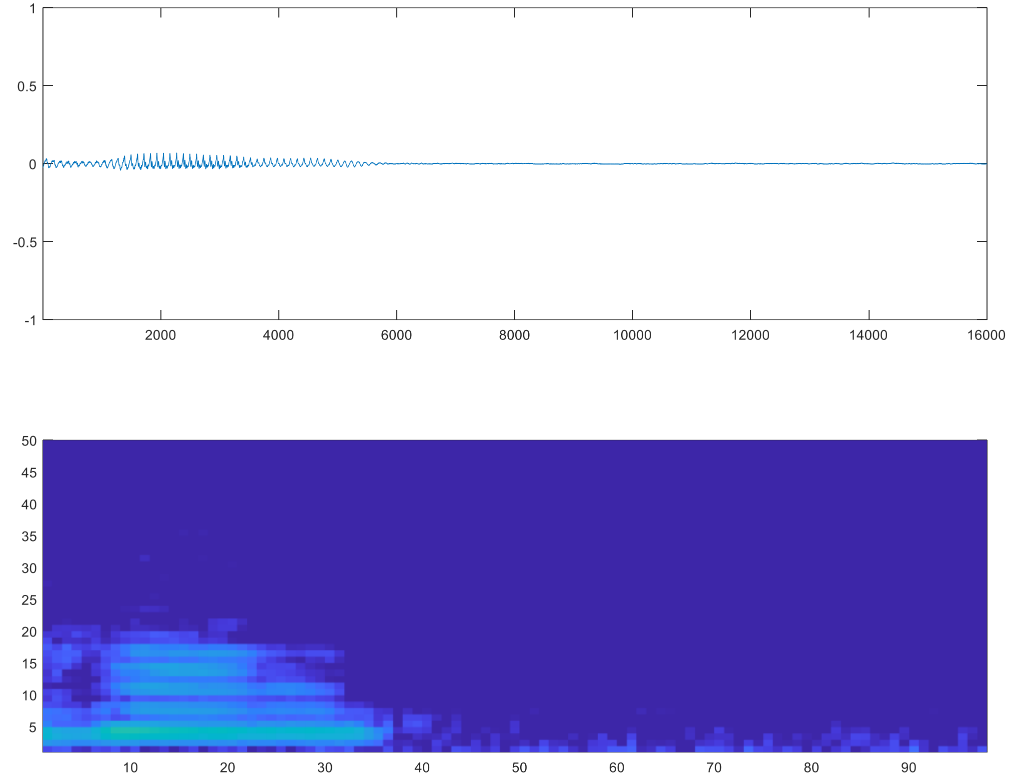
- Split the spectrograms of background noise between the training, validation, and test sets.
- Because the _background_noise_ folder contains only about five and a half minutes of background noise, the background samples in the different data sets are highly correlated.
- To increase the variation in the background noise, you can create your own background files and add them to the folder.
- To increase the robustness of the network to noise, you can also try mixing background noise into the speech files.

```
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";
```

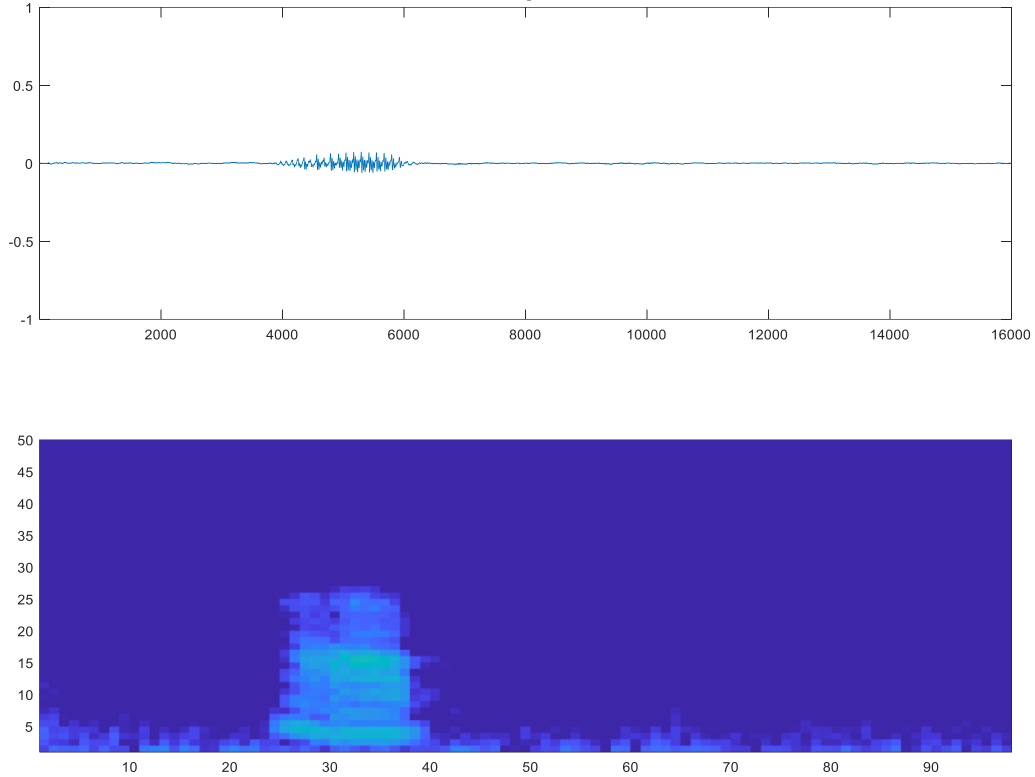
- Plot the distribution of the different class labels in the training and validation sets.

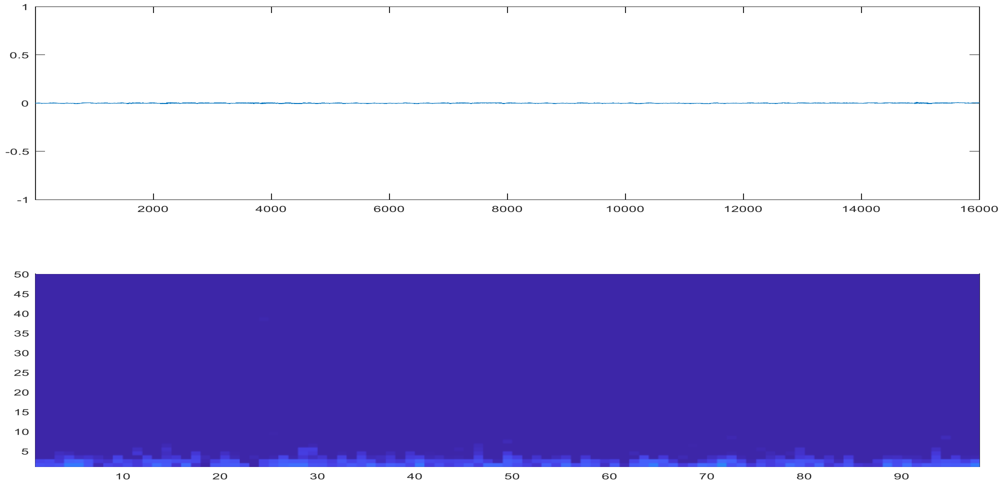


unknown


1
0.5
0
-0.5
-1
2000 4000 6000 8000 10000 12000 14000 16000
50 45 40 35 30 25 20 15 10 5
10 20 30 40 50 60 70 80 90

up


1
0.5
0
-0.5
-1
2000 4000 6000 8000 10000 12000 14000 16000
50 45 40 35 30 25 20 15 10 5
10 20 30 40 50 60 70 80 90


1
0.5
0
-0.5
-1
2000 4000 6000 8000 10000 12000 14000 16000
50 45 40 35 30 25 20 15 10 5
10 20 30 40 50 60 70 80 90