



The
University
Of
Sheffield.

Deep Learning

ACS61011

Deep Learning Assignment: Individual Project (ACS61011)

By

Ankur Tiwari

Registration Number 210158775

Contents

Task1 : Pre-processing data	3
Task 2 : Implementing Basic CNN for speech Recognition.....	7
Task 3	10
3.1 Grid Search	10
3.2 Model Averaging	14
3.2 Add five or more additional words to dataset	20
Task 4	25
Task 5: Implementing Bayesian Optimization.....	25
Conclusion:	29

Task1 : Pre-processing data

The MATLAB file given for data Pre-Processing can be divided into following sub sections:

Loading speech commands Data

```
clear all

% define the random number seed for repeatable results

rng(1,'twister');

% define the folder for the speech dataset
datasetFolder = 'C:\Users\tiwar\MATLAB Drive\coursework1\speechDataReduced';

% Create an audioDatastore that points to the data set
ads = audioDatastore(datasetFolder, ...
    'IncludeSubfolders',true, ...
    'FileExtensions','.wav', ...
    'LabelSource','foldernames');
```

Choose Words to Recognize

```
% define a subset of command words to recognise from the full data set
commands =
categorical(["yes","no","up","down","left","right","on","off","stop","go"]);

% define an index into command words and unknown words
isCommand = ismember(ads.Labels,commands);
isUnknown = ~ismember(ads.Labels,[commands,"_background_noise_"]);

% specify the fraction of unknown words to include - labeling words that
% are not commands as |unknown| creates a group of words that approximates
% the distribution of all words other than the commands. The network uses
% this group to learn the difference between commands and all other words.
includeFraction = 0.1;
mask = rand(numel(ads.Labels),1) < includeFraction;
isUnknown = isUnknown & mask;
ads.Labels(isUnknown) = categorical("unknown");

% create a new data store of only command words and unknown words
adsSubset = subset(ads,isCommand|isUnknown);

% count the number of instances of each word
countEachLabel(adsSubset)
```

Define Training, Validation, and Test data Sets

```
% define split proportion for training, validation and test data
p1 = 0.6; % training data proportion
p2 = 0.4; % validation data proportion
[adsTrain,adsValidation] = splitEachLabel(adsSubset,p1);

% reduce the dataset to speed up training
numUniqueLabels = numel(unique(adsTrain.Labels));
nReduce = 1.2; % Reduce the dataset by a factor of nReduce
adsTrain = splitEachLabel(adsTrain,round(numel(adsTrain.Files) / numUniqueLabels /
nReduce));
adsValidation = splitEachLabel(adsValidation,round(numel(adsValidation.Files) /
numUniqueLabels / nReduce));
```

define object for computing auditory spectrograms from audio data

```

% spectrogram parameters
fs = 16e3; % sample rate of the data set
segmentDuration = 1; % duration of each speech clip (in seconds)
frameDuration = 0.025; % duration of each frame for spectrum calculation
hopDuration = 0.010; % time step between each spectrum

segmentSamples = round(segmentDuration*fs); % number of segment samples
frameSamples = round(frameDuration*fs); % number of frame samples
hopSamples = round(hopDuration*fs); % number of hop samples
overlapSamples = frameSamples - hopSamples; % number of overlap samples

FFTLength = 512; % number of points in the FFT
numBands = 50; % number of filters in the auditory spectrogram

% extract audio features using spectrogram - specifically bark spectrum
afe = audioFeatureExtractor( ...
    'SampleRate',fs, ...
    'FFTLength',FFTLength, ...
    'Window',hann(frameSamples,'periodic'), ...
    'OverlapLength',overlapSamples, ...
    'barkSpectrum',true);
setExtractorParams(afe,'barkSpectrum','NumBands',numBands);

```

Process a file from the dataset to get denormalization factor

```

% apply zero-padding to the audio signal so they are a consistent length of 1 sec
x = read(adsTrain);
numSamples = size(x,1);
numToPadFront = floor( (segmentSamples - numSamples)/2 );
numToPadBack = ceil( (segmentSamples - numSamples)/2 );
xPadded = [zeros(numToPadFront,1,'like',x);x;zeros(numToPadBack,1,'like',x)];

% extract audio features - the output is a Bark spectrum with time across rows
features = extract(afe,xPadded);
[numHops,numFeatures] = size(features);

% determine the denormalization factor to apply for each signal
unNorm = 2/(sum(afe.Window)^2);

```

Feature extraction: read data file, zero pad, then apply spectrogram methods

```

% Training data: read from datastore, zero-pad, extract spectrogram features
subds = partition(adsTrain,1,1);
XTrain = zeros(numHops,numBands,1,numel(subds.Files));
for idx = 1:numel(subds.Files)
    x = read(subds);
    xPadded = [zeros(floor((segmentSamples-size(x,1))/2),1);x;zeros(ceil((segmentSamples-size(x,1))/2),1)];
    XTrain(:, :, :, idx) = extract(afe,xPadded);
end
XTrainC{1} = XTrain;
XTrain = cat(4,XTrainC{:});
% extract parameters from training data
[numHops,numBands,numChannels,numSpec] = size(XTrain);
% Scale the features by the window power and then take the log (with small offset)
XTrain = XTrain/unNorm;
epsil = 1e-6;
XTrain = log10(XTrain + epsil);

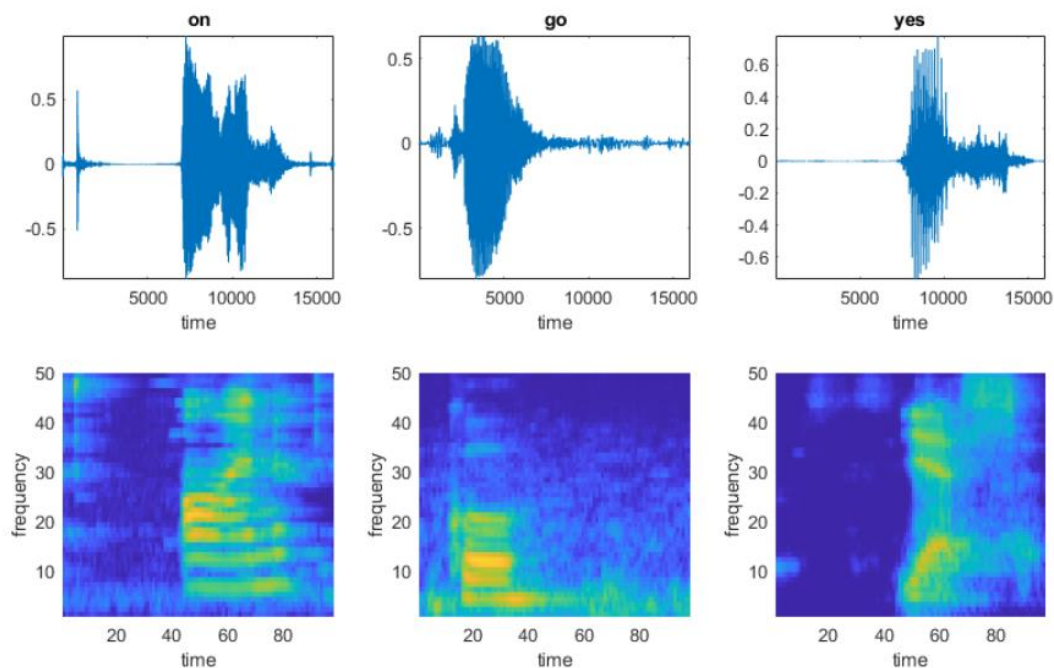
% Validation data: read from datastore, zero-pad, extract spectrogram features
subds = partition(adsValidation,1,1);

```

```

XValidation = zeros(numHops,numBands,1,numel(subds.Files));
for idx = 1:numel(subds.Files)
    x = read(subds);
    xPadded = [zeros(floor((segmentSamples-
size(x,1))/2),1);x;zeros(ceil((segmentSamples-size(x,1))/2),1)];
    XValidation(:,:,1,idx) = extract(afe,xPadded);
end
XValidationC{1} = XValidation;
XValidation = cat(4,XValidationC{:});
XValidation = XValidation/unNorm;
XValidation = log10(XValidation + epsilon);
% Isolate the train and validation labels. Remove empty categories.
YTrain = removecats(adsTrain.Labels);
YValidation = removecats(adsValidation.Labels);
% Visualize Data - plot waveforms and play clips for a few examples
specMin = min(XTrain,[],'all');
specMax = max(XTrain,[],'all');
idx = randperm(numel(adsTrain.Files),3);
figure('Units','normalized','Position',[0.2 0.2 0.6 0.6]);
for i = 1:3
    [x,fs] = audioread(adsTrain.Files{idx(i)});
    subplot(2,3,i)
    plot(x)
    axis tight
    title(string(adsTrain.Labels{idx(i)}))
    xlabel('time')
    subplot(2,3,i+3)
    spect = (XTrain(:,:,1,idx(i)))';
    pcolor(spect)
    caxis([specMin specMax])
    shading flat
    sound(x,fs)
    xlabel('time')
    ylabel('frequency')
    pause(2)
end

```



Add Background Noise Data

```

% Use the audio files in the |_background_noise_|_ folder to create samples
% of background noise
adsBkg = subset(ads,ads.Labels=="_background_noise_");
numBkgClips = 400;

numBkgFiles = numel(adsBkg.Files);
numClipsPerFile = histcounts(1:numBkgClips,linspace(1,numBkgClips,numBkgFiles+1));
Xbkg = zeros(size(XTrain,1),size(XTrain,2),1,numBkgClips,'single');
bkgAll = readall(adsBkg);
ind = 1;

% rescale audio clip with value sampled from volumeRange
volumeRange = log10([1e-4,1]);
for count = 1:numBkgFiles
    bkg = bkgAll{count};
    idxStart = randi(numel(bkg)-fs,numClipsPerFile(count),1);
    idxEnd = idxStart+fs-1;
    gain = 10.^((volumeRange(2)-volumeRange(1))*rand(numClipsPerFile(count),1) +
volumeRange(1));
    for j = 1:numClipsPerFile(count)

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

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

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

        if mod(ind,1000)==0
            disp("Processed " + string(ind) + " background clips out of " +
string(numBkgClips))
        end
        ind = ind + 1;
    end
end
Xbkg = Xbkg/unNorm;
Xbkg = log10(Xbkg + epsil);

% Split spectrograms of background noise between training and validation data
numTrainBkg = floor(0.85*numBkgClips);
numValidationBkg = floor(0.15*numBkgClips);
XTrain(:, :, :, end+1:end+numTrainBkg) = Xbkg(:, :, :, 1:numTrainBkg);
YTrain(end+1:end+numTrainBkg) = "background";
XValidation(:, :, :, end+1:end+numValidationBkg) = Xbkg(:, :, :, numTrainBkg+1:end);
YValidation(end+1:end+numValidationBkg) = "background";

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

figure('Units','normalized','Position',[0.2 0.2 0.5 0.5])
subplot(2,1,1)
histogram(YTrain)
title("Training Label Distribution")
subplot(2,1,2)
histogram(YValidation)
title("Validation Label Distribution")

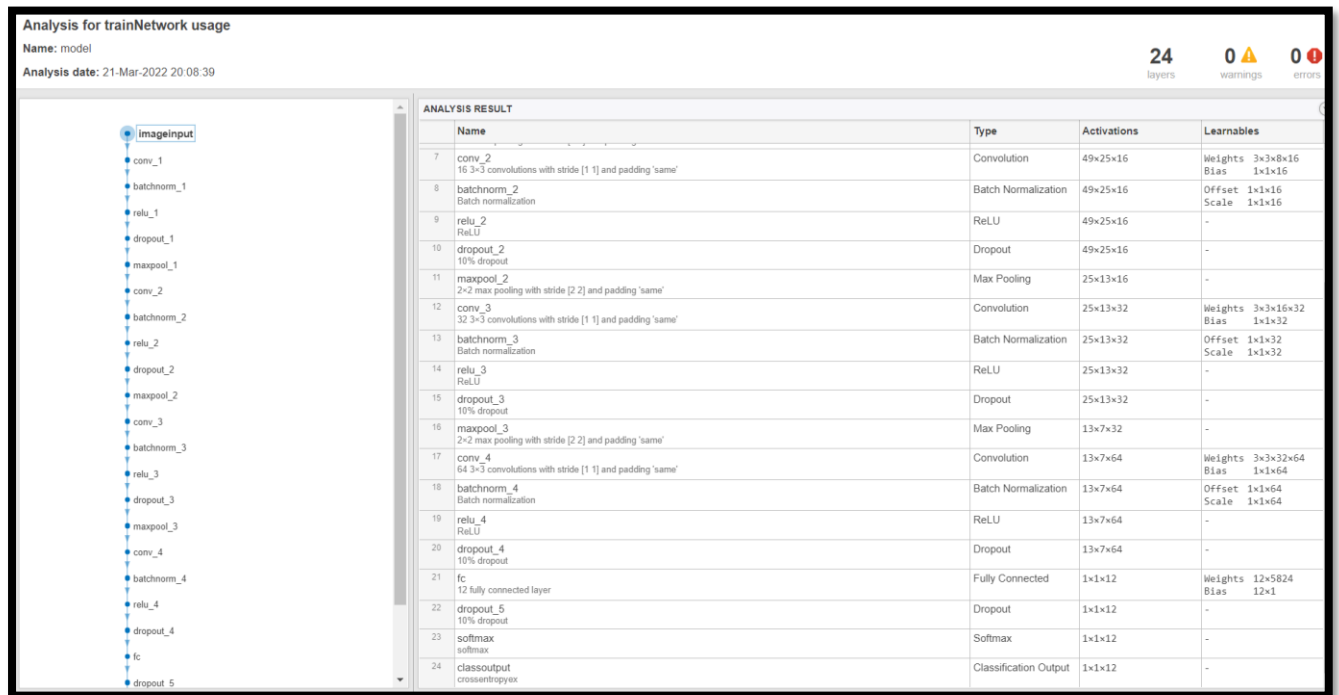
% load ACS61011projectData
save ankur XTrain YTrain XValidation YValidation

```

Task 2 : Implementing Basic CNN for speech Recognition

The objective for this task was to implement a deep learning model on the given dataset. For the implementation of this task following steps were taken:

1. The result of data preprocessing has been saved in a file and later loaded in workspace.
2. A total of 24 layers were implemented increasing the number of filters consecutively in each layer from 8 to 64
3. Each hidden layer constitutes of a dropout layer to reduce the variance



Name	Type	Activations	Learnables
conv_1	Convolution	49x25x16	Weights 3x3x8x16 Bias 1x1x16
batchnorm_1	Batch Normalization	49x25x16	Offset 1x1x16 Scale 1x1x16
relu_1	ReLU	49x25x16	-
dropout_1	Dropout	49x25x16	-
maxpool_1	Max Pooling	25x13x16	-
conv_2	Convolution	25x13x32	Weights 3x3x16x32 Bias 1x1x32
batchnorm_2	Batch Normalization	25x13x32	Offset 1x1x32 Scale 1x1x32
relu_2	ReLU	25x13x32	-
dropout_2	Dropout	25x13x32	-
maxpool_2	Max Pooling	13x7x32	-
conv_3	Convolution	13x7x64	Weights 3x3x32x64 Bias 1x1x64
batchnorm_3	Batch Normalization	13x7x64	Offset 1x1x64 Scale 1x1x64
relu_3	ReLU	13x7x64	-
dropout_3	Dropout	13x7x64	-
maxpool_3	Max Pooling	1x1x12	-
fc	Fully Connected	1x1x12	Weights 12x5824 Bias 12x1
dropout_4	Dropout	1x1x12	-
softmax	Softmax	1x1x12	-
classoutput	Classification Output	1x1x12	-

Figure 1 Model for Task 1

The code is given below:

```
clear all;
load ACS61011projectData.mat
dropout_value=0.1;
model = [

imageInputLayer([98 50 1])
convolution2dLayer([3 3],8,"Padding","same")
batchNormalizationLayer
reluLayer
dropoutLayer(dropout_value)

maxPooling2dLayer([2 2],"Padding","same","Stride",[2 2])
convolution2dLayer([3 3],16,"Padding","same")
batchNormalizationLayer
reluLayer
dropoutLayer(dropout_value)

maxPooling2dLayer([2 2],"Padding","same","Stride",[2 2])
convolution2dLayer([3 3],32,"Padding","same")
batchNormalizationLayer
reluLayer
```

```

dropoutLayer(dropout_value)

maxPooling2dLayer([2 2],"Padding","same","Stride",[2 2])
convolution2dLayer([3 3],64,"Padding","same")
batchNormalizationLayer
reluLayer
dropoutLayer(dropout_value)

fullyConnectedLayer(12)
dropoutLayer(dropout_value)
softmaxLayer

classificationLayer
];

% analyze the network
analyzeNetwork(model)
options = trainingOptions('adam', ...
'MiniBatchSize',128, ...
'MaxEpochs',30, ...
'InitialLearnRate',1e-3, ...
'ValidationData',{XValidation,YValidation},...
'Verbose',true, ...
'Plots','training-progress',...
'Plots','training-progress',...
'ExecutionEnvironment','gpu');
% Train the network using the training data

net = trainNetwork(XTrain,YTrain,model,options);

% Classify the validation images using the trained network
[YPred,probs] = classify(net,XValidation);
accuracy = 100*mean(YPred == YValidation); % accuracy
disp(['Validation Accuracy: ' num2str(accuracy) '%']);

% plot confusion matrix
plotconfusion(YValidation,YPred)

```

Initializing input data normalization.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:04	7.03%	12.55%	3.0265	3.4585	0.0010
4	50	00:00:15	70.31%	48.16%	0.9534	1.5969	0.0010
7	100	00:00:23	85.94%	55.59%	0.5811	1.4879	0.0010
10	150	00:00:30	86.72%	53.37%	0.3963	1.4789	0.0010
14	200	00:00:40	84.38%	57.05%	0.4378	1.4400	0.0010
17	250	00:00:50	89.84%	58.58%	0.3227	1.4971	0.0010
20	300	00:01:03	92.19%	62.51%	0.2356	1.4129	0.0010
24	350	00:01:13	90.62%	65.50%	0.1672	1.3524	0.0010
27	400	00:01:23	92.97%	65.93%	0.1512	1.4011	0.0010
30	450	00:01:32	91.41%	66.95%	0.1278	1.3876	0.0010

Training finished: Max epochs completed.

Validation Accuracy: 63.0231%

Figure 2 Validation Table

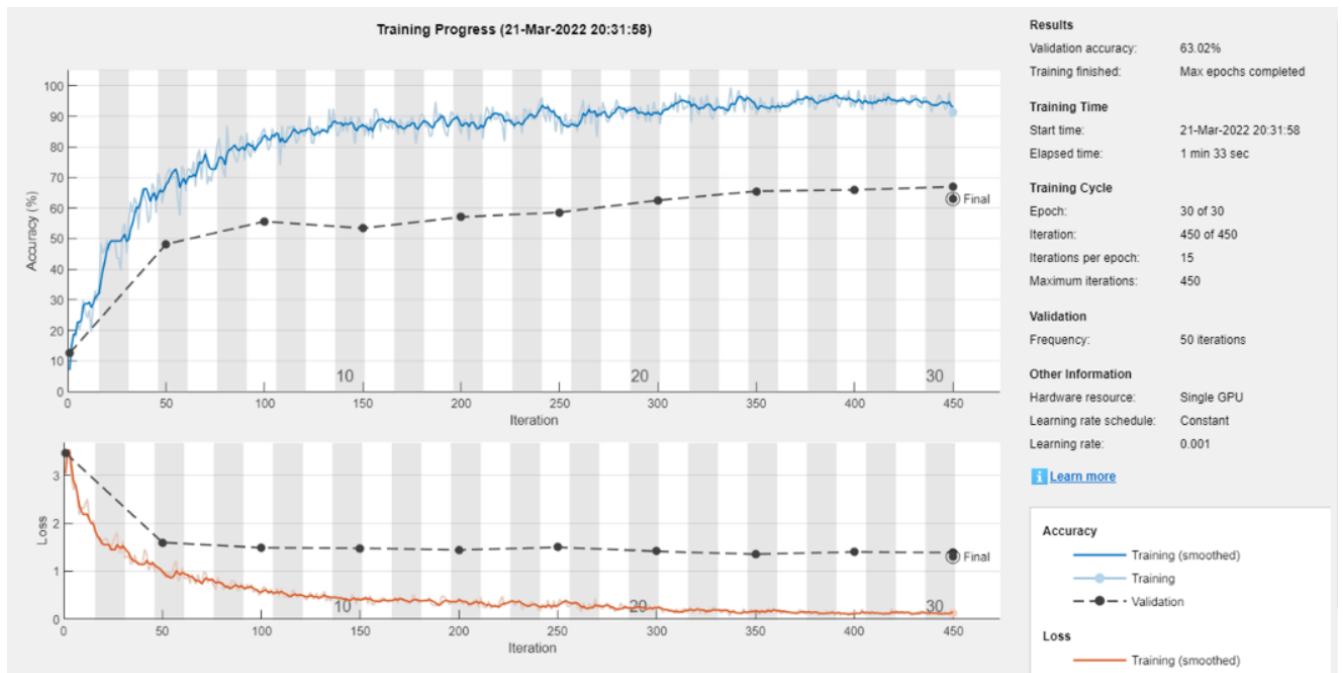


Figure 3 Training and Validation Curve

Confusion Matrix														
Output Class	down	74 6.3%	5 0.4%	2 0.2%	11 0.9%	1 0.1%	0 0.0%	1 0.1%	1 0.1%	2 0.2%	1 0.1%	5 0.4%	0 0.0%	71.8% 28.2%
	go	9 0.8%	67 5.7%	0 0.0%	17 1.5%	0 0.0%	1 0.1%	0 0.0%	2 0.2%	4 0.3%	0 0.0%	4 0.3%	0 0.0%	64.4% 35.6%
	left	2 0.2%	0 0.0%	68 5.8%	2 0.2%	0 0.0%	2 0.2%	9 0.8%	1 0.1%	2 0.2%	1 0.1%	3 0.3%	0 0.0%	75.6% 24.4%
	no	8 0.7%	15 1.3%	2 0.2%	67 5.7%	0 0.0%	1 0.1%	0 0.0%	0 0.0%	2 0.2%	4 0.3%	1 0.1%	0 0.0%	67.0% 33.0%
	off	0 0.0%	2 0.2%	3 0.3%	0 0.0%	54 4.6%	4 0.3%	0 0.0%	6 0.5%	5 0.4%	0 0.0%	0 0.0%	0 0.0%	73.0% 27.0%
	on	0 0.0%	0 0.0%	3 0.3%	2 0.2%	18 1.5%	86 7.3%	1 0.1%	2 0.2%	2 0.2%	0 0.0%	25 2.1%	0 0.0%	61.9% 38.1%
	right	0 0.0%	3 0.3%	14 1.2%	0 0.0%	2 0.2%	0 0.0%	80 6.8%	0 0.0%	1 0.1%	3 0.3%	53 4.5%	0 0.0%	51.3% 48.7%
	stop	2 0.2%	2 0.2%	1 0.1%	0 0.0%	0 0.0%	1 0.1%	1 0.1%	71 6.1%	4 0.3%	0 0.0%	0 0.0%	0 0.0%	86.6% 13.4%
	up	0 0.0%	2 0.2%	1 0.1%	0 0.0%	13 1.1%	0 0.0%	4 0.3%	6 0.5%	71 6.1%	1 0.1%	1 0.1%	0 0.0%	71.7% 28.3%
	yes	0 0.0%	1 0.1%	3 0.3%	1 0.1%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	86 7.3%	0 0.0%	0 0.0%	94.5% 5.5%
	unknown	3 0.3%	0 0.0%	2 0.2%	0 0.0%	1 0.1%	0 0.0%	4 0.3%	3 0.3%	2 0.2%	1 0.1%	5 0.4%	0 0.0%	23.8% 76.2%
	background	3 0.3%	4 0.3%	2 0.2%	1 0.1%	12 1.0%	6 0.5%	1 0.1%	9 0.8%	6 0.5%	4 0.3%	4 0.3%	60 5.1%	53.6% 46.4%
			73.3% 26.7%	66.3% 33.7%	67.3% 32.7%	66.3% 33.7%	53.5% 46.5%	85.1% 14.9%	79.2% 20.8%	70.3% 29.7%	70.3% 29.7%	85.1% 14.9%	5.0% 95.0%	100% 0.0%
		down	go	left	no	off	on	right	stop	up	yes	unknown	background	
														Target Class

Task 3

This task comprises of 3 individual Tasks .

3.1 Grid Search

The objective of this task is to perform a systematic investigation into the effects of adding layers and number of filters to a model using 3x3 grid search.

1. Grid search is performed using 2,3,4 layers and 8,16,32 filters in each layer.
2. The function provided for generating convolution network has been altered to add a dropout layer in each hidden layer.

The code for dynamic network creation has been given below:

```
function [lgraph] =  
createConvNetwork(nlayers,nfilters,filterSize,inputSize,numClasses)  
  
% function [lgraph] =  
% createConvNetwork(nlayers,nfilters,filterSize,inputSize,numClasses)  
%  
% This function creates a layer graph for a convolutional network  
% automatically based on the functions inputs. The network by default  
% creates layers of the form conv2D -> batchnorm -> relu -> maxpool2d  
%  
% function inputs:  
% nlayers: number of convolutional layers  
% nfilters: number of convolutional filters in each layer  
% filterSize: size of the convolutional filter  
% inputSize: size of the input image in rows by columns by depth  
%  
% function output:  
% lgraph: a deep convolutional network defined as a layer graph  
%  
% for help on layergraphs type help layergraph at command prompt or go to  
% https://uk.mathworks.com/help/deeplearning/ref/nnet.cnn.layergraph.html  
% define input layer  
inputLayer = imageInputLayer([inputSize(1) inputSize(2)  
inputSize(3)], 'Name', 'input');  
  
% define output layer  
outputLayer = [fullyConnectedLayer(numClasses, 'Name', 'FCoutput')  
softmaxLayer('Name', 'softmax')  
classificationLayer('Name', 'classLayer')];  
  
% initialise the layer graph  
lgraph = layerGraph;  
  
% add the input layer to the layer graph  
lgraph = addLayers(lgraph, inputLayer);  
inputString = 'input';  
% loop over convolutional layers, creating, adding and connecting them  
for i = 1:nlayers  
  
% define conv layer  
convLayer = [  
convolution2dLayer(filterSize,nfilters, 'Padding', 'same', 'Name', ['conv_'  
num2str(i)])  
batchNormalizationLayer('Name', ['BN_' num2str(i)])  
reluLayer('Name', ['relu_' num2str(i)])  
dropoutLayer(0.1, 'Name', ['dl_' num2str(i)])  
maxPooling2dLayer(3, 'Stride', 2, 'Padding', 'same', 'Name', ['maxpool_' num2str(i)])];  
  
% add layer  
lgraph = addLayers(lgraph, convLayer);
```

```

% connect layer
outputString = ['conv_' num2str(i)];
lgraph = connectLayers(lgraph,inputString,outputString);

% update input string name
inputString = ['maxpool_' num2str(i)];
end
% add the output layer at the end of the network
lgraph = addLayers(lgraph,outputLayer);
lgraph = connectLayers(lgraph,inputString,'FCoutput');

```

Code for Implementing Grid Search:

```

clear all;
load ACS61011projectData.mat

nlayers = [2,3,4]; % number of convolutional layers
nfilters = [8,16,32]; % number of filters in each layer
filterSize = 3; % filter size
inputSize = [98 50 1]; % input image size
numClasses = 12;

% Train the network
for i=1:length(nlayers)
    for j= 1:length(nfilters)
        lgraph =
createConvNetwork(nlayers(i),nfilters(j),filterSize,inputSize,numClasses);
        analyzeNetwork(lgraph);
        options = trainingOptions('adam', ...
            'MaxEpochs',60, ...
            'MiniBatchSize',128, ...
            'Plots','training-progress', ...
            'Verbose',true, ...
            'ValidationData',{XValidation,YValidation}, ...
            'ExecutionEnvironment','gpu');
        trainedNet = trainNetwork(XTrain,YTrain,lgraph,options);
        % Classify the validation images using the trained network
        [YPred,probs] = classify(trainedNet,XValidation);
        accuracy = 100*mean(YPred == YValidation); % accuracy
        disp(['Validation Accuracy: ' num2str(accuracy) '%']);
        accuracy_matrix(i,j)=accuracy;
        % plot confusion matrix
        plotconfusion(YValidation,YPred)
    end
end
End

fprintf("Table showing Validation Accuracy for various Networks");
Number_of_Layers=[nlayers(1),nlayers(2),nlayers(3)];
Filters_8=[accuracy_matrix(1,1),accuracy_matrix(2,1),accuracy_matrix(3,1)];
Filters_16=[accuracy_matrix(1,2),accuracy_matrix(2,2),accuracy_matrix(3,2)];
Filters_32=[accuracy_matrix(1,3),accuracy_matrix(2,3),accuracy_matrix(3,3)];
validation_table=table(Number_of_Layers',Filters_8',Filters_16',Filters_32')

```

some of the results of this has been shown below .

Initializing input data normalization.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:04	4.69%	10.85%	3.1230	2.9895	0.0010
4	50	00:00:08	44.53%	29.46%	1.6123	2.0900	0.0010
7	100	00:00:11	65.62%	37.92%	1.1608	1.8680	0.0010
10	150	00:00:18	82.03%	47.05%	0.6978	1.6831	0.0010
14	200	00:00:24	81.25%	51.58%	0.6631	1.6172	0.0010
17	250	00:00:29	84.38%	49.10%	0.5060	1.7621	0.0010
20	300	00:00:35	90.62%	52.95%	0.3861	1.6740	0.0010
24	350	00:00:40	92.97%	51.15%	0.3405	1.7502	0.0010
27	400	00:00:45	96.09%	50.73%	0.2348	1.7418	0.0010
30	450	00:00:51	94.53%	50.56%	0.1922	1.8305	0.0010
34	500	00:00:56	98.44%	54.57%	0.1457	1.8931	0.0010
37	550	00:01:01	96.09%	53.46%	0.1533	1.9533	0.0010
40	600	00:01:07	97.66%	50.81%	0.1335	2.0213	0.0010
44	650	00:01:12	99.22%	53.37%	0.0907	1.9846	0.0010
47	700	00:01:18	98.44%	54.57%	0.1240	2.4366	0.0010
50	750	00:01:23	97.66%	52.18%	0.1010	2.2052	0.0010
54	800	00:01:29	97.66%	53.46%	0.1194	2.1642	0.0010
57	850	00:01:34	100.00%	52.09%	0.0406	2.1597	0.0010
60	900	00:01:39	96.88%	55.00%	0.0531	2.0690	0.0010

Training finished: Max epochs completed.

Validation Accuracy: 52.8608%

Figure 5 Validation Results for 2 layers with 8 filters

Initializing input data normalization.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:29	10.16%	9.14%	3.7829	2.7685	0.0010
4	50	00:01:07	62.50%	48.51%	1.1391	1.5634	0.0010
7	100	00:01:22	78.12%	60.38%	0.7851	1.2574	0.0010
10	150	00:01:47	89.06%	67.29%	0.3746	1.0674	0.0010
14	200	00:02:08	95.31%	70.88%	0.1970	1.0086	0.0010
17	250	00:02:26	96.88%	73.10%	0.2519	1.0792	0.0010
20	300	00:02:56	98.44%	73.87%	0.1261	1.0045	0.0010
24	350	00:03:54	99.22%	75.58%	0.0904	0.9585	0.0010
27	400	00:04:31	96.88%	75.49%	0.1113	0.9941	0.0010
30	450	00:05:04	98.44%	75.23%	0.0691	1.0141	0.0010
34	500	00:05:35	99.22%	76.17%	0.0403	1.0258	0.0010
37	550	00:06:09	98.44%	77.46%	0.0808	0.9316	0.0010
40	600	00:06:33	100.00%	77.80%	0.0238	0.9677	0.0010
44	650	00:07:09	100.00%	76.69%	0.0257	0.9883	0.0010
47	700	00:07:35	99.22%	78.57%	0.0442	0.9318	0.0010
50	750	00:07:51	100.00%	77.80%	0.0165	0.9992	0.0010
54	800	00:08:13	100.00%	78.91%	0.0078	0.9932	0.0010
57	850	00:08:34	100.00%	78.57%	0.0125	0.9574	0.0010
60	900	00:08:52	100.00%	78.31%	0.0065	0.9934	0.0010

Training finished: Max epochs completed.

Validation Accuracy: 75.8326%

Figure 6 Validation Results for 4 layers 32 Filters.

3.2 Model Averaging

The objective of this task is to evaluate a bagging scheme to regularize the network from previous levels.

- Three models have been implemented
- The first model is the same model used for Task 1
- In second model all the dropout layers have been dropped
- In third model the first model is improvised by adding one more layer with reinstated dropout layers.

Code has been given below:

```
load ACS61011projectData.mat

% sample training data with replacement and train model
% size of training data
XTrainSize = size(XTrain); % get dimensions of training data XTrain
N = XTrainSize(4); % N is the number of training data samples

%====1===1====1===1===1===1===1===1===1===1===1===1===1===1===1===1=
==%

% generate random samples with replacement according to size of training
data
idx = randi([1 N],N,1);
% check this: the percentage of unique samples in idx - usually about 63%
% this indicates we have sampled about 63% of the original data
% if you do this a few different times you get a different 63% each time
uniqueSamples = 100*length(unique(idx))/length(idx);
% create new randomly sampled training data from the indices idx

%First Model
XTrain1 = XTrain(:, :, 1, idx);
YTrain1 = YTrain(idx, 1);
dropout_value=0.1;
model_1 = [

imageInputLayer([98 50 1])
convolution2dLayer([3 3],8,"Padding","same")
batchNormalizationLayer
reluLayer
dropoutLayer(dropout_value)

maxPooling2dLayer([2 2],"Padding","same","Stride",[2 2])
convolution2dLayer([3 3],16,"Padding","same")
batchNormalizationLayer
reluLayer
dropoutLayer(dropout_value)

maxPooling2dLayer([2 2],"Padding","same","Stride",[2 2])
convolution2dLayer([3 3],32,"Padding","same")
batchNormalizationLayer
reluLayer
dropoutLayer(dropout_value)

maxPooling2dLayer([2 2],"Padding","same","Stride",[2 2])
convolution2dLayer([3 3],64,"Padding","same")
batchNormalizationLayer
reluLayer
dropoutLayer(dropout_value)

fullyConnectedLayer(12)
```

[illegible]

[illegible]


```

maxPooling2dLayer([2 2],"Padding","same","Stride",[2 2])
convolution2dLayer([3 3],64,"Padding","same")
batchNormalizationLayer
reluLayer
dropoutLayer(dropout_value)

maxPooling2dLayer([2 2],"Padding","same","Stride",[2 2])
convolution2dLayer([3 3],32,"Padding","same")
batchNormalizationLayer
reluLayer
dropoutLayer(dropout_value)

fullyConnectedLayer(12)
dropoutLayer(dropout_value)
softmaxLayer

classificationLayer
];

% analyze the network
analyzeNetwork(model_3)

options_3 = trainingOptions('adam', ...
'MiniBatchSize',128, ...
'MaxEpochs',30, ...
'InitialLearnRate',1e-3, ...
'ValidationData',{XValidation,YValidation},...
'Verbose',true, ...
'Plots','training-progress',...
'Plots','training-progress',...
'ExecutionEnvironment','gpu');
% Train the network using the training data

net_3 = trainNetwork(XTrain3,YTrain3,model_3,options_3);
[YPred3,probs3] = classify(net_3,XValidation);

Prediction = [YPred1,YPred2,YPred3];
pred = mode(Prediction,2);
accuracy = 100*mean(pred == YValidation); % accuracy
display(['Validation Accuracy: ' num2str(accuracy) '%']);
accuracy_1=100*mean(YPred1 == YValidation);
accuracy_2=100*mean(YPred2 == YValidation);
accuracy_3=100*mean(YPred3 == YValidation);
% plot confusion matrix
plotconfusion(YValidation,pred)

```

Results for 3 models are given below

Initializing input data normalization.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:03	7.03%	8.97%	3.0172	3.1491	0.0010
4	50	00:00:12	78.12%	48.59%	0.7157	1.6216	0.0010
7	100	00:00:19	87.50%	55.25%	0.4039	1.5681	0.0010
10	150	00:00:28	92.97%	56.87%	0.2461	1.5184	0.0010
14	200	00:00:36	92.19%	58.92%	0.1966	1.5005	0.0010
17	250	00:00:44	92.97%	59.01%	0.1911	1.5909	0.0010
20	300	00:00:52	92.97%	61.14%	0.1719	1.5523	0.0010
24	350	00:01:00	96.88%	60.97%	0.0821	1.5859	0.0010
27	400	00:01:07	93.75%	62.25%	0.1599	1.6026	0.0010
30	450	00:01:14	93.75%	62.34%	0.1092	1.6287	0.0010

Training finished: Max epochs completed.

Initializing input data normalization.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:03	7.03%	13.75%	3.0311	7.5577	0.0010
4	50	00:00:14	82.03%	40.56%	0.6032	1.9799	0.0010
7	100	00:00:24	96.88%	45.94%	0.1698	1.8827	0.0010
10	150	00:00:35	78.12%	43.21%	0.5948	2.4446	0.0010
14	200	00:00:45	100.00%	49.70%	0.0107	1.8915	0.0010
17	250	00:00:56	100.00%	51.49%	0.0047	2.0195	0.0010
20	300	00:01:07	100.00%	51.41%	0.0031	1.9772	0.0010
24	350	00:01:15	100.00%	50.73%	0.0031	2.0719	0.0010
27	400	00:01:25	100.00%	51.75%	0.0007	2.0392	0.0010
30	450	00:01:36	100.00%	52.69%	0.0011	2.1622	0.0010

Training finished: Max epochs completed.

Initializing input data normalization.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:03	15.62%	12.72%	3.0015	2.7083	0.0010
4	50	00:00:18	64.84%	47.31%	1.1124	1.6273	0.0010
7	100	00:00:32	83.59%	58.92%	0.6396	1.2576	0.0010
10	150	00:00:46	88.28%	63.19%	0.3658	1.1584	0.0010
14	200	00:01:00	92.19%	66.01%	0.2369	1.1133	0.0010
17	250	00:01:13	93.75%	67.21%	0.1889	1.1106	0.0010
20	300	00:01:30	91.41%	69.60%	0.1706	1.1198	0.0010
24	350	00:01:43	93.75%	70.28%	0.1431	1.1095	0.0010
27	400	00:01:57	96.09%	70.88%	0.1341	1.0800	0.0010
30	450	00:02:12	93.75%	71.73%	0.1481	1.1486	0.0010

Training finished: Max epochs completed.

Validation Accuracy: 63.1939%

Figure 9 Figure showing the validation accuracy for 3 different layers

The accuracy for three models has been calculated individually and also mode of the 3 values has been taken to calculate the overall accuracy

```
accuracy_1 =

60.3757

>> accuracy_2

accuracy_2 =

62.8523

>> accuracy_3

accuracy_3 =

66.0120

>> accuracy

accuracy =

67.0367
```

Figure 10 Accuracy for different models and overall accuracy

		Confusion Matrix												
Output Class	down	73 6.2%	8 0.7%	0 0.0%	10 0.9%	0 0.0%	5 0.4%	3 0.3%	4 0.3%	2 0.2%	2 0.2%	8 0.7%	0 0.0%	63.5% 36.5%
	go	9 0.8%	60 5.1%	2 0.2%	15 1.3%	1 0.1%	2 0.2%	5 0.4%	4 0.3%	1 0.1%	0 0.0%	2 0.2%	0 0.0%	59.4% 40.6%
	left	0 0.0%	0 0.0%	66 5.6%	0 0.0%	1 0.1%	0 0.0%	5 0.4%	3 0.3%	4 0.3%	7 0.6%	1 0.1%	0 0.0%	75.9% 24.1%
	no	7 0.6%	20 1.7%	1 0.1%	63 5.4%	1 0.1%	2 0.2%	0 0.0%	0 0.0%	4 0.3%	2 0.2%	4 0.3%	0 0.0%	60.6% 39.4%
	off	0 0.0%	1 0.1%	0 0.0%	0 0.0%	72 6.1%	4 0.3%	1 0.1%	5 0.4%	11 0.9%	0 0.0%	0 0.0%	0 0.0%	76.6% 23.4%
	on	4 0.3%	0 0.0%	1 0.1%	0 0.0%	8 0.7%	80 6.8%	2 0.2%	0 0.0%	3 0.3%	0 0.0%	23 2.0%	0 0.0%	66.1% 33.9%
	right	0 0.0%	0 0.0%	8 0.7%	0 0.0%	0 0.0%	0 0.0%	75 6.4%	0 0.0%	3 0.3%	1 0.1%	49 4.2%	0 0.0%	55.1% 44.9%
	stop	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.1%	0 0.0%	0 0.0%	60 5.1%	5 0.4%	0 0.0%	0 0.0%	0 0.0%	90.9% 9.1%
	up	0 0.0%	1 0.1%	4 0.3%	0 0.0%	5 0.4%	0 0.0%	2 0.2%	6 0.5%	47 4.0%	1 0.1%	0 0.0%	0 0.0%	71.2% 28.8%
	yes	0 0.0%	1 0.1%	4 0.3%	0 0.0%	0 0.0%	1 0.1%	0 0.0%	0 0.0%	0 0.0%	81 6.9%	0 0.0%	0 0.0%	93.1% 6.9%
	unknown	1 0.1%	0 0.0%	6 0.5%	0 0.0%	1 0.1%	0 0.0%	1 0.1%	0 0.0%	1 0.1%	1 0.1%	4 0.3%	0 0.0%	26.7% 73.3%
	background	7 0.6%	10 0.9%	9 0.8%	13 1.1%	11 0.9%	7 0.6%	7 0.6%	19 1.6%	20 1.7%	6 0.5%	10 0.9%	60 5.1%	33.5% 66.5%
		72.3% 27.7%	59.4% 40.6%	65.3% 34.7%	62.4% 37.6%	71.3% 28.7%	79.2% 20.8%	74.3% 25.7%	59.4% 40.6%	46.5% 53.5%	80.2% 19.8%	4.0% 96.0%	100% 0.0%	63.3% 36.7%
		down	go	left	no	off	on	right	stop	up	yes	unknown	background	
		Target Class												

Figure 11 Confusion matrix

3.2 Add five or more additional words to dataset

The objective of this task is to add five or more words to dataset. This is accomplished by following steps:

- The code of the dataPreProcess file has been altered.
- Model used in Task 1 has been again implemented for this task.
- 4 CNN layers have been used with 8 , 16 , 32 & 64 filters
- A dropout layer has been used after each CNN layer in order to reduce the variance.

The code has been given below:

```
clear

% define the random number seed for repeatable results
rng(1,'twister');

% define the folder for the speech dataset
datasetFolder = 'C:\Users\tiwar\MATLAB Drive\coursework1\speechDataReduced';

% Create an audioDatastore that points to the data set
ads = audioDatastore(datasetFolder, ...
    'IncludeSubfolders',true, ...
    'FileExtensions','.wav', ...
    'LabelSource','foldernames');

% define a subset of command words to recognise from the full data set
commands =
categorical(["yes","no","up","down","left","right","on","off","stop","go","bird","c
at","dog","five","four","three","wow"]);

% define an index into command words and unknown words
isCommand = ismember(ads.Labels,commands);
isUnknown = ~ismember(ads.Labels,[commands,"_background_noise_"]);

% specify the fraction of unknown words to include - labeling words that
% are not commands as |unknown| creates a group of words that approximates
% the distribution of all words other than the commands. The network uses
% this group to learn the difference between commands and all other words.
includeFraction = 0.2;
mask = rand(numel(ads.Labels),1) < includeFraction;
isUnknown = isUnknown & mask;
ads.Labels(isUnknown) = categorical("unknown");

% create a new data store of only command words and unknown words
adsSubset = subset(ads,isCommand|isUnknown);

% count the number of instances of each word
countEachLabel(adsSubset)

% define split proportion for training, validation and test data
p1 = 0.6; % training data proportion
p2 = 0.4; % validation data proportion
[adsTrain,adsValidation] = splitEachLabel(adsSubset,p1);

% reduce the dataset to speed up training
numUniqueLabels = numel(unique(adsTrain.Labels));
nReduce = 4; % Reduce the dataset by a factor of nReduce
adsTrain = splitEachLabel(adsTrain,round(numel(adsTrain.Files) / numUniqueLabels /
nReduce));
adsValidation = splitEachLabel(adsValidation,round(numel(adsValidation.Files) /
numUniqueLabels / nReduce));
```

```

% spectrogram parameters
fs = 16e3; % sample rate of the data set
segmentDuration = 1; % duration of each speech clip (in seconds)
frameDuration = 0.025; % duration of each frame for spectrum calculation
hopDuration = 0.010; % time step between each spectrum

segmentSamples = round(segmentDuration*fs); % number of segment samples
frameSamples = round(frameDuration*fs); % number of frame samples
hopSamples = round(hopDuration*fs); % number of hop samples
overlapSamples = frameSamples - hopSamples; % number of overlap samples

FFTLenght = 512; % number of points in the FFT
numBands = 50; % number of filters in the auditory spectrogram

% extract audio features using spectrogram - specifically bark spectrum
afe = audioFeatureExtractor( ...
    'SampleRate',fs, ...
    'FFTLenght',FFTLenght, ...
    'Window',hann(frameSamples,'periodic'), ...
    'OverlapLength',overlapSamples, ...
    'barkSpectrum',true);
setExtractorParams(afe,'barkSpectrum','NumBands',numBands);

% apply zero-padding to the audio signal so they are a consistent length of 1 sec
x = read(adsTrain);
numSamples = size(x,1);
numToPadFront = floor( (segmentSamples - numSamples)/2);
numToPadBack = ceil( (segmentSamples - numSamples)/2 );
xPadded = [zeros(numToPadFront,1,'like',x);x;zeros(numToPadBack,1,'like',x)];

% extract audio features - the output is a Bark spectrum with time across rows
features = extract(afe,xPadded);
[numHops,numFeatures] = size(features);

% determine the denormalization factor to apply for each signal
unNorm = 2/(sum(afe.Window)^2);

% Training data: read from datastore, zero-pad, extract spectrogram features
subds = partition(adsTrain,1,1);
XTrain = zeros(numHops,numBands,1,numel(subds.Files));
for idx = 1:numel(subds.Files)
    x = read(subds);
    xPadded = [zeros(floor((segmentSamples-
size(x,1))/2),1);x;zeros(ceil((segmentSamples-size(x,1))/2),1)];
    XTrain(:, :, :, idx) = extract(afe,xPadded);
end
XTrainC{1} = XTrain;
XTrain = cat(4,XTrainC{:});

% extract parameters from training data
[numHops,numBands,numChannels,numSpec] = size(XTrain);

% Scale the features by the window power and then take the log (with small offset)
XTrain = XTrain/unNorm;
epsil = 1e-6;
XTrain = log10(XTrain + epsil);

% Validation data: read from datastore, zero-pad, extract spectrogram features
subds = partition(adsValidation,1,1);
XValidation = zeros(numHops,numBands,1,numel(subds.Files));
for idx = 1:numel(subds.Files)
    x = read(subds);
    xPadded = [zeros(floor((segmentSamples-
size(x,1))/2),1);x;zeros(ceil((segmentSamples-size(x,1))/2),1)];
    XValidation(:, :, :, idx) = extract(afe,xPadded);
end
XValidationC{1} = XValidation;
XValidation = cat(4,XValidationC{:});

```

```

XValidation = XValidation/unNorm;
XValidation = log10(XValidation + epsilon);

% Isolate the train and validation labels. Remove empty categories.
YTrain = removecats(adsTrain.Labels);
YValidation = removecats(adsValidation.Labels);

% Visualize Data - plot waveforms and play clips for a few examples
specMin = min(XTrain,[],'all');
specMax = max(XTrain,[],'all');
idx = randperm(numel(adsTrain.Files),3);
figure('Units','normalized','Position',[0.2 0.2 0.6 0.6]);
for i = 1:3
    [x,fs] = audioread(adsTrain.Files{idx(i)});
    subplot(2,3,i)
    plot(x)
    axis tight
    title(string(adsTrain.Labels{idx(i)}))
    xlabel('time')

    subplot(2,3,i+3)
    spect = (XTrain(:, :, 1, idx(i)))';
    pcolor(spect)
    caxis([specMin specMax])
    shading flat

    sound(x,fs)
    xlabel('time')
    ylabel('frequency')
    pause(2)
end

% Use the audio files in the |_background_noise_|_ folder to create samples
% of background noise
adsBkg = subset(ads,ads.Labels=="_background_noise_");
numBkgClips = 400;

numBkgFiles = numel(adsBkg.Files);
numClipsPerFile = histcounts(1:numBkgClips,linspace(1,numBkgClips,numBkgFiles+1));
Xbkg = zeros(size(XTrain,1),size(XTrain,2),1,numBkgClips,'single');
bkgAll = readall(adsBkg);
ind = 1;

% rescale audio clip with value sampled from volumeRange
volumeRange = log10([1e-4,1]);
for count = 1:numBkgFiles
    bkg = bkgAll{count};
    idxStart = randi(numel(bkg)-fs,numClipsPerFile(count),1);
    idxEnd = idxStart+fs-1;
    gain = 10.^((volumeRange(2)-volumeRange(1))*rand(numClipsPerFile(count),1) +
    volumeRange(1));
    for j = 1:numClipsPerFile(count)

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

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

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

        if mod(ind,1000)==0
            disp("Processed " + string(ind) + " background clips out of " +
            string(numBkgClips))
        end
        ind = ind + 1;
    end
end
Xbkg = Xbkg/unNorm;

```

```

Xbkg = log10(Xbkg + epsil);

% Split spectrograms of background noise between training and validation data
numTrainBkg = floor(0.85*numBkgClips);
numValidationBkg = floor(0.15*numBkgClips);
XTrain(:, :, :, end+1:end+numTrainBkg) = Xbkg(:, :, :, 1:numTrainBkg);
YTrain(end+1:end+numTrainBkg) = "background";
XValidation(:, :, :, end+1:end+numValidationBkg) = Xbkg(:, :, :, numTrainBkg+1:end);
YValidation(end+1:end+numValidationBkg) = "background";

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

figure('Units','normalized','Position',[0.2 0.2 0.5 0.5])
subplot(2,1,1)
histogram(YTrain)
title("Training Label Distribution")
subplot(2,1,2)
histogram(YValidation)
title("Validation Label Distribution")

% You now have training and validation data in the variables XTrain, YTrain,
% XValidation and YValidation in the form of images for X and word labels for Y.
%
% suggest at this point you save XTrain, YTrain, XValidation and
% YValidation for future fast loading i.e.
save ACS61011projectData_modified XTrain YTrain XValidation YValidation

%=====Code for implementing model=====
load ACS61011projectData_modified.mat
dropout_value=0.1;
model = [

imageInputLayer([98 50 1])
convolution2dLayer([3 3],8,"Padding","same")
batchNormalizationLayer
reluLayer
dropoutLayer(dropout_value)

maxPooling2dLayer([2 2],"Padding","same","Stride",[2 2])
convolution2dLayer([3 3],16,"Padding","same")
batchNormalizationLayer
reluLayer
dropoutLayer(dropout_value)
maxPooling2dLayer([2 2],"Padding","same","Stride",[2 2])
convolution2dLayer([3 3],32,"Padding","same")
batchNormalizationLayer
reluLayer
dropoutLayer(dropout_value)

maxPooling2dLayer([2 2],"Padding","same","Stride",[2 2])
convolution2dLayer([3 3],64,"Padding","same")
batchNormalizationLayer
reluLayer
dropoutLayer(dropout_value)
fullyConnectedLayer(19)
dropoutLayer(dropout_value)
softmaxLayer
classificationLayer
];
% analyze the network
analyzeNetwork(model)
options = trainingOptions('adam', ...
'MiniBatchSize',128, ...
'MaxEpochs',30, ...
'InitialLearnRate',1e-3, ...

```

```

'ValidationData',{XValidation,YValidation},...
'Verbose',true, ...
'Plots','training-progress',...
'Plots','training-progress',...
'ExecutionEnvironment','gpu');
% Train the network using the training data
net = trainNetwork(XTrain,YTrain,model,options);
% Classify the validation images using the trained network
[YPred,probs] = classify(net,XValidation);
accuracy = 100*mean(YPred == YValidation); % accuracy
disp(['Validation Accuracy: ' num2str(accuracy) '%']);
% plot confusion matrix
plotconfusion(YValidation,YPred)

```

Initializing input data normalization.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:03	3.12%	7.55%	3.6443	3.7131	0.0010
3	50	00:00:08	52.34%	42.35%	1.5906	1.9549	0.0010
5	100	00:00:14	81.25%	53.91%	0.8218	1.6078	0.0010
7	150	00:00:25	81.25%	58.63%	0.6285	1.5079	0.0010
9	200	00:00:38	91.41%	62.21%	0.3896	1.4264	0.0010
11	250	00:00:50	93.75%	64.33%	0.2561	1.3435	0.0010
14	300	00:01:04	89.84%	65.36%	0.3269	1.3779	0.0010
16	350	00:01:14	88.28%	66.50%	0.2459	1.3512	0.0010
18	400	00:01:28	86.72%	66.34%	0.3260	1.4110	0.0010
20	450	00:01:37	91.41%	66.18%	0.2160	1.5024	0.0010
22	500	00:01:48	90.62%	67.43%	0.1784	1.4967	0.0010
24	550	00:02:01	95.31%	69.27%	0.1407	1.4998	0.0010
27	600	00:02:15	91.41%	68.30%	0.1684	1.5937	0.0010
29	650	00:02:30	95.31%	67.43%	0.2456	1.6730	0.0010
30	690	00:02:41	90.62%	68.13%	0.1944	1.4890	0.0010

Training finished: Max epochs completed.

Validation Accuracy: 66.2866%

Figure 12 Table for Validation

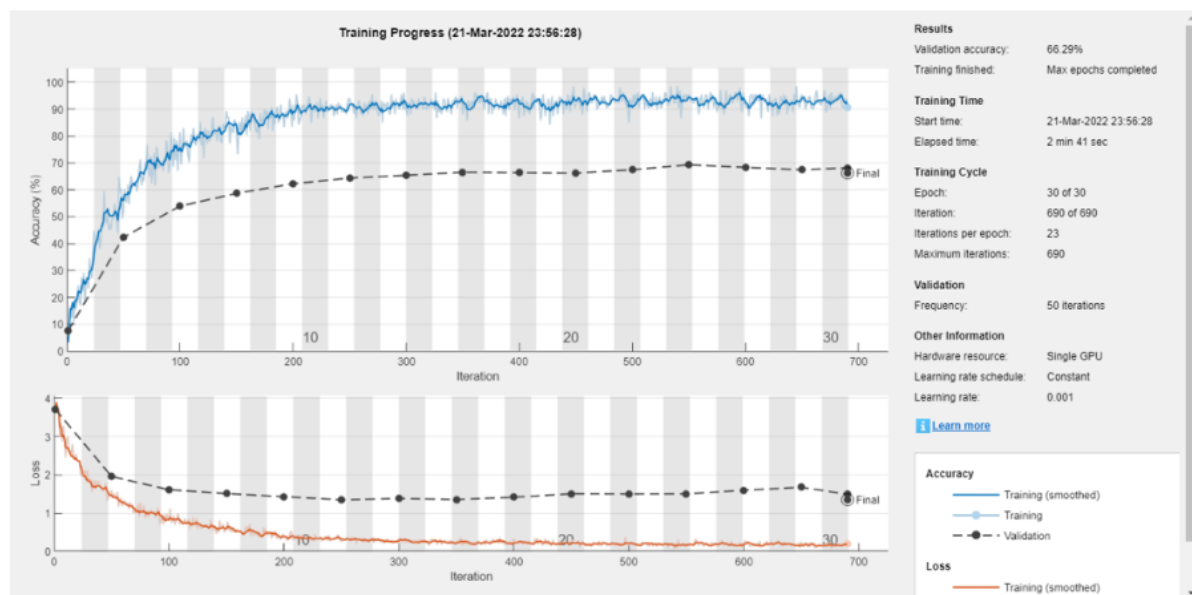


Figure 13 Training and validation curve for added speech data

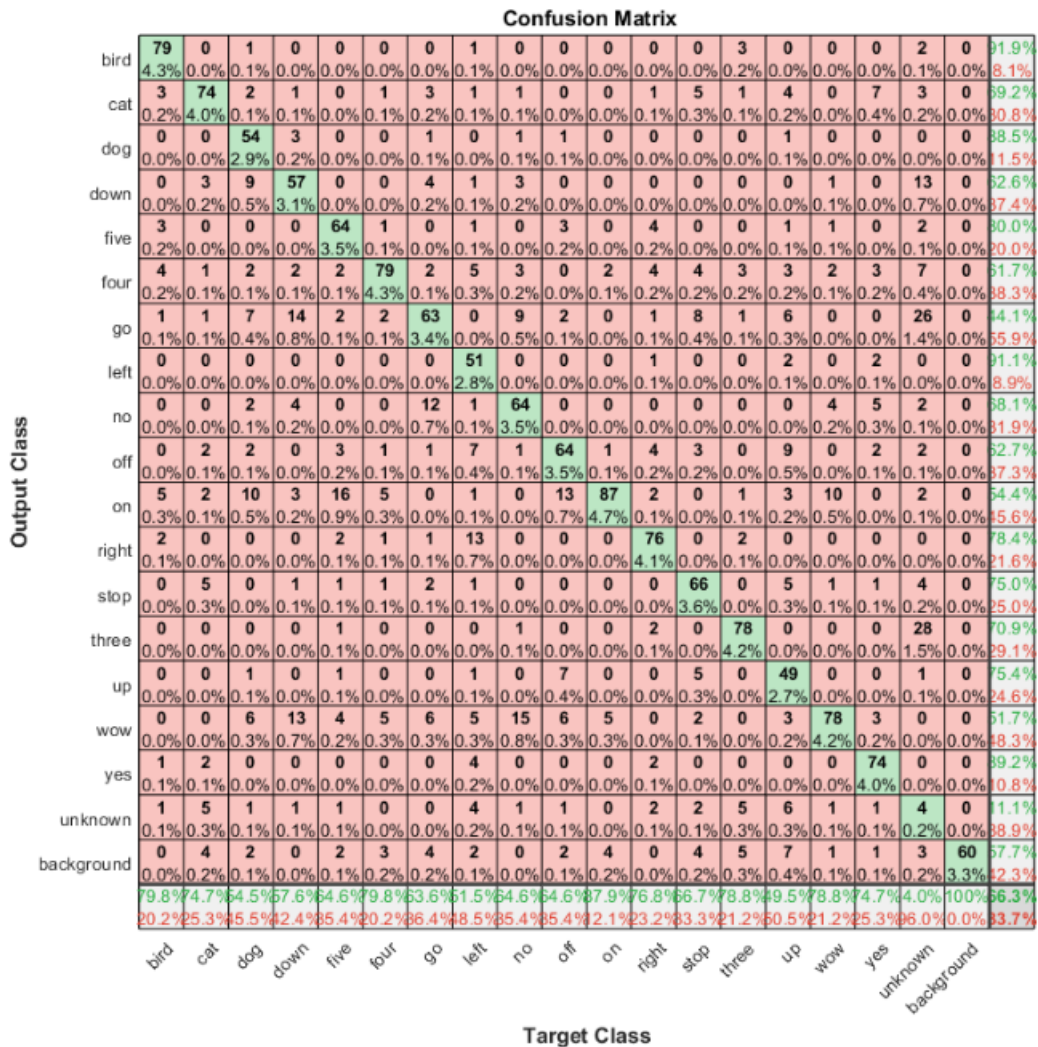


Figure 14 Confusion Matrix

Task 4

Three tasks mentioned in Task 3 have already been accomplished and their results have been shown with respective tasks.

Task 5: Implementing Bayesian Optimization

The objective of this task is to make more accurate model. Bayesian optimization has been implemented for the accomplishment of the above task.

- Bayesian optimization has been implemented to search for optimal value of following hyperparameters.
 - Number of Filters
 - Number of layers
 - Mini Batch Size
 - Learning Rate
 - L2 Regularization Parameter

- In order to iterate over different models, the createConvNetwork function has been used to build models dynamically create models for iteration.
- An extra dropout layer has been added in each dynamically created layer to reduce variance.

The code for performing Bayesian optimization is shown below:

```
load ACS61011projectData.mat

%=====choosing variables to optimize=====

optimVars = [
    optimizableVariable('nlayers',[1 5],'Type','integer')
    optimizableVariable('nfilters',[8 64],'Type','integer')
    optimizableVariable('MiniBatchSize',[64 256],'Type','integer')
    optimizableVariable('InitialLearnRate',[1e-2 1],'Transform','log')
    optimizableVariable('L2Regularization',[1e-10 1e-2],'Transform','log')
];

%=====Perform Beysian Optimization=====

ObjFcn = makeObjFcn(XTrain,YTrain,XValidation,YValidation);
BayesObject = bayesopt(ObjFcn,optimVars, ...
    'MaxTime',14*60*60, ...
    'IsObjectiveDeterministic',false, ...
    'UseParallel',false);

%=====Evaluate Final Network=====

bestIdx = BayesObject.IndexOfMinimumTrace(end);
fileName = BayesObject.UserDataTrace{bestIdx};
savedStruct = load(fileName);
valError = savedStruct.valError;

[YPredicted,probs] = classify(savedStruct.trainedNet,XValidation);
accuracy = 100*mean(YPredicted == YValidation); % accuracy
disp(['Validation Accuracy: ' num2str(accuracy) '%']);

% plot confusion matrix
plotconfusion(YValidation,YPredicted)
```

Make object function is used to create CNN network for optimization testing:

```
function ObjFcn = makeObjFcn(XTrain,YTrain,XValidation,YValidation)
ObjFcn = @valErrorFun;
function [valError,cons,fileName] = valErrorFun(optVars)
    filterSize = 3; % filter size
    dropout_value=0.1;
    inputSize = [98 50 1]; % input image size
    numClasses = 12;
lgraph =
createConvNetwork(optVars.nlayers,optVars.nfilters,filterSize,inputSize,numClasses)
;

% analyze the network
analyzeNetwork(lgraph)
options = trainingOptions('adam', ...
'MiniBatchSize',optVars.MiniBatchSize, ...
'MaxEpochs',30, ...
'InitialLearnRate',optVars.InitialLearnRate, ...
'ValidationData',{XValidation,YValidation},...
'L2Regularization',optVars.L2Regularization, ...
'Verbose',true, ...
```

```

'Plots','training-progress',...
'ExecutionEnvironment','gpu');
% Train the network using the training data

trainedNet= trainNetwork(XTrain,YTrain,lgraph,options);
YPredicted = classify(trainedNet,XValidation);
valError = 1 - mean(YPredicted == YValidation);
fileName = num2str(valError) + ".mat";
save(fileName,'trainedNet','valError','options')
cons = [];

    end
end

```

The summary of optimization results obtained after 30 iterations is shown below

```

Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 6406.0416 seconds
Total objective function evaluation time: 6387.2044

Best observed feasible point:
    nlayers    nfilters    MiniBatchSize    InitialLearnRate    L2Regularization
    -----    -
           5           45           184           0.010108           1.0943e-10

Observed objective function value = 0.19385
Estimated objective function value = 0.19635
Function evaluation time = 327.5514

Best estimated feasible point (according to models):
    nlayers    nfilters    MiniBatchSize    InitialLearnRate    L2Regularization
    -----    -
           5           45           184           0.010108           1.0943e-10

Estimated objective function value = 0.19635
Estimated function evaluation time = 289.4747

Validation Accuracy: 80.6149%

```

Figure 15 Optimization process summary

As seen in the above figure the best estimated feasible value for various parameters is given by Bayesian optimization.

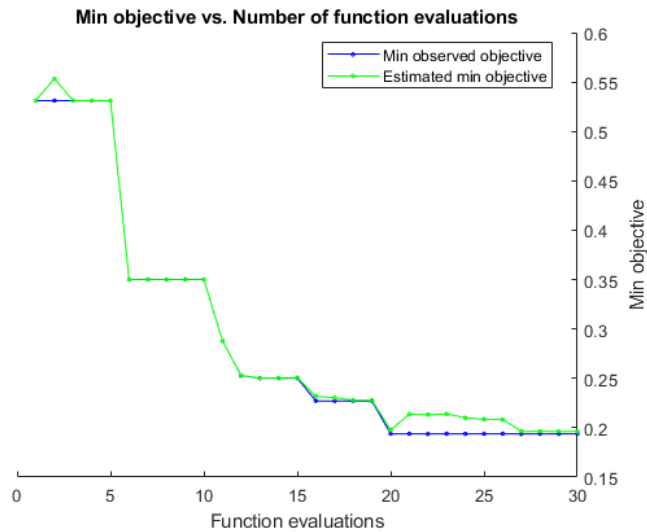


Figure 16 Minimum Objective vs Number of function evaluations.

Finally, the Confusion Matrix obtained for the above optimization problem is shown below:

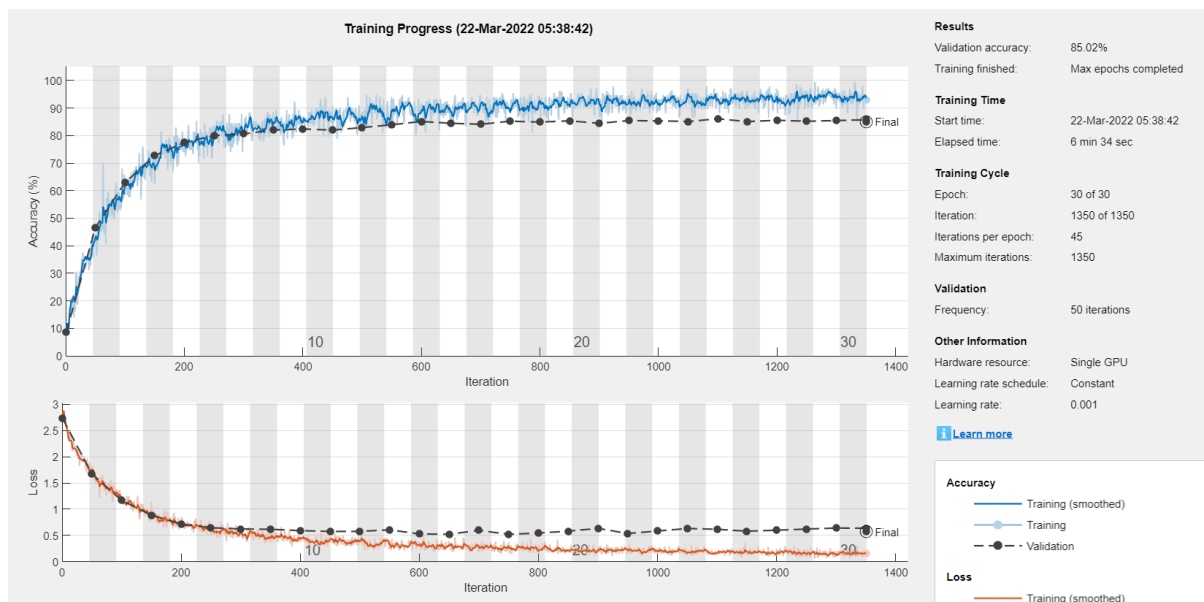
Output Class													
	down	go	left	no	off	on	right	stop	up	yes	unknown	background	
down	83	2	0	5	0	1	0	0	0	0	5	0	86.5%
go	10	87	0	8	0	2	0	3	2	0	2	0	83.5%
left	0	0	89	0	0	1	3	0	0	4	1	0	90.8%
no	3	8	7	83	0	0	1	0	0	1	5	0	92.2%
off	1	0	1	0	87	3	1	1	0	0	0	0	92.6%
on	0	0	0	0	2	86	0	0	0	0	27	0	94.8%
right	0	1	3	2	0	7	93	0	0	1	38	0	95.2%
stop	2	0	0	1	0	1	0	80	2	2	2	0	88.9%
up	0	1	4	0	9	0	0	9	90	0	0	0	89.9%
yes	0	0	0	0	0	1	0	0	0	89	0	0	91.1%
unknown	1	0	3	1	1	3	3	4	0	3	17	0	87.2%
background	1	2	1	0	2	3	1	4	6	1	4	60	90.6%
	82.2%	86.1%	88.1%	82.2%	86.1%	85.1%	82.1%	79.2%	89.1%	88.1%	86.8%	100%	80.6%
	17.8%	13.9%	11.9%	17.8%	13.9%	14.9%	17.9%	20.8%	10.9%	11.9%	13.2%	0.0%	19.4%

Conclusion:

The validation accuracy obtained is around 81 Percent which is not good enough and shows high variance. The best alternative to remove variance is to provide more data for training. We are using only Quarter of the whole data available as in the program given for data pre-processing the data is reduced by a factor of 4 which is shown in figure below:

```
% reduce the dataset to speed up training
numUniqueLabels = numel(unique(adsTrain.Labels));
nReduce = 4; % Reduce the dataset by a factor of nReduce
adsTrain = splitEachLabel(adsTrain,round(numel(adsTrain.Files) / numUniqueLabels / nReduce));
adsValidation = splitEachLabel(adsValidation,round(numel(adsValidation.Files) / numUniqueLabels / nReduce));
```

Hence, in order to provide more data for improving the validation accuracy we can change the value of nReduce and obtain more data. But this strategy has not been implemented in any of the above task to discover the true potential of our optimization algorithm however in order to verify the fact that including more data is the best alternative to reduce the variance the Model used in first task has been used with the dataset but this time during the pre-processing the value of nReduce was set to 1.5 and it has resulted in following optimization curve.



The validation accuracy obtained in this case as apparent is higher than the accuracy obtained by Bayesian Optimization that to with a simplistic model. This shows that using more data is the best alternative to reduce the variance and Bias can be reduced by making larger and deeper network.