Music Genre Classification

Prediction: (none)
## Agenda

<table>
<thead>
<tr>
<th>Why deep learning?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep learning with signal data</td>
</tr>
<tr>
<td>(Demo) Speech Command Recognition</td>
</tr>
<tr>
<td>(Demo) LSTM Networks</td>
</tr>
<tr>
<td>Enabling Features in MATLAB</td>
</tr>
<tr>
<td>Deploying deep learning</td>
</tr>
</tbody>
</table>
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

![Deep Learning Diagram](image-url)
Types of Datasets

- **Tabular Data**
  - ML or LSTM

- **Time Series/Signal Data**
  - 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**
- 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?

- Edges
- Shapes
- Objects

[Diagram showing various stages of processing leading to classification of Flower, Cup, Car, and Tree]
Speech Command Recognition

Using Convolutional Neural Networks
Example: Speech Command Recognition Using Deep Learning

CNN Network for Audio Classification

1. Train

2. Predict

Google speech command dataset

audioDatastore

auditorySpectrogram

audioDeviceReader

auditorySpectrogram

"Up"
Command Recognition Demo
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');

```matlab
datafolder = fullfile('...', 'Dataset');
ads = audiodatastore(datafolder, ...
    '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. Labelling 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. Labelling 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 use 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(ad, indices) to create a datastore that contains only the files and labels indexed by indices. Reduce the datastore ad so that it contains only the commands and the subset of unknown words. Count the number of examples belonging to each class.

```matlab
commands = categorical(['yes', 'no', 'up', 'down', 'left', 'right', 'on', 'off', 'stop', 'go']);
isCommand = ismember(ad.Labels, commands);
isUnknown = ~ismember(ad.Labels, commands, 'background_noise_');

IncludeFraction = 0.2;
mask = rand(numel(ad.Labels), 1) < includeFraction;
isUnknown = isUnknown & mask;
ad.Labels(isUnknown) = categorical('unknown');

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

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.

```matlab
[adTrain, adValidation, adTest] = splitData(ad, 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. numChannels is the number of channels, which is the same as the number of frames in the spectrogram.
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.

```matlab
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.

```matlab
[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.
**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.

```matlab
[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.

```matlab
segmentDuration = 1;
frameDuration = 0.025;
hopDuration = 0.010;
umBands = 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 `eps11`.

```matlab
spectrogram = X;

XTrain = speechSpectrograms(adsTrain, segmentDuration, frameDuration, hopDuration, numBands);
XTrain = log10(XTrain + eps1);

XValidation = speechSpectrograms(adsValidation, segmentDuration, frameDuration, hopDuration, numBands);
XValidation = log10(XValidation + eps1);

XTest = speechSpectrograms(adsTest, segmentDuration, frameDuration, hopDuration, numBands);
XTest = log10(XTest + eps1);

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.

```matlab
[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.

```matlab
segmentDuration = 1;
frameDuration = 0.025;
hopDuration = 0.010;
umBands = 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`.

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

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

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

for i = 1:numFiles

    [~,info] = read(ad);

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

    spec = auditorySpectrogram(x,fs,...
    'WindowLength',frameLength,...
    'OverlapLength',frameLength - hopLength,...
    'NumBands',numBands,...
    'Range',[60,7000],...
    '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:end+(w-1);
    X(:,ind,1,:) = spec;

end

if mod(i,1000) == 0
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.

```matlab
[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.

```matlab
segmentDuration = 1;
frameDuration = 0.025;
hopDuration = 0.010;
umBands = 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`.

```matlab
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.

```matlab
specMin = min(XTrain(:));
specMax = max(XTrain(:));
idx = randperm(size(XTrain,1),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(:,1,1,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.

```matlab
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.

```matlab
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")
```
Convolutional Neural Networks for Small-footprint Keyword Spotting

Tara N. Sainath, Carolina Parada

Abstract

We explore using Convolutional Neural Networks (CNNs) for a small-footprint keyword spotting (KWS) task. CNNs are competitive for KWS since they have been shown to outperform DNNs with far fewer parameters. We consider two applications in our work, one where we limit the number of multiplications of the KWS system and another where we limit the number of parameters. We present new CNN architectures to address the constraints of each application. We find that CNN architectures offer a 27-44% relative improvement in false reject rate compared to a DNN, while still satisfying the constraints of each application.

1. Introduction

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 \( V \in \mathbb{R}^{N_x} \), where \( I \) and \( f \) are the input feature dimension in time and frequency respectively. A weight

The second convolutional filter has a filter size of 3, and max-pooling is performed in the last convolutional layer.

For example, in our task if we want parameters below 250K, a typical architecture is shown in Table 1. We will refer to this as cnn-trad-tfpool3 in this paper. A convolutional, one linear layer and one max-pooling layer is used in particular the pooling in frequency, cost was reduced.

However, a main issue with this architecture is the number of multiplications in the convolutional layer, which is not accounted for in the second layer because of the input, spanning across time, frequency and type of architecture infeasible for poolt KWS tasks where multiplications are even if our application is limited by parameters, other architectures which pool in suit are for KWS. Below we present architectures to address the tasks of limiting parameters.

### Table 4: CNNs for Striding in Time

<table>
<thead>
<tr>
<th>Model</th>
<th>Layer</th>
<th>m</th>
<th>r</th>
<th>n</th>
<th>s</th>
<th>q</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnn-treduc2</td>
<td>conv</td>
<td>16</td>
<td>8</td>
<td>78</td>
<td>2</td>
<td>3</td>
<td>10.0K</td>
</tr>
<tr>
<td>conv</td>
<td>9</td>
<td>4</td>
<td>78</td>
<td>11</td>
<td>1</td>
<td>219.0K</td>
<td></td>
</tr>
<tr>
<td>cnn-treduc3</td>
<td>conv</td>
<td>16</td>
<td>8</td>
<td>100</td>
<td>4</td>
<td>3</td>
<td>12.8K</td>
</tr>
<tr>
<td>conv</td>
<td>5</td>
<td>4</td>
<td>78</td>
<td>11</td>
<td>1</td>
<td>200.0K</td>
<td></td>
</tr>
<tr>
<td>cnn-treduc4</td>
<td>conv</td>
<td>8</td>
<td>126</td>
<td>8</td>
<td>3</td>
<td>16.1K</td>
<td></td>
</tr>
<tr>
<td>conv</td>
<td>5</td>
<td>4</td>
<td>78</td>
<td>11</td>
<td>1</td>
<td>190.5K</td>
<td></td>
</tr>
<tr>
<td>cnn-treduc5</td>
<td>conv</td>
<td>16</td>
<td>8</td>
<td>128</td>
<td>8</td>
<td>3</td>
<td>32.2K</td>
</tr>
</tbody>
</table>

### Table 5: CNNs for Pooling in Time

<table>
<thead>
<tr>
<th>Model</th>
<th>Layer</th>
<th>m</th>
<th>r</th>
<th>n</th>
<th>s</th>
<th>q</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnn-tp000</td>
<td>conv</td>
<td>21</td>
<td>8</td>
<td>94</td>
<td>2</td>
<td>3</td>
<td>5.6M</td>
</tr>
<tr>
<td>conv</td>
<td>6</td>
<td>4</td>
<td>94</td>
<td>11</td>
<td>1</td>
<td>1.8M</td>
<td></td>
</tr>
<tr>
<td>cnn-tp001</td>
<td>conv</td>
<td>15</td>
<td>8</td>
<td>94</td>
<td>3</td>
<td>3</td>
<td>7.1M</td>
</tr>
<tr>
<td>conv</td>
<td>6</td>
<td>4</td>
<td>94</td>
<td>11</td>
<td>1</td>
<td>1.8M</td>
<td></td>
</tr>
<tr>
<td>cnn-tp002</td>
<td>conv</td>
<td>16</td>
<td>8</td>
<td>94</td>
<td>3</td>
<td>3</td>
<td>65.5K</td>
</tr>
<tr>
<td>conv</td>
<td>6</td>
<td>4</td>
<td>94</td>
<td>11</td>
<td>1</td>
<td>1.8M</td>
<td></td>
</tr>
</tbody>
</table>

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. We will refer to these architectures as cnn-tp001 and cnn-tp002. For simplicity, we have omitted certain variables held constant for all experiments, namely time and frequency stride \( s = 1 \) and \( r = 1 \). Notice that by pooling in time, we can increase the number of feature maps \( n \) to keep the total number of parameters constant.
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.

```matlab
classWeights = 1./countcats(YTrain);
classWeights = classWeights./mean(classWeights);
numClasses = numel(categories(YTrain));

dropoutProb = 0.2;
umF = 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;
umF = 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
  maxPooling2dLayer(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.

```matlab
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.

```matlab
trainedNet = trainNetwork(augmnstrnTrain,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.

```matlab
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 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.
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 the 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.

```matlab
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%

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

Validation error: 4.2489%

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.

```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])
```

Confusion Matrix for Validation Data

<table>
<thead>
<tr>
<th></th>
<th>yes</th>
<th>2</th>
<th>9</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>1</td>
<td>232</td>
<td>9</td>
<td>4</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>up</td>
<td></td>
<td>240</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>down</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>255</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>
Confusion Matrix for Validation Data

<table>
<thead>
<tr>
<th>True class</th>
<th>yes</th>
<th>no</th>
<th>up</th>
<th>down</th>
<th>left</th>
<th>right</th>
<th>on</th>
<th>off</th>
<th>stop</th>
<th>go</th>
<th>unknown</th>
<th>background</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>251</td>
<td>2</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>240</td>
<td>255</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>no</td>
<td>2</td>
<td>232</td>
<td>9</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>up</td>
<td>240</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>97.2%</td>
<td>10.8%</td>
<td>97.2%</td>
<td>2.8%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>down</td>
<td></td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>255</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>left</td>
<td></td>
<td>1</td>
<td>240</td>
<td>3</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>93.3%</td>
<td>6.3%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>right</td>
<td></td>
<td>2</td>
<td>4</td>
<td>242</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>94.2%</td>
<td>5.8%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>on</td>
<td></td>
<td>2</td>
<td>252</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>98.4%</td>
<td>1.6%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>off</td>
<td></td>
<td>2</td>
<td>1</td>
<td>235</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>95.5%</td>
<td>4.5%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stop</td>
<td></td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>247</td>
<td>4</td>
<td>2</td>
<td></td>
<td>95.0%</td>
<td>5.0%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>go</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>252</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>96.6%</td>
<td>3.4%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unknown</td>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>88.6%</td>
<td>11.4%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>background</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100.0%</td>
<td></td>
</tr>
</tbody>
</table>

Predicted class

97.3% 93.9% 96.4% 92.1% 94.5% 96.4% 99.2% 98.7% 90.8% 97.3% 84.8% 98.8%
2.7%  6.1%  3.6%  7.9%  5.5%  3.6%  0.8%  1.3%  9.2%  2.7% 15.2%  1.2%
while ishandle(h)

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

    % Compute the spectrogram of the latest audio sample.
    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 buffer.
    [YPredicted,probs] = classify(trainedNet,spec,'Exclu...
    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)
    pccolor(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

“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
## Agenda

<table>
<thead>
<tr>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why deep learning?</td>
</tr>
<tr>
<td>Deep learning with signal data</td>
</tr>
<tr>
<td>(Demo) Speech Command Recognition</td>
</tr>
<tr>
<td>(Demo) LSTM Networks</td>
</tr>
<tr>
<td>Enabling Features in MATLAB</td>
</tr>
<tr>
<td>Deploying deep learning</td>
</tr>
</tbody>
</table>
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

\[ h_t = A(X_t, h_{t-1}, c_{t-1}) \]

\[ c_t = c_{t-1} \]

\[ h_t = A(h_{t-1}, c_t, X_t) \]
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
## Agenda

<table>
<thead>
<tr>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why deep learning?</td>
</tr>
<tr>
<td>Deep learning with signal data</td>
</tr>
<tr>
<td>(Demo) Speech Command Recognition</td>
</tr>
<tr>
<td>(Demo) LSTM Networks</td>
</tr>
<tr>
<td>Enabling Features in MATLAB</td>
</tr>
<tr>
<td>Deploying deep learning</td>
</tr>
</tbody>
</table>
Deep Learning on CPU, GPU, Multi-GPU and Clusters

**How to Target?**

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

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

```matlab
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

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

ads = ...

Datastore with properties:
  Files: {
    ...
    '
    ...
    '
    ...
    ...
    and 64724 more
  }
  Labels: [_background_noise_; _background_noise_; _background_noise_; ... and 64724 more categorical]
  ReadMethod: 'File'
  OutputDataType: 'double'

ads = getNextSubsetDatastore(ads,isCommand|isUnknown);
countEachLabel(ads)
ans = 11x2 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
Load or record audio to visualize and label.

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 find.

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 eunuch dive was far short of perfect.
5. The beauty of the view stunned the young boy.
6. Two blue fish swam in the tank.
Signal Analyzer App R2018a
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 estimation
- Hilbert-Huang transform
<table>
<thead>
<tr>
<th>Agenda</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why deep learning?</td>
</tr>
<tr>
<td>Deep learning with signal data</td>
</tr>
<tr>
<td>(Demo) Speech Command Recognition</td>
</tr>
<tr>
<td>(Demo) LSTM Networks</td>
</tr>
<tr>
<td>Enabling Features in MATLAB</td>
</tr>
<tr>
<td>Deploying deep learning</td>
</tr>
</tbody>
</table>
Deploying Deep Learning Models for Inference

- NVIDIA TensorRT & cuDNN Libraries
- ARM Compute Library
- Intel MKL-DNN Library

![Diagram showing Code Generation process with NVIDIA, ARM, and Intel libraries connected to CNN models.](image)
GPU Coder Fills a Gap in the Deep Learning Solution

Access Data → Preprocess → Select Network → Train → Deploy

Image Acq. → Image Processing → Computer Vision → Neural Network → PCT → GPU Coder

Training → Inference
With GPU Coder, MATLAB is fast

GPU Coder is faster than TensorFlow, MXNet and Pytorch

Single Image Inference (Titan XP, Linux)
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)

- 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

\[ [ft, idx] = \text{pitch(audioIn, fs)}; \]
Voice Activity Detection (VAD)

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

```matlab
>> 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
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

1. Train

2. Predict
Example 3: Speech Denoising with Deep Learning
CNN & Fully Connected Networks for 2-D Regression
Speech Denoising Demo
Summary and Release Timeline