

1 Introduction

TensorFlow Lite is an open source software library for running machine learning models on mobile and embedded devices. For more information, see <https://www.tensorflow.org/lite/>.

The MCUXpresso Software Development Kit (MCUXpresso SDK) includes a comprehensive software package with a pre-integrated eIQ TensorFlow Lite library based on TensorFlow Lite 1.14 version. This document describes the steps to train the TensorFlow model on Gender voice audio samples and classification. The trained TensorFlow model is converted to a source file that can run on i.MXRT board.

Contents

1	Introduction.....	1
2	Model development on TensorFlow	2
3	TensorFlow model conversion.....	9
4	Running inference on edge (i.MXRT)	11
5	Conclusion.....	15
6	Revision History.....	15

1.1 Application details

Gender voice recognition consists of two important parts:

1. Convert audio samples from time-domain waveform to the frequency domain and extract features using MFCC(Mel Frequency Cepstral Coefficients).
2. Run the model based on neural network classifier as DS-CNN to process the extracted features and perform prediction.

1.2 Features extraction using MFCC

An overview of the gender voice recognition process is first to use Mel Frequency Cepstral Coefficients(MFCCs) as the feature extractor to get the 2D fingerprint of the audio. Since the input to the neural network is an image like a 2D audio fingerprint with the horizontal axis denoting the time and the vertical axis denoting the frequency coefficients.

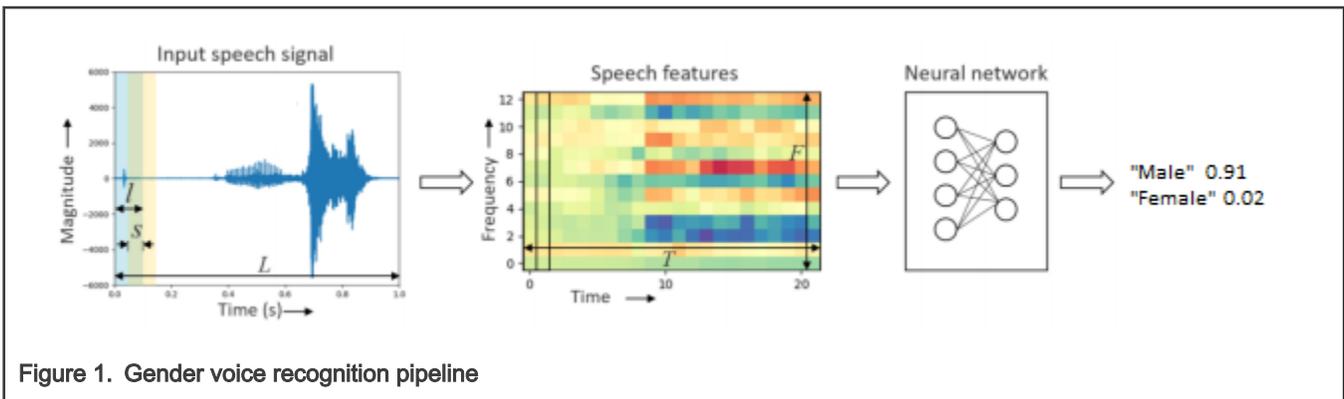


Figure 1. Gender voice recognition pipeline

The following parameters need to define for feature extraction using MFCC:

- Input audio sampling rate: 16000 Hz
- Input audio clip length: 1000 ms (L)
- Spectrogram window size: 40 ms (l)
- Spectrogram window stride: 20 ms (s)



- MFCC coefficient count:10 (F)

Using the above parameters of Input audio clip length(L), Spectrogram window size(I), and Spectrogram window stride(s), the total number of frames for the 2D fingerprint model are:

$$T = \frac{L-I}{s} + 1 = \frac{1000-40}{20} + 1 = 49$$

For 1 frame, MFCC coefficient = 10; then it becomes 490 in size, fingerprint for 49 frames.

On the sliced frame, a window function (Hanning window) is applied to each frame; afterward, Fourier transform on each frame (or more specifically a First Fourier Transform) is applied and the power spectrum is calculated; and subsequently compute the filter banks. To obtain MFCCs, a Discrete Cosine Transform (DCT) is applied to the filter banks retaining 10 numbers of the resulting coefficients while the rest are discarded.

After pre-processing of the audio signal through MFCC; it is fed to inference for gender voice classification.

1.3 Prerequisites

- MCUXpresso 11.2.0 or later
- i.MX RT1060 or i.MX RT1050 board
- Serial Console(Teraterm)
- Gender Voice Recognition Release Package. It contains Python scripts that this document needs to train the model, available in the [Application Note Software](#) section.
- MCUXpresso SDK 2.8.0 or later for i.MX RT1060, available at [MCUXpresso SDK Builder](#).

2 Model development on TensorFlow

TensorFlow is an open-source neural network library written in Python. The developed TensorFlow model can be easily converted into TensorFlow Lite for running the model on edge.

Based upon a sample voice, it will help identify a person's gender as male or female. Often, the human ear can easily detect the difference between a male or female voice within the first few spoken words; this functionality is achieved through a developed model.

This application note describes TensorFlow model training on DS-CNN neural network with audio samples. The TensorFlow model (available with this Application Note) was trained using 8500 pre-recorded samples of male and female voices, speech, and utterances. The audio sample consists of a downloaded sample from <http://festvox.org/cmudict/cmudict/packed/> and recorded samples from 21 different speakers. The samples are processed using Mel-frequency Cepstral Coefficients (MFCC) technique and then applied to an artificial intelligence/machine learning algorithm to learn gender-specific traits and then export the trained model on i.MXRT1060 board and use to run inference to classify gender voice.

2.1 Depthwise Separable Convolutional Neural Network (DS-CNN) model

Depthwise Separable Convolution is an efficient alternative to the standard 3-D convolution operation and has been used to achieve compact network architectures in the area of computer vision. DS-CNN first convolves each channel in the input feature map with a separate 2-D filter and then uses pointwise convolutions (that is, 1x1) to combine the outputs in the depth dimension. By decomposing the standard 3-D convolutions into 2-D convolutions followed by 1-D convolutions, Depthwise Separable Convolutions are more efficient both in the number of parameters and operations, which makes deeper and wider architecture possible even in the resource-constrained microcontroller devices. In this work, we adopt a Depthwise Separable CNN based on the implementation of MobileNet. An average pooling followed by a fully-connected layer is used at the end to provide global interaction and reduce the total number of parameters in the final layer.

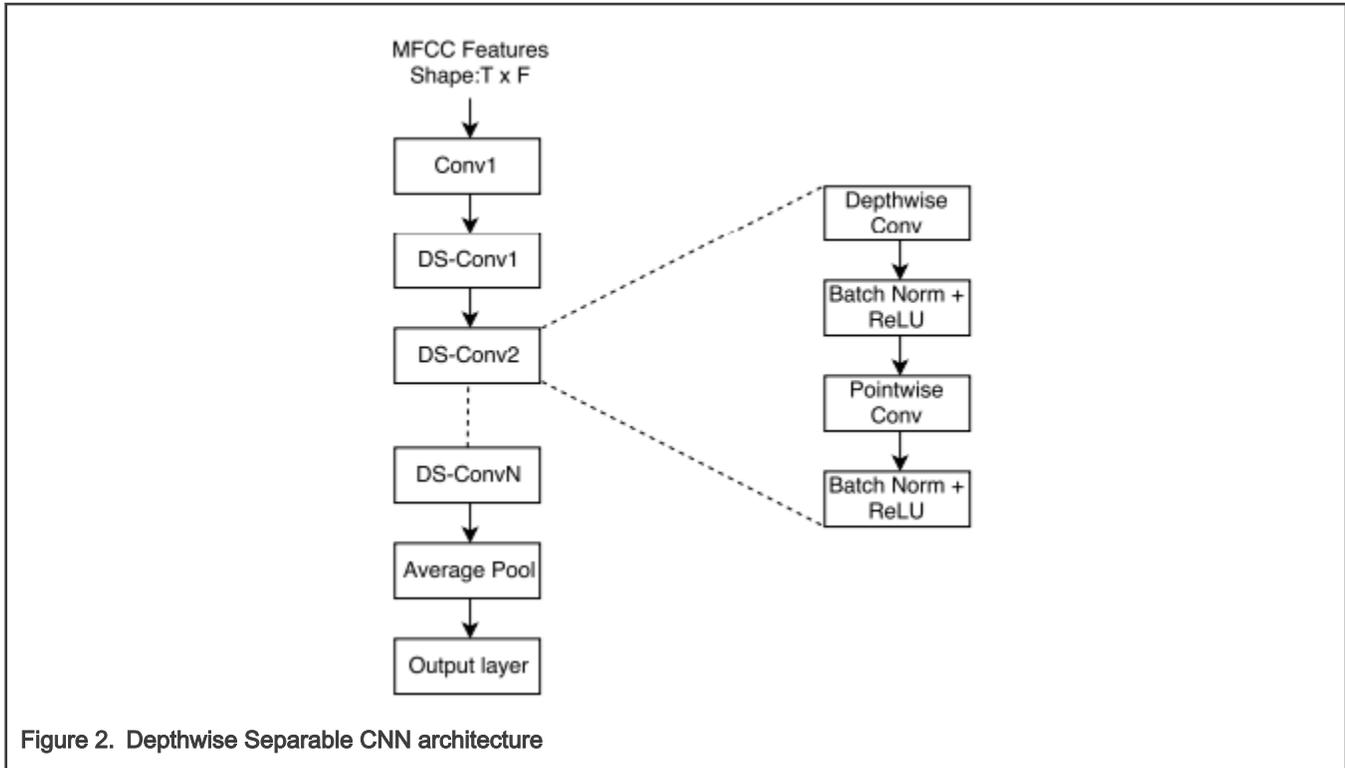


Figure 2. Depthwise Separable CNN architecture

A snippet of DS-CNN code is provided below:

Note that the first layer is always regular convolution of the model, but the remaining layers are all Depthwise Separable Convolutions.

```

for layer_no in range(0,num_layers):
    if layer_no==0:
        net = slim.convolution2d(fingerprint_4d, conv_feat[layer_no], \
            [conv_kt[layer_no], conv_kf[layer_no]], stride=[conv_st[layer_no], \
            conv_sf[layer_no]], padding='SAME', scope='conv_1')
        net = slim.batch_norm(net, scope='conv_1/batch_norm')
    else:
        net = _depthwise_separable_conv(net, conv_feat[layer_no], \
            kernel_size = [conv_kt[layer_no],conv_kf[layer_no]], \
            stride = [conv_st[layer_no],conv_sf[layer_no]], sc='conv_ds_'+str(layer_no))
        t_dim = math.ceil(t_dim/float(conv_st[layer_no]))
        f_dim = math.ceil(f_dim/float(conv_sf[layer_no]))

        net = slim.avg_pool2d(net, [t_dim, f_dim], scope='avg_pool')

net = tf.squeeze(net, [1, 2], name='SpatialSqueeze')
logits = slim.fully_connected(net, label_count, activation_fn=None, scope='fcl')
  
```

Figure 3. DS-CNN code snippet

Implementation of the Depthwise Separable Convolution layer looks like below.

```

def _depthwise_separable_conv(inputs,
                              num_pwc_filters,
                              sc,
                              kernel_size,
                              stride):
    """ Helper function to build the depth-wise separable convolution layer.
    """

    # skip pointwise by setting num_outputs=None
    depthwise_conv = slim.separable_convolution2d(inputs,
                                                  num_outputs=None,
                                                  stride=stride,
                                                  depth_multiplier=1,
                                                  kernel_size=kernel_size,
                                                  scope=sc+'/dw_conv')

    bn = slim.batch_norm(depthwise_conv, scope=sc+'/dw_conv/batch_norm')
    pointwise_conv = slim.convolution2d(bn,
                                       num_pwc_filters,
                                       kernel_size=[1, 1],
                                       scope=sc+'/pw_conv')
    bn = slim.batch_norm(pointwise_conv, scope=sc+'/pw_conv/batch_norm')
    return bn

```

Figure 4. Depthwise Separable Convolution layer

An average pooling followed by a fully-connected layer is used at the end to provide global interaction and reduce the total number of parameters in the final layer.

2.2 Steps for training the DS-CNN model on Gender Voice Dataset

Before we head towards model training, we have to set up a system for the TensorFlow framework for model training.

Windows system should be pre-installed with MCUXpresso IDE v11.2.0 or later, SDK2.8.0, or later with the eIQ component for i.MXRT1060-EVK and a serial terminal emulator (that is, TeraTerm).

The steps for installation of TensorFlow on Windows are listed below:

1. Download and install the Anaconda (appropriate to your system supporting 32-bit or 64-bit) from the link below. The Anaconda has in-built several utility packages that help to run python scripts for the TensorFlow model. After downloading the Anaconda, install it with the default selection options in your system.

<https://www.anaconda.com/products/individual#windows>

NOTE

- a. Make sure that the Anaconda location path is included in the environment system variable.
 - b. Some of the necessary packages need to be installed separately, which can be done by following step 2.
2. Before installing the required packages for developing the TensorFlow model, make sure that the Python version is 3.7.3. Follow the next commands from the Anaconda command prompt to see the python version and downgrade it if required:

```
python -V
conda install python=3.7.3
```

Below are the packages required for developing the TensorFlow model. The below commands are executed from the Anaconda command prompt.

```
python -m conda install tensorflow=1.14.0
python -m conda install tensorflow-datasets
python -m conda install numpy=1.16.4
conda install -c anaconda scikit-learn=0.21.2
```

NOTE

Install the NumPy package only if not already installed from the above mentioned packages.

3. Install Vim v8.1 or latest, for the conversion of **.tfLite** file into **.h** file. There is a binary converter program named **xxd.exe** located inside the Vim package that is required during the conversion of the TensorFlow-Lite (TF-Lite) file into a source file. For windows, install Vim from the below link.

<https://www.vim.org/download.php#pc>

NOTE

Vim package named "gvim82.exe (ftp)" needs to be installed. Add vim location path in the environment system variable.

The steps for model training and gender voice classification are listed below:

1. Data preparation: A total of 4000 audio wave files are available.

Male audio sample files are available at the below link.

http://festvox.org/cmu_arctic/cmu_arctic/packed/cmu_us_rms_arctic-0.95-release.zip

http://festvox.org/cmu_arctic/cmu_arctic/packed/cmu_us_ksp_arctic-0.95-release.zip

Female audio sample files are available at the below link.

http://festvox.org/cmu_arctic/cmu_arctic/packed/cmu_us_sl_t_arctic-0.95-release.zip

http://festvox.org/cmu_arctic/cmu_arctic/packed/cmu_us_clb_arctic-0.95-release.zip

NOTE

After unzip, you will get the audio samples in the **wav** folder.

2. Create a **male** and a **female** folder. Copy male and female wave files that have been downloaded from step 1 above and paste them in the **male** and **female** folder respectively.
3. Create a **tmp** folder as the parent folder in the drive; which will be used for providing a dataset for training the model. For example, **E:\tmp**. Under the **tmp** folder, create a **speech_dataset** folder; copy **male** and **female** folders (created in Step 2 which consists of audio samples) into **speech_dataset** folder.

The folder structure must look like below.

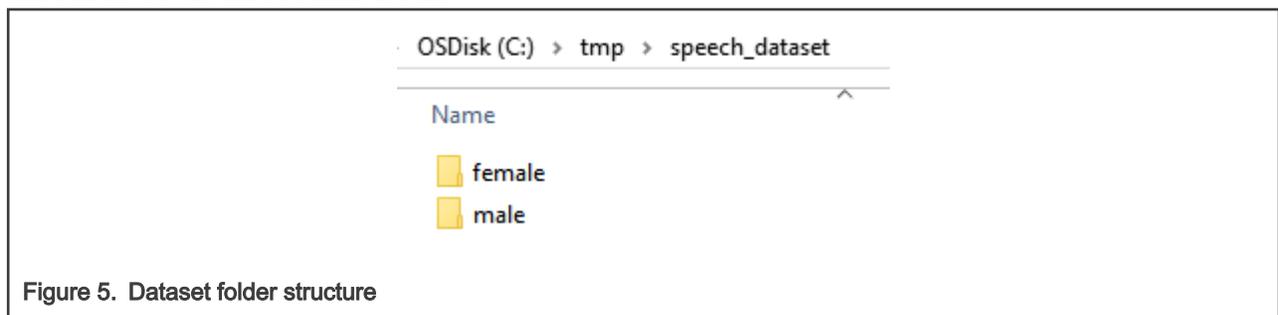


Figure 5. Dataset folder structure

4. Download the model training and gender voice classification script from the below link:

<https://github.com/ARM-software/ML-KWS-for-MCU.git>

After downloading, you will require the below mentioned script from the downloaded folder **ML_KWS-for-MCU**.

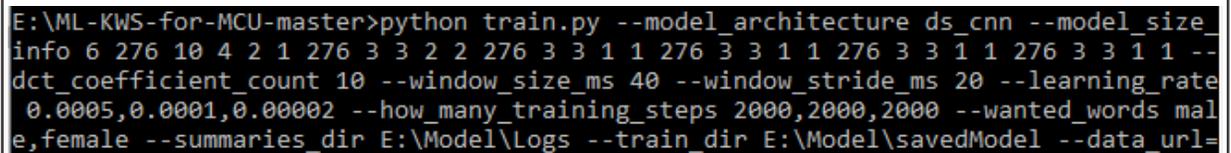
- **train.py** - Used for training the model.
 - **freeze.py** - Used for freezing the trained model checkpoint into a protocol buffer (.pb) file.
 - **label_wav.py** - Used for gender voice classification.
5. To train the model, the command below needs to be executed on the window command prompt. Navigate to the **ML_KWS-for-MCU** folder and execute the command.

```
python train.py --model_architecture ds_cnn --model_size_info
6 276 10 4 2 1 276 3 3 2 2 276 3 3 1 1 276 3 3 1 1
276 3 3 1 1 276 3 3 1 1 --dct_coefficient_count 10 --
window_size_ms 40 --window_stride_ms 20 --learning_rate 0.0005,0.0001,0.00002 --
how_many_training_steps 2000,2000,2000 --wanted_words
male,female --summaries_dir .\path\to\the\Model\Logs --
train_dir .\path\to\the\Model\savedModel
```

Before we start training the model, we need to understand the arguments of the model training command through which we will train the model.

Argument explanation is mentioned below:

- **--model_architecture <ds_cnn>**: model architecture can be provided like cnn, dnn, and so on. It is recommended to use ds_cnn as it uses less memory and processing power.
- **--model_size_info <6 276 10 4 2 1 276 3 3 2 2 276 3 3 1 1 276 3 3 1 1 276 3 3 1 1 276 3 3 1 1 276 3 3 1 1 >**
- **1st argument: <6>** define the number of layers.
- **Argument: <276 10 4 2 1>**: 276 defines the number of features for neural network. Argument <10, 4> defines the size of filter and argument <2, 1> defines the filter stride along x and y axis.
- **--dct_coefficient_count <10>**: consider only 10 filters.
- **--window_size_ms <40>**: window size of 40 milliseconds is considered for feature extraction using MFCC.
- **--window_stride_ms <20>**: the window will stride for 20 milliseconds, so it will have an overlap of 20 milliseconds.
- **--learning_rate <0.0005, 0.0001, 0.00002>** and **--how_many_training_steps <2000, 2000, 2000>**: total number of training steps will be 6000; such training steps will train the model for 6000 times. For example, the 1st step trains the model for the first time with a 0.0005 learning rate, the 2nd step trains the model for the second time with a 0.0001 learning rate, and so on.
- **--wanted_words <male,female>**: name of the audio folder which has audio samples. Based on this folder name, labels are generated in ds_cnn_labels.txt with two additional labels **silence** and **unknown**. Labels generated in ds_cnn_labels.txt act as a classification name.
- **--summaries_dir <path/to/the/Model/Logs>**: training model logs will be saved in this folder.
- **--train_dir <path/to/the/Model/savedModel>**: trained model will be saved in this folder.



```
E:\ML-KWS-for-MCU-master>python train.py --model_architecture ds_cnn --model_size_info 6 276 10 4 2 1 276 3 3 2 2 276 3 3 1 1 276 3 3 1 1 276 3 3 1 1 276 3 3 1 1 --dct_coefficient_count 10 --window_size_ms 40 --window_stride_ms 20 --learning_rate 0.0005,0.0001,0.00002 --how_many_training_steps 2000,2000,2000 --wanted_words male,female --summaries_dir E:\Model\Logs --train_dir E:\Model\savedModel --data_url=
```

Figure 6. Model training command

After the execution of the above command, the below logs will be displayed on the screen once training is completed. Model training takes approximately 10 hours depending upon system configuration.

```

I0630 15:00:46.635765 13504 train.py:269] Confusion Matrix:
[[140  0  0  0]
 [  0 136  4  0]
 [  0  2 686 10]
 [  0  2 18 681]]
I0630 15:00:46.636761 13504 train.py:271] Step 6000: Validation accuracy = 97.86% (N=1679)
I0630 15:00:46.636761 13504 train.py:280] So far the best validation accuracy is 98.27%
I0630 15:00:46.636761 13504 train.py:283] set_size=1822
I0630 15:01:53.077236 13504 train.py:302] Confusion Matrix:
[[152  0  0  0]
 [  0 150  2  0]
 [  0  0 746  8]
 [  0  1 17 746]]
I0630 15:01:53.078235 13504 train.py:304] Final test accuracy = 98.46% (N=1822)
    
```

Figure 7. Model training completion logs

After the model is trained; the model checkpoint will be saved in the **'to/the/folder/Model/savedModel/best'** folder.

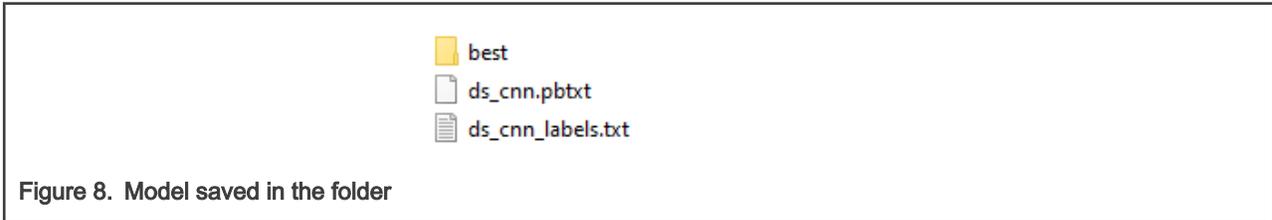


Figure 8. Model saved in the folder

- The protocol buffer (.pb) file will be required further for testing the model to classify the gender as well as to convert it into a TF-Lite file. Therefore, the trained model (checkpoint file) needs to freeze to generate a protocol buffer (.pb) file. The checkpoint file is available at **'to/the/folder/savedModel/best'** path, choose the file which has the maximum numeric number as checkpoint file (For example, ds_cnn_9663.ckpt-3400) among these many files as shown in the below figure.


```
(base) C:\WINDOWS\system32>C:\tmp\speech_dataset\malepython label_wav.py -wav C:\tmp\speech_dataset\male\artic_a0001.wav  
--graph C:\Model\savedModel\ds_cnn.pb --labels C:\Model\savedModel\ds_cnn_labels.txt --how_many_labels 1
```

Figure 11. Gender voice classification script

After the execution of the above command, gender voice classification prediction will be displayed with a score.

```
male (score = 0.99994)
```

Figure 12. Gender voice classification prediction result

3 TensorFlow model conversion

3.1 Converting model in TensorFlowLite format

The trained and saved model should be converted into a TF-Lite compatible format. For this, input tensor name and output tensor name must be known, which is required during the conversion of protocol buffer file (.pb) into the TF-Lite file. Therefore, execute the **check_layer.py** script which is available with this release package in the **script** folder to know the input and output tensor name. Navigate to '**/to/the/folder/release/script**' folder and execute the command below:

```
python check_layer.py --pbfile path/to/the/saveModel/ds_cnn.pb
```

```
E:\release\script>python check_layer.py --pbfile E:\Model\savedModel\ds_cnn.pb
```

Figure 13. Command for checking input/output tensor name

After the execution of the **check-layer.py** script, the below message will be displayed on the screen. Note down the tensors' names marked in red color shown in Figure 14.

Input tensor name: **Reshape**

Output tensor name: **labels_softmax**

```

Tensor("prefix/wav_data:0", dtype=string)
Tensor("prefix/decoded_sample_data:0", shape=(16000, 1), dtype=float32)
Tensor("prefix/AudioSpectrogram:0", shape=(1, 49, 513), dtype=float32)
Tensor("prefix/Mfcc:0", shape=(1, 49, 10), dtype=float32)
Tensor("prefix/Reshape/shape:0", shape=(2,), dtype=int32)
Tensor("prefix/Reshape_0", shape=(1, 490), dtype=float32)
Tensor("prefix/Reshape_1/shape:0", shape=(4,), dtype=int32)
Tensor("prefix/Reshape_1:0", shape=(1, 49, 10, 1), dtype=float32)
Tensor("prefix/DS-CNN/conv_1/weights:0", shape=(10, 4, 1, 172), dtype=float32)
....
....
....
Tensor("prefix/DS-CNN/fc1/biases/read:0", shape=(4,), dtype=float32)
Tensor("prefix/DS-CNN/fc1/MatMul:0", shape=(1, 4), dtype=float32)
Tensor("prefix/DS-CNN/fc1/BiasAdd:0", shape=(1, 4), dtype=float32)
Tensor("prefix/labels_softmax:0", shape=(1, 4), dtype=float32)

```

Figure 14. Message after the execution of the check-layer.py script

Now convert the .pb file into a tflite file by providing input & output tensor name while executing the command as mentioned below:

```

tflite_convert --output_file=path\to\the\Model\savedModel\ds_cnn.tflite --
graph_def_file=path\to\the\Model\savedModel\ds_cnn.pb --input_arrays=Reshape
--output_arrays=labels_softmax

```

```

E:\release\script>tflite_convert --output_file=E:\Model\savedModel\ds_cnn.tflite
--graph_def_file=E:\Model\savedModel\ds_cnn.pb --input_arrays=Reshape --output_arrays=
labels_softmax

```

Figure 15. Command for converting .pb file into TF-Lite

3.2 Converting the TF-Lite model into a source file

TF-Lite model file **ds_cnn.tflite** is used to generate a source file. Open the window command prompt, navigate to the '**to\folder\ Model\savedModel**' folder, and execute the below command:

```

xxd -i ./ds_cnn.tflite > ./ds_cnn_gender_model.h

```

After execution, **ds_cnn_gender_model.h** file will be generated. Add the **const** prefix in front of the unsigned char and the unsigned int declarations as shown below in Figure 16. The **ds_cnn_gender_model.h** file will be required in the below step for running the inference.

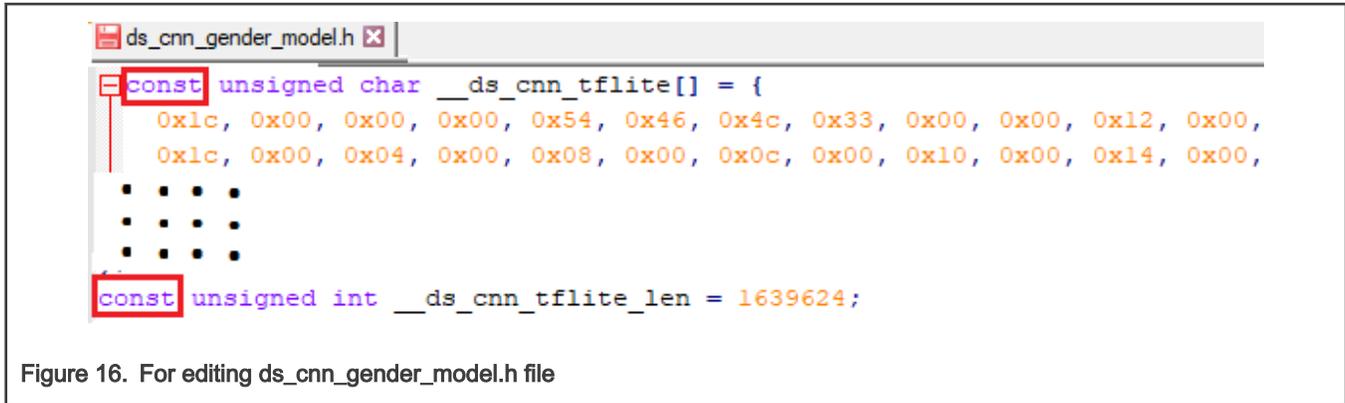


Figure 16. For editing ds_cnn_gender_model.h file

4 Running inference on edge (i.MXRT)

Before proceeding further, it is assumed that you are familiar with the eIQ demo application **tensorflow_lite_kws** and SDK builder to build the SDK_2.8.0_EVK-MIMXRT1060 with eIQ and CMSIS-DSP library component.

Download the SDK_2.8.0_EVK-MIMXRT1060.zip and SDK_2.8.0_EVK-MIMXRT1060_doc.zip folders from the SDK builder. For more information, refer to **eIQ TensorFlow Lite Library User's Guide.pdf** file available in the SDK document zip folder. After that, import **evkmimxrt1060_tensorflow_lite_kws** project available under eIQ component in MCUXpresso IDE.

Perform the below steps for replacing the model in the eIQ **evkmimxrt1060_tensorflow_lite_kws** application and running the inference.

1. Copy and replace **ds_cnn_s_model.h** header files in the application project with the header file **ds_cnn_gender_model.h**, which was generated in the previous step. Your file must look like the figure shown below:

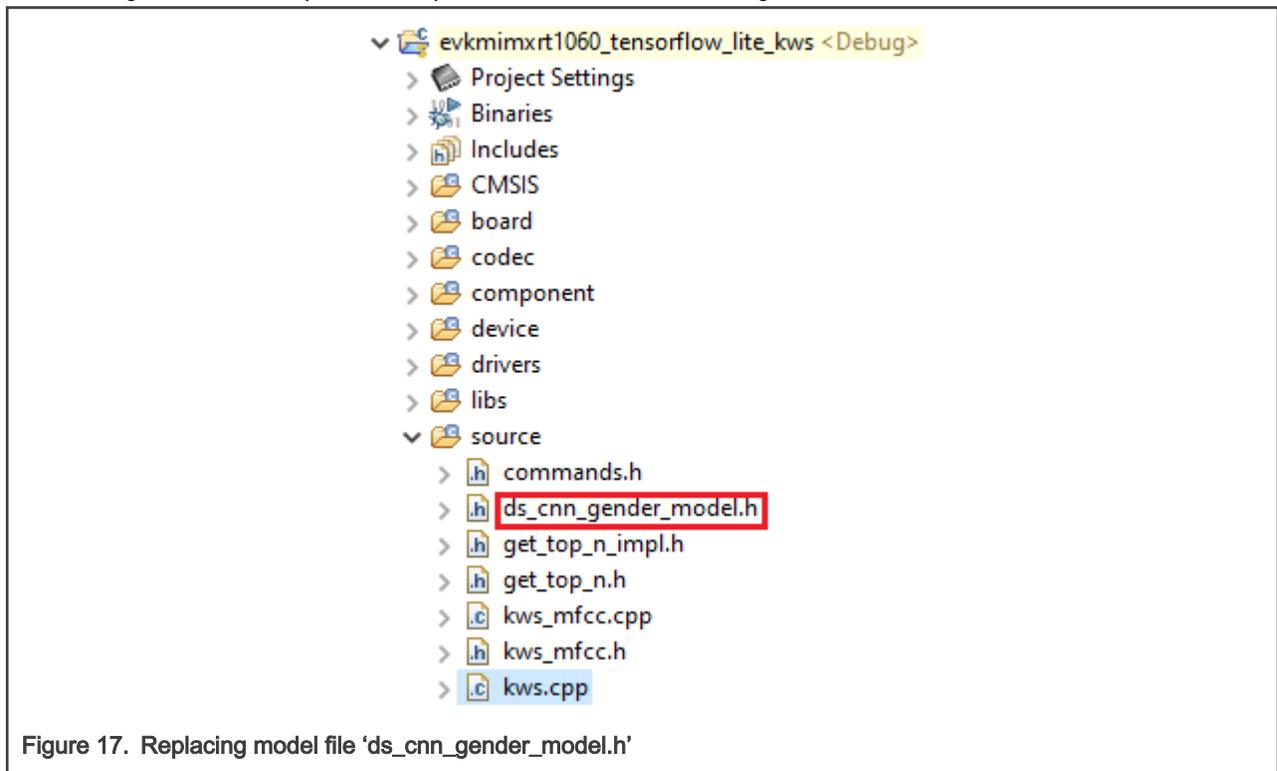


Figure 17. Replacing model file 'ds_cnn_gender_model.h'

2. Comment and add string labels under the **main** function in **kws.cpp** file with the male and female label as shown below:

```

425 // const std::string labels[] = {"Silence", "Unknown", "yes", "no",
426 //   "up", "down", "left", "right", "on", "off", "stop", "go"};
427 const std::string labels[] = {"Silence", "Unknown", "male", "female"};

```

Figure 18. Comment and add labels

```
const std::string labels[] = {"Silence", "Unknown", "male", "female"};
```

3. Comment and add the below statement under the **main** function in the **kws.cpp** file:

```

446 InferenceInit(model, interpreter, &input_tensor, false);
447
448 //LOG(INFO) << "Keyword spotting example using a TensorFlow Lite model.\r\n" << std::endl;
449 LOG(INFO) << "Gender voice recognition example using a TensorFlow Lite model.\r\n" << std::endl;
450 LOG(INFO) << "Detection threshold: " << DETECTION_TRESHOLD << "%\r\n";
451
452 LOG(INFO) << "\r\nStatic data processing:\r\n" << std::endl;
453
454 //RunInference(&kws_mfcc, (int16_t*)OFF, labels, model, interpreter, input_tensor);
455 //RunInference(&kws_mfcc, (int16_t*)RIGHT, labels, model, interpreter, input_tensor);
456 RunInference(&kws_mfcc, (int16_t*)WAVE_DATA, labels, model, interpreter, input_tensor);
457 /* Clock setting for LPI2C */

```

Figure 19. Comment and add a statement for WAVE_DATA

```
LOG(INFO) << "Gender voice recognition example using a TensorFlow Lite model.\r\n" << std::endl;
```

```
RunInference(&kws_mfcc, (int16_t*)WAVE_DATA, labels, model, interpreter, input_tensor);
```

4. Comment and add model buffer name under **InferenceInit** function in **kws.cpp** file

```

5 void InferenceInit(std::unique_ptr<tflite::FlatBufferModel> &model,
6                   std::unique_ptr<tflite::Interpreter> &interpreter,
7                   TfliteTensor** input_tensor, bool isVerbose)
8 {
9   //model = tflite::FlatBufferModel::BuildFromBuffer((const char*)ds_cnn_s_model, ds_cnn_s_model_len);
10  model = tflite::FlatBufferModel::BuildFromBuffer((const char*)_ds_cnn_tflite, __ds_cnn_tflite_len);

```

Figure 20. Comment and add model buffer name

```
model = tflite::FlatBufferModel::BuildFromBuffer((const char*)
__ds_cnn_tflite, __ds_cnn_tflite_len);
```

5. Comment on the below statement and declare input tensors as shown in the below figure under the **RunInference** function in the **kws.cpp** file.

```

368     int input_bytes = input_tensor->bytes;
369
370     /* Experimental min, max range for quantization of the input.*/
371     // float min= -247.0;
372     // float max = 30.0;
373     // for (int i = 0; i < input_bytes; i++)
374     // {
375     //     input_tensor->data.uint8[i] = (uint8_t)round((255 * 1.0 * (float)(in[i] - min)) / (max - min));
376     // }
377
378     int input = interpreter->inputs()[0];
379     float* input_voice = interpreter->typed_tensor<float>(input);
380
381     for (int i = 0; i < input_bytes/4 ; i++){
382         input_voice[i] = in[i];
383     }
384
385     auto start = GetTimeInUS();
    
```

Figure 21. Declare input tensors

```

int input = interpreter->inputs()[0];
float* input_voice = interpreter->typed_tensor<float>(input);

for(int i = 0; i < input_bytes/4; i++){
    input_voice[i] = in[i];
}
    
```

6. Comment and add the below statement for function **GetTopN**; under **RunInference** function in **kws.cpp** file as shown in the below figure:

```

/* Assume output dims to be something like (1, 1, ... , size) */
auto output_size = output_dims->data[output_dims->size - 1];

// GetTopN<uint8_t>(interpreter->typed_output_tensor<uint8_t>(0),
//                 output_size, 1, threshold,
//                 &top_results, false);

GetTopN<float>(interpreter->typed_output_tensor<float>(0),
              output_size, 1, threshold,
              &top_results, true);
    
```

Figure 22. Replacing input/output tensor types

```

GetTopN<float>(interpreter->typed_output_tensor<float>(0),
              output_size, 1, threshold,
              &top_results, true);
    
```

7. Comment and add the gender model library in the include section:

```

42 // #include "ds_cnn_s_model.h"
43 #include "ds_cnn_gender_model.h"
    
```

Figure 23. Include library

```

#include "ds_cnn_gender_model.h"
    
```

8. Perform the below steps to execute the gender voice classification on the edge (i.MXRT board).
 - a. Generate an audio wave buffer array using the **wave_to_array.py** script. It is available in the **script** folder in the release package. After the execution of **wave_to_array.py**, it will generate audio wave buffer array **WAVE_DATA**

in **commands.h** file. File **commands.h** will be generated in the **'/release/audio'** folder. Execute the command below:

```
python wave_to_array.py --wave male1.wav
```

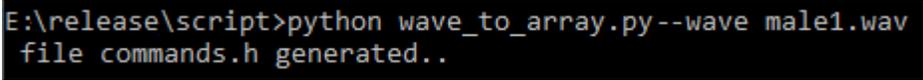


Figure 24. Audio wave buffer file command

- b. Copy and replace audio wave buffer array file **commands.h** in your eIQ **evkmimxrt1060_tensorflow_lite_kws** demo application, which was generated in the above **step a**. Your file must look similar to the image shown below:

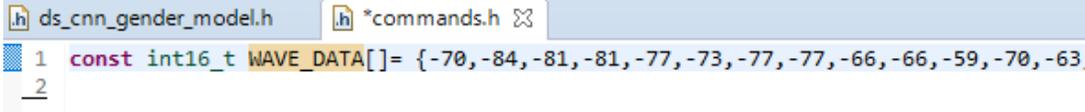


Figure 25. Audio wave buffer file

- c. Save the changes in the project file, build the project, and flash the binary in i.MXRT1060 board using debug mode.

The message will be displayed in the TeraTerm serial terminal for gender voice prediction. Here, the male is the output which is also known as predicted class by the model for input **artic_a0001.wav** with inference execution time 838 milliseconds as shown below.

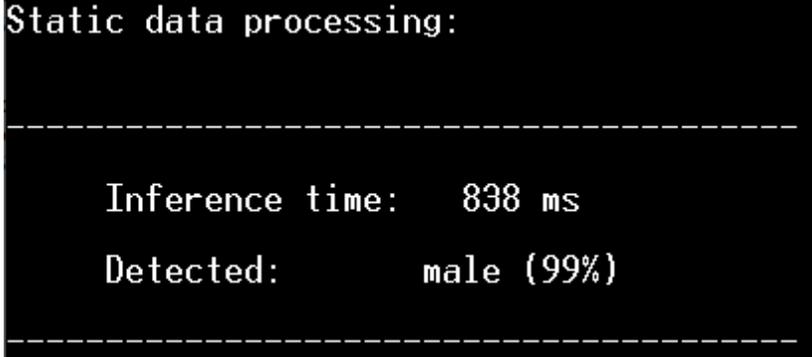
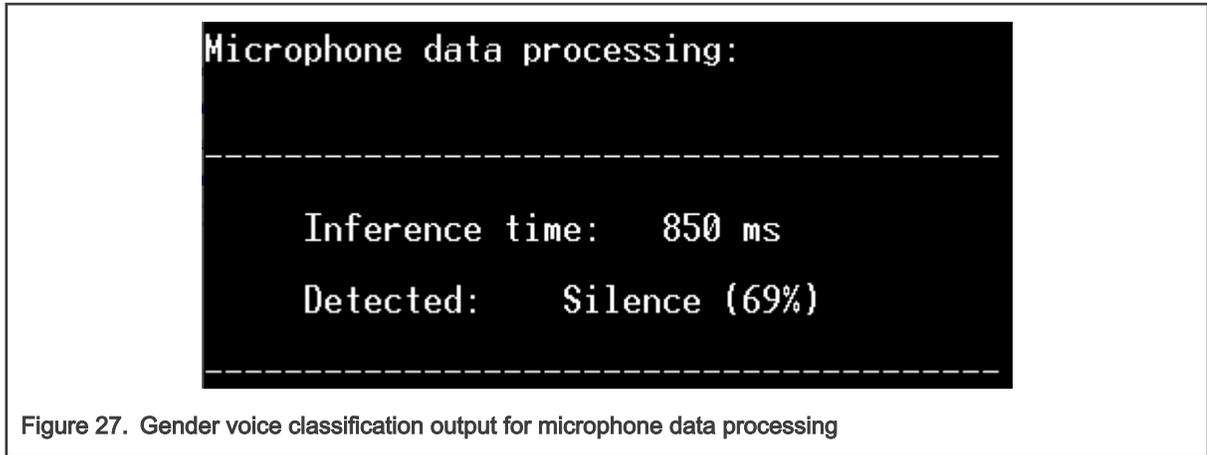


Figure 26. Gender voice classification output on TeraTerm serial terminal for static data

- d. Below logs are shown in the figure for microphone data processing when no male/female is speaking. Thus, the “silence” as the class is detected with accuracy in TeraTerm.

NOTE

As real-time data; the i.MXRT board microphone will capture audio data every second. Hence, the continuous logs will be displayed in TeraTerm.



- e. If any male/female is speaking nearby the i.MXRT board while microphone data processing is going on; then the microphone on the board will capture spoken audio of male/female and will display the result here with detected accuracy as real-time data in TeraTerm

5 Conclusion

This document demonstrates the TensorFlow model trained with the gender (male & female) voice samples to classify gender voices as male and female by such trained models in real-time and static data. It also demonstrates the conversion of a trained model to source files to run on i.MXRT platform using TF-Lite inference. Generated TF-Lite model runs on both i.MXRT1050 and i.MXRT1060 boards.

6 Revision History

Table 1. Revision history

Revision number	Date	Substantive changes
0	12/2020	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© NXP B.V. 2020.

All rights reserved.

For more information, please visit: <http://www.nxp.com>

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 12/2020
Document identifier: AN13065