User's Guide:
A Convolutional Neural Network Voice Activity Detector (VAD) Smartphone App for Hearing Improvement Studies

A. SEHGAL AND N. KEHTARNAVAZ
UNIVERSITY OF TEXAS AT DALLAS

FEBURARY 2018

This work was supported by the National Institute of the Deafness and Other Communication Disorders (NIDCD) of the National Institutes of Health (NIH) under the award number 1R01DC015430-01. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH.

Table of Contents

Introduction

This user's guide discusses how to use a convolutional neural network (CNN) voice activity detector (VAD) app that was developed at the University of Texas at Dallas. Both iOS and Android versions are covered in the user's guide. The VAD app uses signal features that had been found effective as the input to a convolutional neural network classifier. The theoretical details of the voice activity detection algorithm and its real-time implementation aspects are published in the following open access journal paper:

> A. Sehgal and N. Kehtarnavaz, "A Convolutional Neural Network Smartphone App for Real-time Voice Activity Detection," *IEEE Access*, open access journal, 2018.

In addition to the results reported in the above paper, Appendix A in this user's guide provides additional results of the performance of the developed VAD app.

This guide is divided into two parts. The first part covers the iOS version of the VAD app and the second part covers the Android version. Each part consists of four sections. In the first section, the GUI of the app is explained. The second section explains the code flow and how an audio stream is processed to achieve voice activity detection. Noting that voice activity detection is done in a supervised manner, the third section shows how to use a Python training code to generate the CNN model classifier.

Voice Activity Detection Folder Description

The code for the Android and iOS versions of the app along with the code for training the CNN classifier accompany this user's guide. Fig. 1 lists the contents of the code folder. These contents include:

- "CNN-VAD-Android" is the Android Studio project. To open the project, open Android Studio, click on "Open an existing Android Studio Project" and navigate to the project "CNN-VAD-Android".
- "CNN-VAD-iOS" is the iOS Xcode project. To open the project, double click on "CNN_VAD.xcodeproj" inside the folder.
- "Training Code" contains the Matlab and python scripts required for the purpose of re-training the CNN model, if desired.

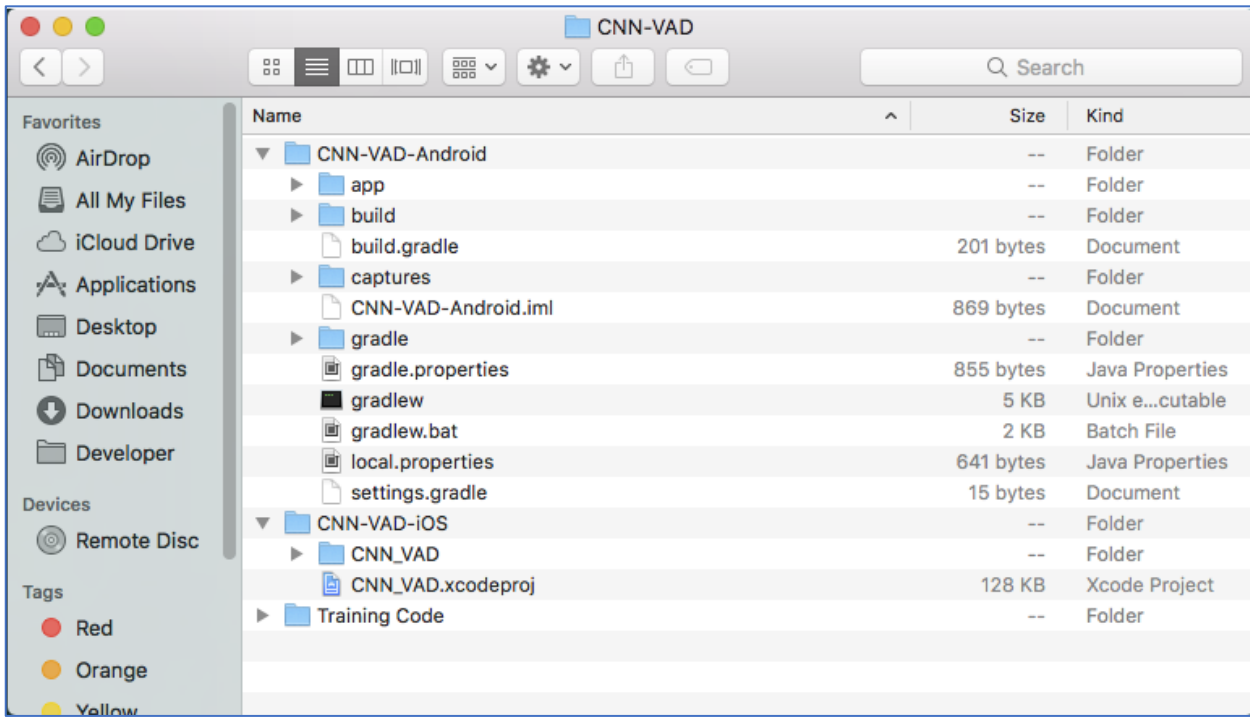
CNN-VAD
Search
Favorites
AirDrop
All My Files
iCloud Drive
Applications
Desktop
Documents
Downloads
Developer
Devices
Remote Disc
Tags
Red
Orange
Yellow
Name
Size
Kind
CNN-VAD-Android -- Folder
app -- Folder
build -- Folder
build.gradle 201 bytes Document
captures -- Folder
CNN-VAD-Android.iml 869 bytes Document
gradle -- Folder
gradle.properties 855 bytes Java Properties
gradlew 5 KB Unix e...cutable
gradlew.bat 2 KB Batch File
local.properties 641 bytes Java Properties
settings.gradle 15 bytes Document
CNN-VAD-iOS -- Folder
CNN_VAD -- Folder
CNN_VAD.xcodeproj 128 KB Xcode Project
Training Code -- Folder

Fig. 1

Part 1
iOS

Section 1: iOS GUI

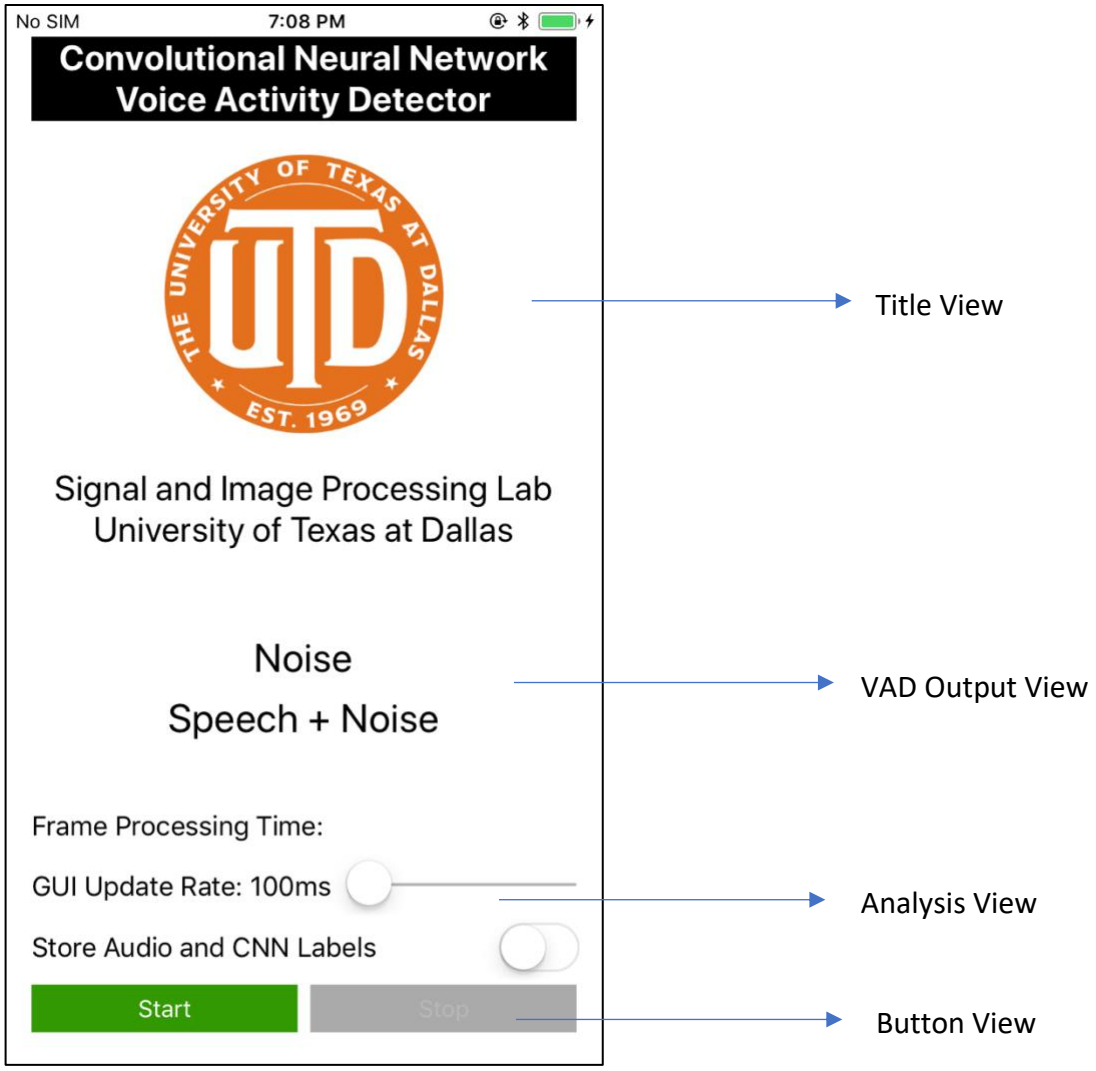

Fig. 2

This section discusses the GUI of the developed voice activity detection app and the options included in it, see Fig. 2. The GUI consists of 4 views:

1. Title View
2. VAD Output View
3. Analysis View
4. Button View

1.1 Title View

The title view displays the title of the app, the university logo and the lab name where the app was developed.

1.2 VAD Output View

The status view updates on a real-time basis, giving feedback to the user whether a current audio signal is noise or speech+noise and how much processing time is taken per audio frame. The displayed frame processing time is averaged over 0.5 second. The app allows adjusting the status view update rate.

1.3 Analysis View

This view allows the user to adjust the analysis options of the app.

- The user can adjust the GUI update rate of the app. High update rates reflect a more balanced VAD output, that is processing time with less fluctuations and more gradual output changes. Low updates reflect more instantaneous changes or fluctuations in the VAD output and processing time.
- The user is given the option to store or record the audio along with the CNN labels. These settings can be accessed using iTunes, as shown in Fig. 3, via the following steps:
    - Connect your device to the computer.
    - In iTunes, navigate to the device.
    - Select File Sharing and then select the app “CNN_VAD”.
    - You can download the files from there by selecting the files and clicking “Save To…”

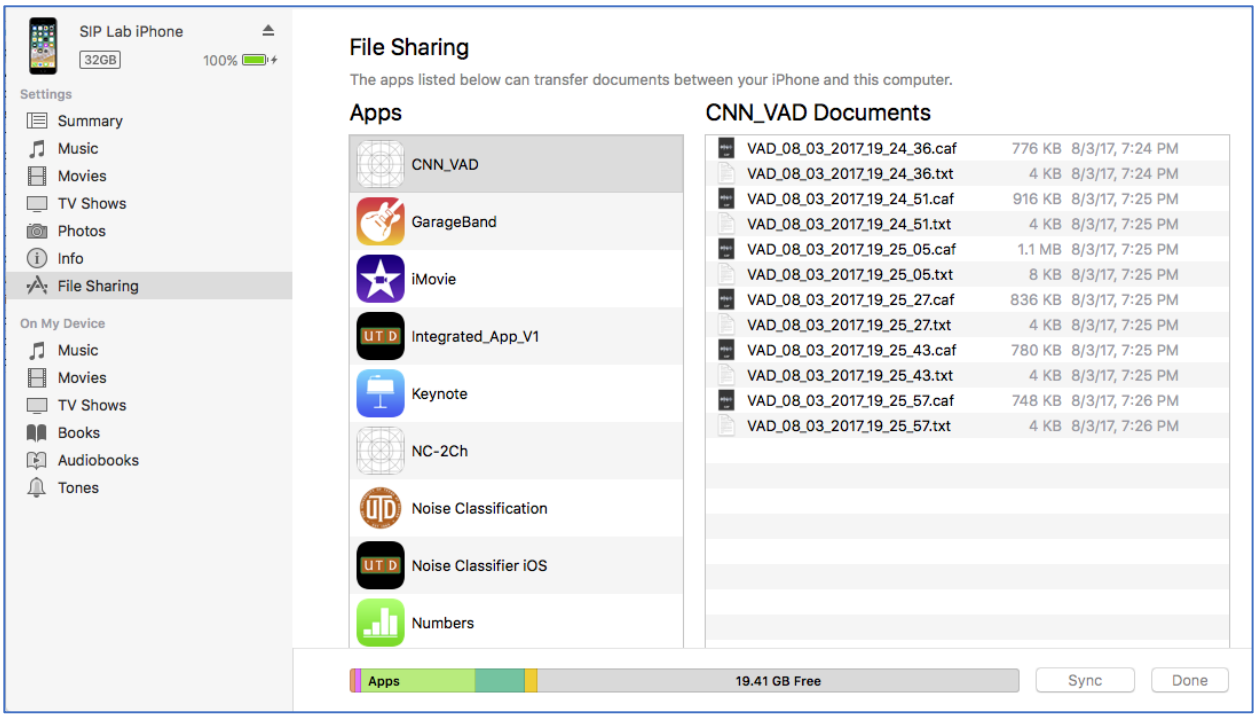

Fig. 3

1.4 Button View

This view contains the buttons that allow one to start a CNN VAD session or stop it. Once started, the Analysis view is disabled and one can make changes after stopping the session.

Section 2: Code Flow

This section discusses the app code flow. The app is designed to make the components modular and the code blocks or modules can be easily replaced. One can view the code by running “CNN_VAD.xcodeproj” in the folder “CNN_VAD_iOS” as shown in Fig. 1. The code is divided into 3 sections as shown in the project navigator in Fig. 4:

- View
- Audio IO
- Feature Extraction

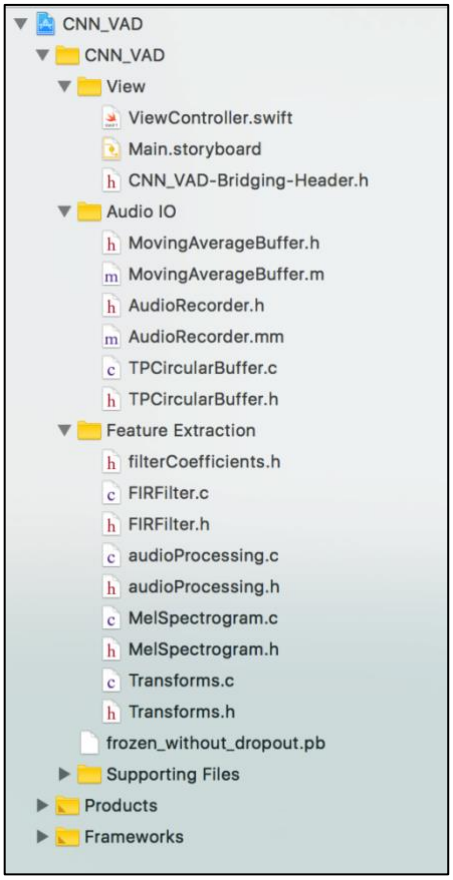

Fig. 4

2.1 View

The app view is done through the GUI. The initial setup of the GUI along with the screen interaction methods, e.g. button touch, slider adjust, etc., are mentioned here. The GUI consists of 3 parts:

- **Main.storyboard**: This denotes the layout of the GUI.
- **ViewController.swift**: This connects the GUI elements to actions. It is coded in Swift.
- **CNN-VAD-Bridging-Header.h**: This is a bridging header that allows calling the Objective-C functions declared in Audio IO in the Swift ViewController file.

2.2 Audio IO

The Audio IO section of the code handles the initialization of the audio setup of the app and the synchronous callbacks that collect and transmit audio from the smartphone:

- **MovingAverageBuffer**: This is a multi-purpose Objective-C class that creates an object to calculate the moving and total average of a data stream. In this app, it is used to average the processing time and VAD output to display on the GUI.
- **AudioRecorder**: This is the main component of the Audio IO. This class creates the audio i/o setup and initiates the input and output synchronous callbacks. The audio setup is designed to collect and play audio at the rate of 64 samples at 48 kHz to have the lowest latency. It also contains the CNN classification code.
- **TPCircularBuffer**: This is a data structure that synchronizes the audio i/o with the audio processing framework. The CNN VAD is a multi-rate signal processing application and the circular buffer enables synchronization of the modules. The input circular buffer collects 64 samples from the microphone till 12.5 ms or 600 samples of audio data are collected. These samples are then processed and fed to the output circular buffer and played back via the speaker/headphones at the rate of 64 samples.

2.3 Feature Extraction

The feature extraction code comprises the engine of the app. It processes each incoming frame and outputs whether the frame is speech+noise or noise. It is divided into the following parts:

- **audioProcessing**: This is the main component of the native code. It initializes all the settings and then assigns how each module should be called in order to classify audio frames. The following modules are used for the voice activity detection:
    - **FIRFilter**: This module is used to bandlimit the incoming and outgoing audio signal.
    - **Transforms**: This module computes the incoming FFT of audio frames.
    - **MelSpectrogram**: This module is used to extract the features required by the VAD and to create Log Mel Energy Spectrum images.
    - **filterCoefficients**: These are the filter coefficients for the FIRFilter module.

The above modules are written in a modular manner allowing one to add and replace audio processing, to perform feature extraction and voice activity detection tasks with ease.

Section 3: Adding Dependencies

To run the CNN inference code, Tensorflow C++ API is used. The Tensorflow API can be downloaded or cloned from this website:

https://www.tensorflow.org/install/

For this project Tensorflow version 1.3 was used and the following steps were taken to build the static library and implement it into the project:

- Execute the following command from the Tensorflow directory:

  `tensorflow/contrib/makefile/build_all_ios.sh`

  This will build the three static libraries required for iOS: `libtensorflow-core.a`, `libprotobuf.a` and `libprotobuf-lite.a`.
- After the static libraries are built, link it to the iOS project. Open the project and navigate to **Build Settings** in the **Project Settings**.
- Under **Other Linker Flags**, change the path to where your Tensorflow repository is located as depicted in Fig. 4.

Fig. 5

- Navigate to **Header Search Paths** and update their paths to your Tensorflow repository as shown in Fig. 5.

Fig. 6

- Apart from the above two changes, the other changes that need to be made are:
    - **Enable Bitcode** set to No
    - **Warnings/Documentation Comments** set to No.
    - **Warnings/Deprecated Functions** set to No.
- Now, build the project and run it on the connected iOS device.

Part 2

Android

Section 1: Android GUI

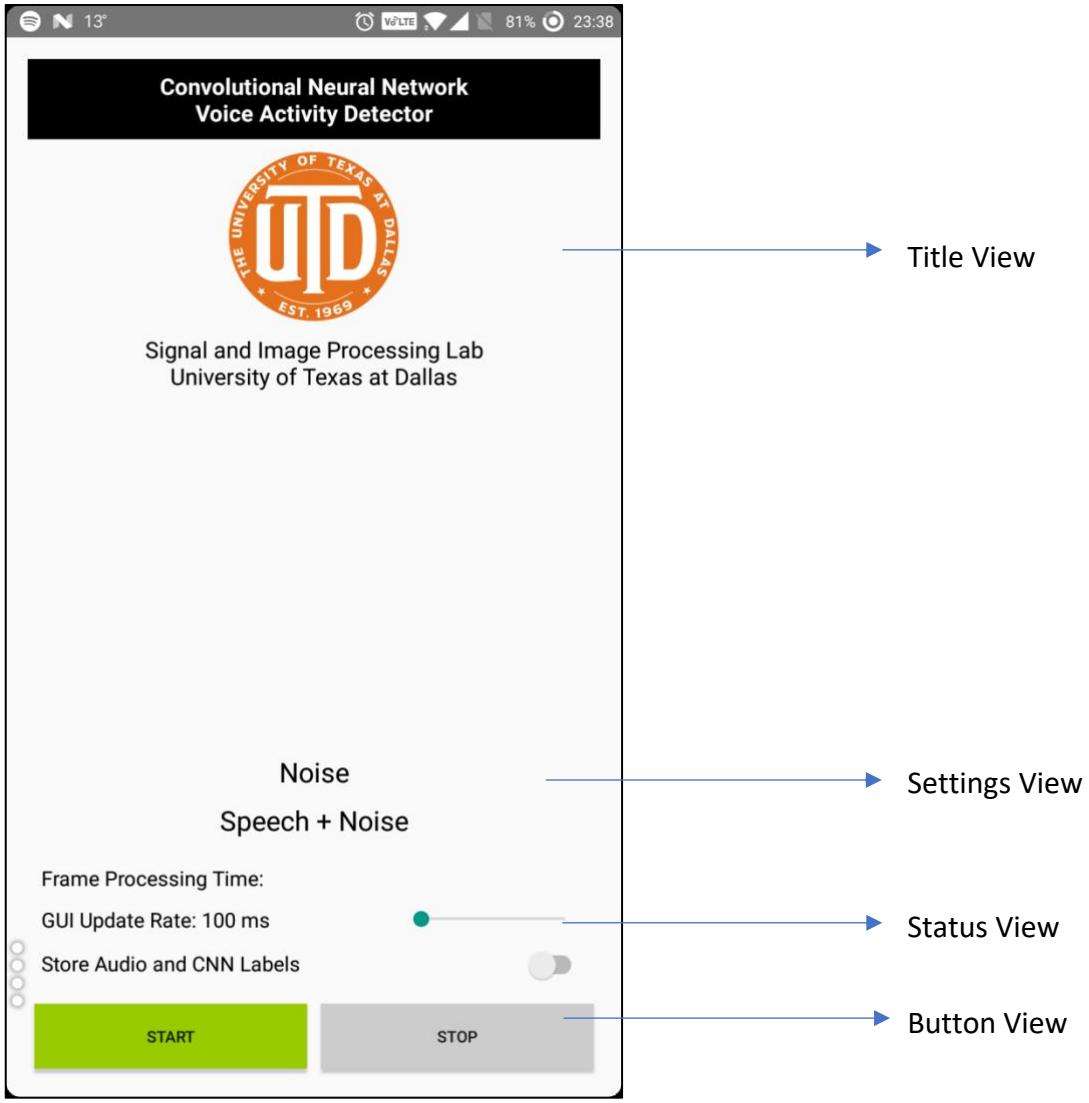

Fig. 6

This section discusses the GUI of the Android version of the app and the features of the GUI, see Fig. 6. The GUI is divided into 4 views:

1. Title View
2. VAD Output View
3. Analysis View
4. Button View

1.1 Title View

The title view displays the title of the app, the university logo and the lab name where the app was developed.

1.2 VAD Output View

The status view updates on a real-time basis, giving feedback to the user whether a current audio signal is noise or speech+noise and how much processing time is taken per audio frame. The displayed frame processing time is averaged over 0.5 second. The app allows adjusting the status view update rate.

1.3 Analysis View

This view allows the user to adjust the analysis options of the app.

- The user can adjust the GUI update rate of the app. High update rates reflect a more balanced VAD output, that is processing time with less fluctuations and more gradual output changes. Low updates reflect more instantaneous changes or fluctuations in the VAD output and processing time.
- The user can also store or record the audio along with the CNN labels. These files can be accessed in the device memory in a folder labelled “CNN-VAD-Android”.

1.4 Button View

This view contains the buttons that allow one to start a CNN VAD session or stop it. Once started, the Analysis view is disabled and one can make changes after stopping the session.

Section 2: Code Flow

To be able to run the app, it is necessary to have the Superpowered SDK on the smartphone device. This SDK can be acquired from the following website:

http://superpowered.com/

The Android shell can be opened in Android Studio by taking the following steps:

- Start Android Studio
- Select “Open an existing project”
- Navigate to the folder “CNN-VAD-Android”
- Click open

After the project is opened, navigate to the file “local.properties” in the project. In the project, make sure that “sdk.dir” (Android SDK), “ndk.dir” (Android NDK) and “superpowered.dir” (Superpowered Audio Engine) point to the proper location on the hard disk. Clean your project and rebuild it so that the project can be run on the smartphone device.

2.1 Project Organization

The project is organized in the following way in Android Studio which can be seen under the view “Android” in the project navigator as depicted in Fig. 7:

- **java**: The java folder contains the file “MainActivity.java”. This file handles all the operations of the app and allows one to link the GUI to the native code.
- **cpp**: This folder contains the native code. This folder has 2 subfolders.
    - **AndroidIO**: This subfolder contains the Superpowered source root files which control the audio interface to the app.
    - **jni**: This subfolder includes all the native code required to start the audio I/O and also the voice activity detection files. This code provides the audio processing part of the app.
    - **assets**: The assets folder contains the trained CNN model.
    - **jniLibs**: This folder contains the libraries required to run the CNN inference. All the folders for various Android devices are included.

Both the **assets** and **jniLibs** folder are crucial to run the app. These are included as part of the project and it is not required to run any code to compile the .so files as is the case for the iOS version.

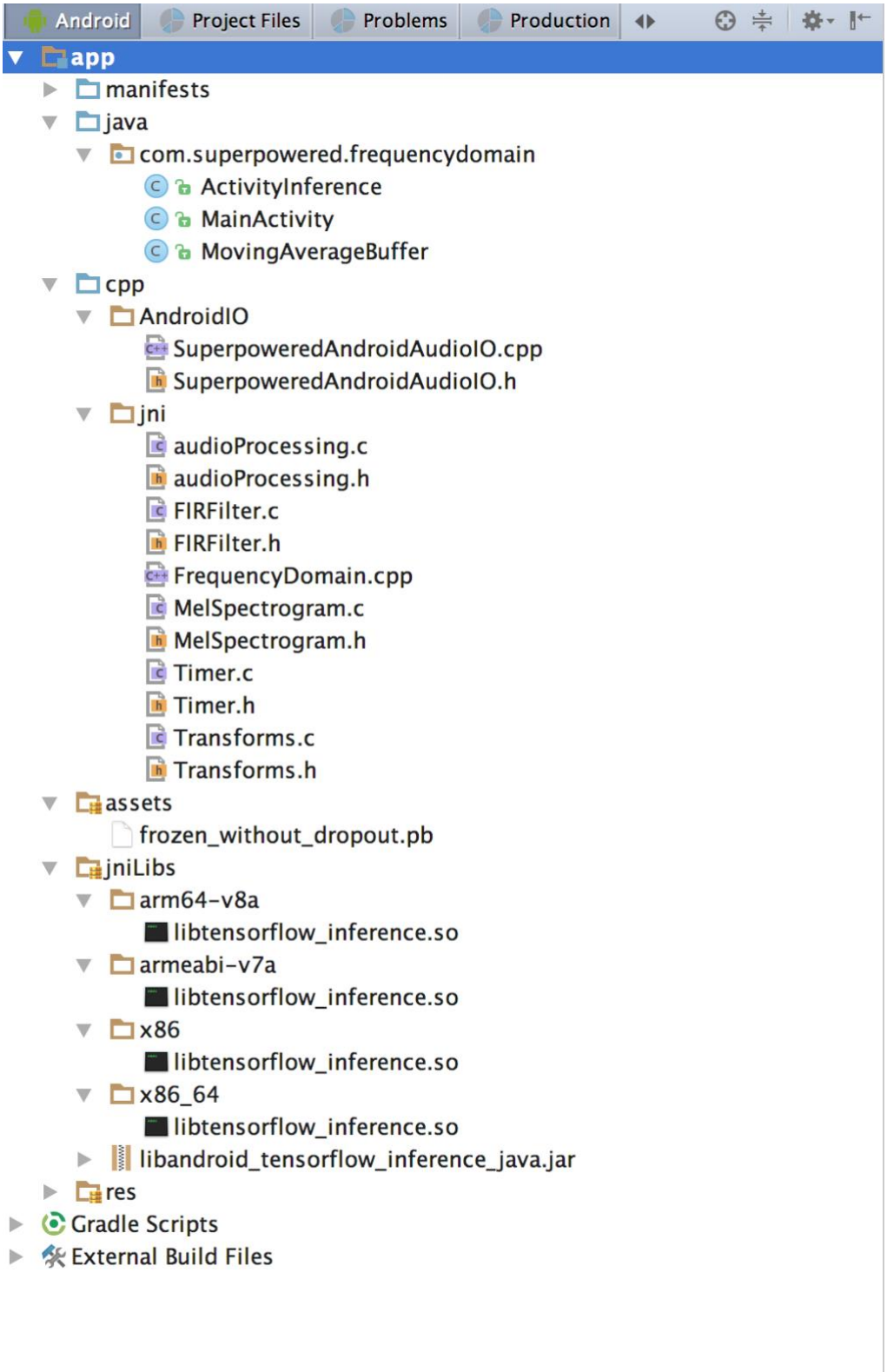
Android
Project Files
Problems
Production
app
manifests
java
com.superpowered.frequencydomain
ActivityInference
MainActivity
MovingAverageBuffer
cpp
AndroidIO
SuperpoweredAndroidAudioIO.cpp
SuperpoweredAndroidAudioIO.h
jni
audioProcessing.c
audioProcessing.h
FIRFilter.c
FIRFilter.h
FrequencyDomain.cpp
MelSpectrogram.c
MelSpectrogram.h
Timer.c
Timer.h
Transforms.c
Transforms.h
assets
frozen_without_dropout.pb
jniLibs
arm64-v8a
libtensorflow_inference.so
armeabi-v7a
libtensorflow_inference.so
x86
libtensorflow_inference.so
x86_64
libtensorflow_inference.so
libandroid_tensorflow_inference_java.jar
res
Gradle Scripts
External Build Files

Fig. 7

2.2 Java Code

- **ActivityInference:** This class is used to run the CNN code and classify whether the incoming audio contains speech or not.
- **MainActivity:** This class is used to initialize all the GUI elements and link the audio processing code written in C and C++ with the GUI elements written in Java.
- **MovingAverageBuffer**: This is a multi-purpose java class which creates an object to calculate the moving and total average of a data stream. In this app, it is used to average the processing time and VAD output to display on the GUI.

2.3 Native Code

The native code is the engine of the app. It processes each incoming frame and outputs the detected noise type. It is divided into the following parts:

- **FrequencyDomain:** This is the C++ file that acts as a bridge between the native code and the java GUI. This file is responsible for creating the audio I/O interface with the settings appearing in the GUI, for providing the audio data to the processing code frame-by-frame, and for providing the result back to the GUI.
- **audioProcessing**: This is the main component of the native code. It initializes all the settings and then assigns how each module should be called in order to classify audio frames. The following modules are used for the voice activity detection:
    - **FIRFilter**: This module is used to bandlimit the incoming and outgoing audio signal.
    - **Transforms**: This module computes the incoming FFT of audio frames.
    - **MelSpectrogram**: This module is used to extract the features required by the VAD and to create the Log Mel Energy Spectrum image.
    - **filterCoefficients**: These are the filter coefficients for the FIRFilter module.

The above modules are written in a modular manner allowing one to add and replace audio processing, and to perform feature extraction and noise classification tasks with ease.

Part 3
Training

Section 1: Creating CNN Model

The CNN VAD uses a pre-trained model that can be used by both iOS and Android shells without any alteration. To train the model, noisy speech data first need to be generated from existing noise and speech databases. This noisy speech data are used to extract features and the features are then used to train the CNN model to be running on smartphones. The scripts required for achieving the above are shown in Fig. 8. The folder named "Functions" contains the secondary functions required for the scripts to run.

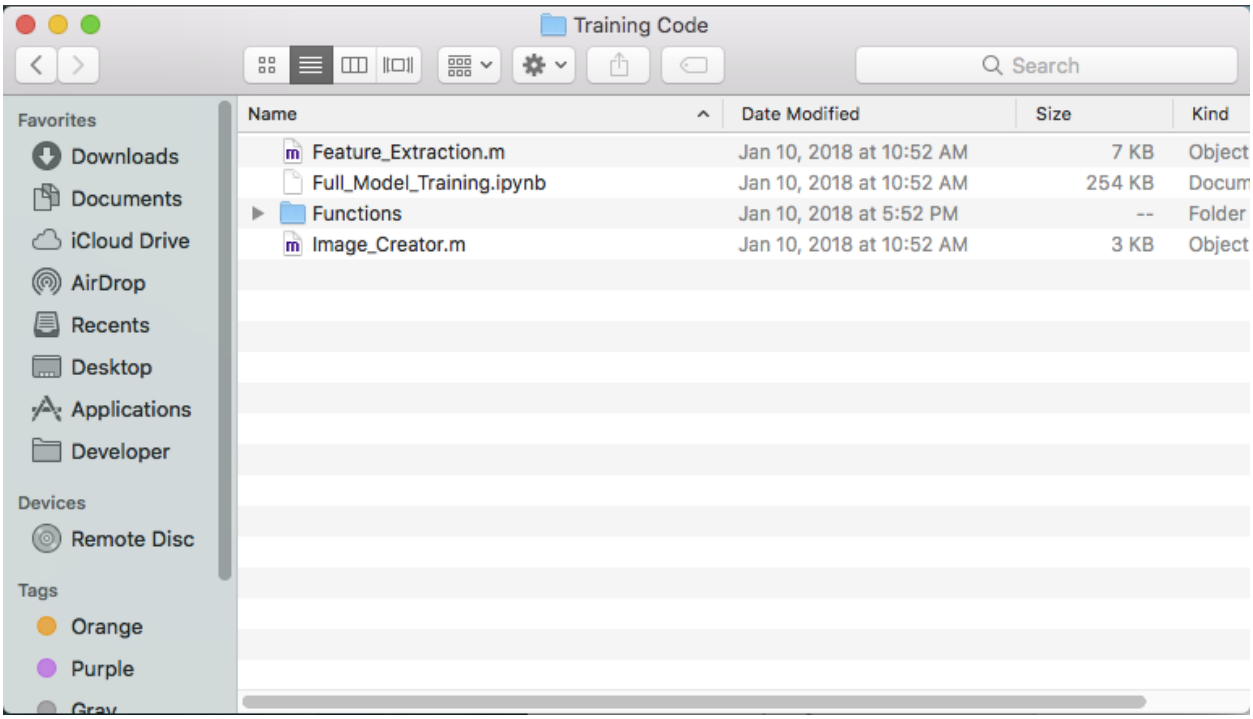

Fig. 8

- To create a noisy speech database, noise and speech data are needed. To train the CNN model, the following databases were used:
    - Speech: PNNC Speech Corpus
      https://depts.washington.edu/phonlab/resources/pnnc/pnnc1/
    - Noise: DCASE 2017 Noise Database
      https://zenodo.org/record/400515
- These databases were used to generate noisy speech at different SNRs. The SNR can be specified in the script "Feature_Extraction.m". This script generates the log-mel energy

features per frame and stores them as a .mat file named “VAD_features_SNR_[specified SNR].mat”.

- The generated features are then used to generate the images using the script “Image_Creator”. The images are generated in dimensions of 40x40 with a new image generated after every 5 frames. These images are stored in a .mat file named “VAD_Training_data_SNR_[specified SNR].mat”.
- After the images are generated, the classifier is trained using the Python notebook script “Full_Model_Training.ipynb”. In this script, all the training data extracted from the Matlab scripts are provided which can be used to train the CNN model.The training script creates the graph and stores the weights as checkpoints based on the number of epochs completed. Once a specified number of epochs is reached, as illustrated in Fig. 9, the model can be frozen for inference.

```
        for i in np.arange(0, nDataSamples, trainBatchSize):
            x_batch = data[idx[i:i+trainBatchSize], :, :]
            y_batch = labels[idx[i:i+trainBatchSize]]

            feed = {x: x_batch,
                    Y_: y_batch,
                   learning_rate: learningRates[epoch],
                   keep_prob: 0.75}
            sess.run(train_op, feed_dict=feed)

            if i%50*trainBatchSize == 0:
                feed = {x: x_batch,
                        Y_: y_batch,
                        learning_rate: learningRates[epoch],
                        keep_prob: 1.0}

                train_accuracy, loss_value = sess.run([accuracy, cross_entropy], feed_dict=feed)
                print("epoch: %2d step: %6d, Training Accuracy: %3.2f, loss: %6.4f" % \
                      (epoch, i, train_accuracy*100, loss_value))

        tf.gfile.MakeDirs(checkpoint_dir + '/model' + str(epoch))
        checkpoint_file = os.path.join(checkpoint_dir + '/model' + str(epoch), "model")
        saver.save(sess, checkpoint_file)
        print("**** SAVED MODEL ****")
        print("**** COMPLETED EPOCH ****")
epoch: 11 step: 883200, Training Accuracy: 98.83, loss: 26.5467
epoch: 11 step: 896000, Training Accuracy: 99.22, loss: 24.2738
epoch: 11 step: 908800, Training Accuracy: 97.66, loss: 39.1677
epoch: 11 step: 921600, Training Accuracy: 98.63, loss: 36.7975
epoch: 11 step: 934400, Training Accuracy: 98.44, loss: 31.3061
epoch: 11 step: 947200, Training Accuracy: 98.05, loss: 40.8064
epoch: 11 step: 960000, Training Accuracy: 97.27, loss: 41.2463
epoch: 11 step: 972800, Training Accuracy: 99.22, loss: 20.1818
epoch: 11 step: 985600, Training Accuracy: 98.44, loss: 34.1938
epoch: 11 step: 998400, Training Accuracy: 97.27, loss: 46.0204
epoch: 11 step: 1011200, Training Accuracy: 98.24, loss: 35.3947
epoch: 11 step: 1024000, Training Accuracy: 97.66, loss: 36.6739
epoch: 11 step: 1036800, Training Accuracy: 96.29, loss: 59.9130
epoch: 11 step: 1049600, Training Accuracy: 98.05, loss: 33.2217
epoch: 11 step: 1062400, Training Accuracy: 97.66, loss: 46.9731
epoch: 11 step: 1075200, Training Accuracy: 97.27, loss: 56.4713
epoch: 11 step: 1088000, Training Accuracy: 97.85, loss: 35.1326
**** SAVED MODEL ****
**** COMPLETED EPOCH ****

In [ ]:
```

Fig. 9

- After the training is completed, the graph is to be frozen with the desired weights and the frozen graph should then be used to run the CNN on a smartphone device. Freezing the graph means that all the training modules are removed and only the inference part of the CNN is kept.

  The graph is stored as a .pbtxt file and the trained weights and biases after every epoch are stored in the folders named model[epoch number], see Fig. 10. Once the model reaches a desired accuracy, the graph can be frozen to use the trained weights and biases from the epoch with the highest accuracy.

  The steps needed to freeze the graph can be found at:
  https://www.tensorflow.org/extend/tool_developers/#freezing

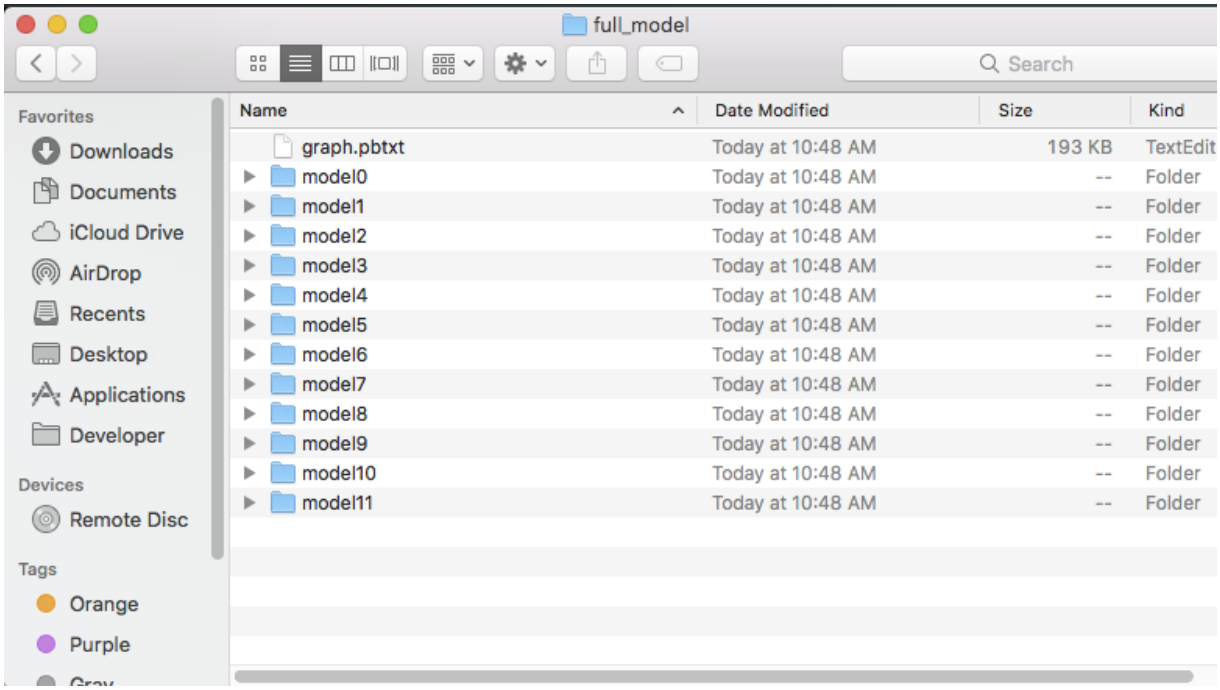

Fig. 10