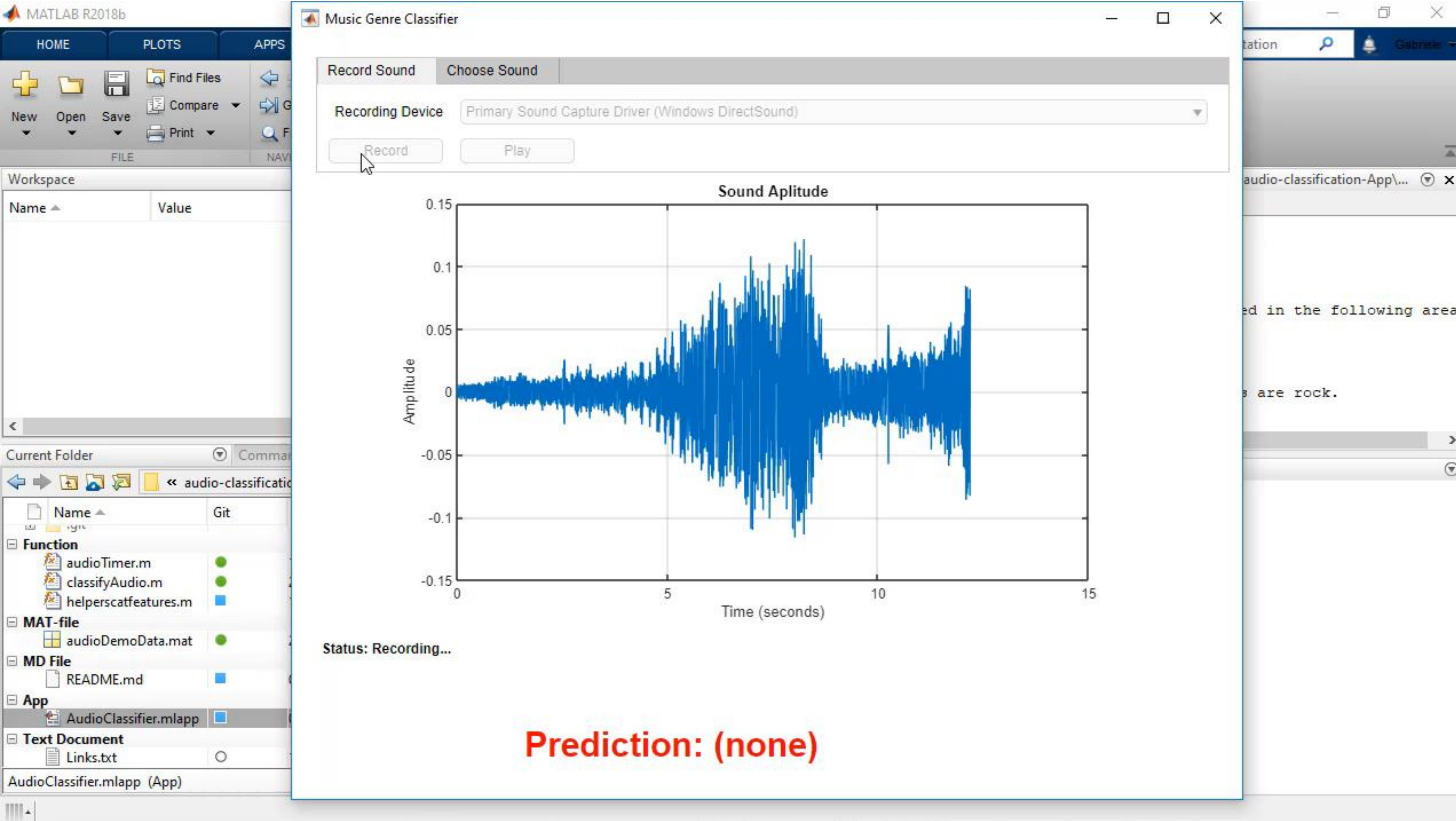
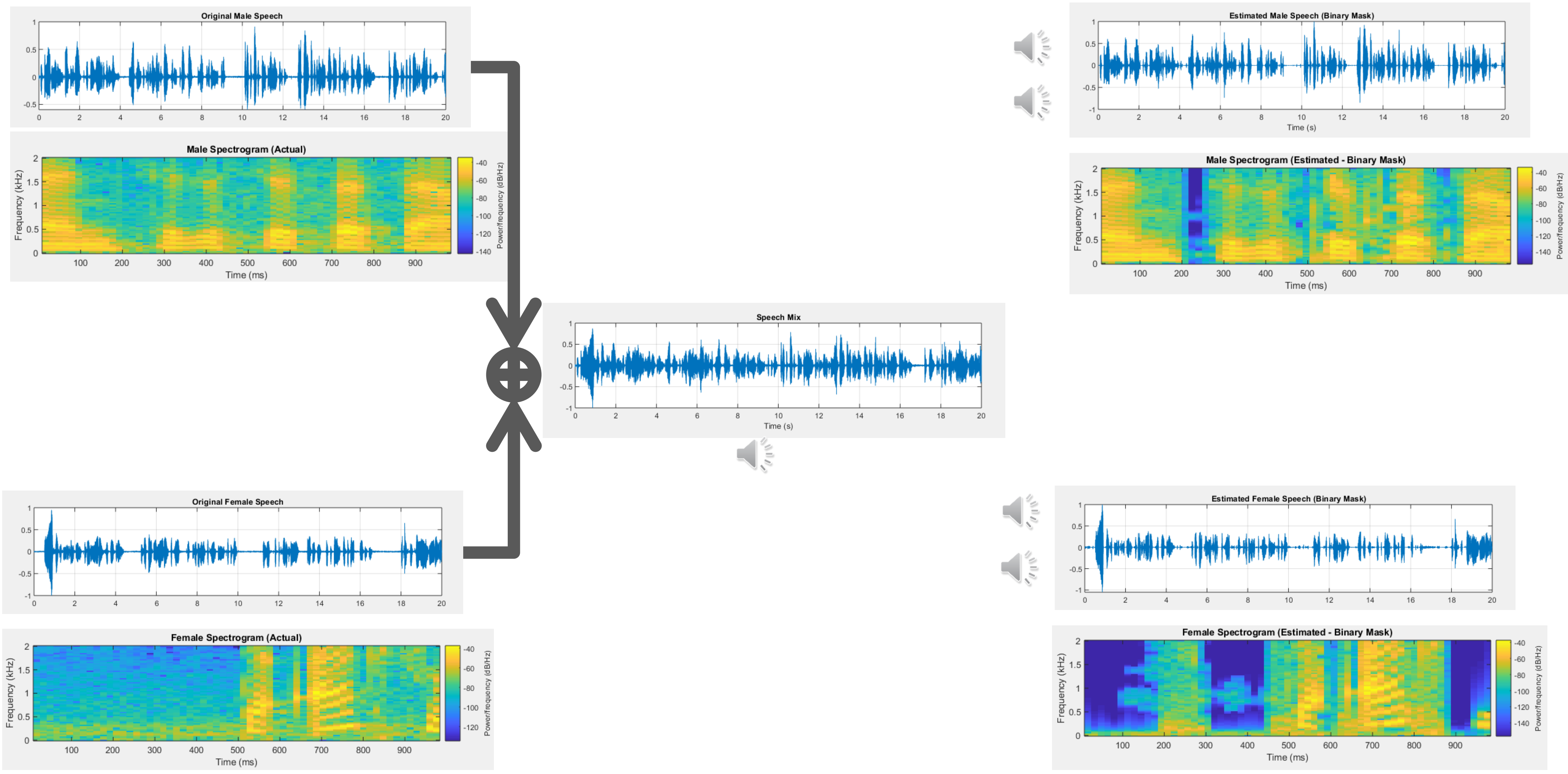
MATLAB EXPO 2018

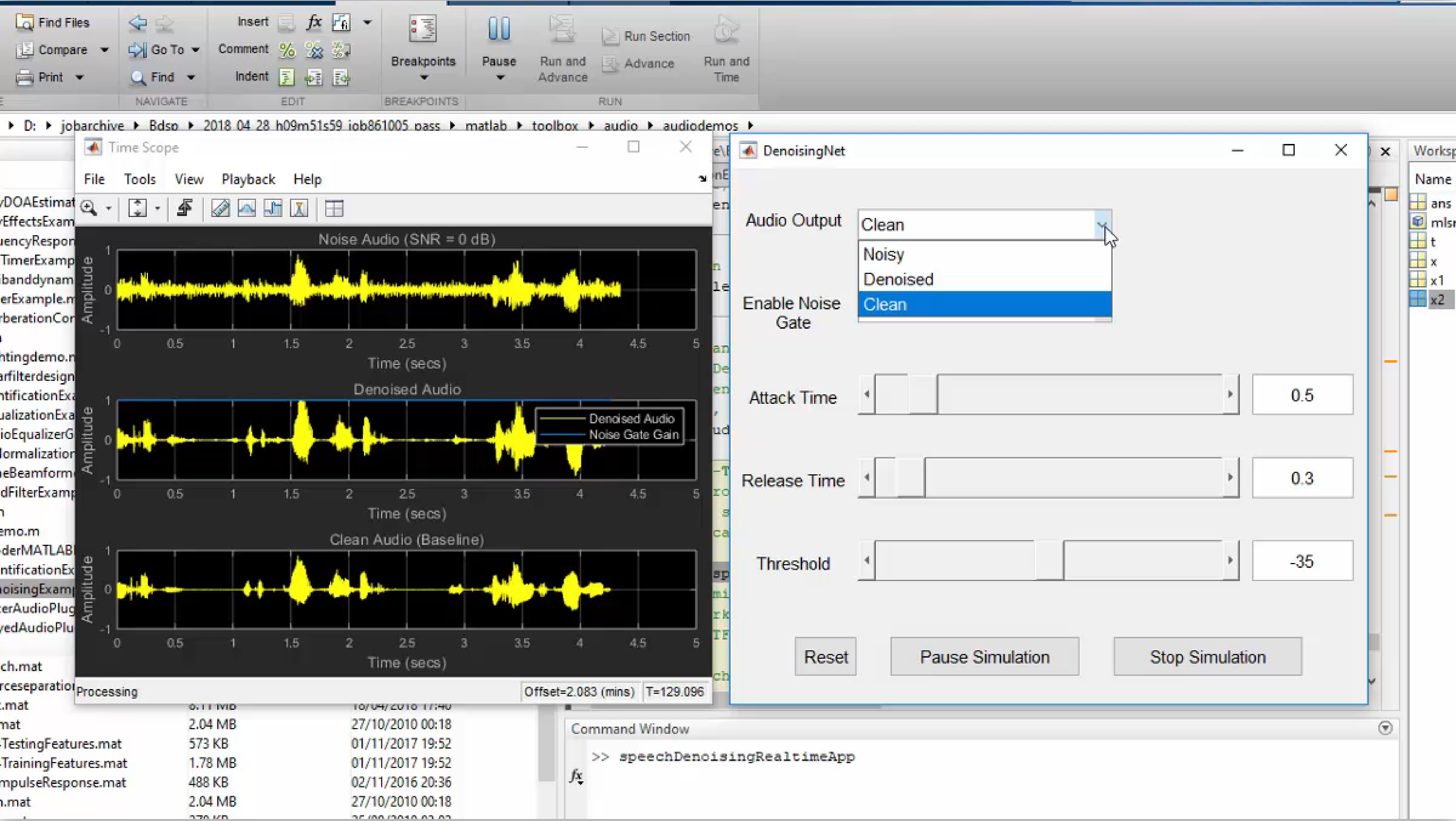
Master Class: Deep Learning for Signals

Abhijit Bhattacharjee
Senior Application Engineer
MathWorks



MATLAB R2018b
HOME
PLOTS
APPS
New
Open
Save
Find Files
Compare
Print
FILE
Workspace
Name
Value
Current Folder
Function
audioTimer.m
classifyAudio.m
helperscatfeatures.m
MAT-file
audioDemoData.mat
MD File
README.md
App
AudioClassifier.mlapp
Text Document
Links.txt
AudioClassifier.mlapp (App)
Music Genre Classifier
Record Sound
Choose Sound
Recording Device
Primary Sound Capture Driver (Windows DirectSound)
Record
Play
Sound Aplitude
Amplitude
Time (seconds)
Status: Recording...
Prediction: (none)
ed in the following area
are rock.


Original Male Speech
Male Spectrogram (Actual)
Original Female Speech
Female Spectrogram (Actual)
Speech Mix
Estimated Male Speech (Binary Mask)
Male Spectrogram (Estimated - Binary Mask)
Estimated Female Speech (Binary Mask)
Female Spectrogram (Estimated - Binary Mask)
Frequency (kHz)
Time (ms)
Time (s)
Power/frequency (dB/Hz)



Find Files
Compare
Print
Go To
Find
NAVIGATE
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Pause
Run and Advance
Run Section
Advance
Run and Time
RUN
D: ▸ jobarchive ▸ Bdsp ▸ 2018_04_28_h09m51s59_job861005_pass ▸ matlab ▸ toolbox ▸ audio ▸ audiodemos ▸
Time Scope
File Tools View Playback Help
Noise Audio (SNR = 0 dB)
Denoised Audio
Denoised Audio
Noise Gate Gain
Clean Audio (Baseline)
Amplitude
Time (secs)
Processing
Offset=2.083 (mins)
T=129.096
DenoisingNet
Audio Output
Clean
Noisy
Denoised
Clean
Enable Noise Gate
Attack Time
0.5
Release Time
0.3
Threshold
-35
Reset
Pause Simulation
Stop Simulation
Command Window
>> speechDenoisingRealtimeApp
Name
ans
t
x
x1
x2
2.04 MB 27/10/2010 00:18
TestingFeatures.mat 573 KB 01/11/2017 19:52
TrainingFeatures.mat 1.78 MB 01/11/2017 19:52
mpulseResponse.mat 488 KB 02/11/2016 20:36
n.mat 2.04 MB 27/10/2010 00:18

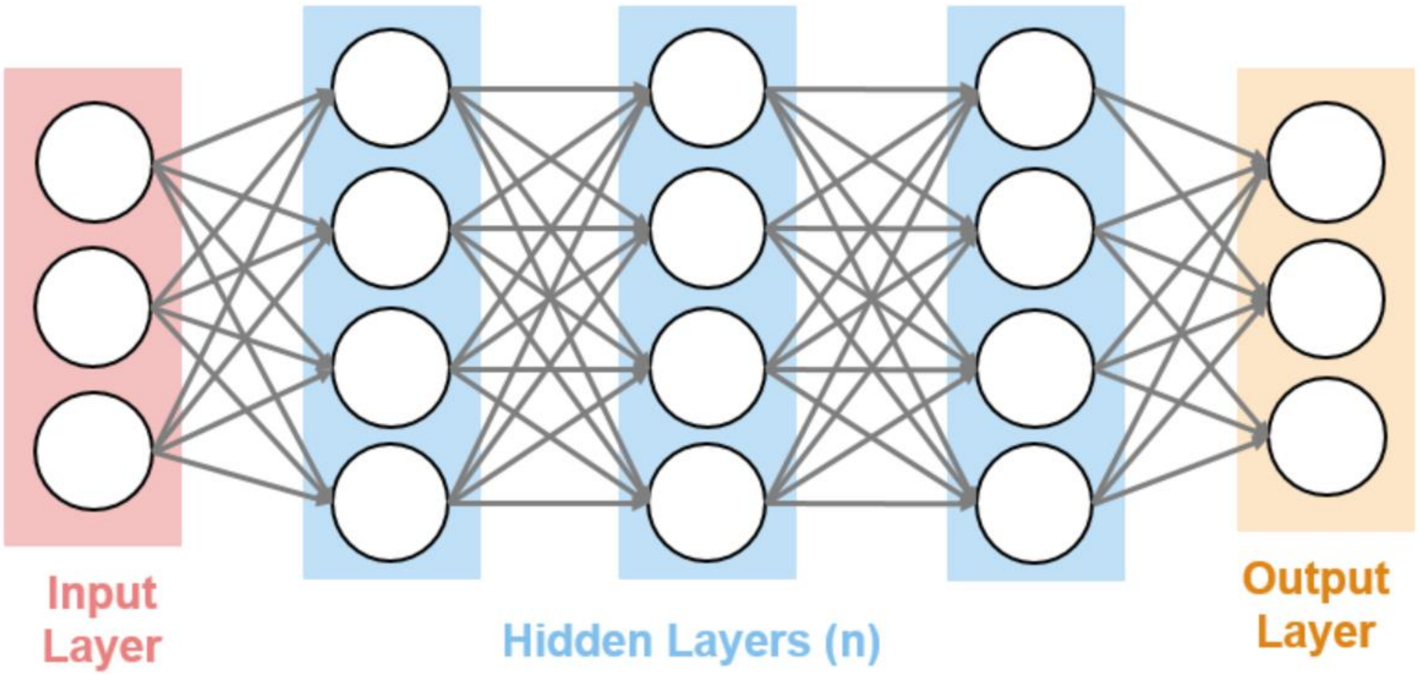
Agenda

Why deep learning?
Deep learning with signal data
(Demo) Speech Command Recognition
(Demo) LSTM Networks
Enabling Features in MATLAB
Deploying deep learning

What is Deep Learning?

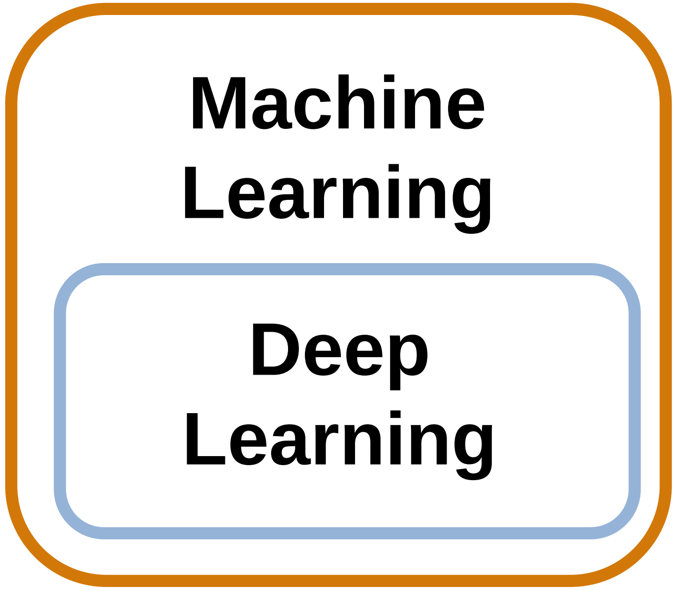
Deep learning is a type of machine learning in which a model learns to perform tasks directly from image, time-series or text data.

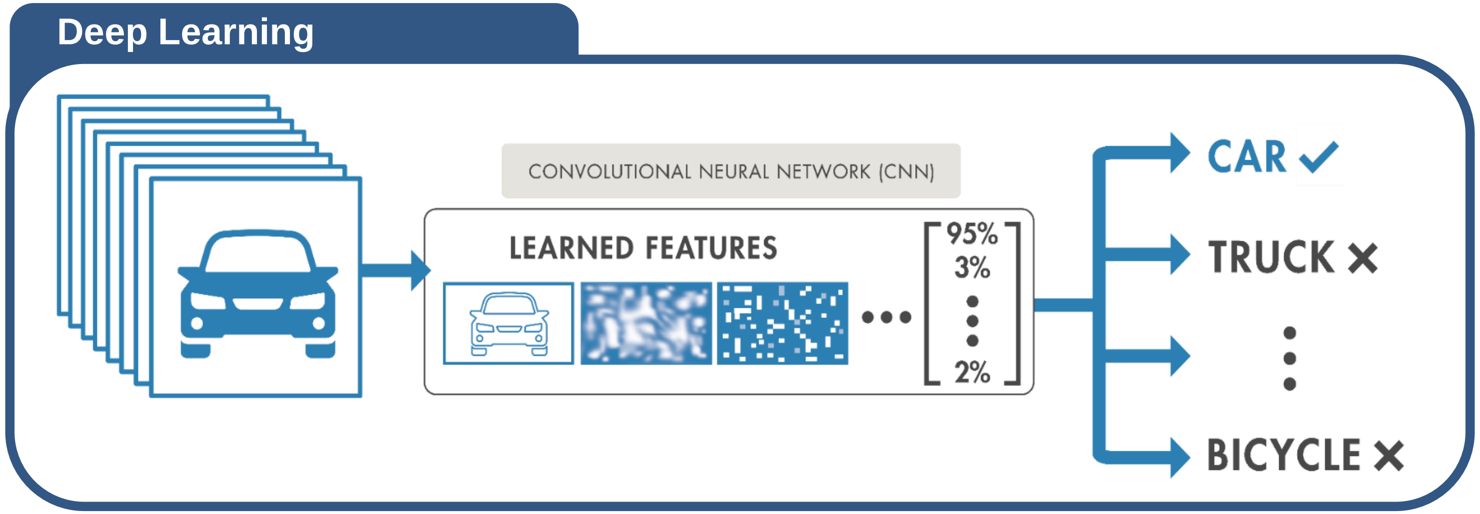
Deep learning is usually implemented using a **neural network architecture.**



What is Deep Learning?

- Subset of machine learning with automatic feature extraction
  - Can learn features directly from data
  - More Data = better model





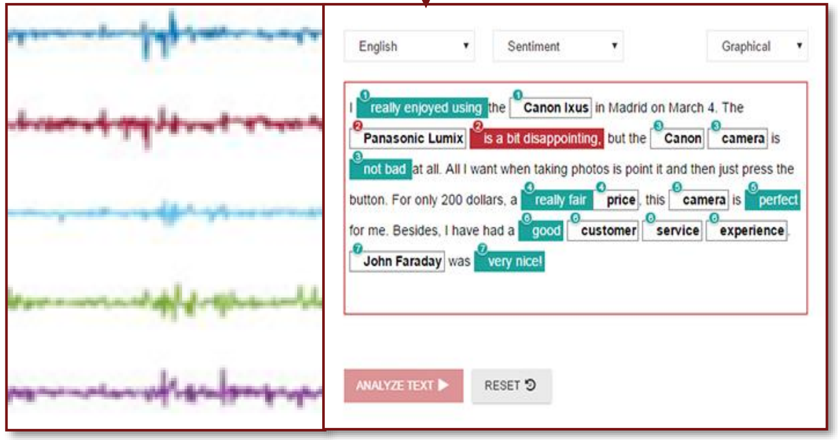
Types of Datasets

Tabular Data

ID	WC_TA	RE_TA	EBIT_TA	MVE_BVTD	S_TA	Industry	Rating
62394	0.013	0.104	0.036	0.447	0.142	3	BB
48608	0.232	0.335	0.062	1.969	0.281	8	A
42444	0.311	0.367	0.074	1.935	0.366	1	A
48631	0.194	0.263	0.062	1.017	0.228	4	BBB
43768	0.121	0.413	0.057	3.647	0.466	12	AAA
39255	-0.117	-0.799	0.01	0.179	0.082	4	CCC
62236	0.087	0.158	0.049	0.816	0.324	2	BBB
39354	0.005	0.181	0.034	2.597	0.388	7	AA
40326	0.47	0.752	0.07	11.596	1.12	8	AAA
51681	0.11	0.337	0.045	3.835	0.812	4	AAA

ML or LSTM

Time Series/ Signal

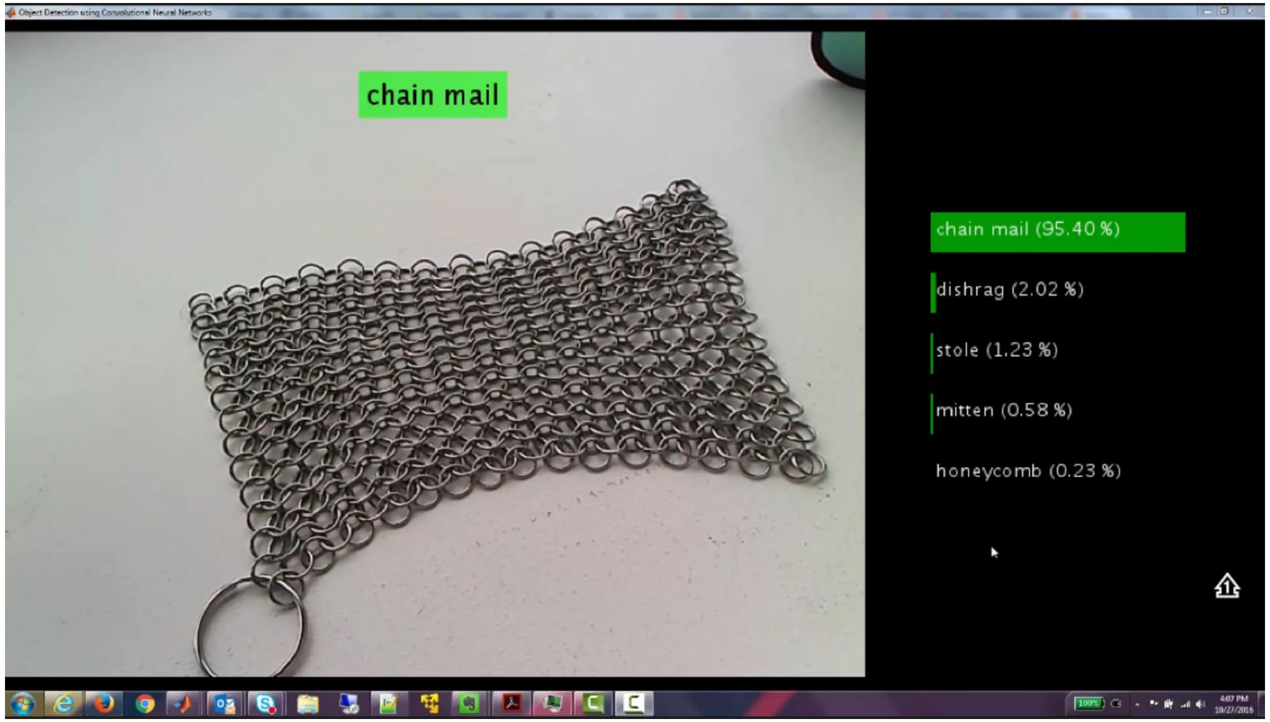


LSTM or CNN

Image Data

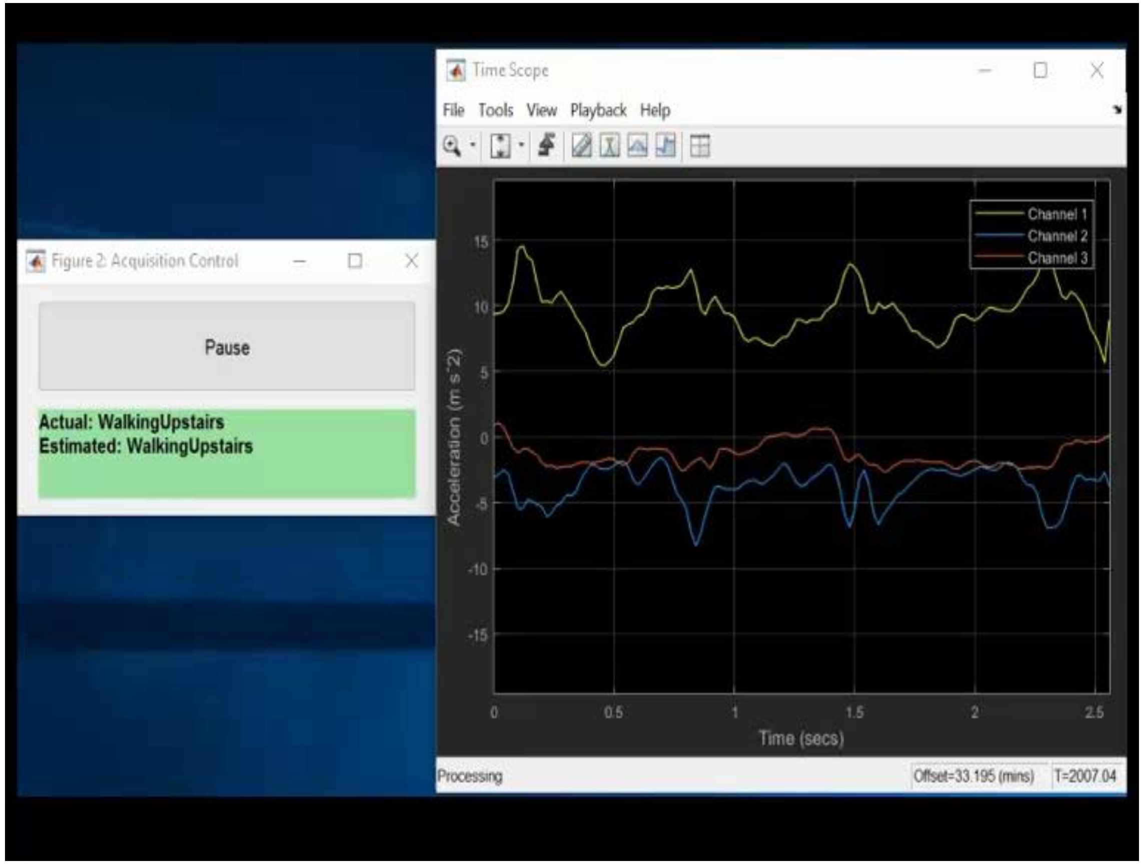
CNN

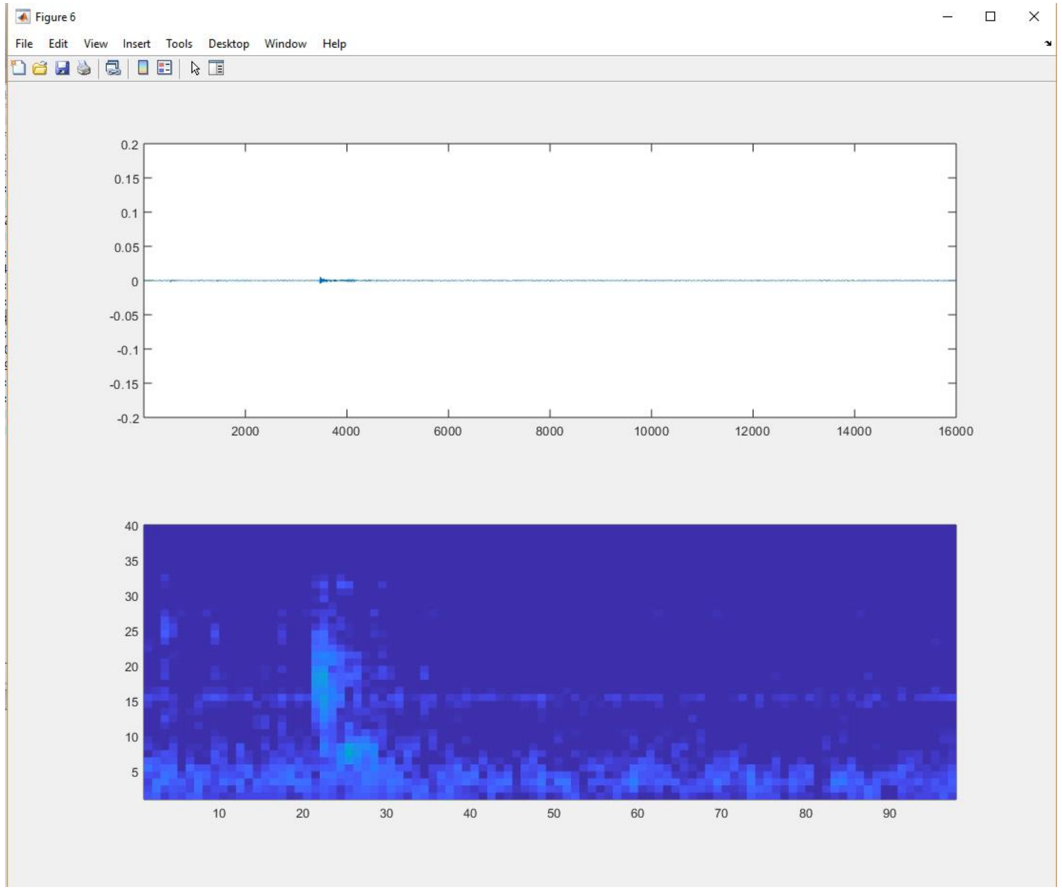
Image Example: Object recognition using deep learning



Training (GPU)	Millions of images from 1000 different categories
Prediction	Real-time object recognition using a webcam connected to a laptop

Signals Example: Analyzing signal data using deep learning





Signal Classification using LSTMs

Speech Recognition using CNNs

Deep Learning Workflow

ACCESS AND EXPLORE DATA → **LABEL AND PREPROCESS DATA** → **DEVELOP PREDICTIVE MODELS** → **INTEGRATE MODELS WITH SYSTEMS**

Access and Explore Data

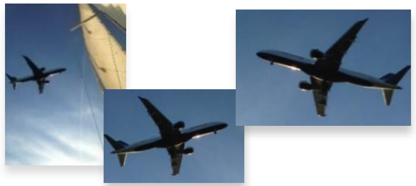
Files

Databases

Sensors

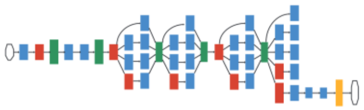
Label and Preprocess Data

Data Augmentation/ Transformation



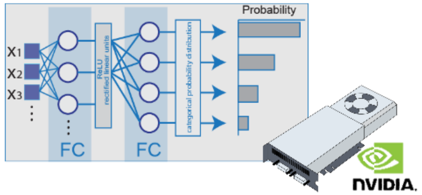
Labeling Automation

Import Reference Models



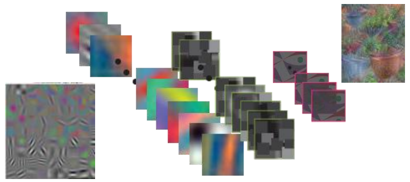
Develop Predictive Models

Hardware-Accelerated Training



Hyperparameter Tuning

Network Visualization

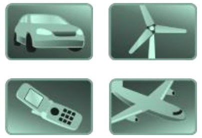


Integrate Models with Systems

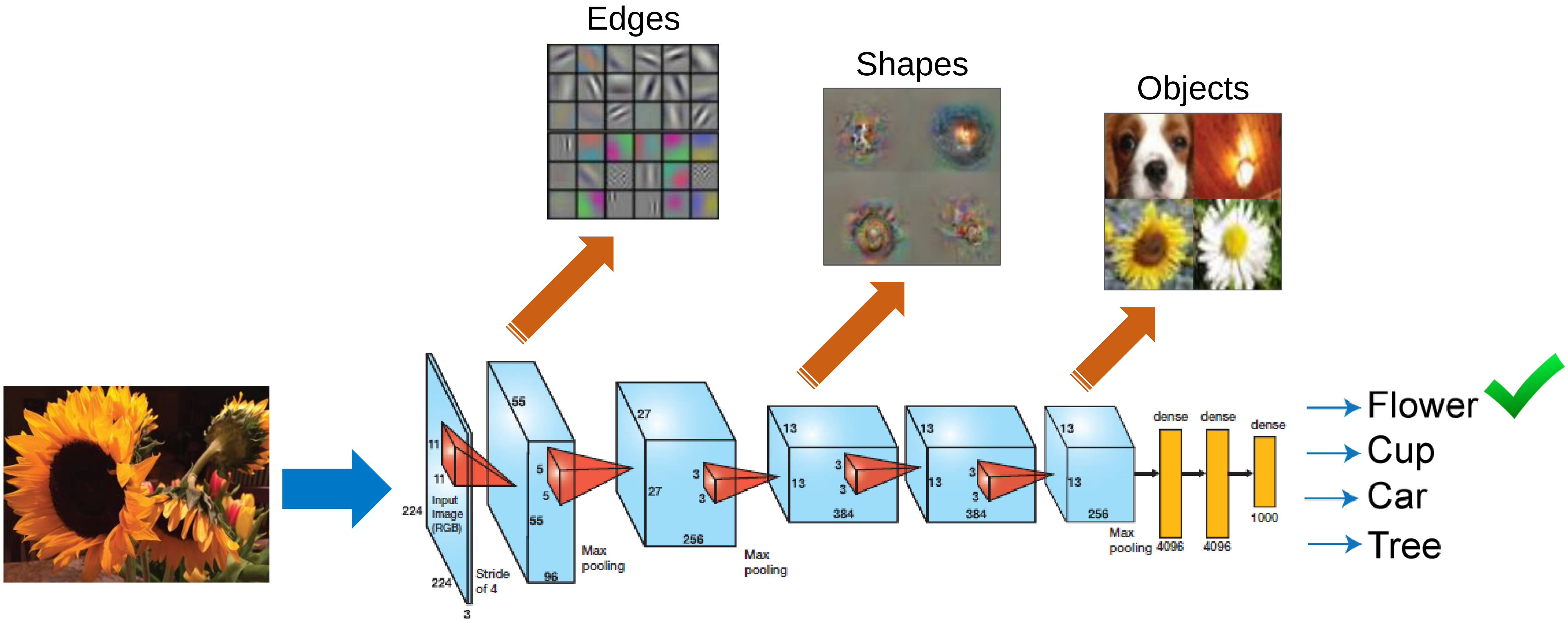
Desktop Apps

Enterprise Scale Systems

Embedded Devices and Hardware

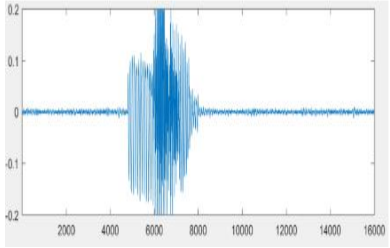


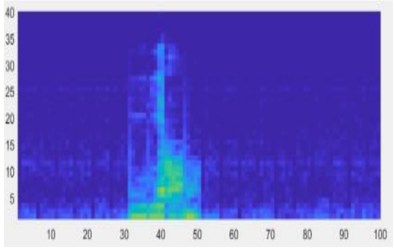
How Does a Convolutional Neural Network Work?

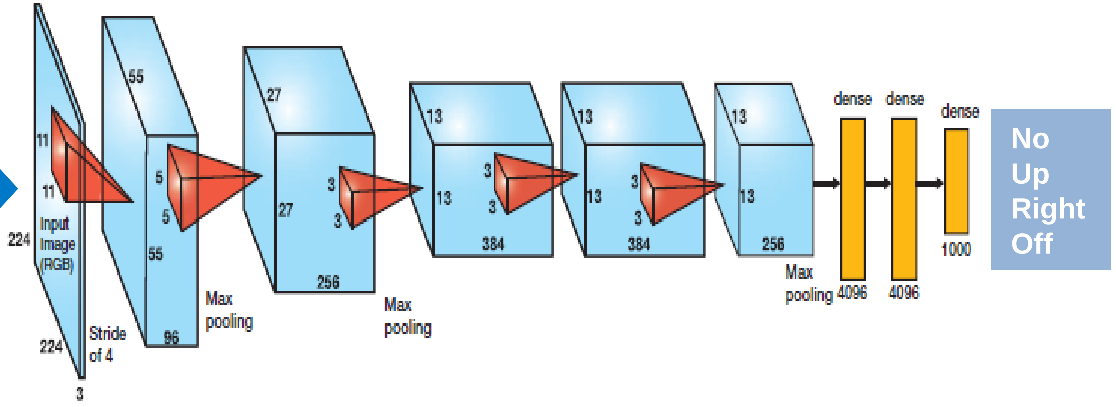


Speech Command Recognition

Using Convolutional Neural Networks



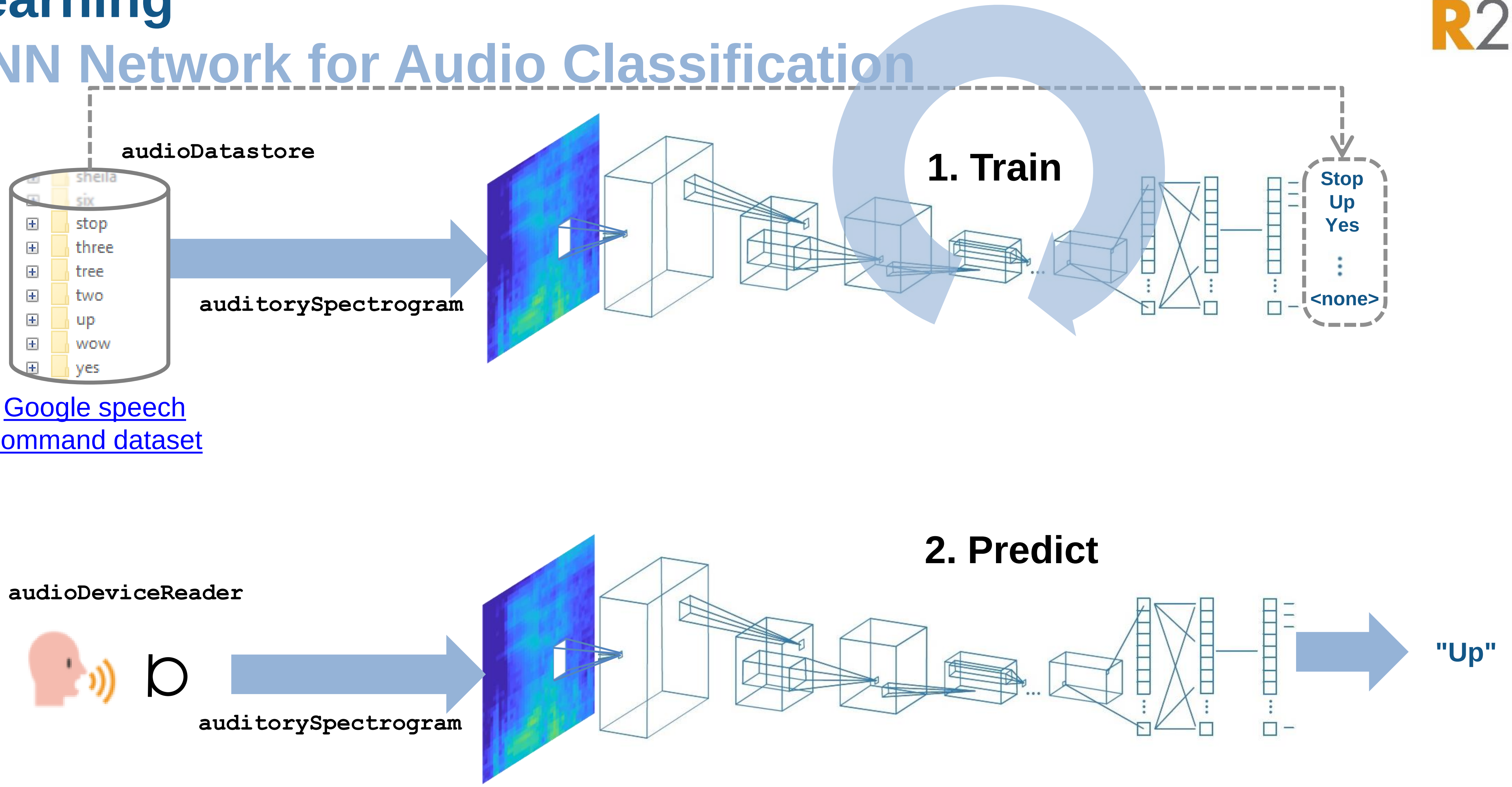




Example: Speech Command Recognition Using Deep Learning

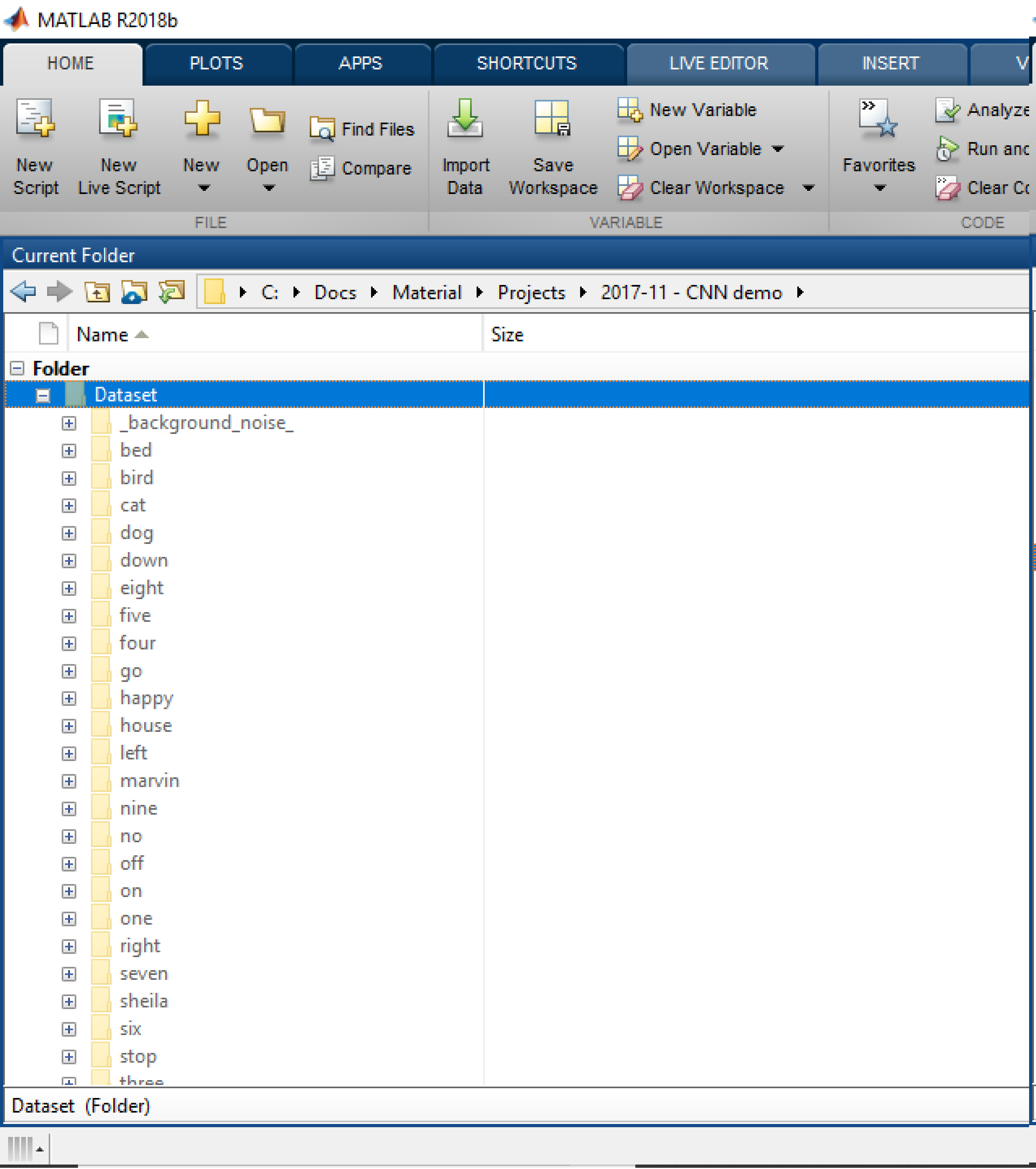
CNN Network for Audio Classification

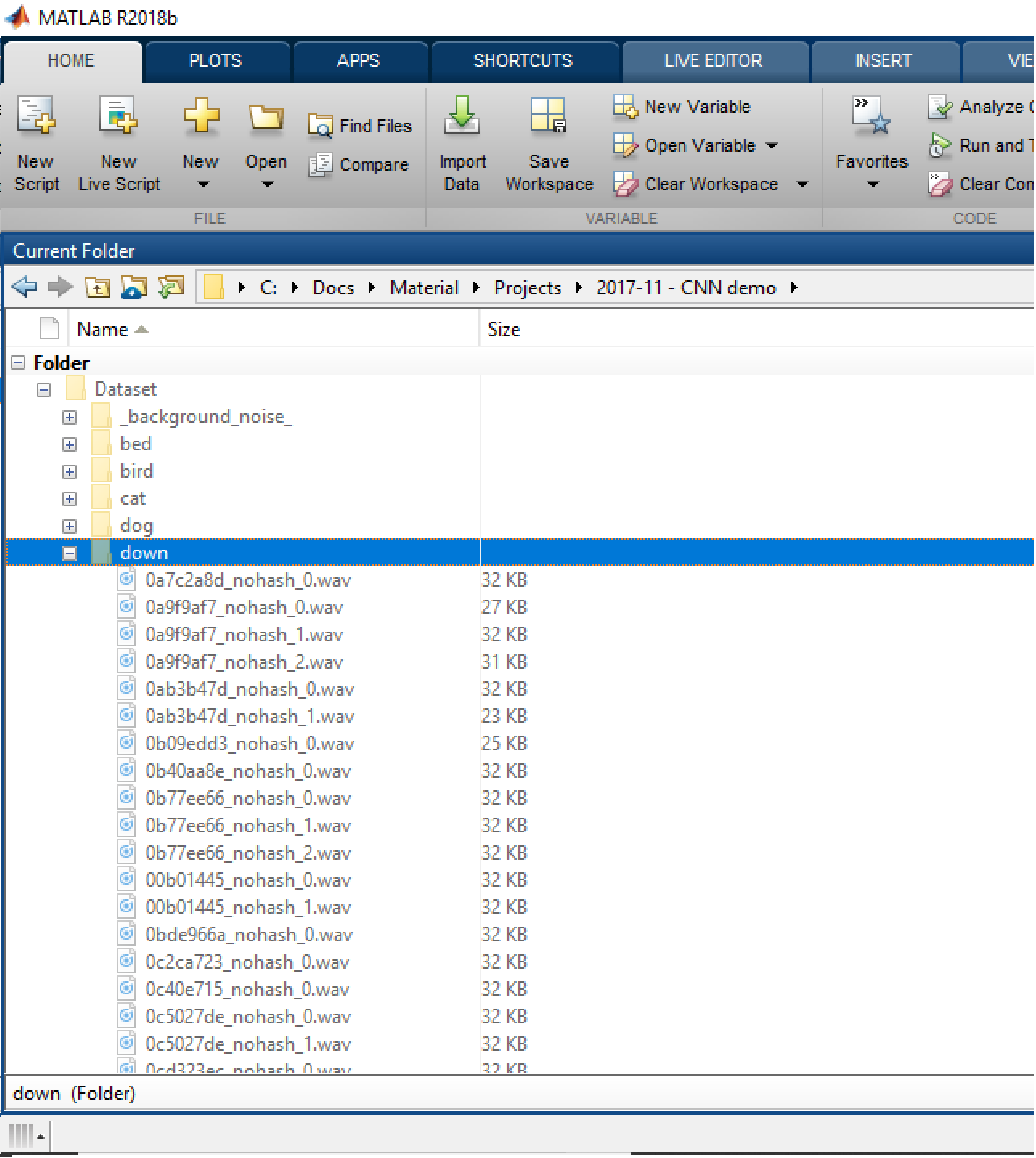
R2018b



[Google speech command dataset]

Command Recognition Demo


MATLAB R2018b
HOME
PLOTS
APPS
SHORTCUTS
LIVE EDITOR
INSERT
New Script
New Live Script
New
Open
Find Files
Compare
FILE
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
VARIABLE
Favorites
CODE
Current Folder
C: Docs Material Projects 2017-11 - CNN demo
Name
Size
Folder
Dataset
_background_noise_
bed
bird
cat
dog
down
eight
five
four
go
happy
house
left
marvin
nine
no
off
on
one
right
seven
sheila
six
stop
Dataset (Folder)


MATLAB R2018b
HOME
PLOTS
APPS
SHORTCUTS
LIVE EDITOR
INSERT
New Script
New Live Script
New
Open
Find Files
Compare
FILE
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
VARIABLE
Favorites
CODE
Current Folder
C: Docs Material Projects 2017-11 - CNN demo
Name
Size
Folder
Dataset
_background_noise_
bed
bird
cat
dog
down
0a7c2a8d_nohash_0.wav 32 KB
0a9f9af7_nohash_0.wav 27 KB
0a9f9af7_nohash_1.wav 32 KB
0a9f9af7_nohash_2.wav 31 KB
0ab3b47d_nohash_0.wav 32 KB
0ab3b47d_nohash_1.wav 23 KB
0b09edd3_nohash_0.wav 25 KB
0b40aa8e_nohash_0.wav 32 KB
0b77ee66_nohash_0.wav 32 KB
0b77ee66_nohash_1.wav 32 KB
0b77ee66_nohash_2.wav 32 KB
00b01445_nohash_0.wav 32 KB
00b01445_nohash_1.wav 32 KB
0bde966a_nohash_0.wav 32 KB
0c2ca723_nohash_0.wav 32 KB
0c40e715_nohash_0.wav 32 KB
0c5027de_nohash_0.wav 32 KB
0c5027de_nohash_1.wav 32 KB
down (Folder)

Speech Command Recognition Using Deep Learning

This example shows how to train a simple deep learning model that detects the presence of speech commands in audio. The example uses the Speech Commands Dataset [1] to train a convolutional neural network to recognize a given set of commands.

To run the example, you must first download the data set. If you do not want to download the data set or train the network, then you can load a pretrained network by opening this example in MATLAB® and typing `load('commandNet.mat')` at the command line. After loading the network, go directly to the last section of this example, *Detect Commands Using Streaming Audio from Microphone*.

Load Speech Commands Data Set

Download the data set from http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz and extract the downloaded file. Set `datafolder` to the location of the data. Use `audioDatastore` to create a datastore that contains the file names and the corresponding labels. Use the folder names as the label source. Specify the read method to read the entire audio file. Create a copy of the datastore for later use. datafolder = fullfile(tempdir,'speech_commands_v0.01');

```
datafolder = fullfile('..','Dataset');
ads = audioDatastore(datafolder, ...
    'IncludeSubfolders',true, ...
    'FileExtensions','.wav', ...
    'LabelSource','foldernames')
```

```
ads = 
  audioDatastore with properties:

                       Files: {
                              ' ...\2017-11 - CNN demo\Dataset\_background_noise_\doing_the_dishes.wav';
                              ' ...\2017-11 - CNN demo\Dataset\_background_noise_\dude_miaowing.wav';
                              ' ...\2017-11 - CNN demo\Dataset\_background_noise_\exercise_bike.wav'
                               ... and 64724 more
                              }
                      Labels: [_background_noise_; _background_noise_; _background_noise_ ... and 64724 more categorical]
[redacted]
```

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 networks 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 `includeFraction` of the unknown words in the training

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 networks 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 `includeFraction` of the unknown words in the training set. Do not include the longer files with background noise in the training set yet. Background noise will be added in a separate step later.

Use `subset(ads,indices)` to create a datastore that contains only the files and labels indexed by `indices`. Reduce the datastore `ads` so that it contains only the commands and the subset of unknown words. Count the number of examples belonging to each class.

```
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");

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

Split Data into Training, Validation, and Test Sets

The data set folder contains text files, which list the audio files to be used as the validation and test sets. These predefined validation and test sets do not contain utterances of the same word by the same person, so it is better to use these predefined sets than to select a random subset of the whole data set. Use the supporting function `splitData` to split the datastore into training, validation, and test sets based on the list of validation and test files located in the data set folder.

[redacted]

Compute Speech Spectrograms

To prepare the data for efficient training of a convolutional neural network, convert the speech waveforms to log-bark auditory spectrograms.

Define the parameters of the spectrogram calculation. `segmentDuration` is the duration of each speech clip (in seconds). `frameDuration` is the duration of each frame for spectrogram calculation. `hopDuration` is the time step between each column of the spectrogram. `numBands` is the number of log-bark filters and equals the height of each

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 networks 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 `includeFraction` of the unknown words in the training set. Do not include the longer files with background noise in the training set yet. Background noise will be added in a separate step later.

Use `subset(ads,indices)` to create a datastore that contains only the files and labels indexed by `indices`. Reduce the datastore `ads` so that it contains only the commands and the subset of unknown words. Count the number of examples belonging to each class.

```
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");

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

Split Data into Training, Validation, and Test Sets

The data set folder contains text files, which list the audio files to be used as the validation and test sets. These predefined validation and test sets do not contain utterances of the same word by the same person, so it is better to use these predefined sets than to select a random subset of the whole data set. Use the supporting function `splitData` to split the datastore into training, validation, and test sets based on the list of validation and test files located in the data set folder.

```
[adsTrain,adsValidation,adsTest] = splitData(ads,datafolder);
```

Compute Speech Spectrograms

To prepare the data for efficient training of a convolutional neural network, convert the speech waveforms to log-bark auditory spectrograms.

Define the parameters of the spectrogram calculation. `segmentDuration` is the duration of each speech clip (in seconds). `frameDuration` is the duration of each frame for spectrogram calculation. `hopDuration` is the time step between each column of the spectrogram. `numBands` is the number of log-bark filters and equals the height of each

Split Data into Training, Validation, and Test Sets

The data set folder contains text files, which list the audio files to be used as the validation and test sets. These predefined validation and test sets do not contain utterances of the same word by the same person, so it is better to use these predefined sets than to select a random subset of the whole data set. Use the supporting function `splitData` to split the datastore into training, validation, and test sets based on the list of validation and test files located in the data set folder.

```
[adsTrain,adsValidation,adsTest] = splitData(ads,datafolder);
```

Compute Speech Spectrograms

To prepare the data for efficient training of a convolutional neural network, convert the speech waveforms to log-bark auditory spectrograms.

Define the parameters of the spectrogram calculation. `segmentDuration` is the duration of each speech clip (in seconds). `frameDuration` is the duration of each frame for spectrogram calculation. `hopDuration` is the time step between each column of the spectrogram. `numBands` is the number of log-bark filters and equals the height of each spectrogram.

```
segmentDuration = 1;
frameDuration = 0.025;
hopDuration = 0.010;
numBands = 40;
```

Compute the spectrograms for the training, validation, and test sets by using the supporting function `speechSpectrograms`. The `speechSpectrograms` function uses `auditorySpectrogram` for the spectrogram calculations. To obtain data with a smoother distribution, take the logarithm of the spectrograms using a small offset `epsil`.

```
[redacted]

XTrain = speechSpectrograms(adsTrain,segmentDuration,frameDuration,hopDuration,numBands);
XTrain = log10(XTrain + epsil);

XValidation = speechSpectrograms(adsValidation,segmentDuration,frameDuration,hopDuration,numBands);
XValidation = log10(XValidation + epsil);

XTest = speechSpectrograms(adsTest,segmentDuration,frameDuration,hopDuration,numBands);
XTest = log10(XTest + epsil);

YTrain = adsTrain.Labels;
YValidation = adsValidation.Labels;
YTest = adsTest.Labels;
```

Split Data into Training, Validation, and Test Sets

The data set folder contains text files, which list the audio files to be used as the validation and test sets. These predefined validation and test sets do not contain utterances of the same word by the same person, so it is better to use these predefined sets than to select a random subset of the whole data set. Use the supporting function `splitData` to split the datastore into training, validation, and test sets based on the list of validation and test files located in the data set folder.

```
[adsTrain,adsValidation,adsTest] = splitData(ads,datafolder);
```

Compute Speech Spectrograms

To prepare the data for efficient training of a convolutional neural network, convert the speech waveforms to log-bark auditory spectrograms.

Define the parameters of the spectrogram calculation. `segmentDuration` is the duration of each speech clip (in seconds). `frameDuration` is the duration of each frame for spectrogram calculation. `hopDuration` is the time step between each column of the spectrogram. `numBands` is the number of log-bark filters and equals the height of each spectrogram.

```
segmentDuration = 1;
frameDuration = 0.025;
hopDuration = 0.010;
numBands = 40;
```

Compute the spectrograms for the training, validation, and test sets by using the supporting function `speechSpectrograms`. The `speechSpectrograms` function uses `auditorySpectrogram` for the spectrogram calculations. To obtain data with a smoother distribution, take the logarithm of the spectrograms using a small offset `epsil`.

```
epsil = 1e-6;

XTrain = speechSpectrograms(adsTrain,segmentDuration,frameDuration,hopDuration,numBands);
XTrain = log10(XTrain + epsil);

XValidation = speechSpectrograms(adsValidation,segmentDuration,frameDuration,hopDuration,numBands);
XValidation = log10(XValidation + epsil);

XTest = speechSpectrograms(adsTest,segmentDuration,frameDuration,hopDuration,numBands);
XTest = log10(XTest + epsil);

YTrain = adsTrain.Labels;
YValidation = adsValidation.Labels;
YTest = adsTest.Labels;
```

MATLAB R2018b

HOME | PLOTS | APPS | SHORTCUTS | EDITOR | PUBLISH | VIEW

Cleanup

Search Documentation

Gabriele

Editor - C:\Docs\Material\Projects\2017-11 - CNN demo\R2018b Shipping\speechSpectrograms.m

speechSpectrograms.m

```matlab
function X = speechSpectrograms(ads,segmentDuration,frameDuration,hopDuration,numBands)

disp("Computing speech spectrograms...");

numHops = ceil((segmentDuration - frameDuration)/hopDuration);
numFiles = length(ads.Files);
X = zeros([numBands,numHops,1,numFiles],'single');

for i = 1:numFiles

    [x,info] = read(ads);

    fs = info.SampleRate;
    frameLength = round(frameDuration*fs);
    hopLength = round(hopDuration*fs);

    spec = auditorySpectrogram(x,fs, ...
        'WindowLength',frameLength, ...
        'OverlapLength',frameLength - hopLength, ...
        'NumBands',numBands, ...
        'Range',[50,7000], ...
        'WindowType','Hann', ...
        'WarpType','Bark', ...
        'SumExponent',2);

    % If the spectrogram is less wide than numHops, then put spectrogram in
    % the middle of X.
    w = size(spec,2);
    left = floor((numHops-w)/2)+1;
    ind = left:left+w-1;
    X(:,ind,1,i) = spec;

    if mod(i,1000) == 0
```

2 usages of "x" found

speechSpectrograms | Ln 18 | Col 6

Split Data into Training, Validation, and Test Sets

The data set folder contains text files, which list the audio files to be used as the validation and test sets. These predefined validation and test sets do not contain utterances of the same word by the same person, so it is better to use these predefined sets than to select a random subset of the whole data set. Use the supporting function `splitData` to split the datastore into training, validation, and test sets based on the list of validation and test files located in the data set folder.

```
[adsTrain,adsValidation,adsTest] = splitData(ads,datafolder);
```

Compute Speech Spectrograms

To prepare the data for efficient training of a convolutional neural network, convert the speech waveforms to log-bark auditory spectrograms.

Define the parameters of the spectrogram calculation. `segmentDuration` is the duration of each speech clip (in seconds). `frameDuration` is the duration of each frame for spectrogram calculation. `hopDuration` is the time step between each column of the spectrogram. `numBands` is the number of log-bark filters and equals the height of each spectrogram.

```
segmentDuration = 1;
frameDuration = 0.025;
hopDuration = 0.010;
numBands = 40;
```

Compute the spectrograms for the training, validation, and test sets by using the supporting function `speechSpectrograms`. The `speechSpectrograms` function uses `auditorySpectrogram` for the spectrogram calculations. To obtain data with a smoother distribution, take the logarithm of the spectrograms using a small offset `epsil`.

```
epsil = 1e-6;

XTrain = speechSpectrograms(adsTrain,segmentDuration,frameDuration,hopDuration,numBands);
XTrain = log10(XTrain + epsil);

XValidation = speechSpectrograms(adsValidation,segmentDuration,frameDuration,hopDuration,numBands);
XValidation = log10(XValidation + epsil);

XTest = speechSpectrograms(adsTest,segmentDuration,frameDuration,hopDuration,numBands);
XTest = log10(XTest + epsil);

YTrain = adsTrain.Labels;
YValidation = adsValidation.Labels;
YTest = adsTest.Labels;
```

Visualize Data

Plot the waveforms and spectrograms of a few training examples. Play the corresponding audio clips.

```
specMin = min(XTrain(:));
specMax = max(XTrain(:));
idx = randperm(size(XTrain,4),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+2 specMax])
    shading flat

    sound(x,fs)
    pause(2)
end
```

Training neural networks is easiest when the inputs to the network have a reasonably smooth distribution and are normalized. To check that the data distribution is smooth, plot a histogram of the pixel values of the training data.

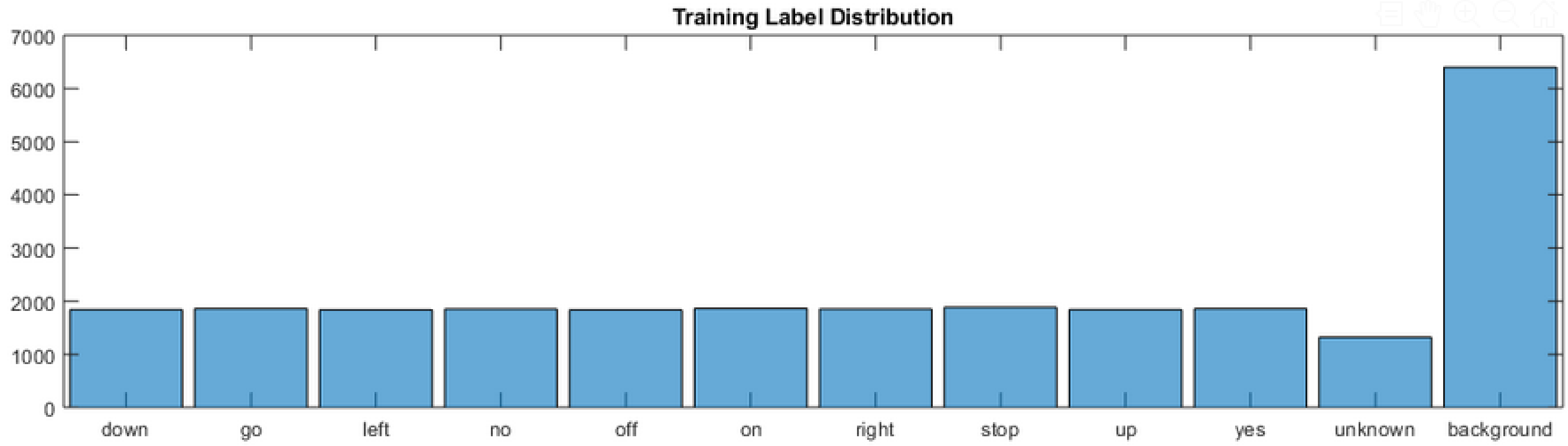
```
figure
histogram(XTrain,'EdgeColor','none','Normalization','pdf')
axis tight
ax = gca;
ax.YScale = 'log';
xlabel("Input Pixel Value")
ylabel("Probability Density")
```

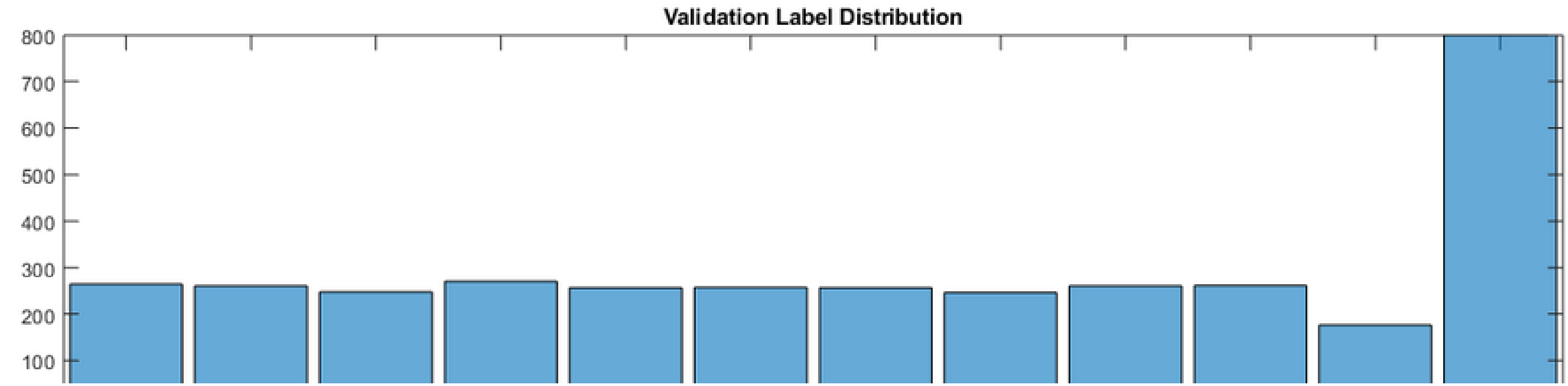
Add Background Noise Data

The network must be able not only to recognize different spoken words but also to detect if the input contains silence or background noise.

Plot the distribution of the different class labels in the training and validation sets. The test set has a very similar distribution to the validation set.

```
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")
```





INTERSPEECH 2015

Convolutional Neural Networks for Small-footprint Keyword Spotting

Tara N. Sainath, Carolina Parada

Google,
{tsainat

Abstract

We explore using Convolutional Neural Networks (CN
a small-footprint keyword spotting (KWS) task. CNN
tractive for KWS since they have been shown to out
DNNs with far fewer parameters. We consider two
applications in our work, one where we limit the nu
multiplications of the KWS system, and another where
the number of parameters. We present new CNN archi
to address the constraints of each applications. We find
CNN architectures offer between a 27-44% relative i
ment in false reject rate compared to a DNN, while fit
the constraints of each application.

1. Introduction

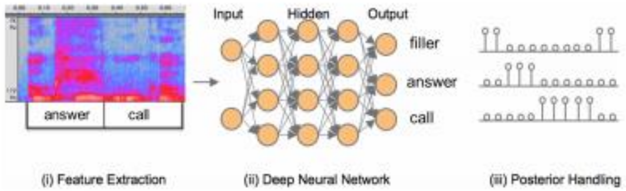


Figure 1: Framework of Deep KWS system, components from left to right: (i) Feature Extraction (ii) Deep Neural Network (iii) Posterior Handling

3. CNN Architectures

In this section, we describe CNN architectures as an alternative to the DNN described in Section 2. The feature extraction and posterior handling stages remain the same as Section 2.

3.1. CNN Description

A typical CNN architecture is shown in Figure 2. First, we are given an input signal $\mathbf{V} \in \Re^{t \times f}$, where t and f are the input feature dimension in time and frequency respectively. A weight

The second convolutional filter has a filte
quency, and no max-pooling is performed

For example, in our task if we want
of parameters below 250K, a typical arc
tecture is shown in Table 1. We will refe
as cnn-trad-fpool3 in this paper. T
convolutional, one linear low-rank and on
tion 5, we will show the benefit of this a
particularly the pooling in frequency, com

However, a main issue with this arc
number of multiplies in the convolutional
acerbated in the second layer because of
put, spanning across time, frequency and
type of architecture is infeasible for pow
footprint KWS tasks where multiplies are
even if our application is limited by para
tiplies, other architectures which pool in
suited for KWS. Below we present alter
tures to address the tasks of limiting para

type	m	r	n	p	q
conv	20	8	64	1	3

model	layer	m	r	n	s	q	Params
cnn-tstride2	conv	16	8	78	**2**	3	10.0K
	conv	9	4	78	1	1	219.0K
	lin	-	-	32	-	-	20.0K
cnn-tstride4	conv	16	8	100	**4**	3	12.8K
	conv	5	4	78	1	1	200.0K
	lin	-	-	32	-	-	25.6K
cnn-tstride8	conv	16	8	126	**8**	3	16.1K
	conv	5	4	78	1	1	190.5K
	lin	-	-	32	-	-	32.2K

Table 4: CNNs for Striding in Time

3.4.2. *Pooling in Time*

An alternative to striding the filter in time is to pool in time, by a non-overlapping amount. Table 5 shows configurations as we vary the pooling in time p. We will refer to these architectures as cnn-tpool2 and cnn-tpool4. For simplicity, we have omitted certain variables held constant for all experiments, namely time and frequency stride $s = 1$ and $v = 1$. Notice that by pooling in time, we can increase the number of feature maps n to keep the total number of parameters constant.

model	layer	m	r	n	p	q	Params
cnn-tpool2	conv	21	8	94	**2**	3	5.6M
	conv	6	4	94	1	1	1.8M
	lin	-	-	32	-	-	65.5K
cnn-tpool3	conv	15	8	94	**3**	3	7.1M
	conv	6	4	94	1	1	1.6M
	lin	-	-	32	-	-	65.5K

Table 5: CNNs for Pooling in Time

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 enforces (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.

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));

dropoutProb = 0.2;
numF = 12;
layers = [
    imageInputLayer(imageSize)

    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')
```

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 enforces (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.

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));

dropoutProb = 0.2;
numF = 12;
layers = [
    imageInputLayer(imageSize)

    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')
```

```
dropoutProb = 0.2;
numF = 12;
layers = [
    imageInputLayer(imageSize)

    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([1 13])

    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. If you do not have a GPU, then training the network can take time.

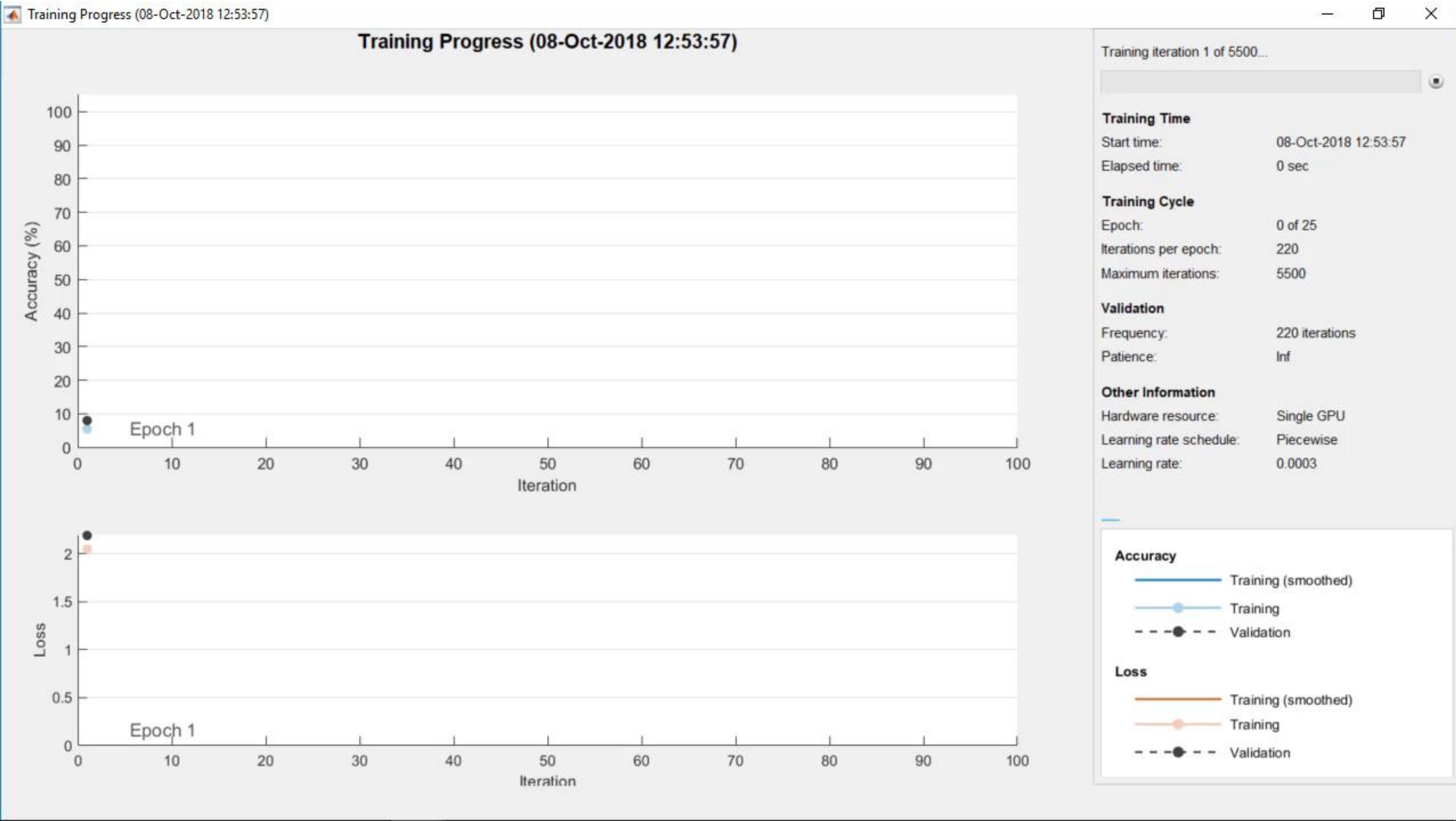
```
trainedNet = trainNetwork(augimdsTrain,layers,options);
```

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. This limitation particularly applies to the `unknown` category, which contains utterances of only a small number of words.

```
YValPred = classify(trainedNet,XValidation);
[redacted]
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 class by using column and row summaries. Sort the classes of the confusion matrix. The largest confusion is between unknown words and commands, *up* and *off*, *down* and *no*, and *go* and *no*.


Training Progress (08-Oct-2018 12:53:57)
Training Progress (08-Oct-2018 12:53:57)
Accuracy (%)
Epoch 1
Iteration
Loss
Epoch 1
Iteration
Training iteration 1 of 5500...
Training Time
Start time: 08-Oct-2018 12:53:57
Elapsed time: 0 sec
Training Cycle
Epoch: 0 of 25
Iterations per epoch: 220
Maximum iterations: 5500
Validation
Frequency: 220 iterations
Patience: Inf
Other Information
Hardware resource: Single GPU
Learning rate schedule: Piecewise
Learning rate: 0.0003
Accuracy
Training (smoothed)
Training
Validation
Loss
Training (smoothed)
Training
Validation

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. This limitation particularly applies to the `unknown` category, which contains utterances of only a small number of words.

```
YValPred = classify(trainedNet,XValidation);
validationError = mean(YValPred ~= YValidation);
YTrainPred = classify(trainedNet,XTrain);
trainError = mean(YTrainPred ~= YTrain);
disp("Training error: " + trainError*100 + "%")
```

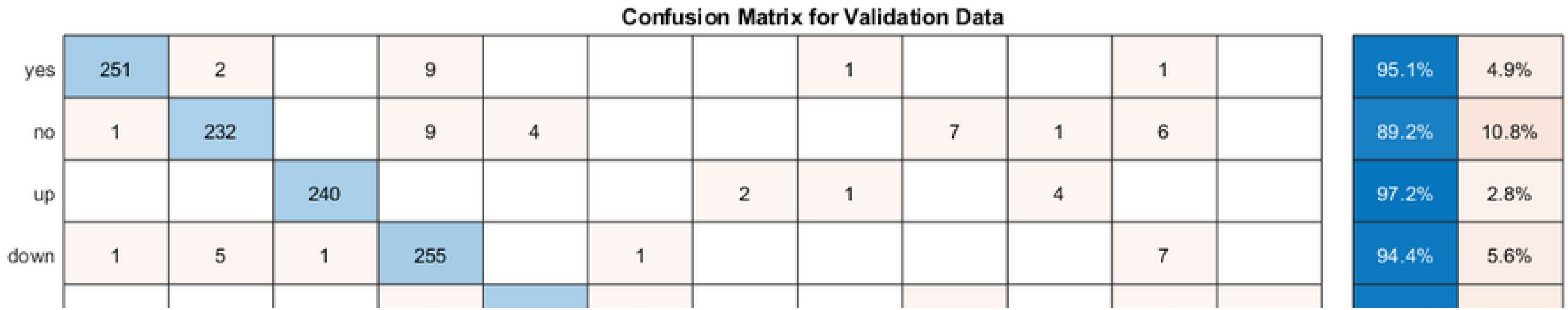
```
Training error: 1.8736%
```

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

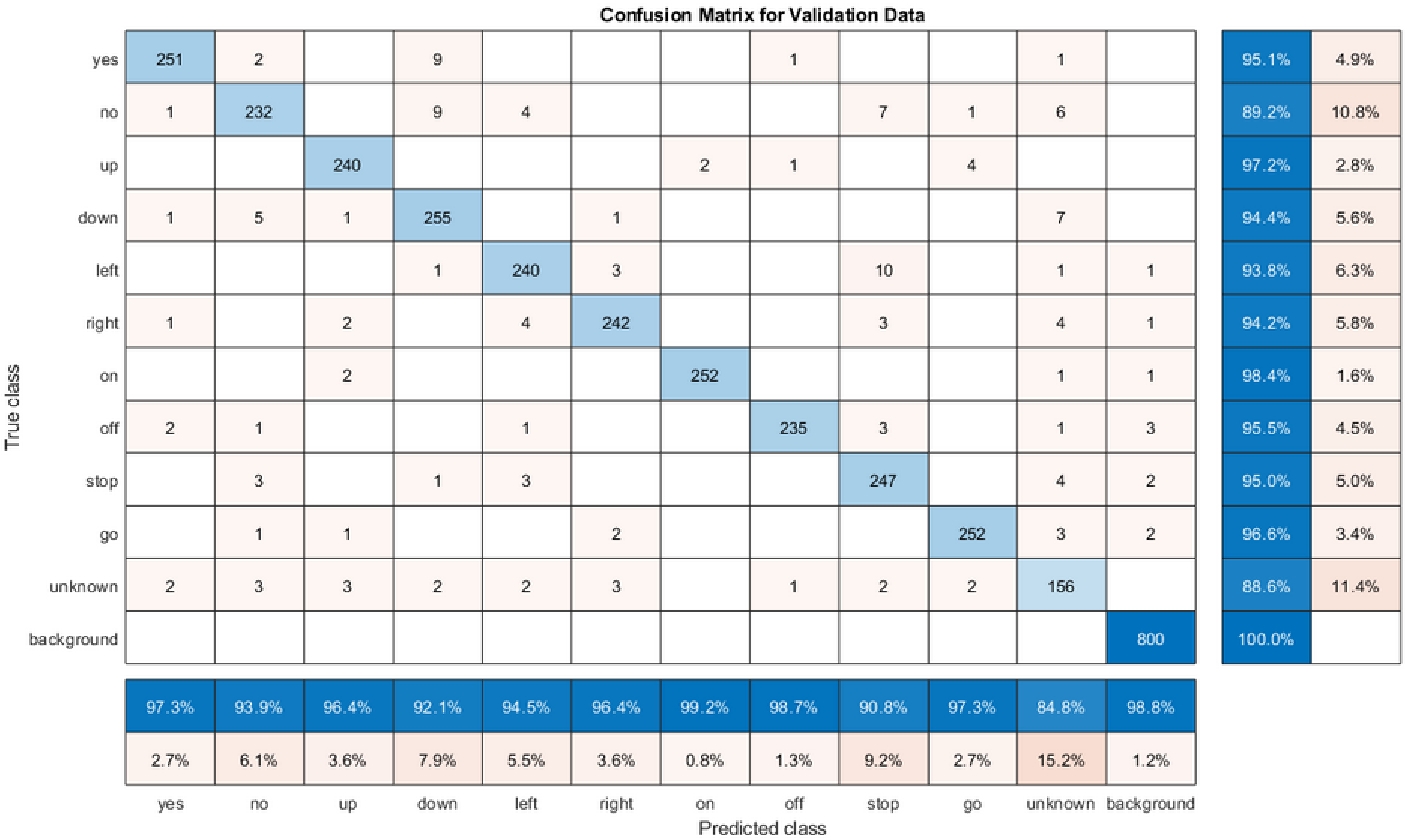
```
Validation error: 4.2499%
```

Plot the confusion matrix. Display the precision and recall for each class by using column and row summaries. Sort the classes of the confusion matrix. The largest confusion is between unknown words and commands, *up* and *off*, *down* and *no*, and *go* and *no*.

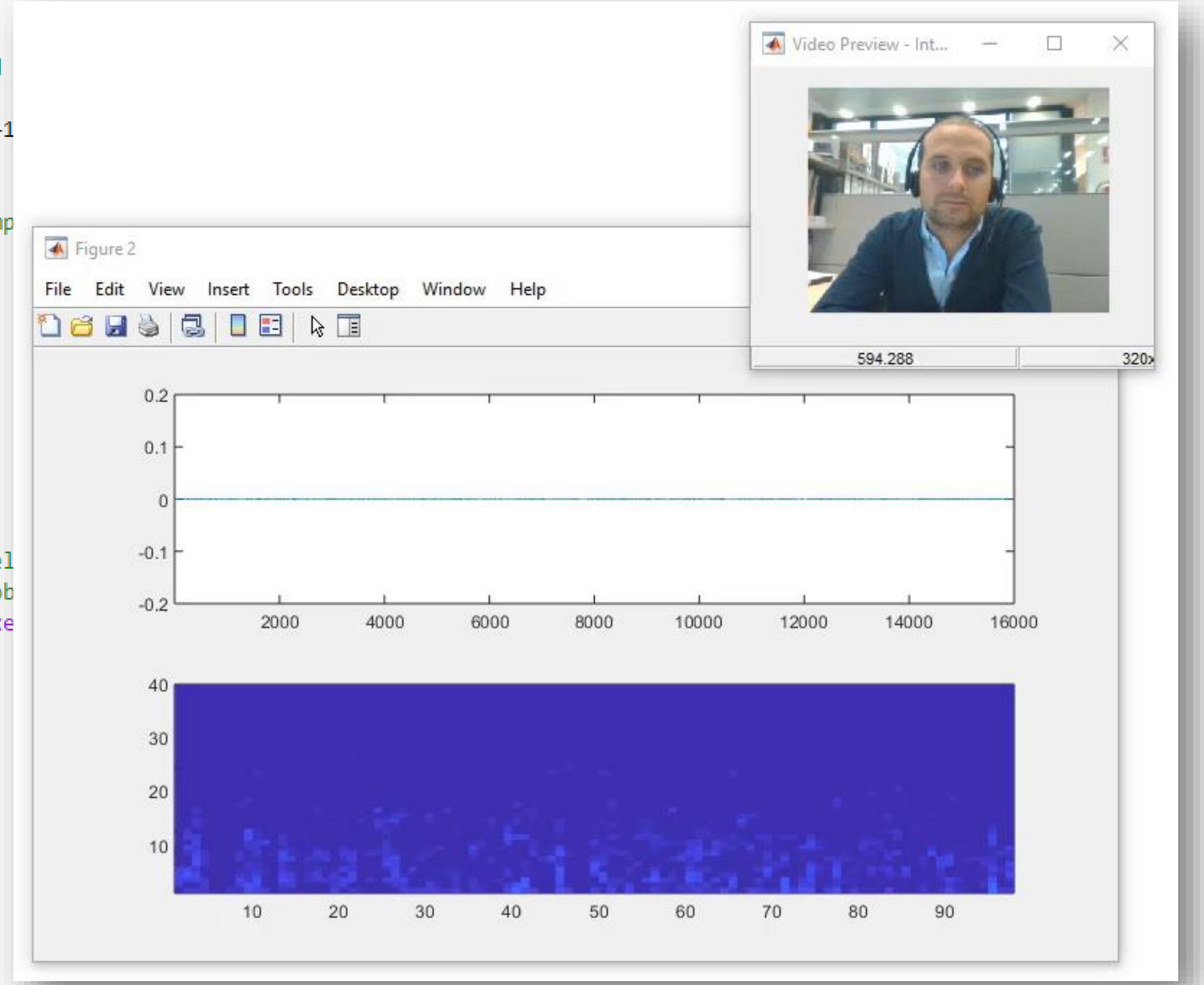
```
figure('Units','normalized','Position',[0.2 0.2 0.5 0.5]);
cm = confusionchart(YValidation,YValPred);
cm.Title = 'Confusion Matrix for Validation Data';
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';
sortClasses(cm, [commands,"unknown","background"])
```



```matlab
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"])
```



```
while ishandle(h)

    % Extract audio samples from the audio device and
    x = audioIn();
    waveBuffer(1:end-numel(x)) = waveBuffer(numel(x)+1
    waveBuffer(end-numel(x)+1:end) = x;

    % Compute the spectrogram of the latest audio samp
    spec = auditorySpectrogram(waveBuffer,fs, ...
        'WindowLength',frameLength, ...
        'OverlapLength',frameLength-hopLength, ...
        'NumBands',numBands, ...
        'Range',[50,7000], ...
        'WindowType','Hann', ...
        'WarpType','Bark', ...
        'SumExponent',2);
    spec = log10(spec + epsil);

    % Classify the current spectrogram, save the label
    % and save the predicted probabilities to the prob
    [YPredicted,probs] = classify(trainedNet,spec,'Exe
    YBuffer(1:end-1)= YBuffer(2:end);
    YBuffer(end) = YPredicted;
    probBuffer(:,1:end-1) = probBuffer(:,2:end);
    probBuffer(:,end) = probs';

    % Plot the current waveform and spectrogram.
    subplot(2,1,1);
    plot(waveBuffer)
    axis tight
    ylim([-0.2,0.2])

    subplot(2,1,2)
    pcolor(spec)
    caxis([specMin+2 specMax])
    shading flat

    % Now do the actual command detection by performing a very simple
    % thresholding operation. Declare a detection and display it in the
```



MATLAB Based Algorithm Wins the 2017 PhysioNet/CinC Challenge to Automatically Detect Atrial Fibrillation

Challenge

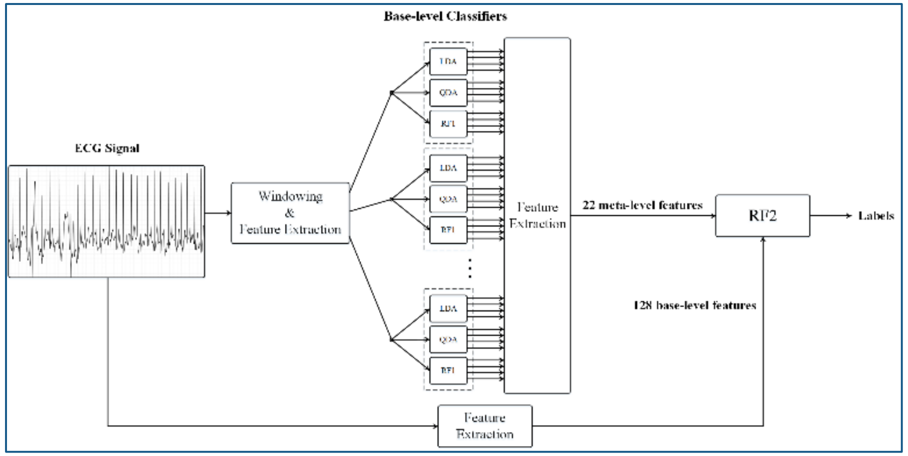
Design an algorithm that uses machine learning to detect atrial fibrillation and other abnormal heart rhythms in noisy, single-lead ECG recordings

Solution

Use MATLAB to analyze ECG data, extract features using signal processing and wavelet techniques, and evaluate different machine learning algorithms to train and implement a best-in-class classifier to detect AF

Results

- First place in PhysioNet/CinC Challenge achieved
- ECG data visualized in multiple domains
- Feature extraction accelerated with parallel processing



Block diagram for Black Swan's atrial fibrillation detection algorithm.

"I don't think MATLAB has any strong competitors for signal processing and wavelet analysis. When you add in its statistics and machine learning capabilities, it's easy to see why nonprogrammers enjoy using MATLAB, particularly for projects that require combining all these methods."

- Ali Bahrami Rad, Aalto University

[Link to user story](#)

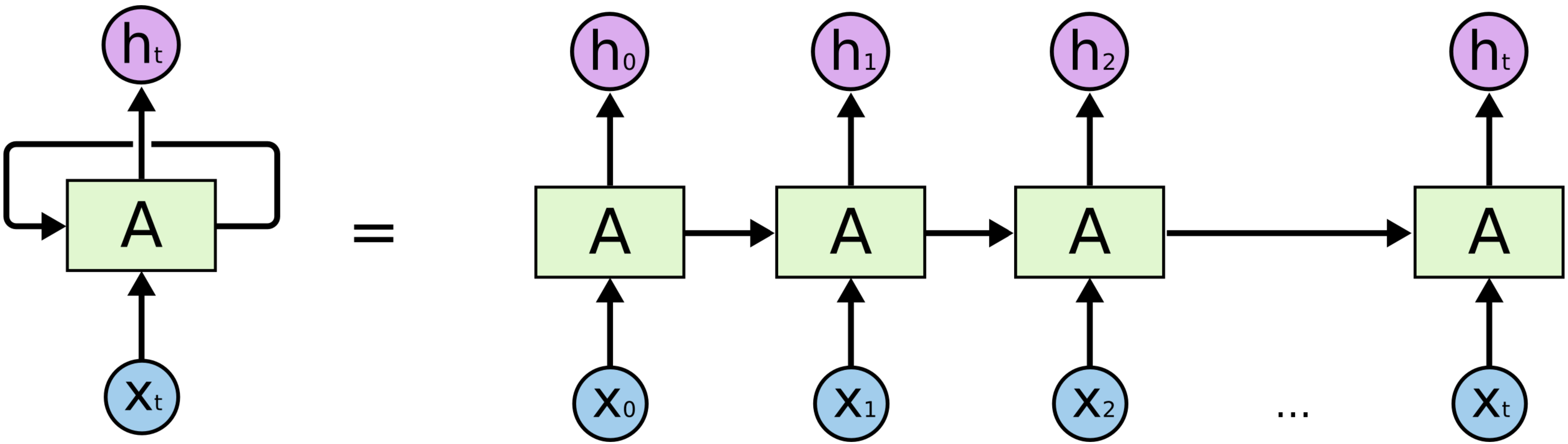
Agenda

Why deep learning?
Deep learning with signal data
(Demo) Speech Command Recognition
(Demo) LSTM Networks
Enabling Features in MATLAB
Deploying deep learning

I was born in France. I speak __________ ?

Recurrent Neural Networks

- Take previous data into account when making new predictions
- Signals, text, time series



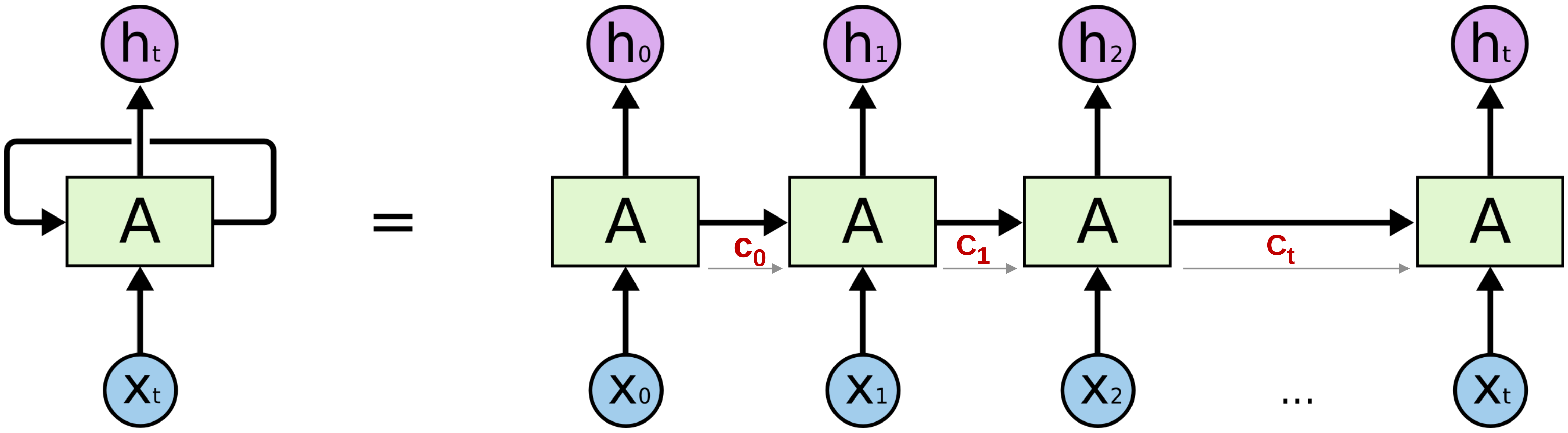
I was born in France…

[2000 words]

… I speak ____________ ?

Long Short Term Memory Networks

- RNN that carries a memory cell throughout the process
- Sequence Problems



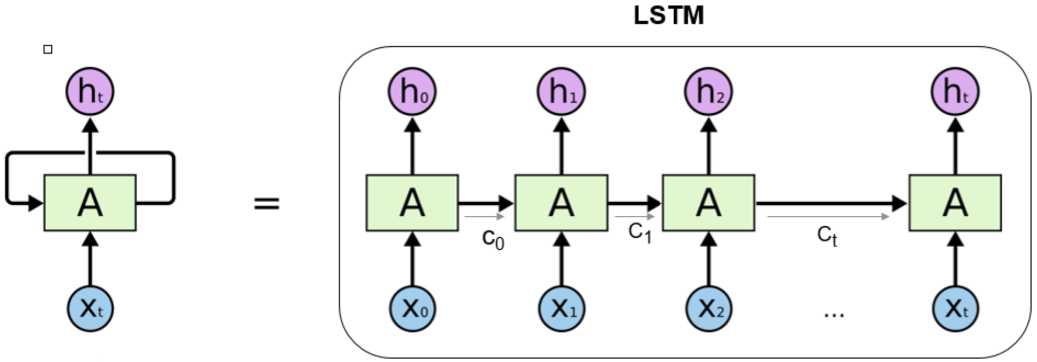
LSTM

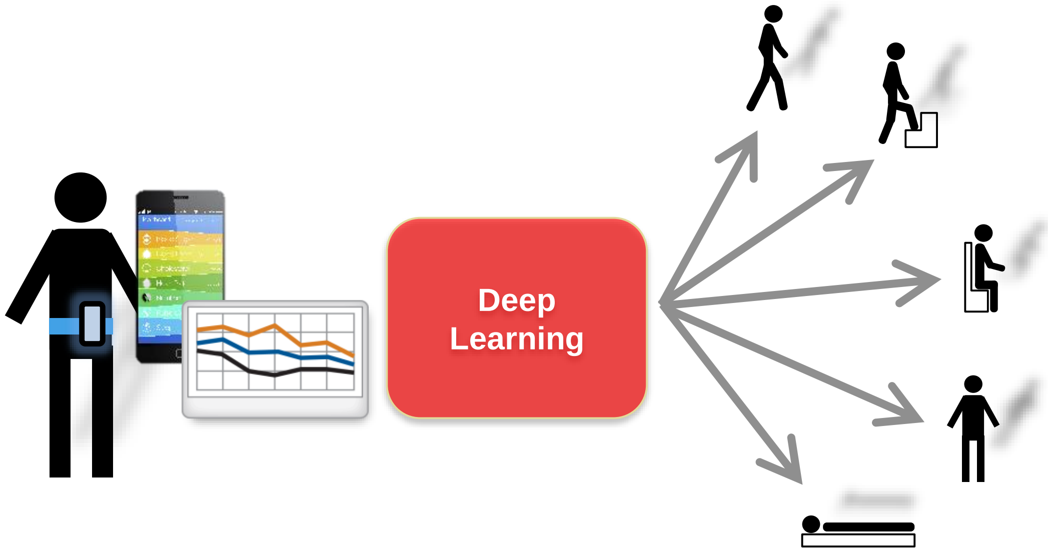
Demo

Time Series Classification (Human Activity Recognition)

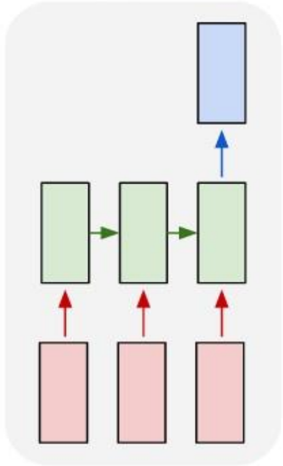
Long short-term memory networks

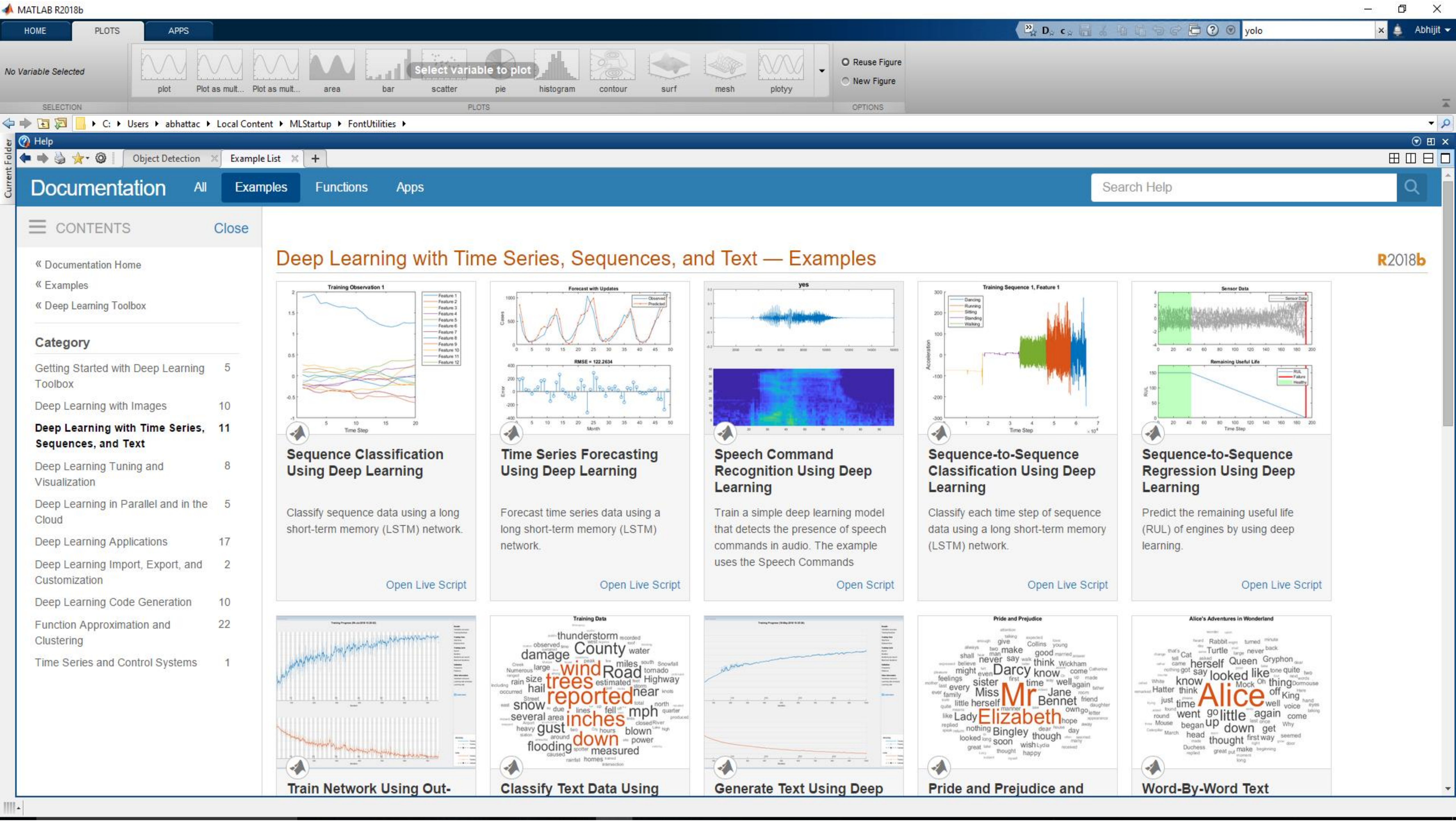
- Dataset is accelerometer and gyroscope signals captured with a smartphone
- Data is a collection of time series with 9 channels





sequence to one




MATLAB R2018b
HOME
PLOTS
APPS
No Variable Selected
Select variable to plot
plot
Plot as mult...
Plot as mult...
area
bar
scatter
pie
histogram
contour
surf
mesh
plotyy
Reuse Figure
New Figure
SELECTION
PLOTS
OPTIONS
C: ▸ Users ▸ abhattac ▸ Local Content ▸ MLStartup ▸ FontUtilities ▸
Help
Object Detection
Example List
Documentation
All
Examples
Functions
Apps
Search Help
CONTENTS
Close
« Documentation Home
« Examples
« Deep Learning Toolbox
Category
Getting Started with Deep Learning Toolbox 5
Deep Learning with Images 10
Deep Learning with Time Series, Sequences, and Text 11
Deep Learning Tuning and Visualization 8
Deep Learning in Parallel and in the Cloud 5
Deep Learning Applications 17
Deep Learning Import, Export, and Customization 2
Deep Learning Code Generation 10
Function Approximation and Clustering 22
Time Series and Control Systems 1
Deep Learning with Time Series, Sequences, and Text — Examples
R2018b
Sequence Classification Using Deep Learning
Classify sequence data using a long short-term memory (LSTM) network.
Open Live Script
Time Series Forecasting Using Deep Learning
Forecast time series data using a long short-term memory (LSTM) network.
Open Live Script
Speech Command Recognition Using Deep Learning
Train a simple deep learning model that detects the presence of speech commands in audio. The example uses the Speech Commands
Open Script
Sequence-to-Sequence Classification Using Deep Learning
Classify each time step of sequence data using a long short-term memory (LSTM) network.
Open Live Script
Sequence-to-Sequence Regression Using Deep Learning
Predict the remaining useful life (RUL) of engines by using deep learning.
Open Live Script
Train Network Using Out-
Classify Text Data Using
Generate Text Using Deep
Pride and Prejudice and
Word-By-Word Text

Agenda

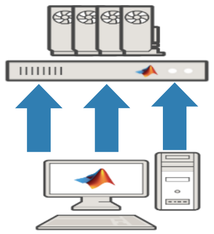
Why deep learning?
Deep learning with signal data
(Demo) Speech Command Recognition
(Demo) LSTM Networks
Enabling Features in MATLAB
Deploying deep learning

Deep Learning on CPU, GPU, Multi-GPU and Clusters

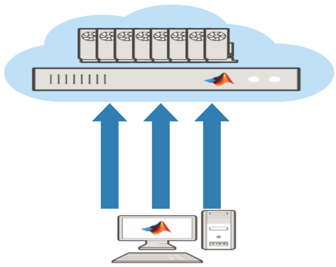
Single
CPU

Single CPU
Single GPU

Single CPU, Multiple GPUs



On-prem server with
GPUs



Cloud GPUs
(AWS)

How to target?

```
opts = trainingOptions('sgdm', ...
    'MaxEpochs', 100, ...
    'MiniBatchSize', 250, ...
    'InitialLearnRate', 0.00005, ...
'ExecutionEnvironment', 'auto' );
```

```
opts = trainingOptions('sgdm', ...
    'MaxEpochs', 100, ...
    'MiniBatchSize', 250, ...
    'InitialLearnRate', 0.00005, ...
'ExecutionEnvironment', 'multi-gpu' );
```

```
opts = trainingOptions('sgdm', ...
    'MaxEpochs', 100, ...
    'MiniBatchSize', 250, ...
    'InitialLearnRate', 0.00005, ...
'ExecutionEnvironment', 'parallel' );
```

Audio Datastore

- Programmatic interface to large collections of audio files
- (Optional) auto-generation of labels from folder names
- Label-based indexing and partitioning
- Random file sampling
- Automatic sequential file reading
- Parallel access for pre-processing, feature extraction, or data augmentation

Folder
- Dataset
  - _background_noise_
  - bed
  - bird
  - cat
  - dog
  - down
  - eight
  - five
  - four
  - go
  - happy
  - house
  - left
  - marvin
  - nine
  - no
  - off
  - on
  - one
  - right
  - seven
  - sheila
  - six
  - stop
  - three
  - tree
  - two
  - up
  - wow
  - yes
  - zero

```
ads = audioDatastore('.\Dataset', ...
    'IncludeSubfolders',true, ...
    'FileExtensions','.wav', ...
    'LabelSource','foldernames')
```

```
ads =

  Datastore with properties:

                 Files: {
                        ' ...\Temp\speech_commands_v0.01\_backg
                        ' ...\Local\Temp\speech_commands_v0.01\
                        ' ...\Local\Temp\speech_commands_v0.01\
                         ... and 64724 more
                        }
                Labels: [_background_noise_; _background_noise_
            ReadMethod: 'File'
        OutputDataType: 'double'
```

```
ads = getSubsetDatastore(ads,isCommand|isUnknown);
countEachLabel(ads)

ans =
11×2 table
Label      Count

    _______    _____
    down       2359
    go         2372
    left       2353
    no         2375
    off        2357
    on         2367
    right      2367
    stop       2380
    unknown    4143
    up         2375
    yes        2377
```

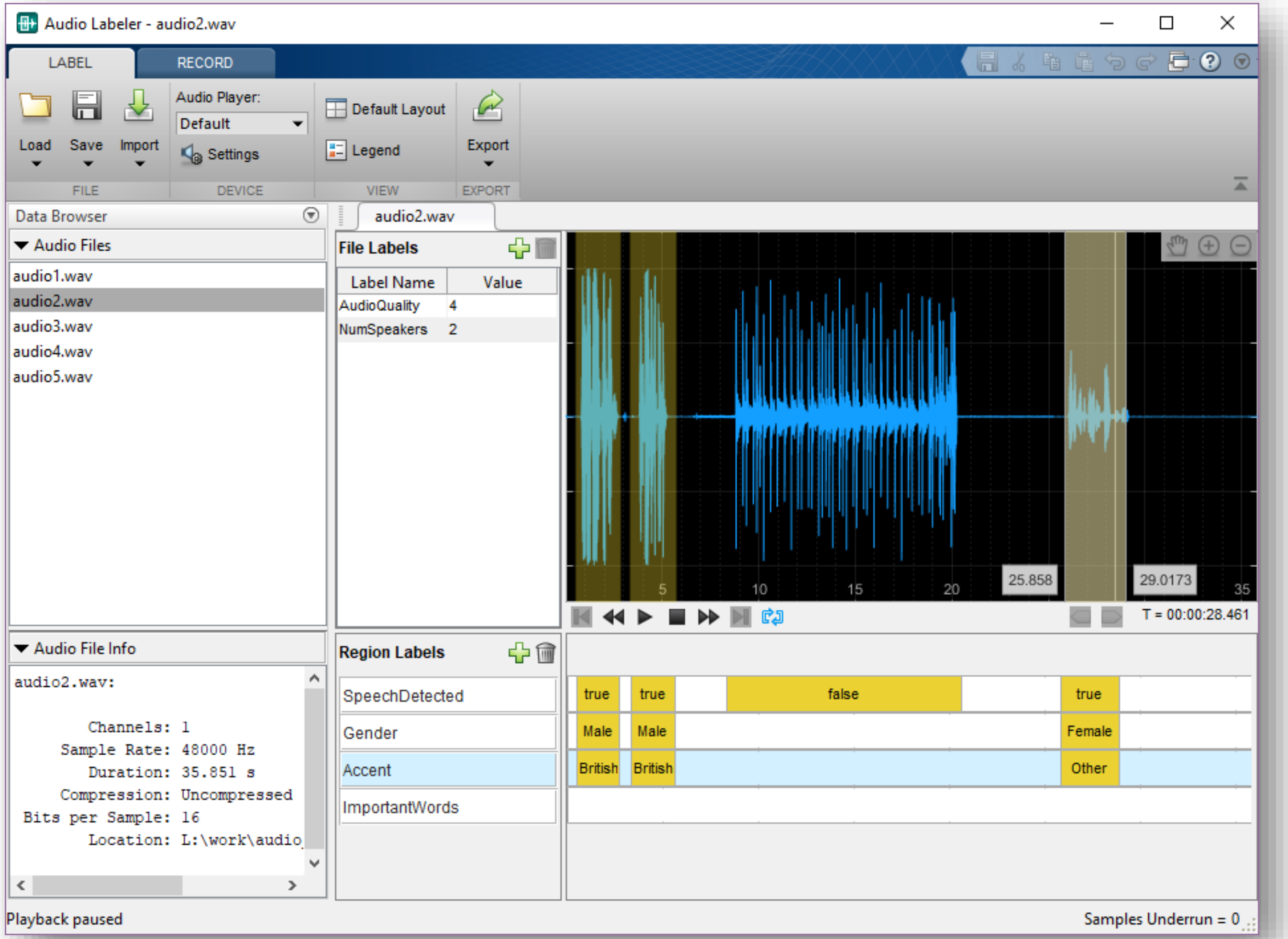
```
[adsTrain,adsValidation,adsTest] = splitData(ads,datafolder)
```

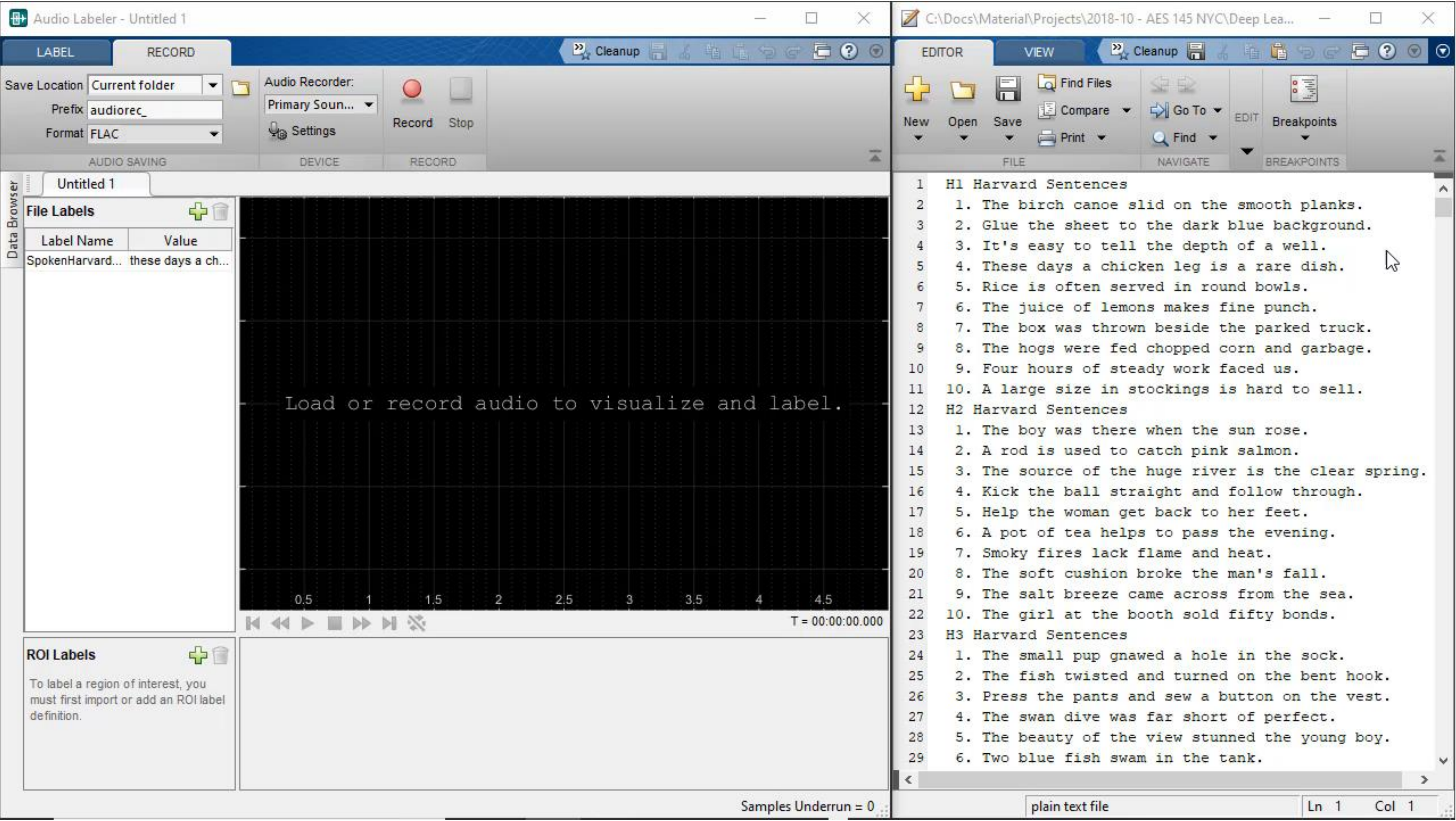
“I love to label and preprocess my data”

~ Said no engineer, ever.

Audio Labeler

- Work on collections of recordings or record new audio directly within the app
- Navigate dataset and playback interactively
- Define and apply labels to
  - Entire files
  - Regions within files
- Import and export audio folders, label definitions and datastores



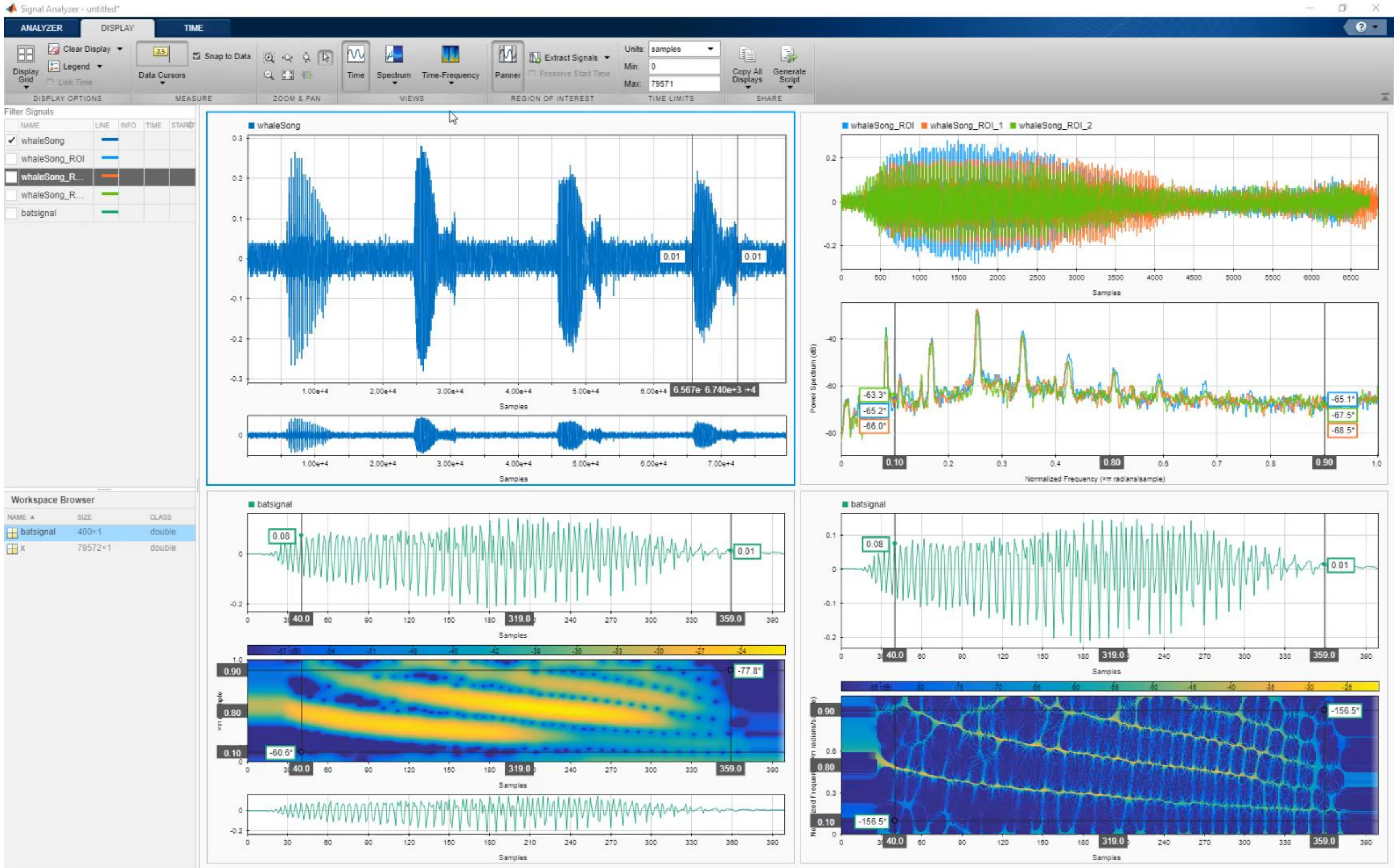

Audio Labeler - Untitled 1
LABEL
RECORD
Cleanup
Save Location
Current folder
Prefix
audiorec_
Format
FLAC
AUDIO SAVING
Audio Recorder:
Primary Soun...
Settings
DEVICE
Record
Stop
RECORD
Data Browser
Untitled 1
File Labels
Label Name
Value
SpokenHarvard...
these days a ch...
Load or record audio to visualize and label.
0.5 1 1.5 2 2.5 3 3.5 4 4.5
T = 00:00:00.000
ROI Labels
To label a region of interest, you must first import or add an ROI label definition.
Samples Underrun = 0
C:\Docs\Material\Projects\2018-10 - AES 145 NYC\Deep Lea...
EDITOR
VIEW
Cleanup
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
EDIT
Breakpoints
BREAKPOINTS
H1 Harvard Sentences
1. The birch canoe slid on the smooth planks.
2. Glue the sheet to the dark blue background.
3. It's easy to tell the depth of a well.
4. These days a chicken leg is a rare dish.
5. Rice is often served in round bowls.
6. The juice of lemons makes fine punch.
7. The box was thrown beside the parked truck.
8. The hogs were fed chopped corn and garbage.
9. Four hours of steady work faced us.
10. A large size in stockings is hard to sell.
H2 Harvard Sentences
1. The boy was there when the sun rose.
2. A rod is used to catch pink salmon.
3. The source of the huge river is the clear spring.
4. Kick the ball straight and follow through.
5. Help the woman get back to her feet.
6. A pot of tea helps to pass the evening.
7. Smoky fires lack flame and heat.
8. The soft cushion broke the man's fall.
9. The salt breeze came across from the sea.
10. The girl at the booth sold fifty bonds.
H3 Harvard Sentences
1. The small pup gnawed a hole in the sock.
2. The fish twisted and turned on the bent hook.
3. Press the pants and sew a button on the vest.
4. The swan dive was far short of perfect.
5. The beauty of the view stunned the young boy.
6. Two blue fish swam in the tank.
plain text file
Ln 1
Col 1

Signal Analyzer App R2018a

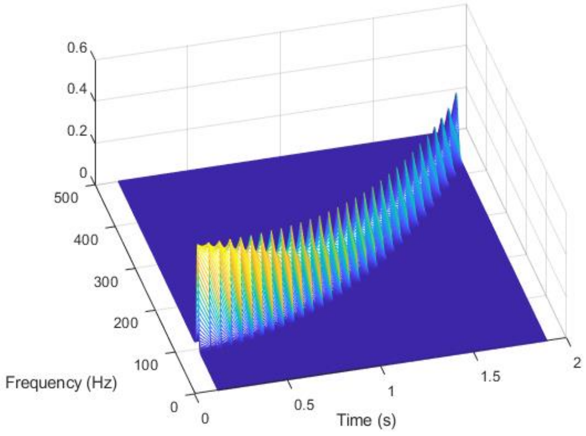
Signal Processing Toolbox

Analyze multiple signals in time, frequency and time-frequency domains

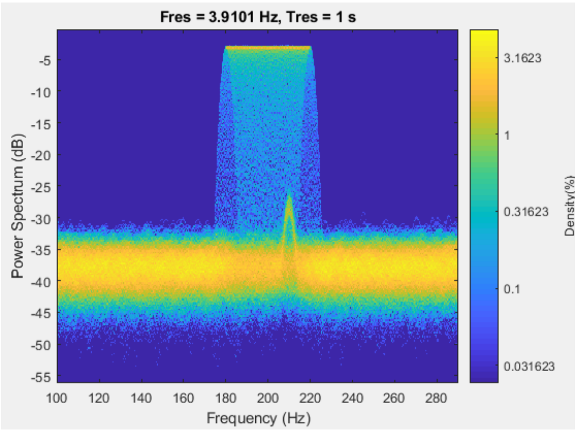
- **Navigate through signals**
- **Extract regions of interest**
- **Generate MATLAB scripts**



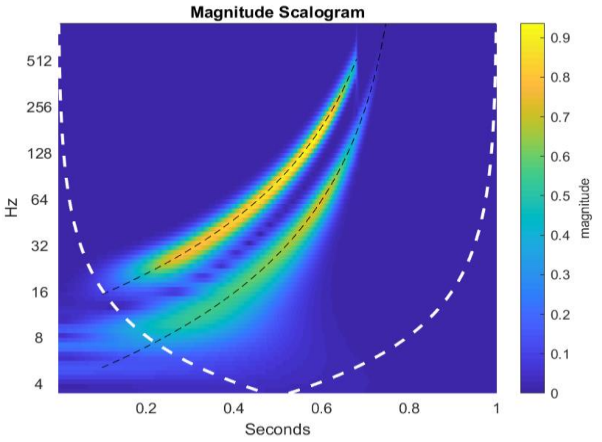
Feature Extraction: Time-Frequency Analysis



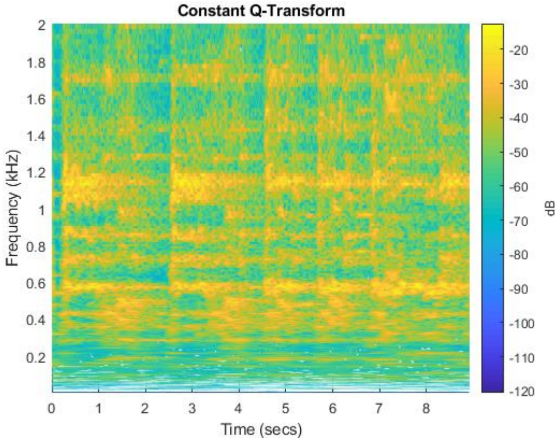
Reassigned spectrogram and synchrosqueezing



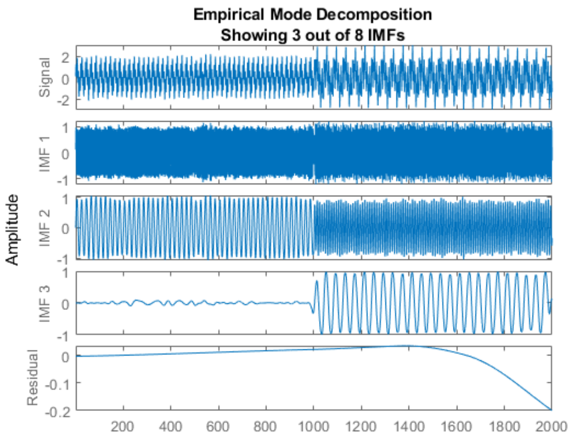
Persistent spectrum



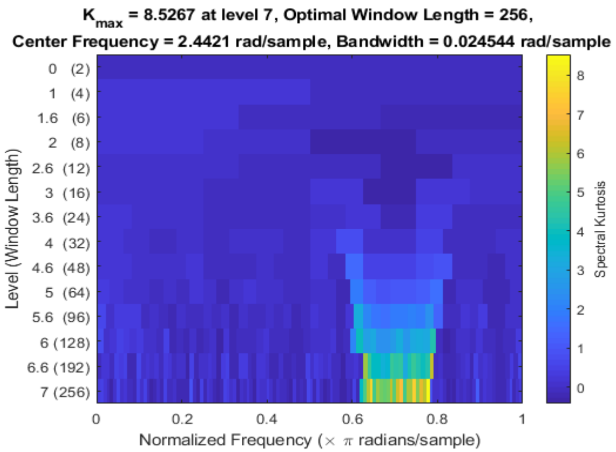
Wavelet scalogram



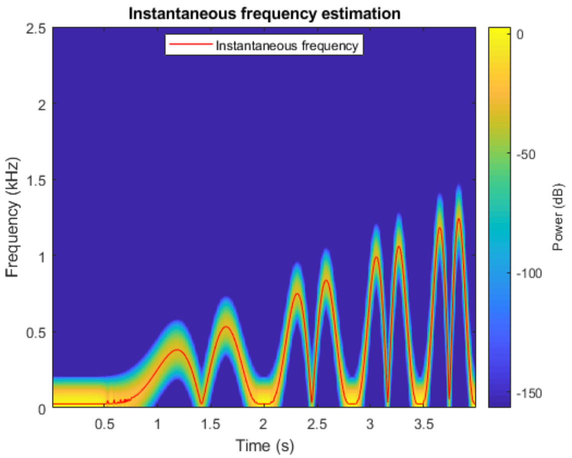
Constant Q transform



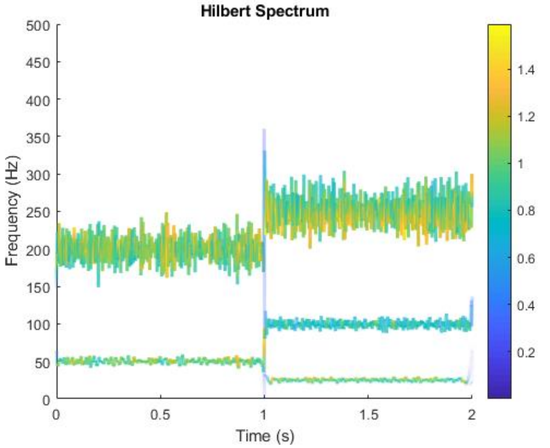
Empirical mode decomposition



Kurtogram



Instantaneous frequency
Instantaneous BW



Hilbert-Huang transform

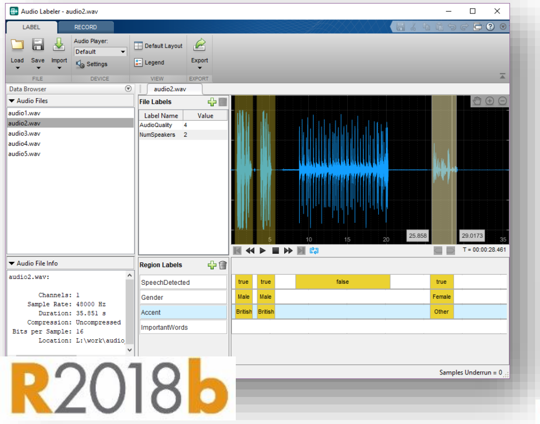
MathWorks®

Enabling Features Summary

(For Audio Applications) Audio System Toolbox
(For Signal Applications) Signal Processing Toolbox

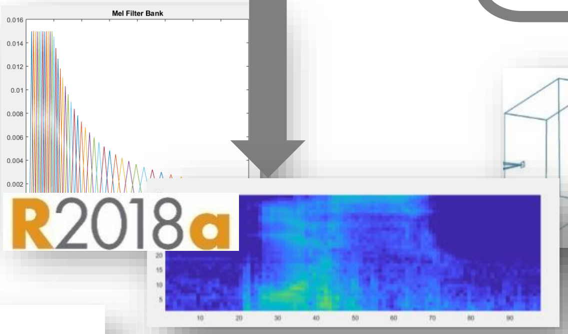
Neural Network TB
Stats & ML TB

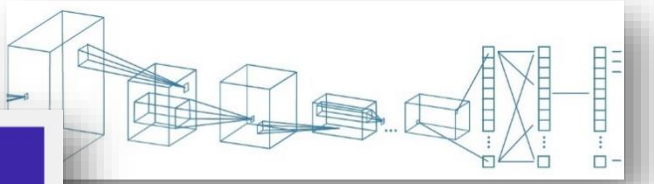
Record & store → Ground Truth Labeling → Read & Partition → Process & augment → Extract Features → Train → Predict → Prototype!




```
ads =

  Datastore with properties:

              Files: {
                     ' ...\Temp\speech_commands_v0.01\_background_noise_\doing_the_dishes.wav';
                     ' ...\Local\Temp\speech_commands_v0.01\_background_noise_\dude_miaowing.wav';
                     ' ...\Local\Temp\speech_commands_v0.01\_background_noise_\exercise_bike.wav'
                      ... and 64724 more
                     }
             Labels: [_background_noise_; _background_noise_; _background_noise_ ... and 64724 more
```

R2018b

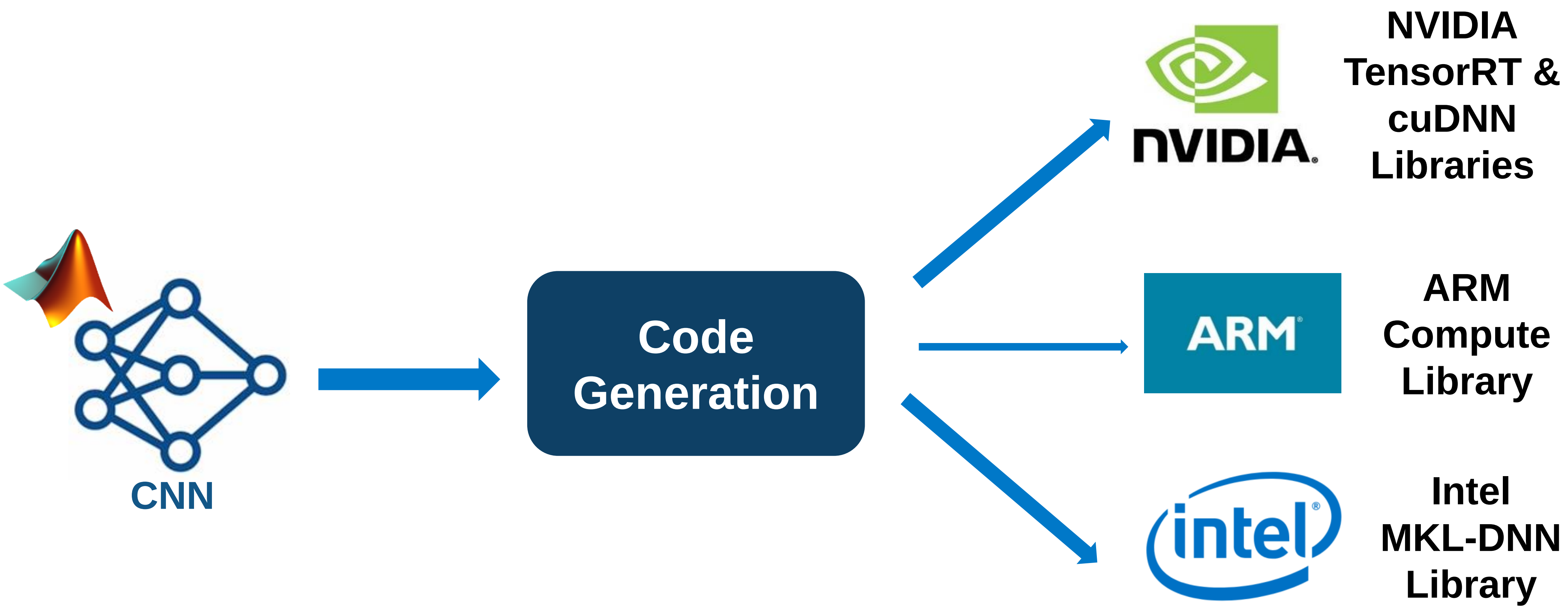




Agenda

Why deep learning?
Deep learning with signal data
(Demo) Speech Command Recognition
(Demo) LSTM Networks
Enabling Features in MATLAB
Deploying deep learning

Deploying Deep Learning Models for Inference



GPU Coder Fills a Gap in the Deep Learning Solution

Training

Inference

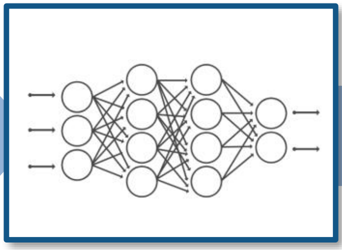
Access Data

Image Acq.

Preprocess

Image Processing

Computer Vision

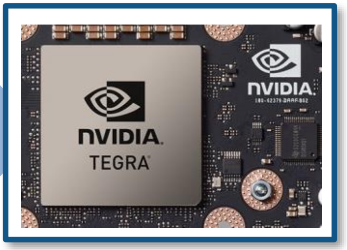


Select Network

Neural Network

Train

PCT

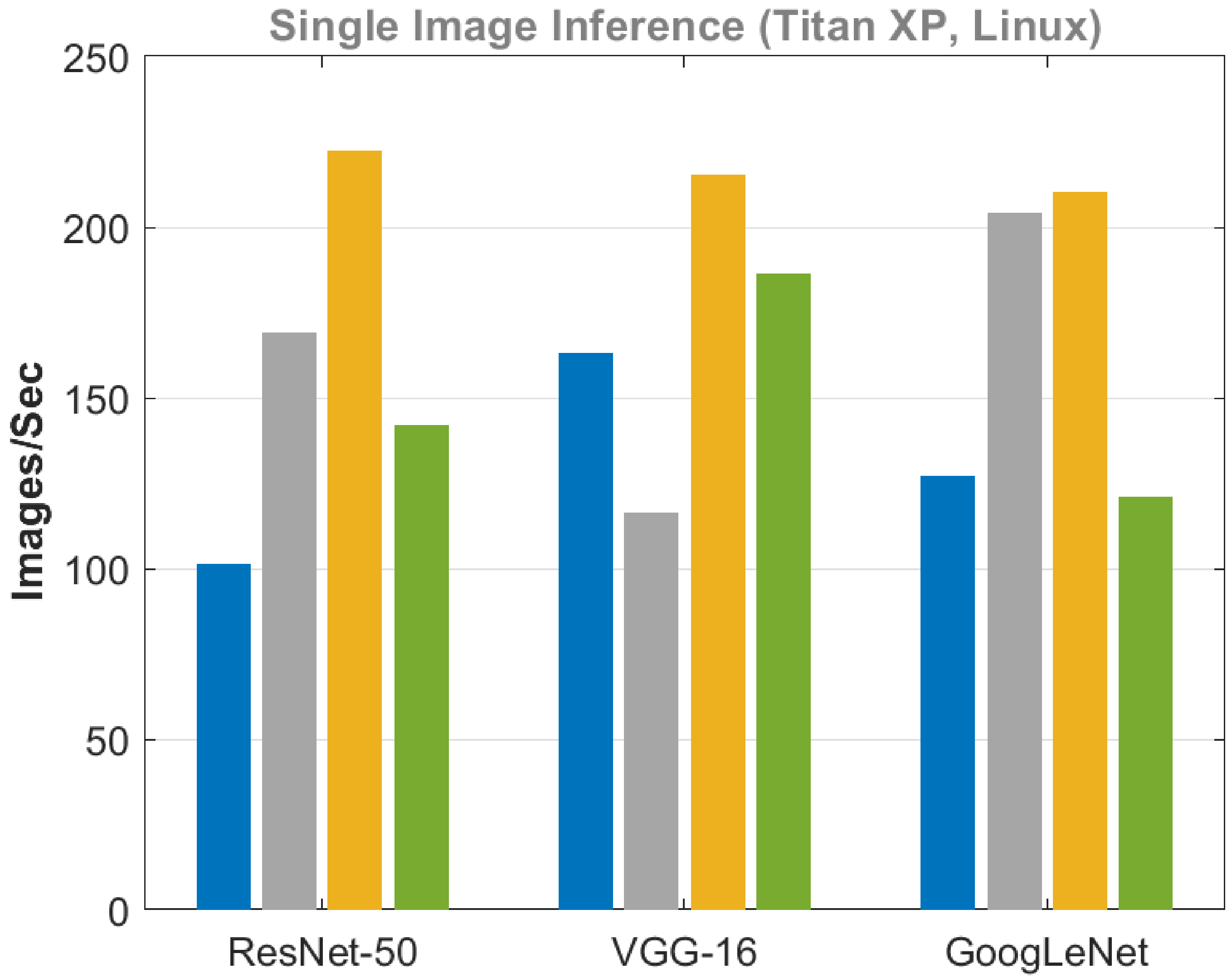


Deploy

GPU Coder

NVIDIA

With GPU Coder, MATLAB is fast



GPU Coder is faster than TensorFlow, MXNet and Pytorch

- TensorFlow
- MXNet
- GPU Coder
- PyTorch

Intel® Xeon® CPU 3.6 GHz - NVIDIA libraries: CUDA9 - cuDNN 7 - Frameworks: TensorFlow 1.8.0, MXNet 1.2.1, PyTorch 0.3.1

MathWorks can help you do Deep Learning

Free resources

- **Guided evaluations with a MathWorks deep learning engineer**
- Proof-of-concept projects
- **Deep learning hands-on workshop**
- Seminars and technical deep dives
- Deep learning onramp course

More options

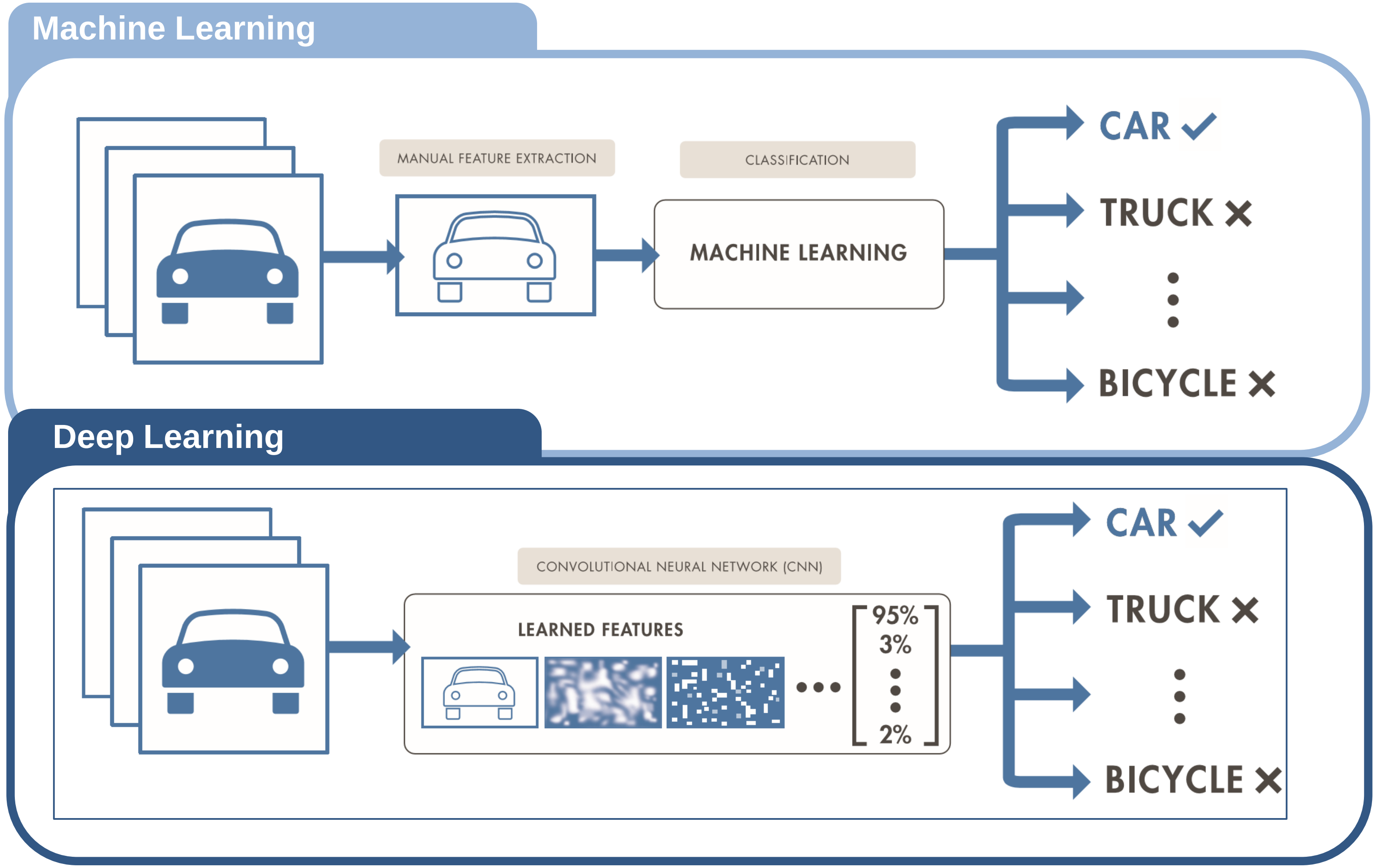
- Consulting services
- Training courses
- Technical support
- Advanced customer support
- Installation, enterprise, and cloud deployment
- MATLAB for Deep Learning

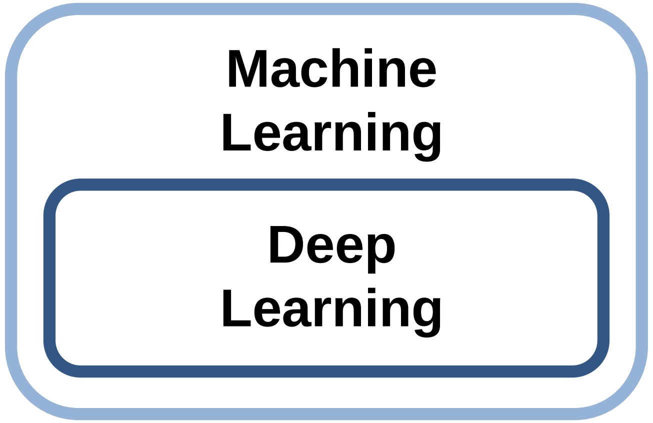
Thank you!

Extra Slides

Machine learning vs. deep learning

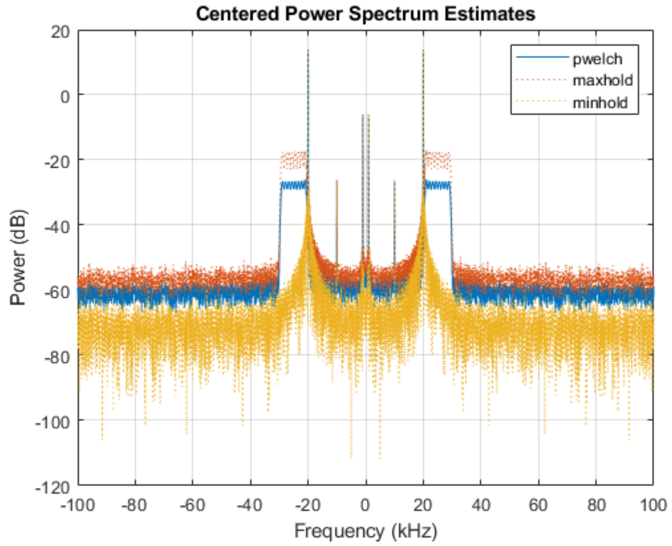
Deep learning performs **end-to-end learning** by learning **features, representations and tasks** directly from **images, text, sound, and signals**



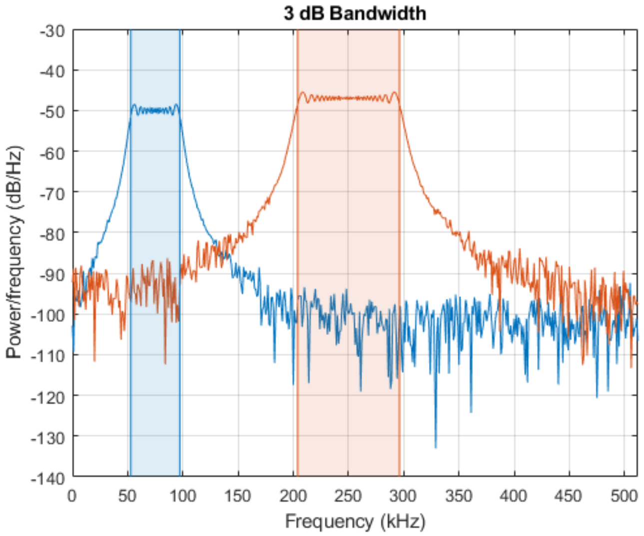


Subset of machine learning with automatic feature extraction

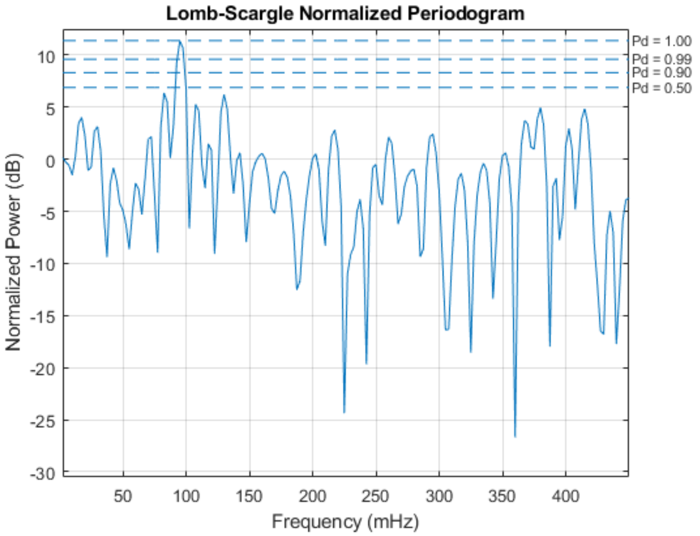
Feature Extraction: Spectral Analysis



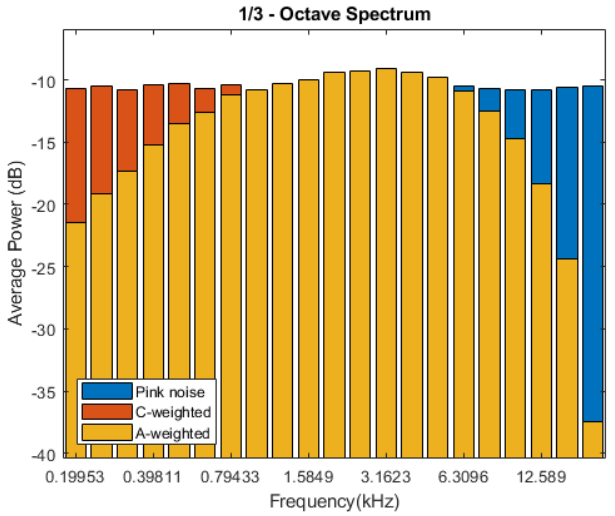
Welch periodogram



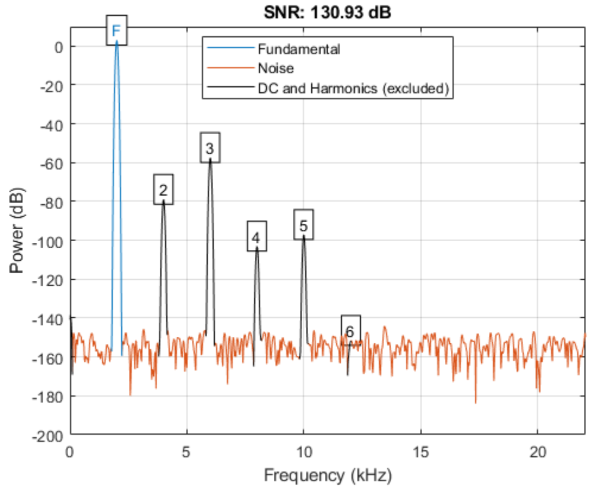
BW measurements



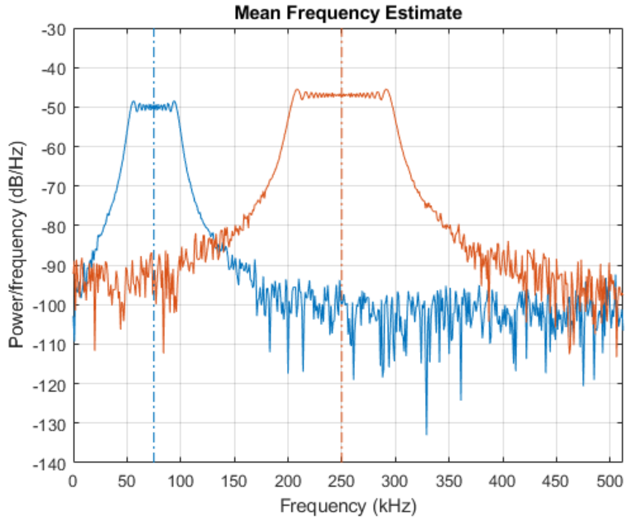
Lomb periodogram



Octave spectrum



Harmonic distortion

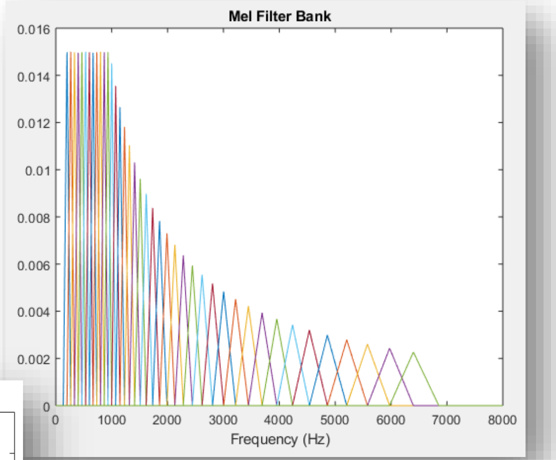


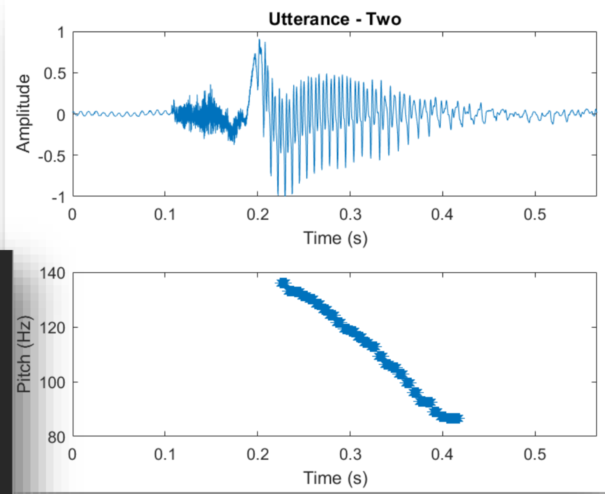
Spectral statistics

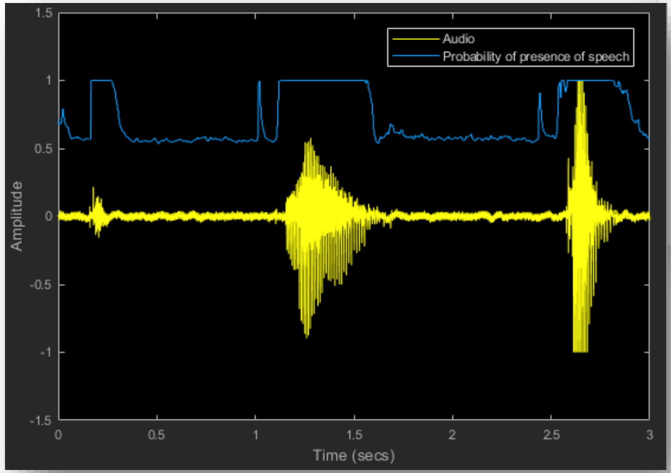
Features Extraction and Signal Segmentation

For Machine and Deep Learning

- Mel Frequency Cepstral Coefficients (MFCC)
- Pitch
- Voice Activity Detection



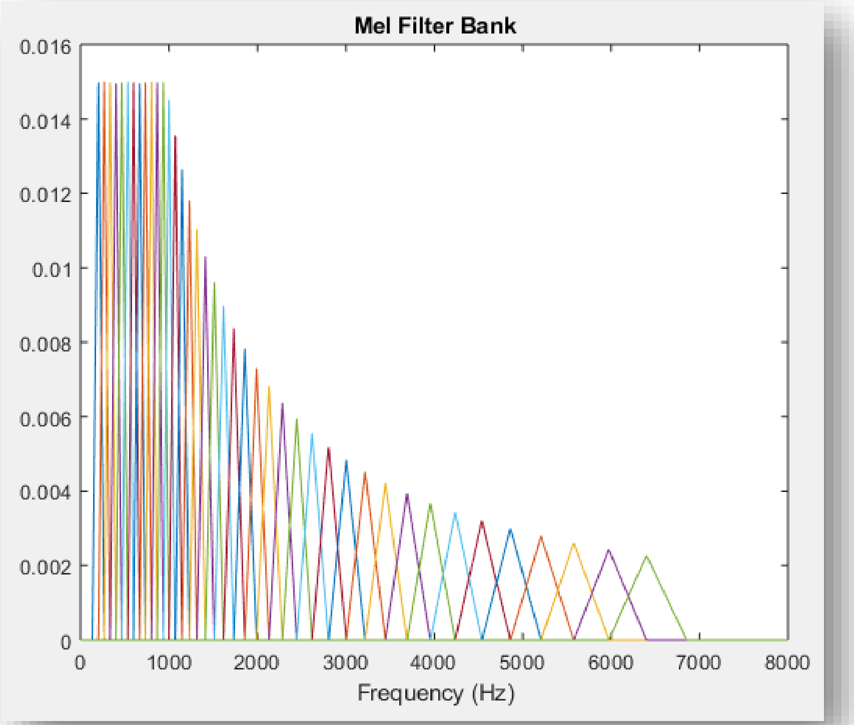




Mel Frequency Cepstral Coefficients (MFCC)

R2018a

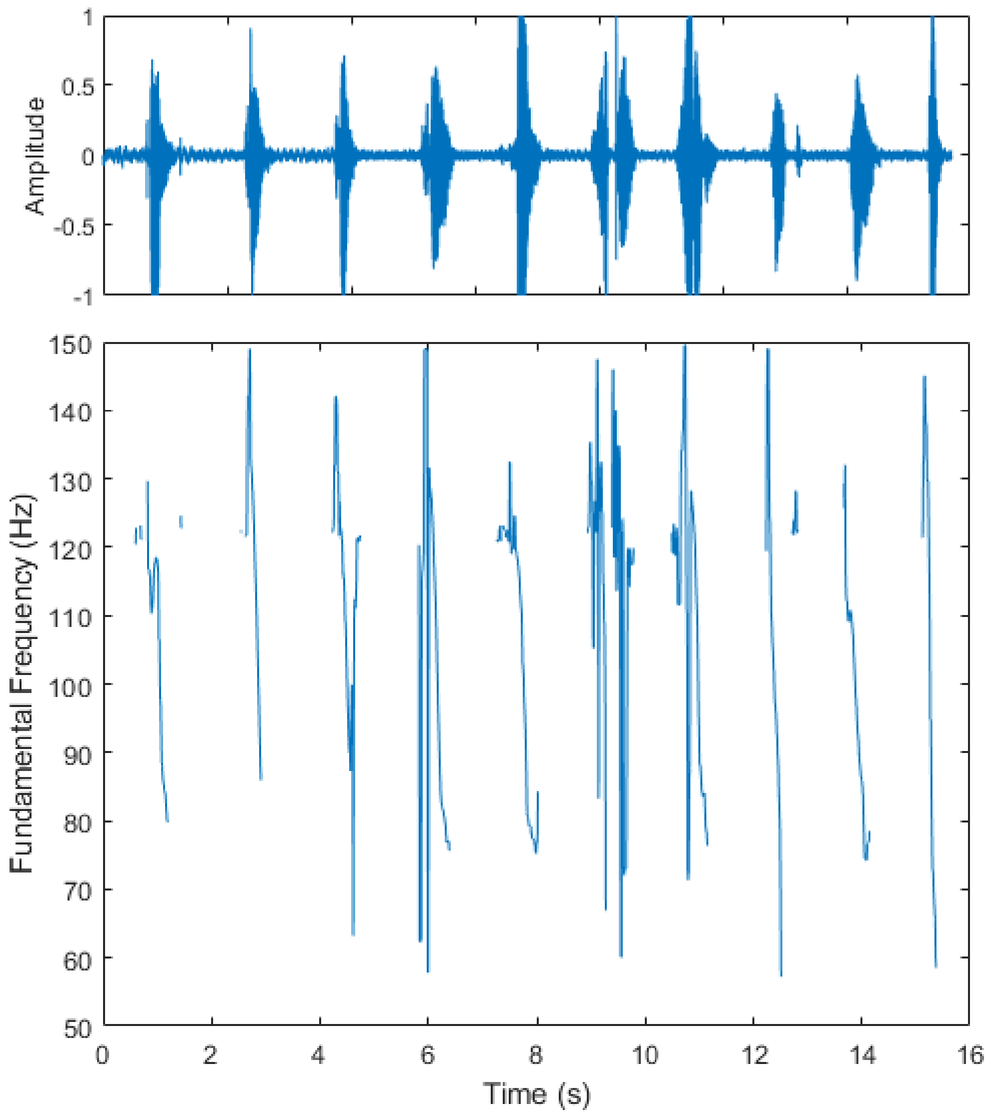
- Variation of "perceptually-adjusted" spectrum content over time
- Most common voice and speech features for Machine Learning applications
- Option of:
  - `mfcc` function,
  - `cepstralFeatureExtractor` System object
  - Cepstral Feature Extractor block (Simulink)
- Returns:
  - MFCC
  - "Delta" (= `mfcc[n]-mfcc[n-1]`)
  - "Delta delta" (= `delta[n]-delta[n-1]`)



Pitch Extraction

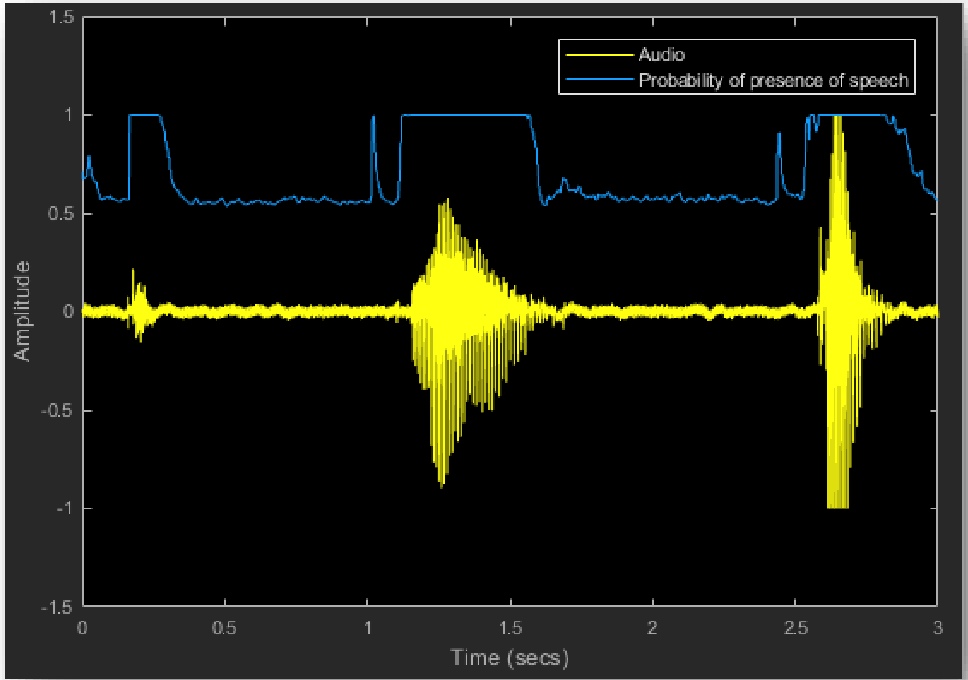
- Estimate audio pitch over time
- Popular feature for machine learning in voice and speech processing
- Choice of 5 different popular algorithms
  - `'NCF'` – Normalized Correlation Function
  - `'PEF'` – Pitch Estimation Filter
  - `'CEP'` – Cepstrum Pitch Determination
  - `'LHS'` – Log-Harmonic Summation
  - `'SRH'` – Summation of Residual Harmonics

```
[ft, idx] = pitch(audioIn, fs);
```



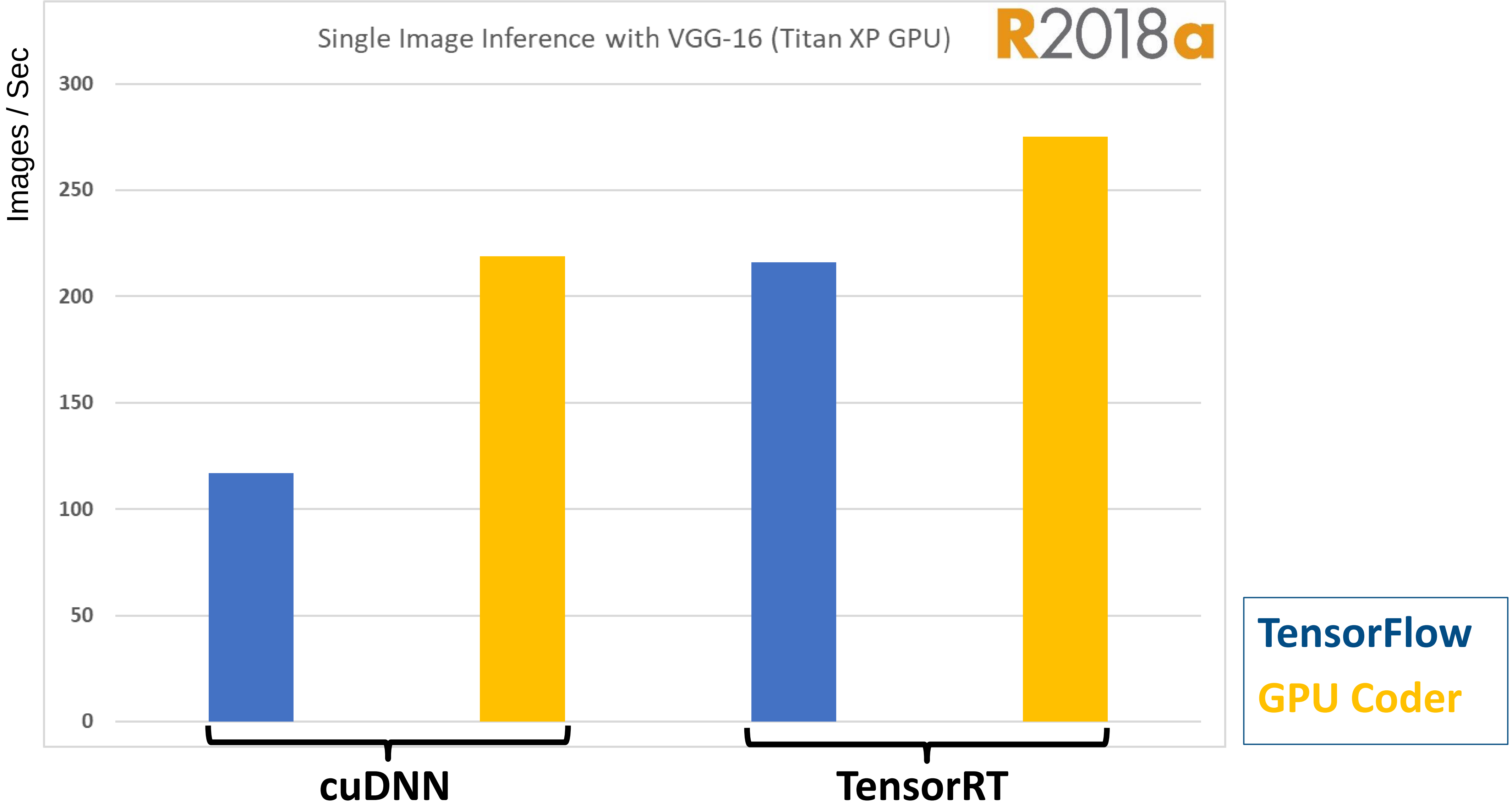
Voice Activity Detection (VAD)

- Standard algorithm to segment voice and speech signals
- `voiceActivityDetector` System object for MATLAB
- Voice Activity Detector block for Simulink



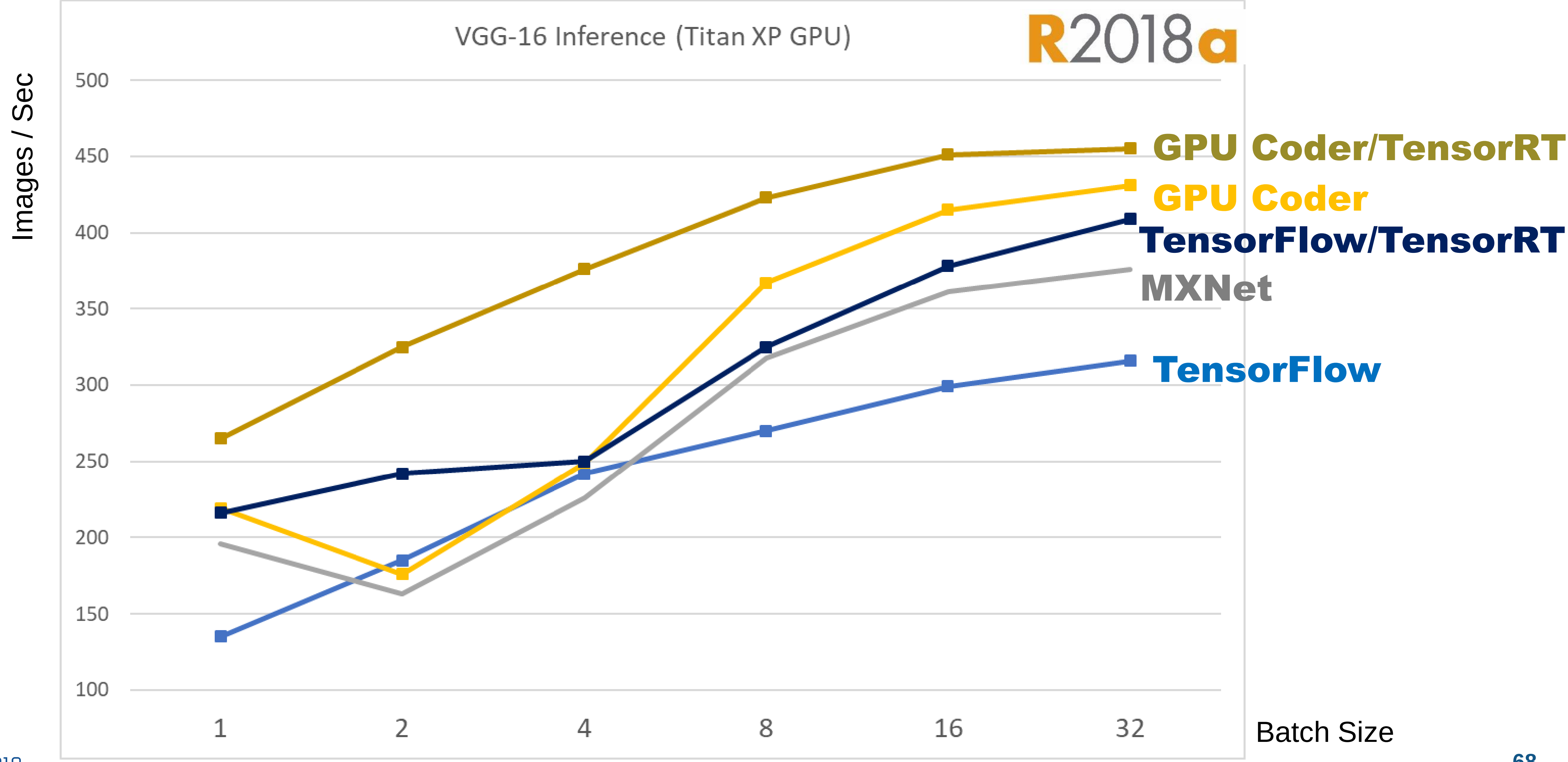

```
>> vad = voiceActivityDetector

vad =

  voiceActivityDetector with properties:

                    InputDomain: 'Time'
                         Window: 'Hann'
                      FFTLength: []
    SilenceToSpeechProbability: 0.2000
    SpeechToSilenceProbability: 0.1000
```

TensorRT speeds up inference for TensorFlow and GPU Coder

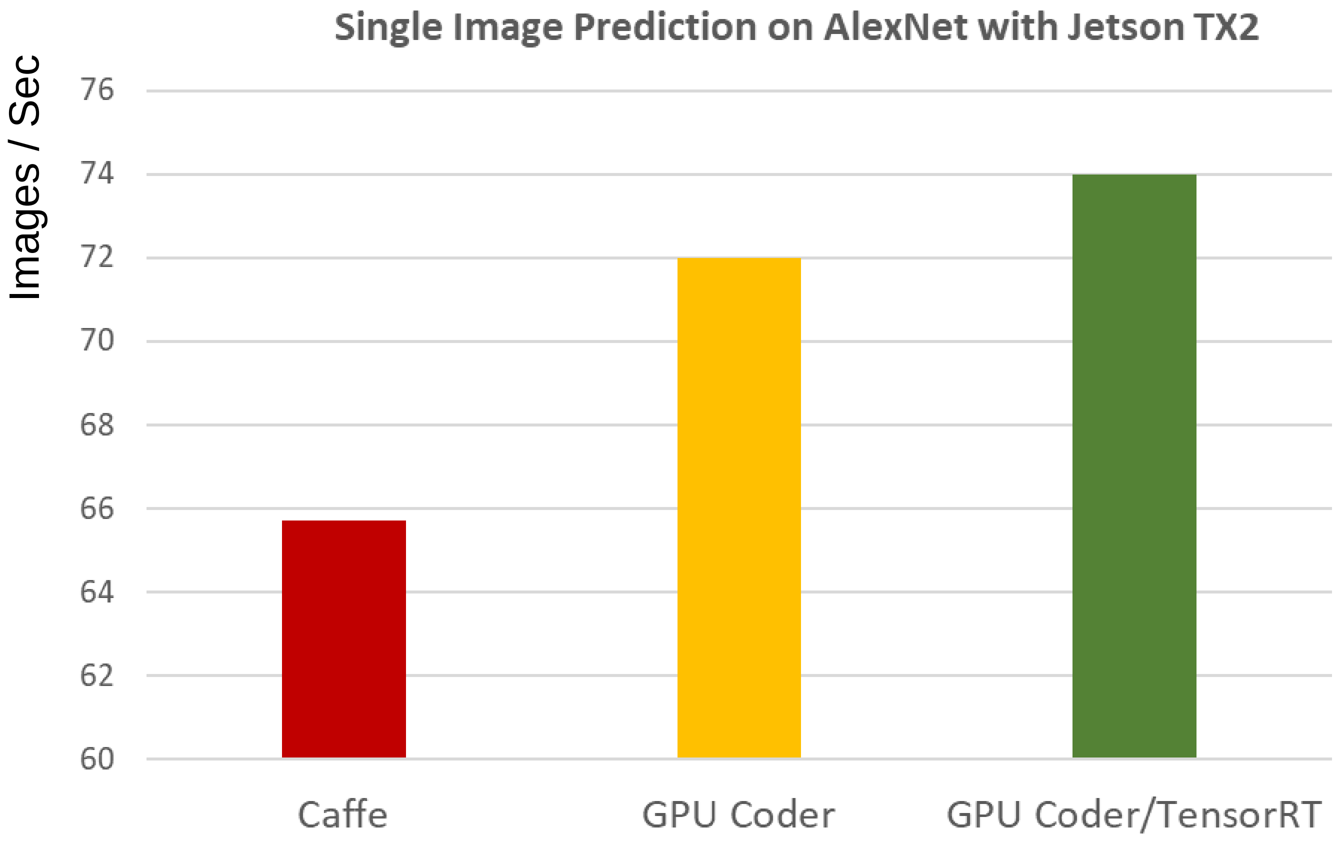


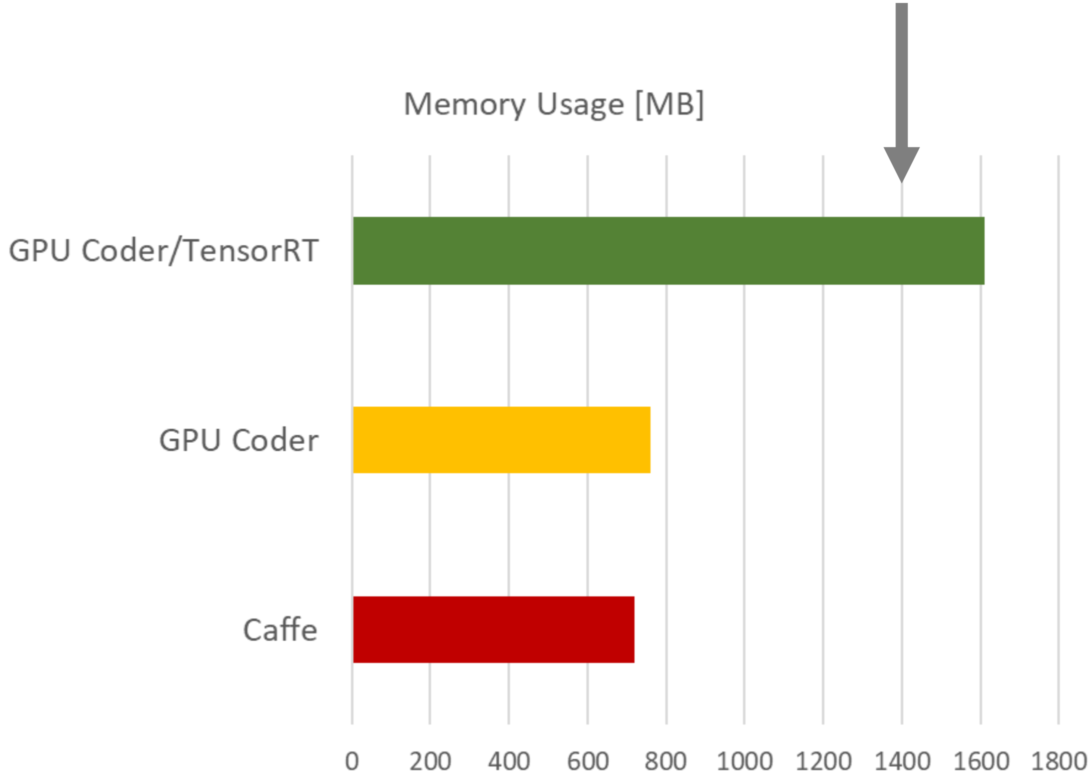
Intel® Xeon® CPU 3.6 GHz - NVIDIA libraries: CUDA9 - cuDNN 7 - Frameworks: TensorFlow 1.6.0, MXNet 1.1.0, MATLAB 18a

GPU Coder with TensorRT provides best performance across all batch sizes



Embedded: GPU Coder also fast

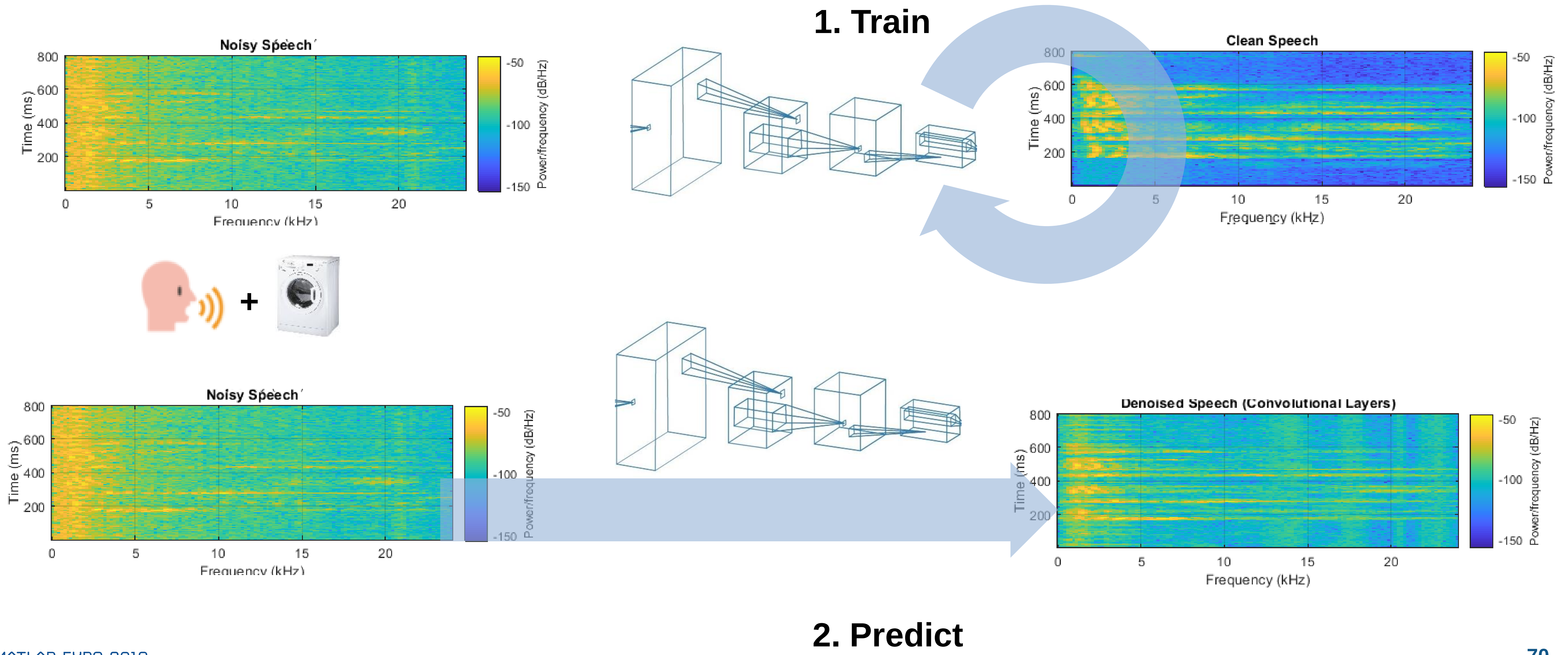




NVIDIA libraries: CUDA9 - cuDNN 7 – Jetpack 3.2.1 - Frameworks: TensorFlow 1.6.0, MXNet 1.1.0, MATLAB 18a

Example: Speech Denoising with Deep Learning

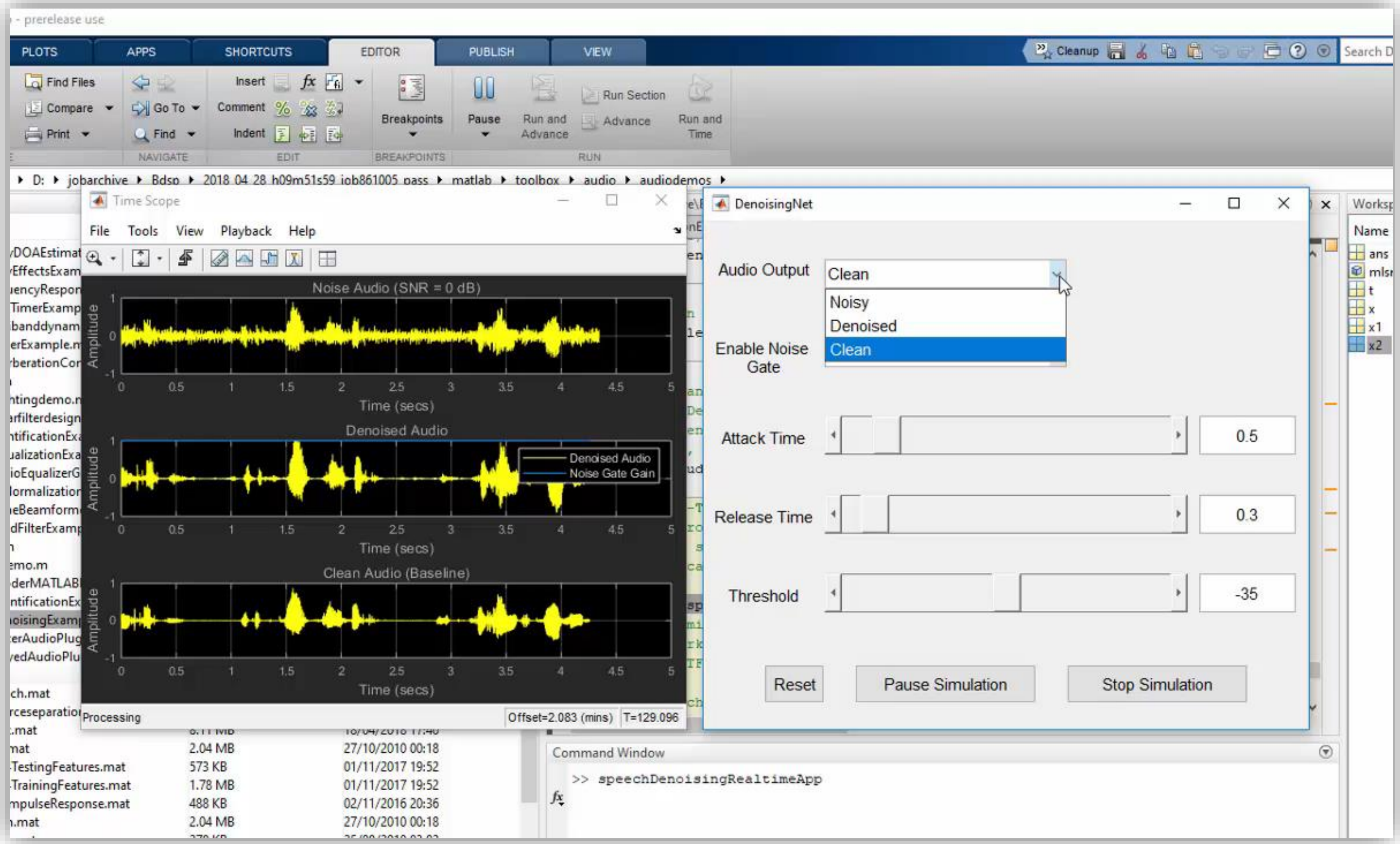
CNN & Fully Connected Networks for 2-D Regression

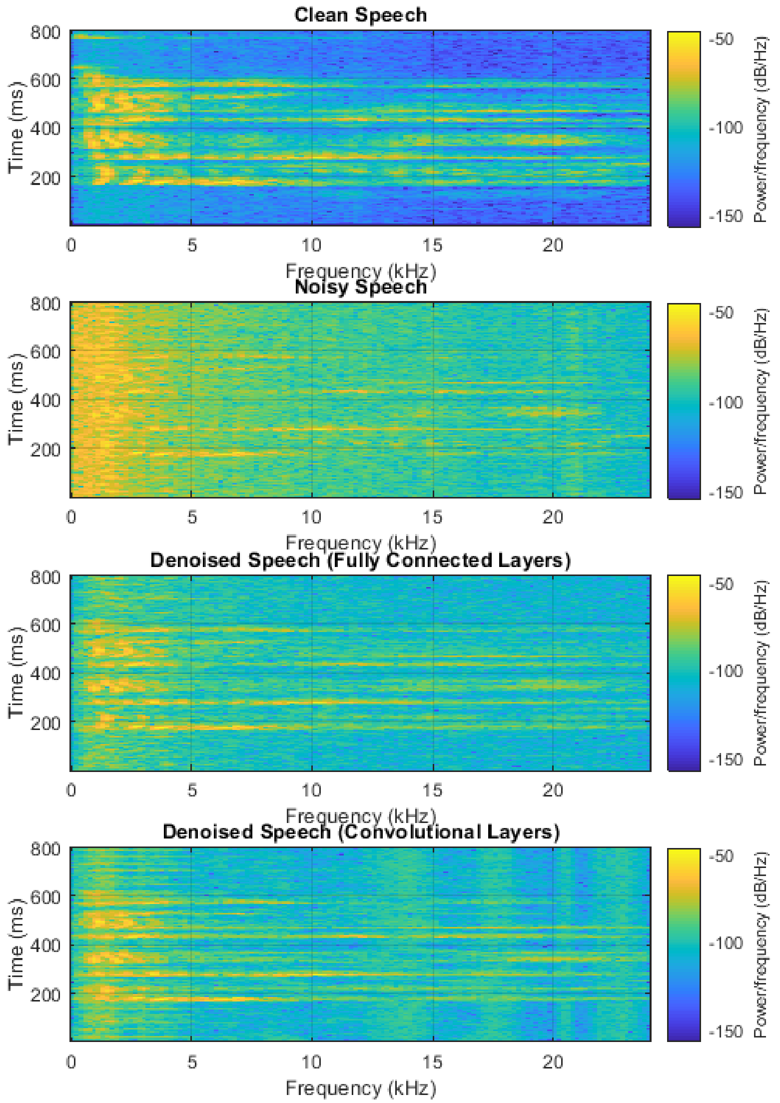


R2018b

Example 3: Speech Denoising with Deep Learning

CNN & Fully Connected Networks for 2-D Regression





Speech Denoising Demo

Extras

Summary and Release Timeline