Contents

Deep learning system for automatic speech recognition

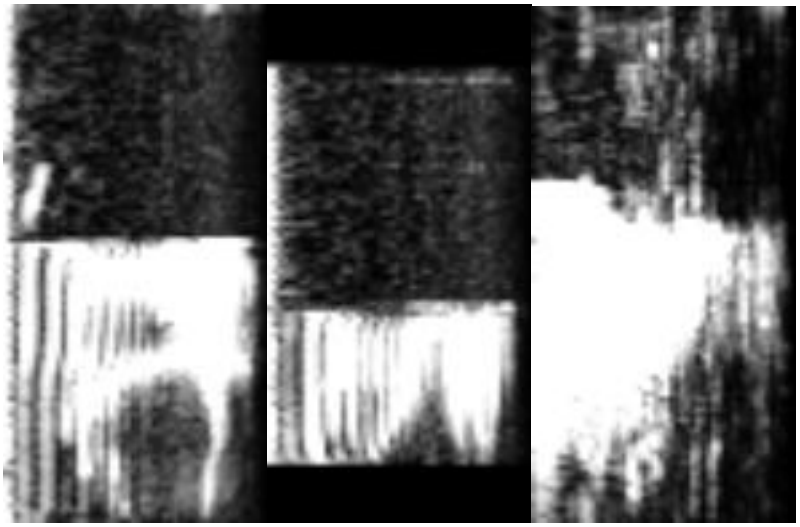
1. Introduction

Speech recognition is a fundamental task in natural language processing (NLP) and has become increasingly important with the rise of smart assistants and voice-controlled devices. Deep learning has shown great promise in improving the accuracy of speech recognition systems. This technical report presents an analysis of a deep learning-based approach to speech recognition, focusing on the design, implementation and evaluation of different neural network architectures and training strategies.

2. Methodology

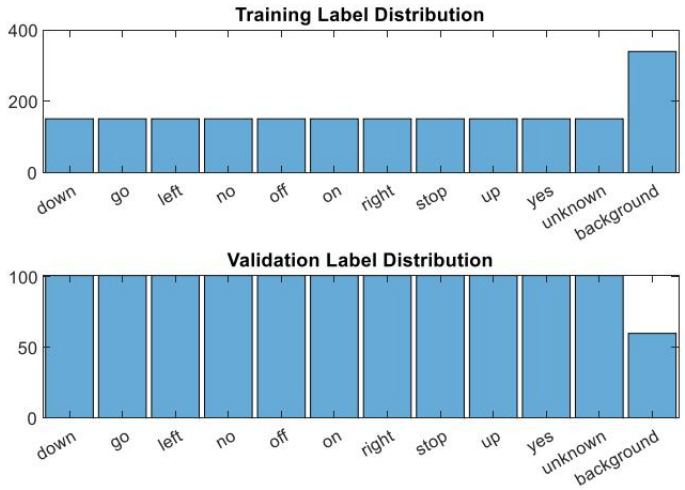
The study used a large dataset of speech recordings, including both clean and noisy speech. The dataset was divided into training, and validation sets.

The input data has been pre-processed into spectrograms of speech 'images' like this:



Down Go No

speech waveforms in the time-domain and corresponding speech image spectrogram that forms the input data. There are 12 word classes, including unknown class and background class, the distribution shown in the plots:



Different neural network architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformer networks, were trained using different training strategies, including supervised learning and transfer learning. For this study, I designed CNNs for speech recognition using Matlab for the following reasons:

(1) Local feature extraction: CNNs are able to extract local features from speech signals, which can capture the patterns and characteristics of speech sounds. This is achieved by convolving a set of filters over small segments of the input signal to detect specific features, such as phonemes or syllables.

(2) Translation invariance: CNNs are invariant to small shifts in the input signal, which can be useful in speech recognition where the same sound can be pronounced in slightly different ways. This property allows CNNs to capture the same feature regardless of its position in the input.

(3) Parameter sharing: CNNs share parameters across the entire network, reducing the number of trainable parameters required for the model. This makes CNNs more efficient to train and less prone to overfitting, which is especially important in speech recognition where large amounts of data are required.

(4) Hierarchical representation: CNNs are capable of learning hierarchical representations of speech signals, which can capture both low-level features (such as phonemes and syllables) and high-level features (such as words and phrases). This allows CNNs to model the complex relationships between different speech components and make accurate predictions.

The tasks accomplished in the technical report are:

• Design, implement and evaluate a basic deep learning classifier system to recognise spoken words from spectrogram image inputs and achieve an accuracy of >=60% on the validation data.

• Use a 2x2 grid search to perform hyperparameter tuning in the model of:

-the number of convolutional layers.

-the number of filters per layer.

• Reduce the computational complexity of the model using depthwise separable convolutions and 1x1 convolutions, and improve model generalisation.

3. Experiments and Results

3.1 Basic deep learning classifier system for speech recognition:

The model has 16 layers and 34.2k learnables:

• The input layer takes in an speech image with a size of 98 x 50 x 1.

• The first convolutional layer applies 16 filters to the input. each with a size of 3 x 3 pixels. The ReLU activation function is applied to the output of each filter, and batch normalize the data before next layer.

• The first max pooling layer reduces the size of the output from the first convolutional layer by applying a 2 x 2 window with stride 2 (i.e. the window moves 2 pixels at a time) and selecting the maximum value in each window.

• The second convolutional layer applies 32 filters to the output of the first max pooling layer, each with a size of 3 x 3 pixels and stride 2. The ReLU activation function is applied to the output of each filter, and batch normalize the data before next layer.

• The third convolutional layer applies 64 filters to the output of the second convolutional layer, each with a size of 3 x 3 pixels and stride 2. The ReLU activation function is applied to the output

of each filter, and batch normalize the data before next layer.

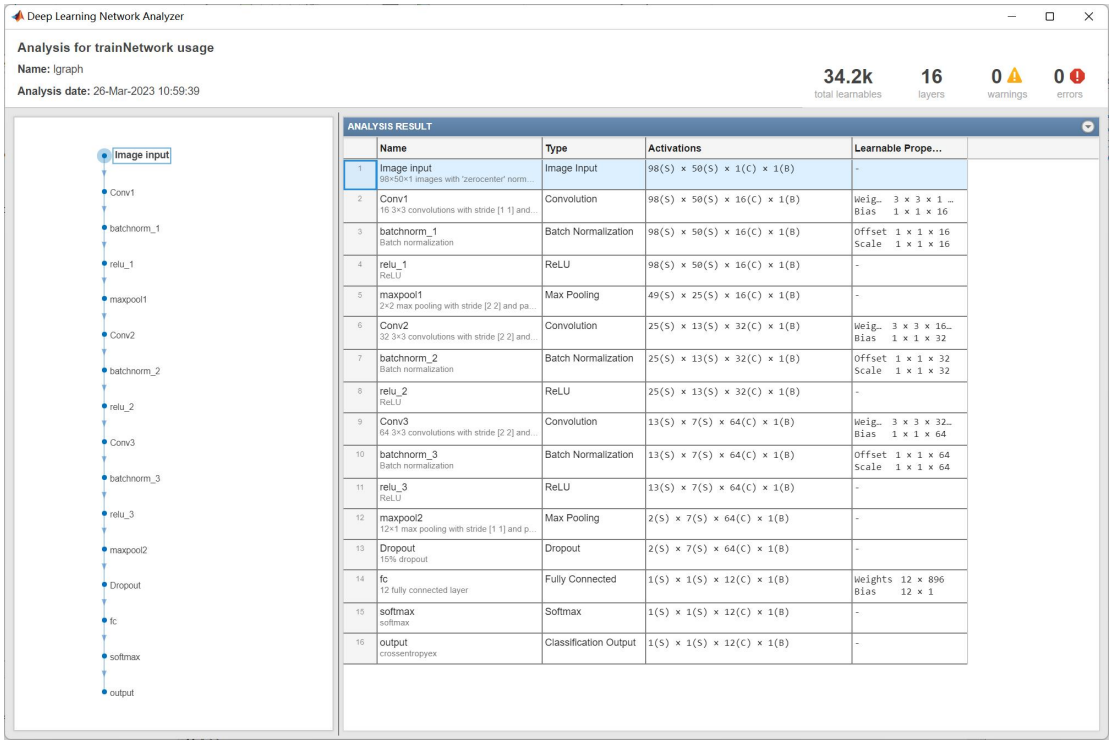
• The second max pooling layer reduces the size of the output from the third convolutional layer in the same way as the first max pooling layer.

• The dropout layer makes some neural function stop working to avoid overfitting.

• The fully connected layer has 12 units.

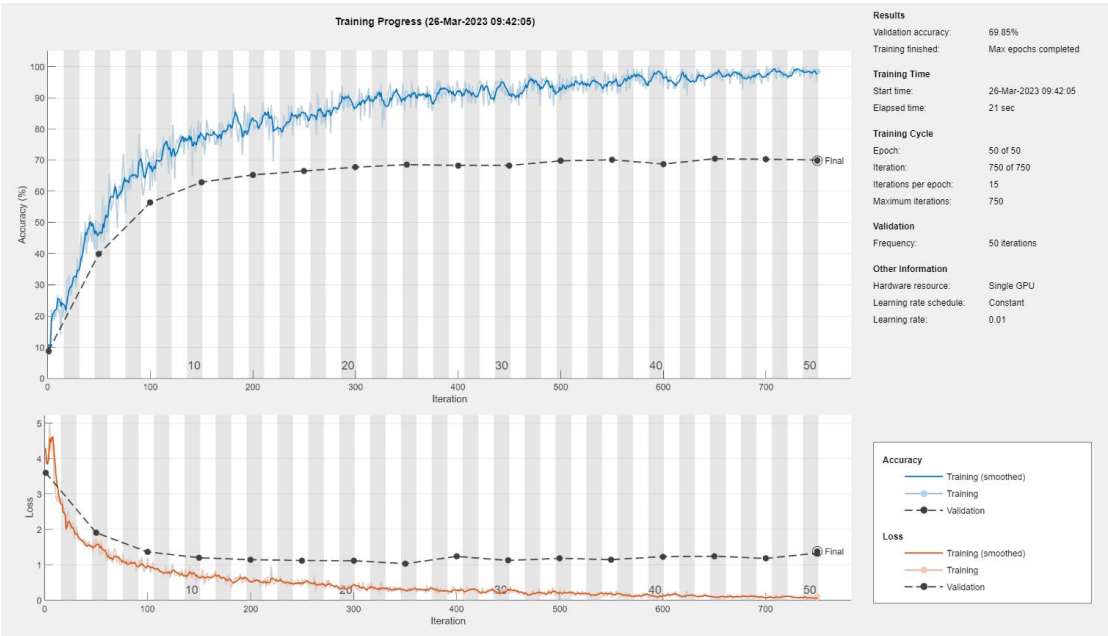
• The output layer has 12 units (assuming a 12-class classification problem) and uses a softmax activation function to produce a probability distribution over the classes.

Network structure:



	Name	Type	Activations	Learnable Prope...
1	Image input 98×50×1 images with 'zerocenter' norm...	Image Input	98(S) × 50(S) × 1(C) × 1(B)	-
2	Conv1 16 3×3 convolutions with stride [1 1] and...	Convolution	98(S) × 50(S) × 16(C) × 1(B)	Weig... 3 × 3 × 1 ... Bias 1 × 1 × 16
3	batchnorm_1 Batch normalization	Batch Normalization	98(S) × 50(S) × 16(C) × 1(B)	Offset 1 × 1 × 16 Scale 1 × 1 × 16
4	relu_1 ReLU	ReLU	98(S) × 50(S) × 16(C) × 1(B)	-
5	maxpool1 2×2 max pooling with stride [2 2] and pa...	Max Pooling	49(S) × 25(S) × 16(C) × 1(B)	-
6	Conv2 32 3×3 convolutions with stride [2 2] and...	Convolution	25(S) × 13(S) × 32(C) × 1(B)	Weig... 3 × 3 × 16... Bias 1 × 1 × 32
7	batchnorm_2 Batch normalization	Batch Normalization	25(S) × 13(S) × 32(C) × 1(B)	Offset 1 × 1 × 32 Scale 1 × 1 × 32
8	relu_2 ReLU	ReLU	25(S) × 13(S) × 32(C) × 1(B)	-
9	Conv3 64 3×3 convolutions with stride [2 2] and...	Convolution	13(S) × 7(S) × 64(C) × 1(B)	Weig... 3 × 3 × 32... Bias 1 × 1 × 64
10	batchnorm_3 Batch normalization	Batch Normalization	13(S) × 7(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64
11	relu_3 ReLU	ReLU	13(S) × 7(S) × 64(C) × 1(B)	-
12	maxpool2 12×1 max pooling with stride [1 1] and p...	Max Pooling	2(S) × 7(S) × 64(C) × 1(B)	-
13	Dropout 15% dropout	Dropout	2(S) × 7(S) × 64(C) × 1(B)	-
14	fc 12 fully connected layer	Fully Connected	1(S) × 1(S) × 12(C) × 1(B)	Weights 12 × 896 Bias 12 × 1
15	softmax softmax	Softmax	1(S) × 1(S) × 12(C) × 1(B)	-
16	output crossentropyex	Classification Output	1(S) × 1(S) × 12(C) × 1(B)	-

Training progress:

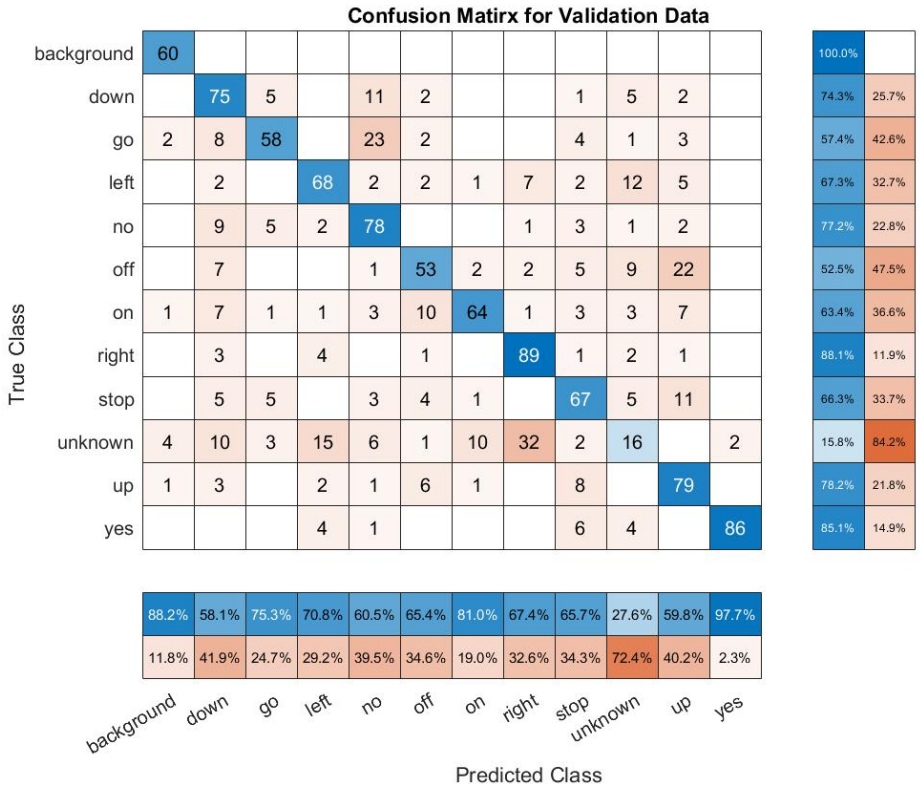


I trained this model on a single GPU like Nvidia RTX3060, some Hyperparameters are:

- Initial learning rate: 0.01
- Max epochs: 50
- Validation Frequency: 50
- Learning rate drop period: 20
- Learning rate drop factor: 0.1

The model achieved an accuracy of 69.85% on the validation data.

Confusion plot:



3.2 Grid search to perform hyperparameter tuninig in the model of:

- the number of layers.
- the number of filters per layer.

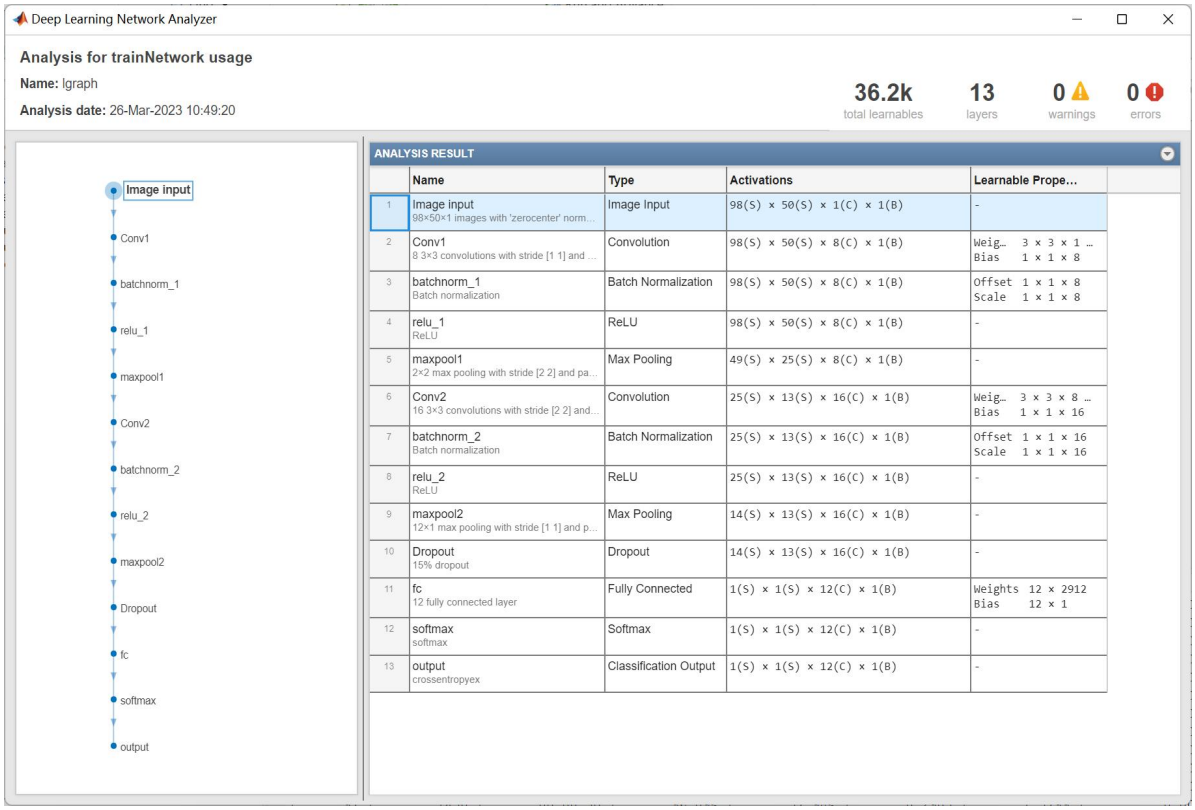
To avoid run into computational difficulties, I took 2x2 grid search as follows:

Num of conv layers: 2 Num of filters : 8, 16,⋯⋯	Num of conv layers: 2 Num of filters : 16, 32,⋯⋯
Num of conv layers: 3 Num of filters: 8, 16,⋯⋯	Num of conv layers: 3 Num of filters : 16, 32,⋯⋯

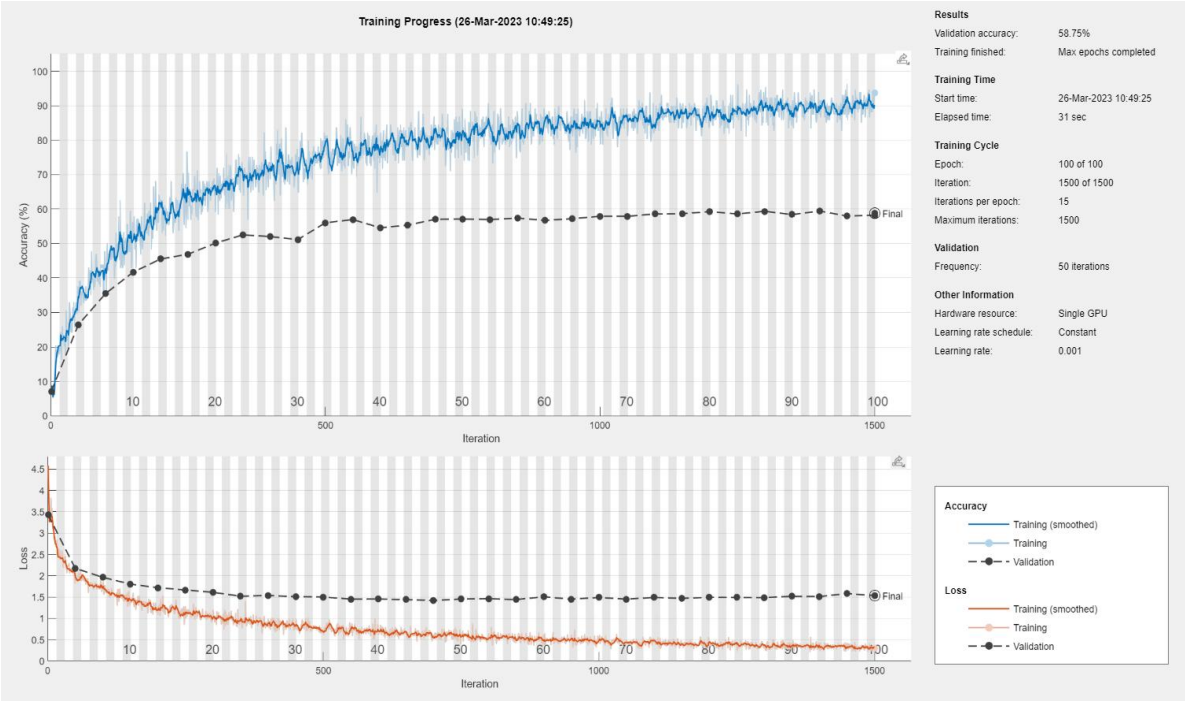
The number of convolution layers: 2

The number of filters:8, 16,……

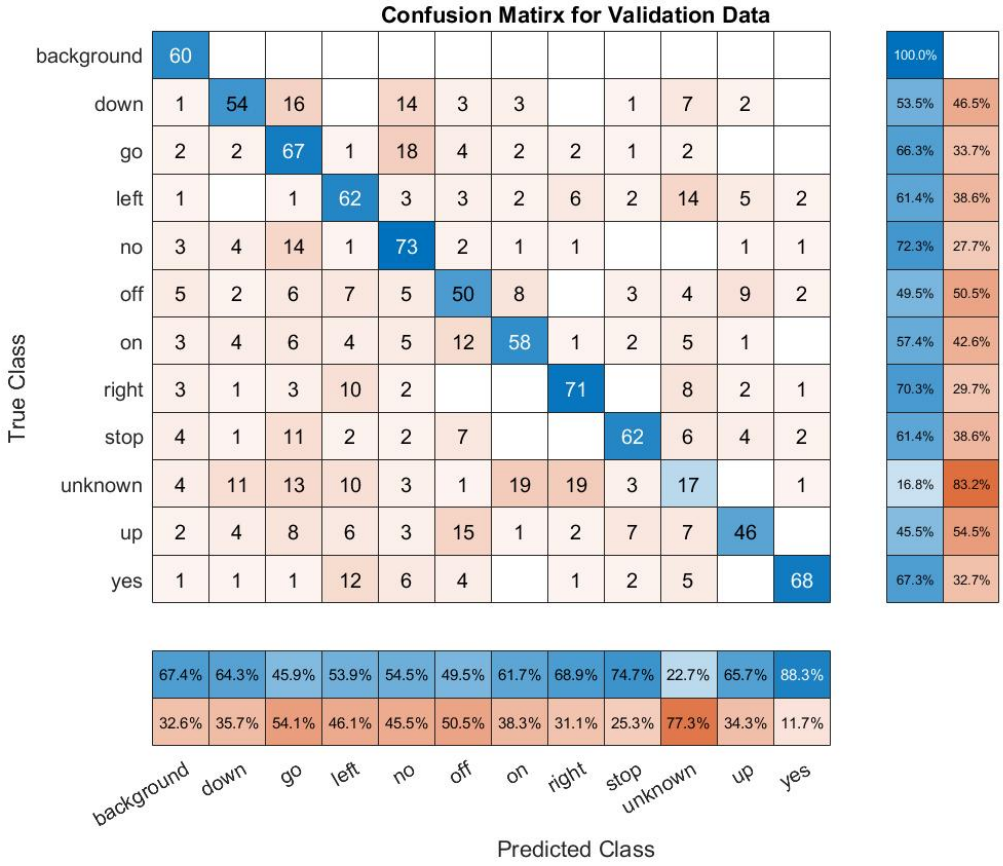
Network structure:



Training progress:



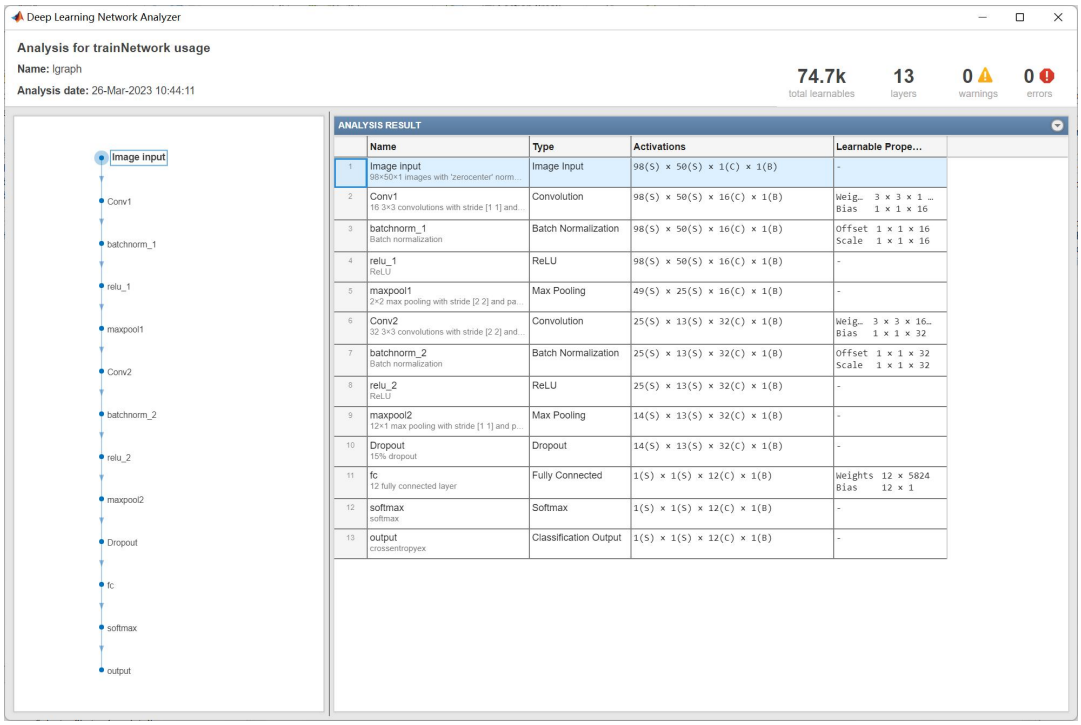
Confusion plot:



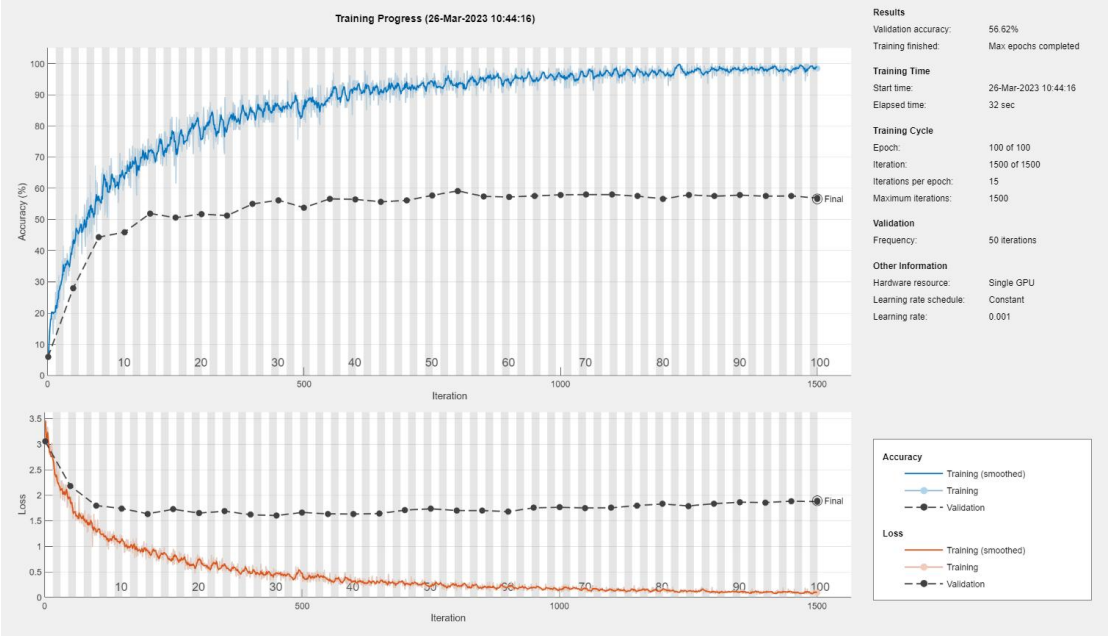
The number of convolution layers: 2

The number of filters:16,32,……

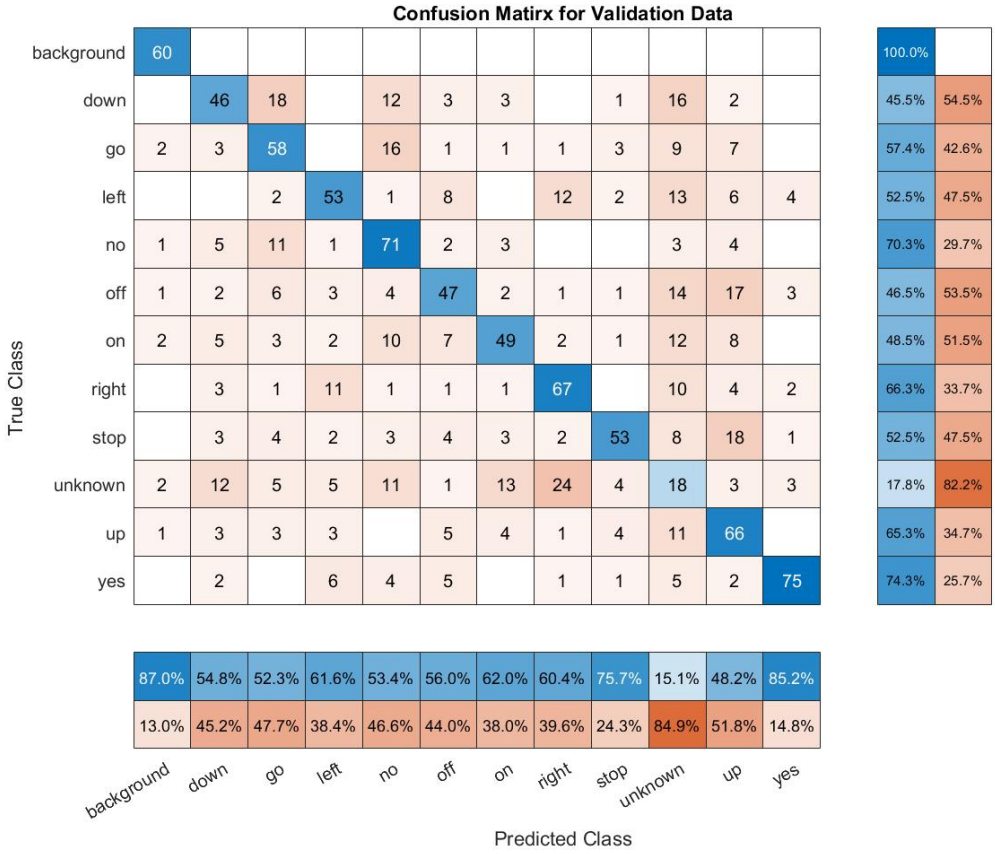
Network structure:



Training progress:



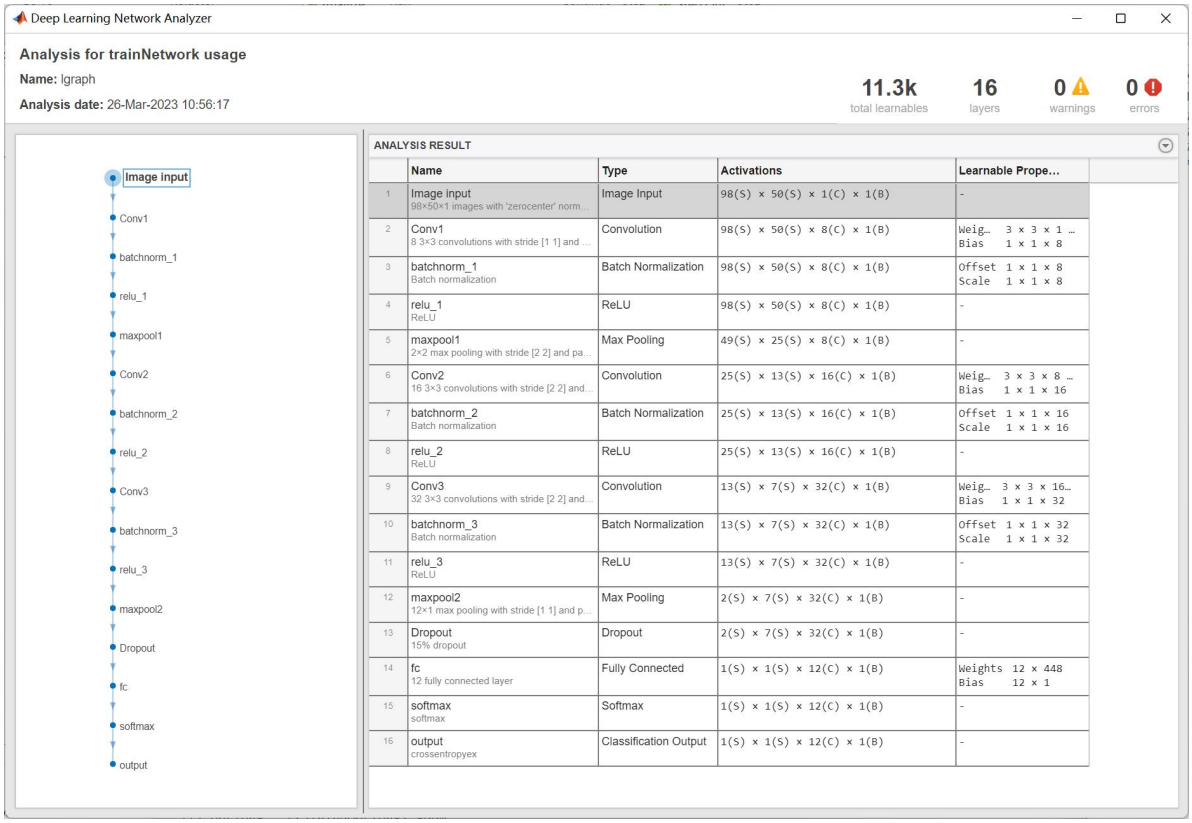
Confusion plot:



The number of convolution layers: 3

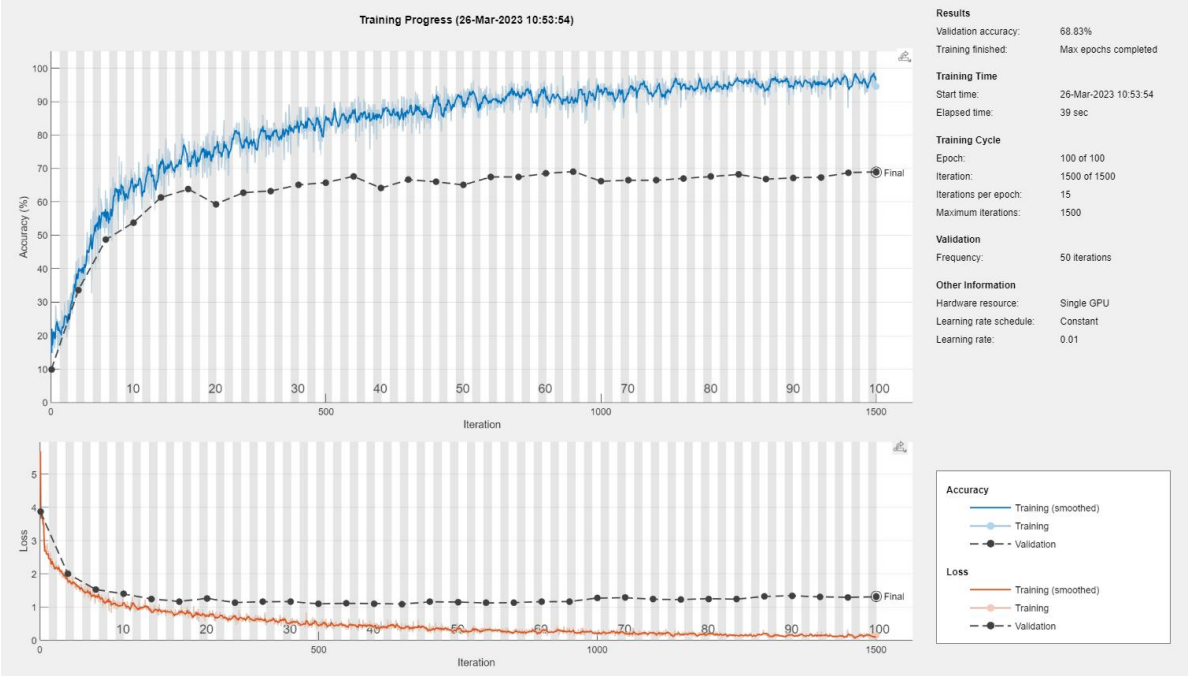
The number of filters:8,16,……

Network structure:

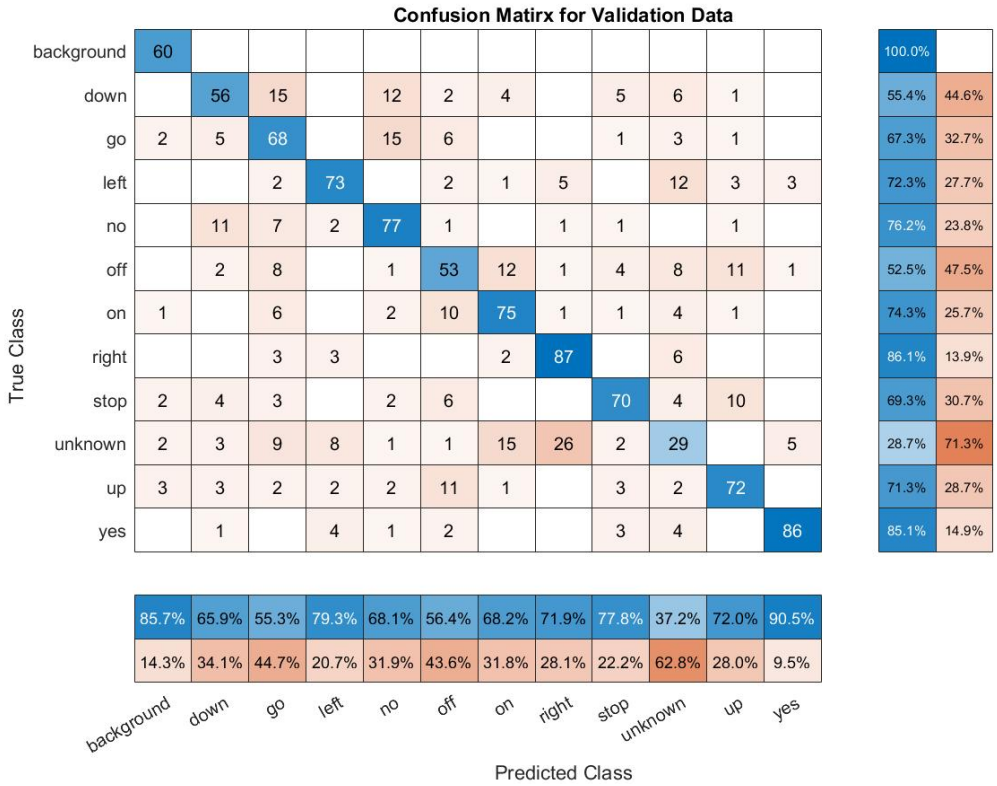


	Name	Type	Activations	Learnable Prope...
1	Image input 98×50×1 images with 'zerocenter' norm...	Image Input	98(S) × 50(S) × 1(C) × 1(B)	-
2	Conv1 8 3×3 convolutions with stride [1 1] and ...	Convolution	98(S) × 50(S) × 8(C) × 1(B)	Weig... 3 × 3 × 1 ... Bias 1 × 1 × 8
3	batchnorm_1 Batch normalization	Batch Normalization	98(S) × 50(S) × 8(C) × 1(B)	Offset 1 × 1 × 8 Scale 1 × 1 × 8
4	relu_1 ReLU	ReLU	98(S) × 50(S) × 8(C) × 1(B)	-
5	maxpool1 2×2 max pooling with stride [2 2] and pa...	Max Pooling	49(S) × 25(S) × 8(C) × 1(B)	-
6	Conv2 16 3×3 convolutions with stride [2 2] and...	Convolution	25(S) × 13(S) × 16(C) × 1(B)	Weig... 3 × 3 × 8 ... Bias 1 × 1 × 16
7	batchnorm_2 Batch normalization	Batch Normalization	25(S) × 13(S) × 16(C) × 1(B)	Offset 1 × 1 × 16 Scale 1 × 1 × 16
8	relu_2 ReLU	ReLU	25(S) × 13(S) × 16(C) × 1(B)	-
9	Conv3 32 3×3 convolutions with stride [2 2] and...	Convolution	13(S) × 7(S) × 32(C) × 1(B)	Weig... 3 × 3 × 16... Bias 1 × 1 × 32
10	batchnorm_3 Batch normalization	Batch Normalization	13(S) × 7(S) × 32(C) × 1(B)	Offset 1 × 1 × 32 Scale 1 × 1 × 32
11	relu_3 ReLU	ReLU	13(S) × 7(S) × 32(C) × 1(B)	-
12	maxpool2 12×1 max pooling with stride [1 1] and p...	Max Pooling	2(S) × 7(S) × 32(C) × 1(B)	-
13	Dropout 15% dropout	Dropout	2(S) × 7(S) × 32(C) × 1(B)	-
14	fc 12 fully connected layer	Fully Connected	1(S) × 1(S) × 12(C) × 1(B)	Weights 12 × 448 Bias 12 × 1
15	softmax softmax	Softmax	1(S) × 1(S) × 12(C) × 1(B)	-
16	output crossentropyex	Classification Output	1(S) × 1(S) × 12(C) × 1(B)	-

Training progress:



Confusion plot:

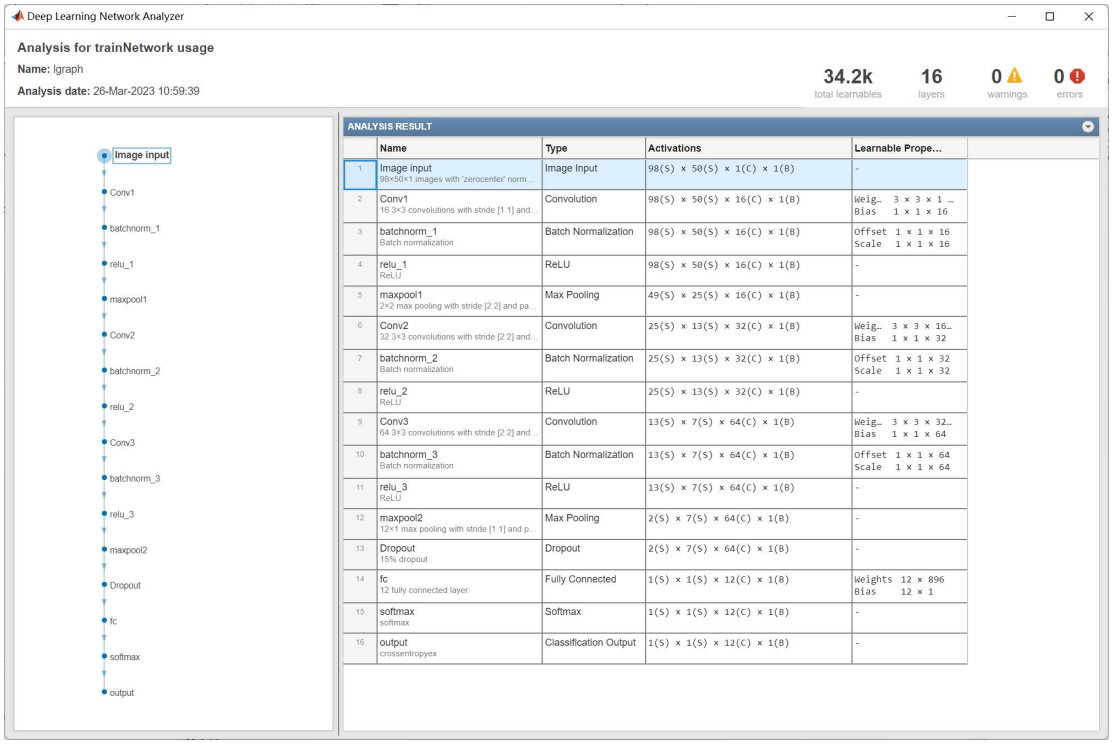


The number of convolution layers: 3
The number of filters of first convolution layer:16:
The details of this model are mentioned above(3.1).

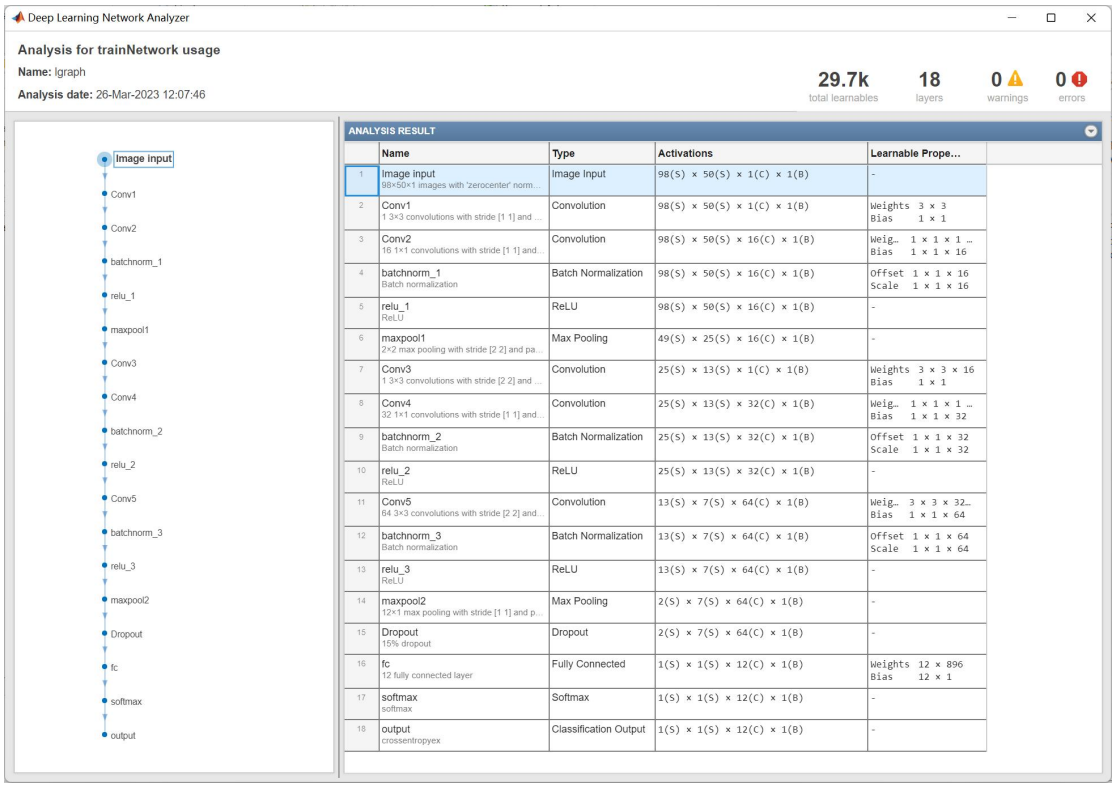
3.3 Reduce the computational complexity:
In this study, I applied depthwise separable convolutions and 1x1 convolutions to reduce the model complexity compared to the best model from **3.2**.
Depthwise separable convolution is a type of convolutional neural network (CNN) layer that consists of two distinct operations: a depthwise convolution and a pointwise convolution, which can reduce computational cost and memory usage.
The first operation in a depthwise separable convolution layer is the depthwise convolution. This is a spatial convolution operation that applies a separate convolutional filter to each input channel, resulting in a set of output feature maps that are channel-wise independent. Essentially, it applies a filter to each channel of the input image independently, which allows the network to learn more specialized features for each channel. The depthwise convolution thus reduces the number of parameters in the network by a factor equal to the number of output channels.
The second operation in a depthwise separable convolution layer is the pointwise convolution. This is a 1x1 convolution operation that applies a linear transformation to the depthwise -convolved output, mapping it to a new feature space. This step can be thought of as a conventional layer, but applied to the output of the depthwise convolution instead of the original input image.
The combination of depthwise and pointwise convolutions enables a more efficient and effective learning process for CNNs. By separating the channel-wise and spatial information, depthwise

separable convolutions can learn more specialized features with fewer parameters than traditional convolutional layers, leading to faster training and better generalization performance.

Compare with CNNs with traditional convolutional layers:



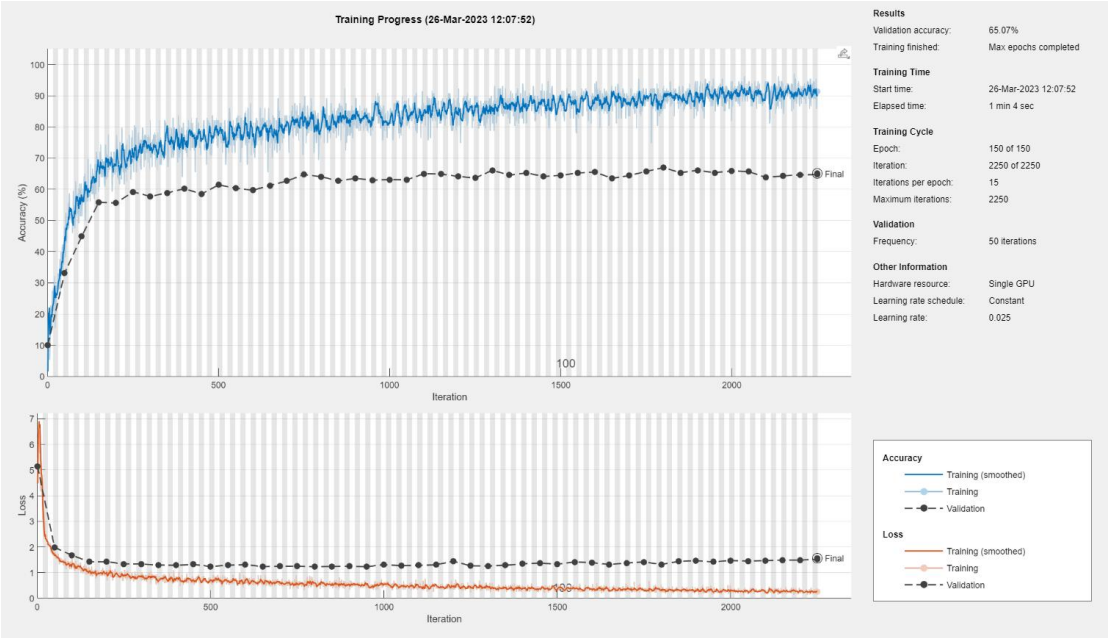
CNN with traditional convolutional layers



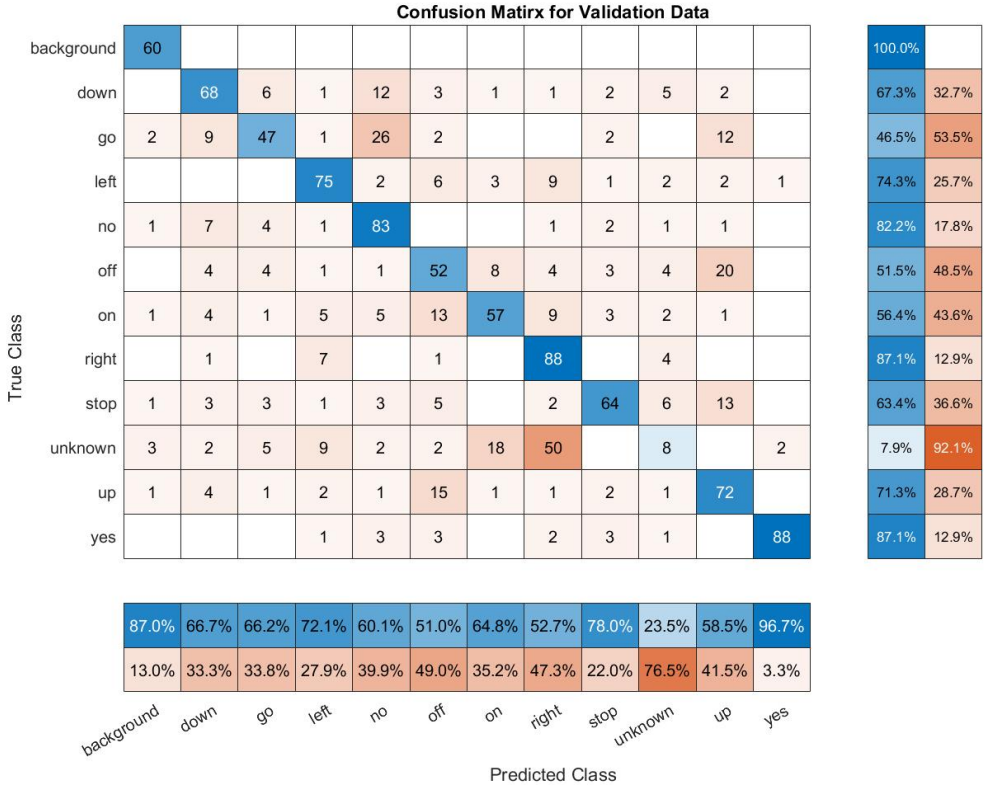
CNN with depthwise separable convolutional layers

As the experiment results shows, the CNN with traditional convolutional layers has 34.2k learnabels, whereas the CNN with depthwise separable convolutional layers has 29.7k learnables. Hence, the refined model has been reduced computational complexity comparing to the best model in 3.2.

Training progress:



Confusion plot:



4. Code:

Matlab:

```
clear all;
close all force;

% define the random number seed for repeatable results
rng(1,'twister');
timePoolSize = 12;

%% Load Speech Data

% create an image data store from the raw images
imdsTrain = imageDatastore('C:\Users\Administrator\Desktop\DL_for_speech\speechImageData\speechImageData\TrainData',...
"IncludeSubfolders",true,"LabelSource","foldernames")

% create an image validation data store from the validation images
imdsVal = imageDatastore('C:\Users\Administrator\Desktop\DL_for_speech\speechImageData\speechImageData\ValData',...
"IncludeSubfolders",true,"LabelSource","foldernames")



% your code here...
% build network

% 2x2 grid search:
% conv layers 2 + fliters 8  conv layers 3 + fliters 8
% conv layers 2 + fliters 16  conv layers 3 + fliters 16

% The number of convolution layers: 2
% The number of filters of first convolution layer:8
% Net= [
%     imageInputLayer([98 50 1],'Name','Image input')
%     convolution2dLayer([3 3],8,'Padding','same','Name','Conv1')
%     batchNormalizationLayer('Name','batchnorm_1')
%     reluLayer('Name','relu_1')
%     maxPooling2dLayer(2,'Stride',2,'Padding','same','Name','maxpool1')
%
%     convolution2dLayer([3 3],16,'Padding','same','Stride', 2 ,'Name','Conv2')
%     batchNormalizationLayer('Name','batchnorm_2')
%     reluLayer('Name','relu_2')
%     maxPooling2dLayer([timePoolSize 1],'Name','maxpool2')
%
```

```
%     dropoutLayer(0.15,'Name','Dropout')
%     fullyConnectedLayer(12,'Name','fc')
%     softmaxLayer('Name','softmax')
%     classificationLayer('Name','output')
%     ];

% The number of convolution layers: 2
% The number of filters of firsr convolution layer:16
% Net= [
%     imageInputLayer([98 50 1],'Name','Image input')
%     convolution2dLayer([3 3],16,'Padding','same','Name','Conv1')
%     batchNormalizationLayer('Name','batchnorm_1')
%     reluLayer('Name','relu_1')
%     maxPooling2dLayer(2,'Stride',2,'Padding','same','Name','maxpool1')
%
%     convolution2dLayer([3 3],32,'Padding','same','Stride', 2 ,'Name','Conv2')
%     batchNormalizationLayer('Name','batchnorm_2')
%     reluLayer('Name','relu_2')
%     maxPooling2dLayer([timePoolSize 1],'Name','maxpool2')
%
%     dropoutLayer(0.15,'Name','Dropout')
%     fullyConnectedLayer(12,'Name','fc')
%     softmaxLayer('Name','softmax')
%     classificationLayer('Name','output')
%     ];

% The number of convolution layers: 3
% The number of filters of first convolution layer:8
% Net= [
%     imageInputLayer([98 50 1],'Name','Image input')
%     convolution2dLayer([3 3],8,'Padding','same','Name','Conv1')
%     batchNormalizationLayer('Name','batchnorm_1')
%     reluLayer('Name','relu_1')
%     maxPooling2dLayer(2,'Stride',2,'Padding','same','Name','maxpool1')
%
%     convolution2dLayer([3 3],16,'Padding','same','Stride', 2 ,'Name','Conv2')
%     batchNormalizationLayer('Name','batchnorm_2')
%     reluLayer('Name','relu_2')
%     convolution2dLayer([3 3],32,'Padding','same','Stride', 2 ,'Name','Conv3')
%     batchNormalizationLayer('Name','batchnorm_3')
%     reluLayer('Name','relu_3')
%     maxPooling2dLayer([timePoolSize 1],'Name','maxpool2')
%
%     dropoutLayer(0.15,'Name','Dropout')
```

```
%     fullyConnectedLayer(12,'Name','fc')
%     softmaxLayer('Name','softmax')
%     classificationLayer('Name','output')
%     ];

% The number of convolution layers: 3
% The number of filters of first convolution layer:16
% Net= [
%     imageInputLayer([98 50 1],'Name','Image input')
%     convolution2dLayer([3 3],16,'Padding','same','Name','Conv1')
%     batchNormalizationLayer('Name','batchnorm_1')
%     reluLayer('Name','relu_1')
%     maxPooling2dLayer(2,'Stride',2,'Padding','same','Name','maxpool1')
%
%     convolution2dLayer([3 3],32,'Padding','same','Stride', 2 ,'Name','Conv2')
%     batchNormalizationLayer('Name','batchnorm_2')
%     reluLayer('Name','relu_2')
%     convolution2dLayer([3 3],64,'Padding','same','Stride', 2 ,'Name','Conv3')
%     batchNormalizationLayer('Name','batchnorm_3')
%     reluLayer('Name','relu_3')
%     maxPooling2dLayer([timePoolSize 1],'Name','maxpool2')
%
%     dropoutLayer(0.15,'Name','Dropout')
%     fullyConnectedLayer(12,'Name','fc')
%     softmaxLayer('Name','softmax')
%     classificationLayer('Name','output')
%     ];

%  Reduce computational complexity by depthwise separable convolutions and
%  1x1 convolutions.
Net= [
    imageInputLayer([98 50 1],'Name','Image input')
    % depthwise separable convolutions
    convolution2dLayer([3 3],1,'Padding','same','Name','Conv1')
    convolution2dLayer([1 1],16,'Padding','same','Name','Conv2')
    batchNormalizationLayer('Name','batchnorm_1')
    reluLayer('Name','relu_1')
    maxPooling2dLayer(2,'Stride',2,'Padding','same','Name','maxpool1')

    % depthwise separable convolutions
    convolution2dLayer([3 3],1,'Padding','same','Stride', 2 ,'Name','Conv3')
    convolution2dLayer([1 1],32,'Padding','same','Name','Conv4')
    batchNormalizationLayer('Name','batchnorm_2')
    reluLayer('Name','relu_2')
```

```
    convolution2dLayer([3 3],64,'Padding','same','Stride', 2 ,'Name','Conv5')
    batchNormalizationLayer('Name','batchnorm_3')
    reluLayer('Name','relu_3')
    maxPooling2dLayer([timePoolSize 1],'Name','maxpool2')

    dropoutLayer(0.15,'Name','Dropout')
    fullyConnectedLayer(12,'Name','fc')
    softmaxLayer('Name','softmax')
    classificationLayer('Name','output')
    ];

%Visualize the network
lgraph = layerGraph(Net);
analyzeNetwork(lgraph);

% Configuration
options = trainingOptions('sgdm', ...
    'InitialLearnRate',0.025, ...
    'LearnRateDropFactor',0.1, ...
    'LearnRateDropPeriod',20, ...
    'MaxEpochs',150, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsVal, ...
    'ValidationFrequency',50, ...
    'Verbose',true, ...
    'Plots','training-progress');

net = trainNetwork(imdsTrain,Net,options);

% evaluation
YPred = classify(net,imdsVal);
YValidation = imdsVal.Labels;
accuracy = sum(YPred == YValidation)/numel(YValidation);
fprintf('Accuracy: %.2f%%\n',accuracy*100);

figure
cm = confusionchart(YValidation,YPred);
cm.ColumnSummary ="column-normalized";
cm.RowSummary = "row-normalized";
title('Confusion Matirx for Validation Data');
% cm.Normalization = 'absolute';
```