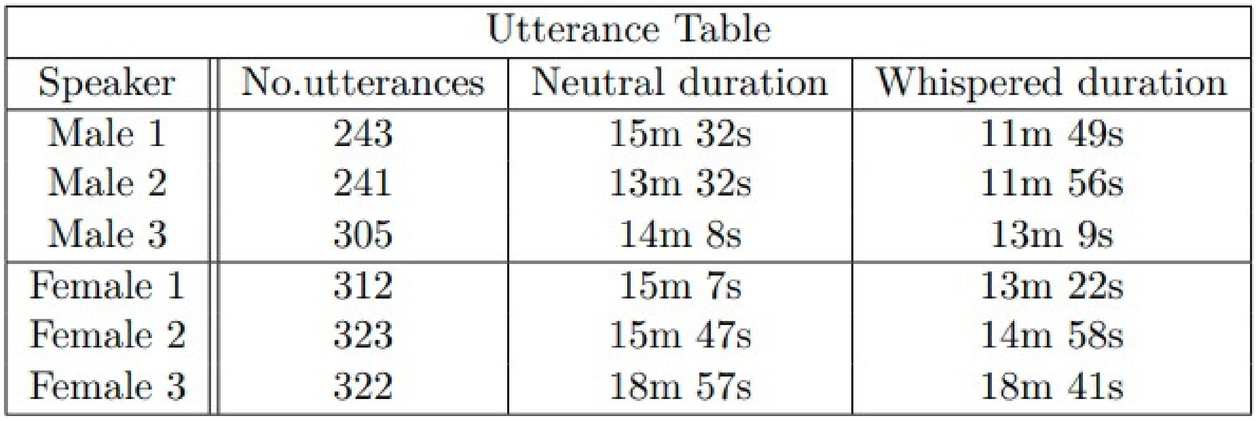
Setting up a new conversion

In `setup_new` folder, there are two subfolders. One fore single speaker conversion, the other for gender generalized speaker conversion. The only difference is in the “speaker” folders, which containt the pairs of whispered and neutral utterances. In preparation of the new custom models, the first thing to be done is silence removal. That is done using the `sil_rm.py` script, which takes three input arguments: `-n, -f, -sr` standing for number of utterance pairs, frame period and speaker, respectively. For clarification, the number of utterance pairs for each speaker is specified in this table:



Utterance Table

Speaker	No.utterances	Neutral duration	Whispered duration
Male 1	243	15m 32s	11m 49s
Male 2	241	13m 32s	11m 56s
Male 3	305	14m 8s	13m 9s
Female 1	312	15m 7s	13m 22s
Female 2	323	15m 47s	14m 58s
Female 3	322	18m 57s	18m 41s

Frame period can be experimented with, but the default value is set to 5.0, and it is recommended to stick with it. As per speakers – the speaker name should be specified in the format in which the folders are named i.e. `male_1, female_3` etc. For the gender generalized training, only two options are: `male` and `female`. With 789 utterance pairs for male and 957 pairs for female.

The silence removal script creates a `speaker_sr` folder (speaker name dependent) from which the next script loads the files for WORLD analysis. This script `floader.py` requires the speaker and number of pairs arguments as well. This script takes long to execute, since it opens every single pair of utterances, analyzes the audio and saves the features into a file. Next step is to separate the features into separate files (for any experimental uses that might come to mind) and, more importantly, pad the feature sequences to the length of the longest one. This is crucial for sequence-wise mini-batch training. The script that allows that, `sep_ftr.py` takes only the speaker argument, and creates all necessary folders. Next, the padded sequences need to be assigned into either Train, Test or Validation (True_test) datasets. That is the work for

`train_test_sep.py` script, which as well takes only the `-sr` argument. It is important to mention that the training loop defined in `main.py` uses two other python scripts – `dataTest.py` and `lstm.py` – the first one constructs the Datasets using PyTorch utilities and makes any datatype conversions and array-wise operations necessary. The second one defines the Neural Network module consisting of one LSTM and one linear layer.

The training loop defined in `main.py` only takes a speaker argument, but the setting can be tweaked a bit. Namely, the number of epochs, learning rate, number of threads (workers), the batchsize and weight decay can be experimented with. This training loop requires a CUDA architecture support on host computer, otherwise the training won’t start. If the host computer does not have the necessary hardware, the code can be tweaked on the 32nd line of the `main.py` script by changing the `model.cuda()` to `model.cpu()` as well as in similar manner editing the 68th and 69th lines which parse the variables into the model. After the model is trained, the inference can be done using the `test_models.py` script which, again, takes only the speaker argument as input. This script will take the True_test utterances and make the conversion over them.

The above mentioned process executed in terminal can look something like this:

```
$ python3 sil_rm.py -n 243 -f 5 -sr male_1
```

…

```
$ python3 floader.py -n 243 -sr male_1
```

…

```
$ python3 sep_ftr.py -sr male_1
```

…

```
$ python3 train_test_sep.py -sr male_1
```

…

```
$ python3 main.py -sr male_1
```

…

```
$ python3 test_models.py -sr male_1
```

Converting using already trained models

In the `conversion` folder, there are two scripts and a model folder. The model folder contains all trained models for each speaker, as well as the generalized ones. The two scripts, as their names suggest, work either with the single speaker or the generalized ones. These scripts can convert any provided wav file, as long as it is in the same folder. Example usage looks like this:

```
$ python3 final_convertor_single.py -f example.wav -g male -s
male_1
```

or, for generalized model:

```
$ python3 final_convertor_gen.py -f example.wav -g male
```

In each case, the script creates a `example_converted.wav` file with the converted speech.

The application

The second subfolder called `app` contains an `app.apk` file. This is the application instalation package, that can be installed on Android system. Because the application (due to the fact it sends raw voice data on server) could not be sanctioned by Google Play, it is unsigned and potentional warning messages can pop up. The user may as well need to allow the installation of unknown applications when prompted. When the application is installed, it can be run. First the user needs to specify the gender (necessary for model selection). Then user can record an utterance by pressing Start Recording button. It is advised to keep these utterances short (under 5 seconds) since the server is running on a low-budget platform and may not have the processing power to complete the conversion before the http request times out. If that happens, the application says the conversion failed, and user is prompted to try again. After recording the desired utterance, by pressing Stop And Convert button, the application sends the data to the Flask server, where the conversion is done. User is prompted to wait. If the conversion failes, as mentioned, the user can try again. If the conversion is successful, the converted audio can be played back using the Play Converted (and paused by the Stop Playing) button. Note that the quality of synthesized speech may vary greatly as described in the thesis text.

The source for application

The app folder also contains source codes for both Flask server and the application.

The `app-source` folder contains the Android Studio project with all dependencies and build utilities. Note that Android Studio IDE is required to run the application from the project source code. To do that, simply open up Android Studio and in `File > Open` find the way to the `app-source` directory. The project will load and the application can be run virtually on emulated device (which needs to be installed) but I wouldn't recommend that – the image takes up several GB of space and cannot process sound from your OS input. Better practice is to connect your smartphone. Turn on the developer mode in your Android OS settings – that is described here:

`https://developer.android.com/studio/debug/dev-options`

After that connect your phone via USB to your computer. The Android Studio automatically connects and switches automatic build device to your physical phone. After that, press build and run button and the application should start (after building gradle and downloading dependencies). Note that the application will be kept on even if disconnecting from the computer, so be sure to exit it properly.

The source for server

Lastly, the application sends the data to a Flask server running on Heroku. The source folder contains three python scripts: `app.py, conv.py` and our `lstm.py`.

The first script defines the flask server and essentially calls upon the other two in order to execute the conversion. The flask application works on two addresses running on port 5000– `/post` which recieves the http request from the android application. The request body contains the source whispered utterance and gender specification. The file is saved and `conv.py` is called upon. After the conversion is done, the converted waveform is saved. The second address `/get` simply returns the audio file to the user. The server can be setup from the `server-source` directory directly. One way, which is used in the thesis – is creating a Heroku account and following these steps:

https://stackabuse.com/deploying-a-flask-application-to-heroku/

except the `pip freeze requirements` which is unnecessary, since the requirements file is already present.

A faster and easier (recommended) way to test the server is running the Flask server locally. That can be done using a virtual environment (which should be set up in the same folder)

The setup is done followingly:

```
$ python -m venv venv/
$ source venv/bin/activate
```

Now the virtual environment should be set up.

Run these following commands to install gunicorn and flask

```
$ pip install flask
$ pip install gunicorn
```

Now, run the flask server using

`$ flask run` in the virtual environment terminal

The server should be running on localhost (typically 127.0.0.1:5000)

Now, place a convertible wav file (could be the example.wav from the conversion section) into the same folder as the flask app.

Now run this `curl` command (install `curl` if needed):

```
curl -X POST localhost:5000/post -F 'file=@example.wav' -i
```

The conversion should be done on the server side and the file saved in the server folder.