Documentation

# Adding Speech and Noise

## Prerequisites

* Normalises speech and noise to reduce or enhance audio to a certain range for accurate noisy speech.
* If required, downsampling audio to 16kHz for both speech and noise will both either increase or decrease the sampling rate of the audio to 16kHz (If not done, then the sampling rate may be higher)

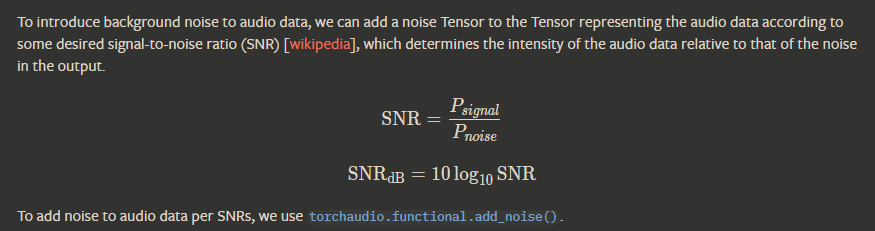
## Steps

### Step 1) Randomise the noise files.

### Step 2) In order of files from speech and noise load, the waveforms

### Step 3) From a random point in the noise waveform, make the size of the noise the same as the speech (Depends if the duration of noise is longer than the duration of speech)

### Step 4) At all SNR levels, add the noise to speech using torch audio to add\_noise



<https://pytorch.org/audio/main/generated/torchaudio.functional.add_noise.html>

# CNN for noisy speech

## First model

<https://github.com/jeffprosise/Deep-Learning/blob/master/Audio%20Classification%20(CNN).ipynb>

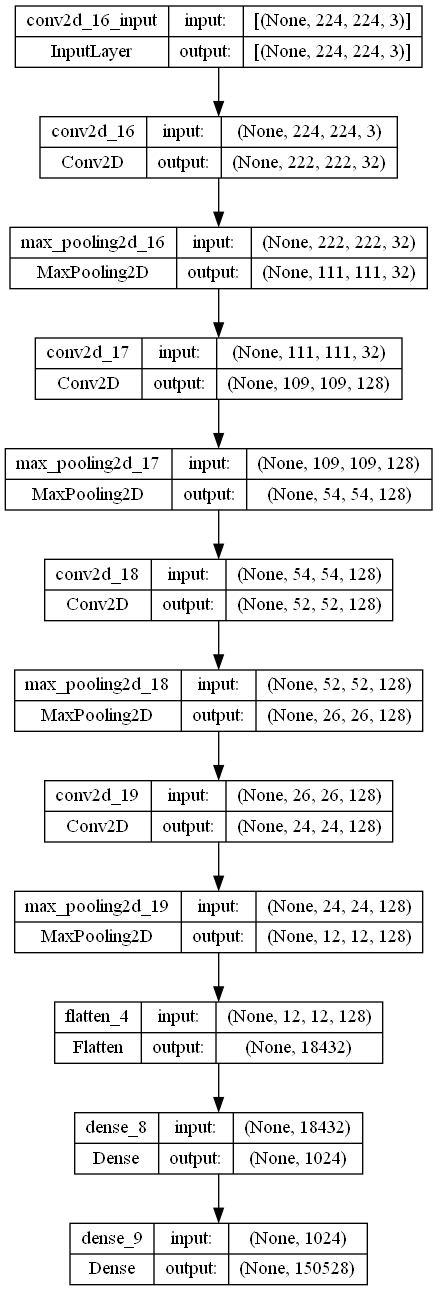
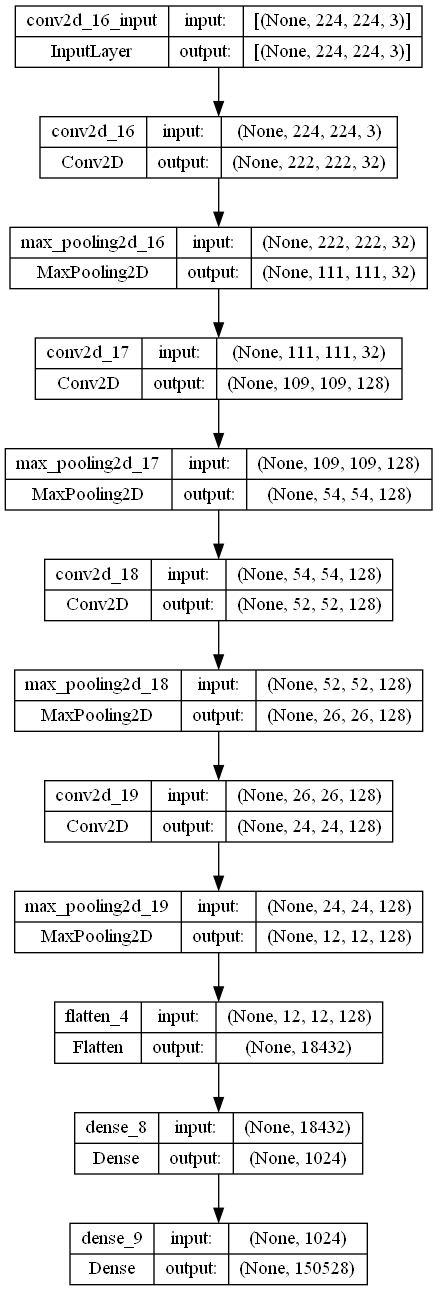
The method used is an image-image regression in machine learning, which is used to extract features of noisy speech with the help of clean speech.

### Prerequisites

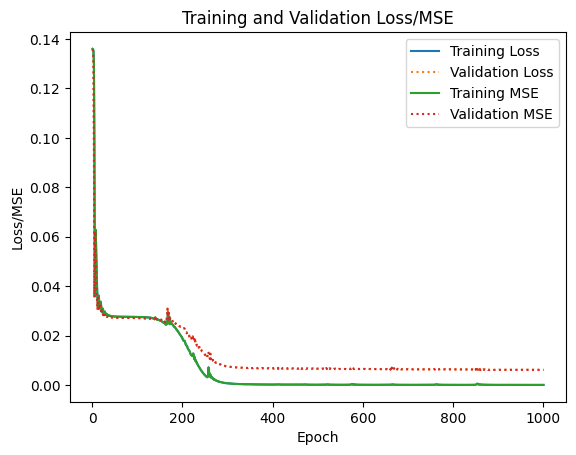
* Requires both noisy speech and clean speech wav files

### Steps

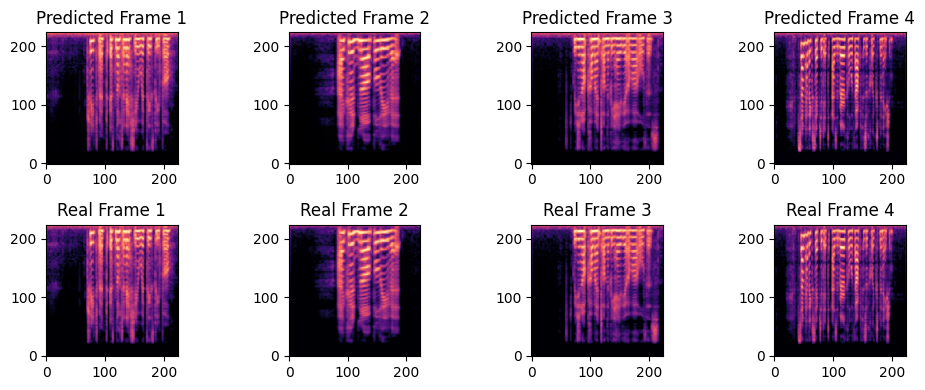
1. Create the spectrograms using the audio data and save them as png
2. Save all noisy speech images into x and clean speech images into y in the size of (224, 224, 3) [A standard size of CNN training]
3. Dividing all the images by 255 to make the values in the range of 0 to 1
4. Split the training and testing data by 80/20, respectively.
5. Then create the model with an input size of 224,224,3 and an output of 224\*224\*3 = 150,528



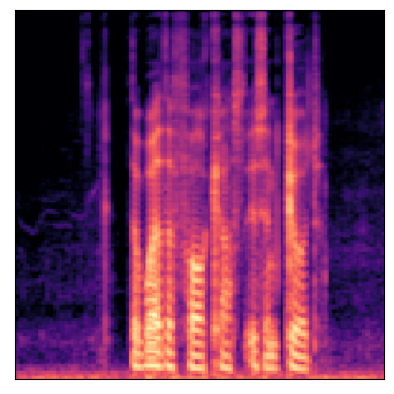
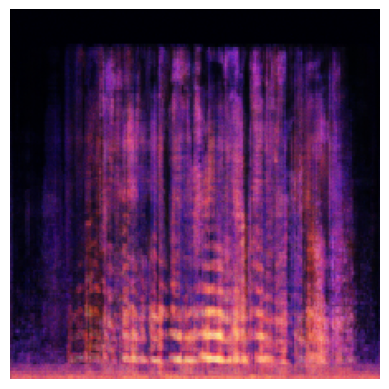
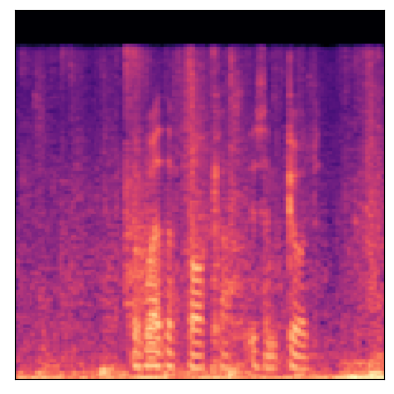
1. The model is compiled with a loss of mean squared error (MSE)
2. The model fits with a batch size of 1000 and an epoch of 1000
3. After training, the model resulted in a curve showing no loss of MSE after about 250 training iterations, which could mean that training with the 150 noisy data resulted in overfitting.
4. While the validation resulted in a constant 0.01 loss as the continuous training did nothing after 300 epochs



1. For the validation frames, it shows similar results with minimal loss



1. However, with new testing data, the loss is significantly more.



Noisy speech | Predicted speech | Clean Speech

### Issues

* The preprocessing method of reducing the size of the spectrograms when training is good for classification as the features are the important aspects but for regression, it results in features being lost and may result in the end audio being inaudible.
  + The speech should be windowed instead of losing information.
  + We also need to remove one of the used noises for testing.

References

<https://github.com/lucko515/speech-recognition-neural-network/blob/master/vui_notebook.ipynb>

<https://github.com/haoxiangsnr/Wave-U-Net-for-Speech-Enhancement>

<https://github.com/jerrygood0703/speech-enhancement-WGAN>

<https://github.com/betegon/Wave-U-Net-For-Speech-Enhancement-1>

<https://github.com/jeffprosise/Deep-Learning/blob/master/Audio%20Classification%20(CNN).ipynb>

Gives nice image of layers

<https://github.com/EncoraDigital/SAB-cnn-audio-denoiser/blob/master/SpeechDenoiserCNN.ipynb>

<https://paperswithcode.com/task/speech-denoising>

<https://malaya-speech.readthedocs.io/en/latest/load-speech-enhancement.html>

<https://bamblebam.medium.com/audio-classification-and-regression-using-pytorch-48db77b3a5ec>

<https://www.tensorflow.org/tutorials/keras/save_and_load>

<https://github.com/chrisdonahue/wavegan>

# Creating the dataset

## Prerequisites

* Normalises speech and noise to reduce or enhance audio to a certain range for accurate noisy speech.
* If required, downsampling audio to 16kHz for both speech and noise will both either increase or decrease the sampling rate of the audio to 16kHz (If not done, then the sampling rate may be higher)
* DEMAND provides 15 recordings that are 5 mins
* VCTK provides 109 native English speakers with approximately 400 passages each of about 5 - 10 seconds

## Info

* Any number of dataset
* Randomised method to create noisy speech and clear speech
* SNR = 0dB, -3 dB, -6dB and -9dB
* Excel contains sample number, name, length, type of speech, SNR, noise type, noise channel, speaker ID and passage ID.

## Steps

1. Create files for all data and classification with respective Excel sheet
2. Locate speech and noise
3. Normalise and downsample speech from 48K to 16K and normalise noise functions.
4. Add noise to speech function.
5. Create the dataset and add information to the Excel sheet.

# Classification model

The model uses the classification of both clean and noisy speech

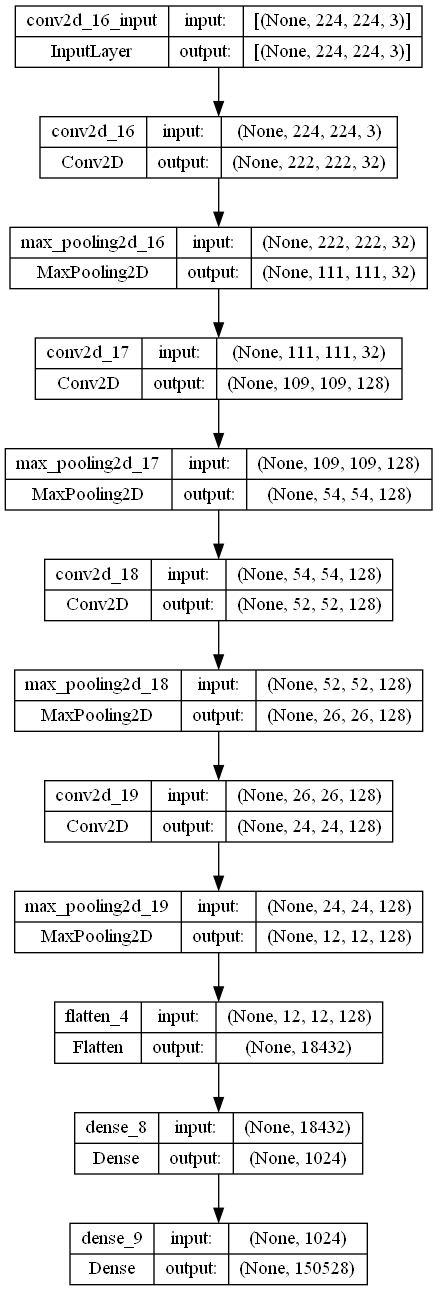
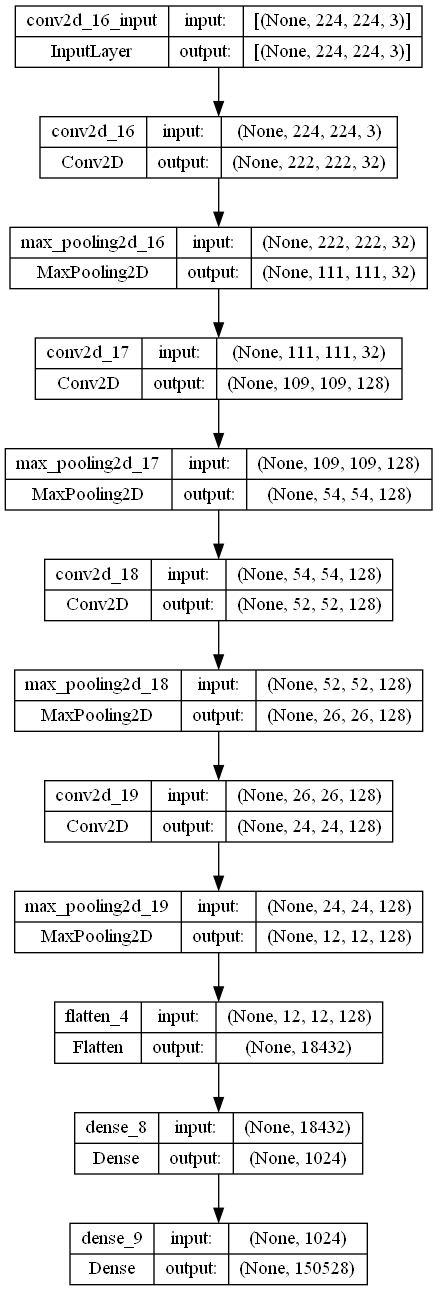
<https://github.com/jeffprosise/Deep-Learning/blob/master/Audio%20Classification%20(CNN).ipynb>

### Prerequisites

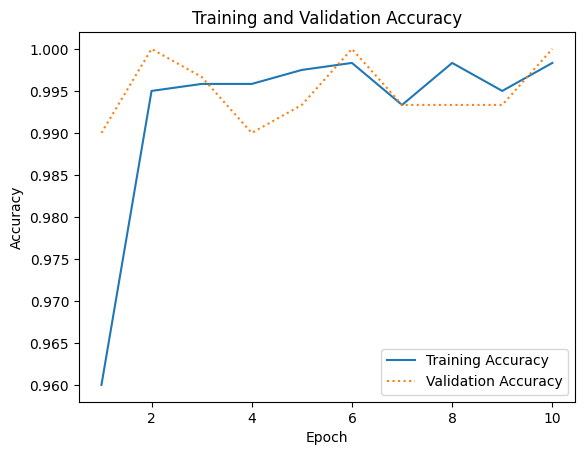
* Requires noisy and clean speech dataset for training
* Requires clean and noisy speech dataset for testing

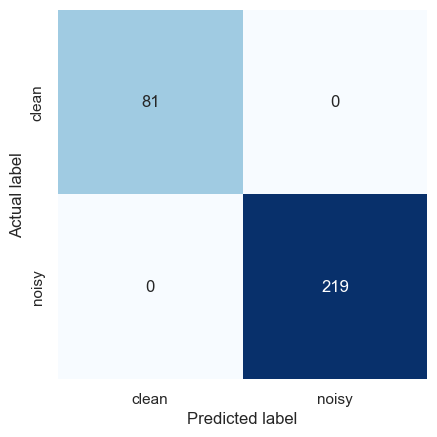
### Steps

1. Functions for creating spectrograms and PNGs from WAV files
2. Converting wav to png and saving images as x and labels as y (0 is clean and 1 is noisy)
3. Splitting the dataset into training and testing 80/20, respectively

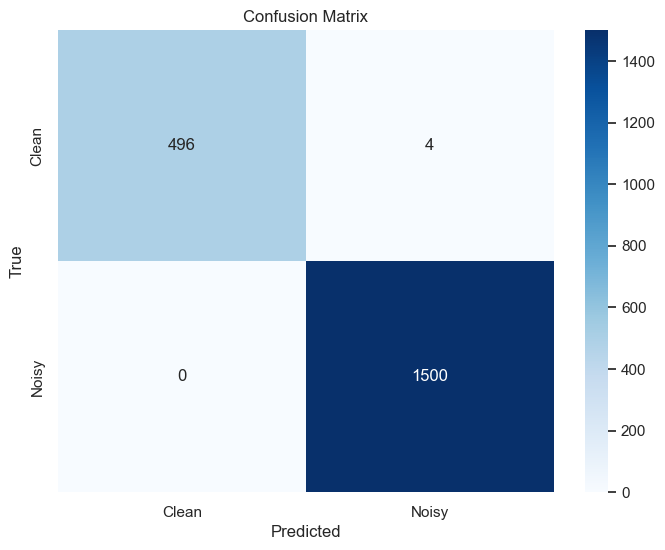


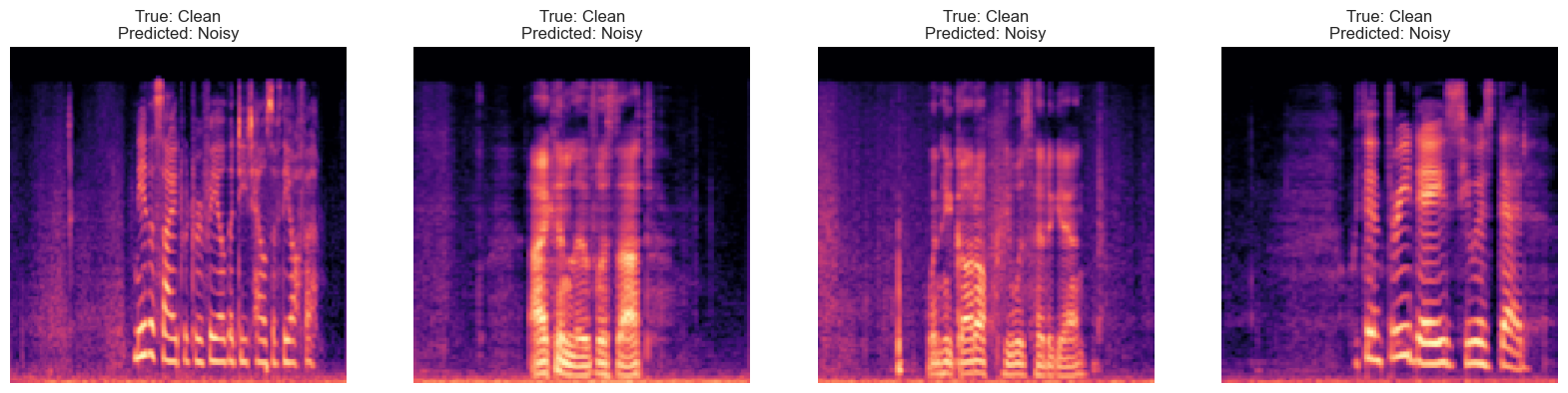
1. Training the model with the batch size and epoch number of 10
2. Evaluation





1. Saving and loading the model





### Considerations

* This method clearly has not trained enough with either clean speech that has small background noise or clean speech that has continuous audio

# Classification model remake

## Motivation

* The classification model distinguishes between noisy and clean speech by understanding that clean speech usually doesn’t start until after 2 seconds of the audio, and only a section of the spectrogram is created when clean.
* Our GMM model may not result in the same speech as what the clean speech is represented as, so we would require tolerance to distinguish between different SNR dBs

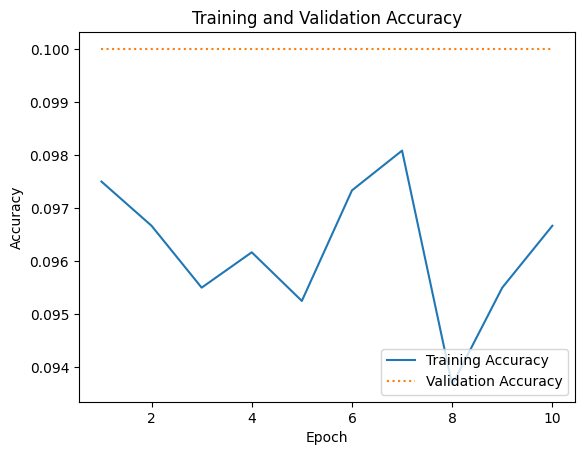
## Methods

* Creating more classifications for different SNRs
* Using a speech detection method before starting the creation of a spectrogram

## METHOD 1)

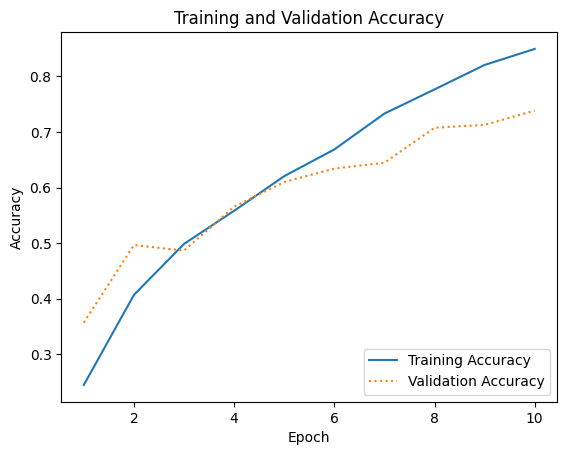
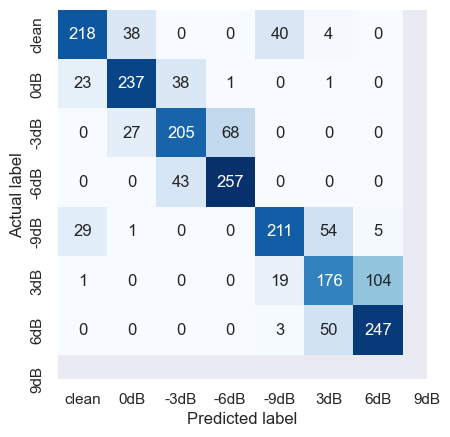
Creating more classifications for different SNRs

### 1.1) Using a range of lower SNRs, such as from 0 to -9dB



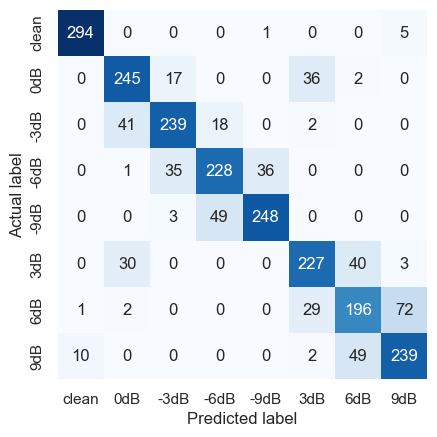
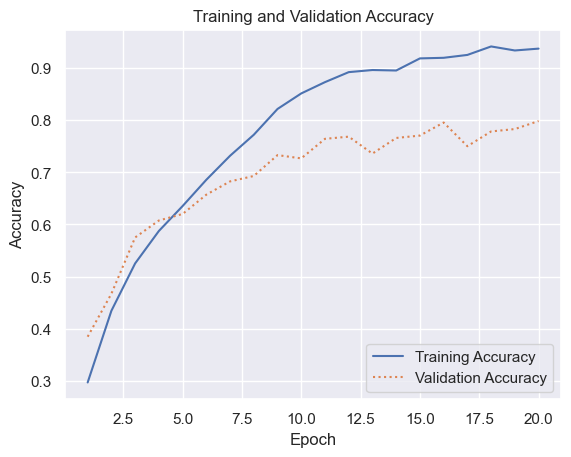
* All of the spectrograms resulted in a prediction of -7dB 1/10 accuracy, but that was only because it thought everything was -7dB

### 1.2) Using a range of SNR from 9 to -9dB



* Having a wider range of SNRs with the values being spread results in more accurate results of 0.7 accuracies after training with 10 epochs and 10 batches
  + As the method still has room for improvement, training will progress.

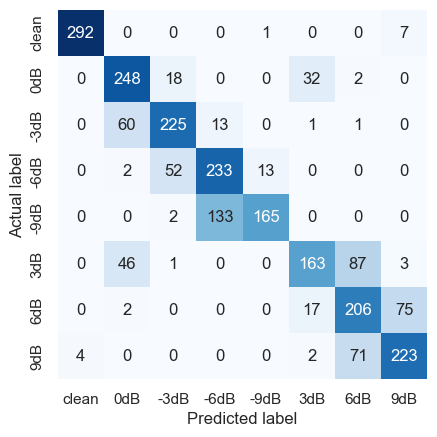
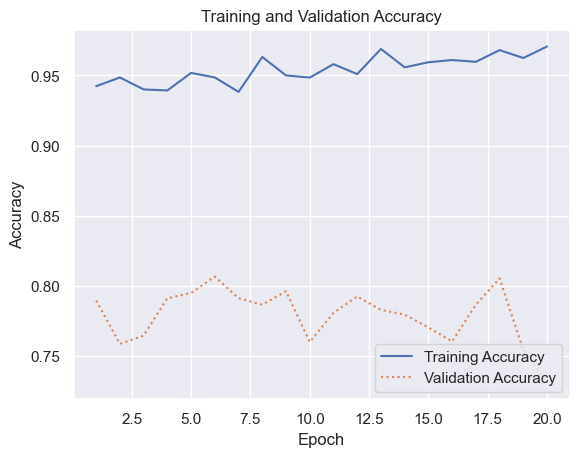
### 1.3) Increasing the epoch to 20



* After 20 epochs, the validation accuracy increased by less than the training accuracy resulting in the overfitting of the data.
* This can be represented by the images below, where the epoch is further increased by 20 epochs, training accuracy steadily increases while the validation accuracy range between 0.8 and 0.7

#### Fix

* Increase the data
* Make sure that the training and testing data are independent of each other



### 1.3) Increasing the amount of data to 1500

https://github.com/YeonwooSung/GAN\_Implementation/blob/master/src/wavegan/utils.py

# Classification model using different spectrograms

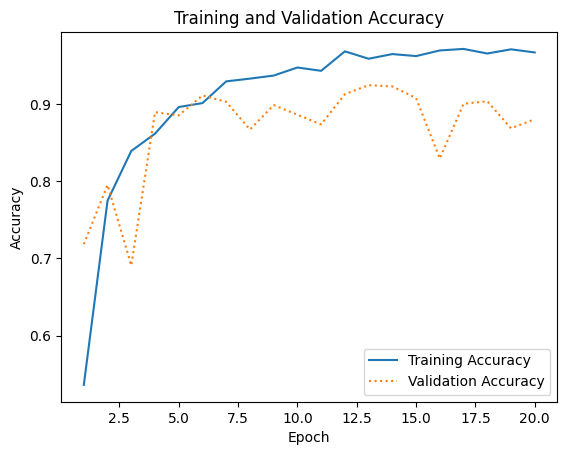
## Motivation

* Catherine suggested that using a wideband spectrogram may result in

## Method

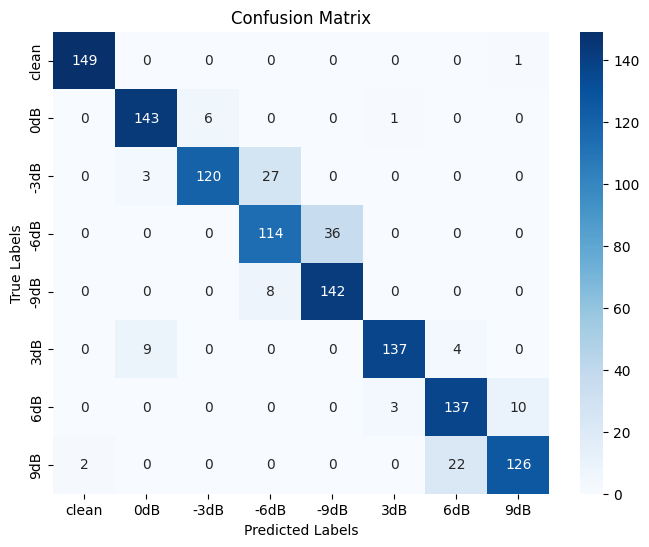
* Window\_length from 0 to 320
* Hop\_length from 512 to 100
* n\_FFT (Length of FFT window) from 2048 to 1024
* 1500 data for each SNR level

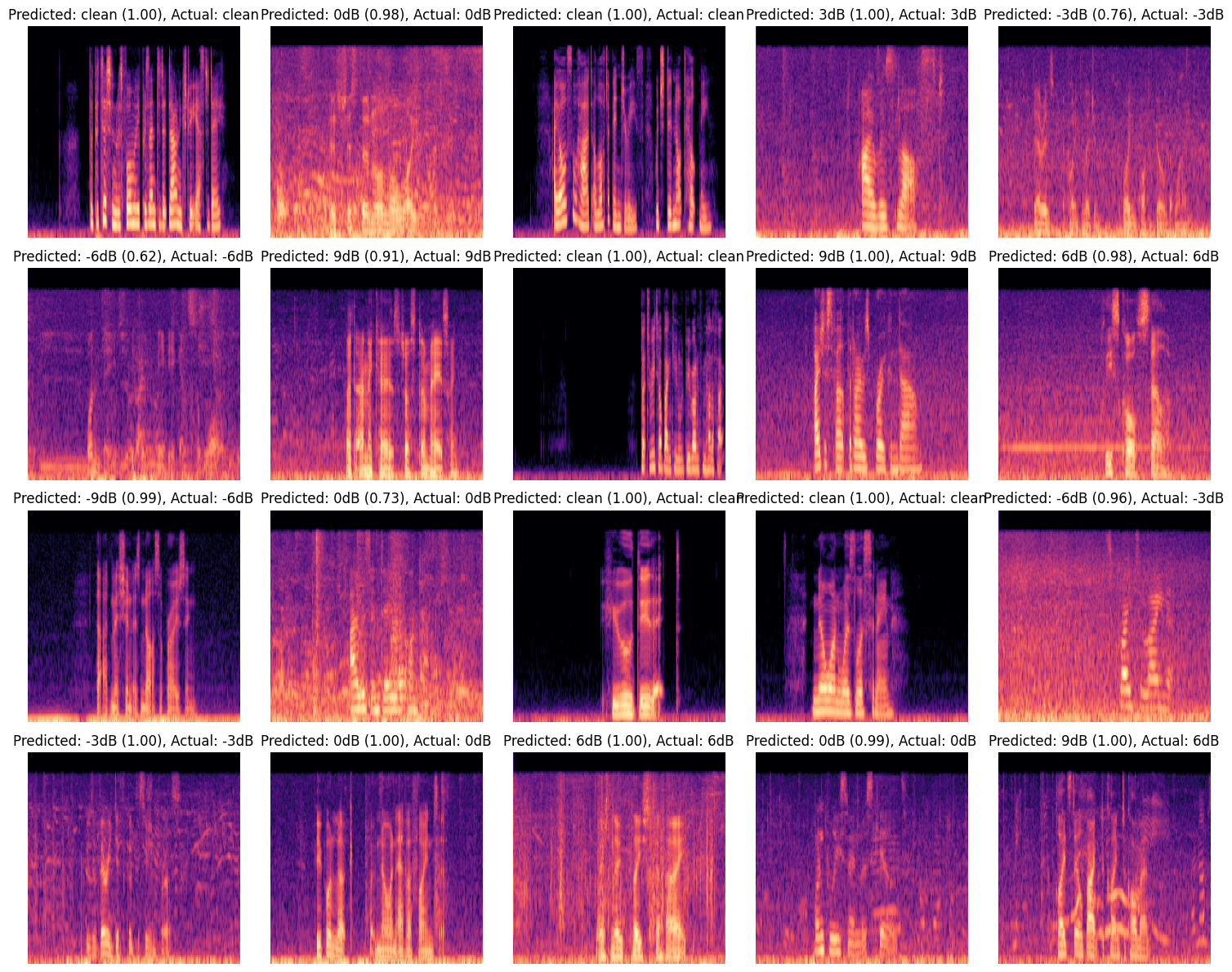
## Result



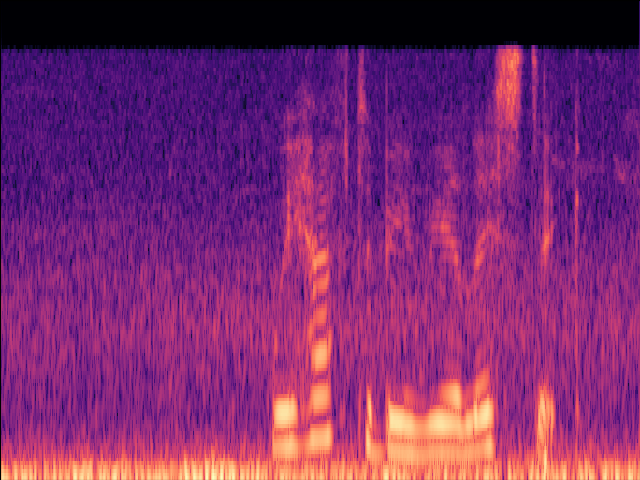
Not much increase in the accuracy as we already had higher accuracy

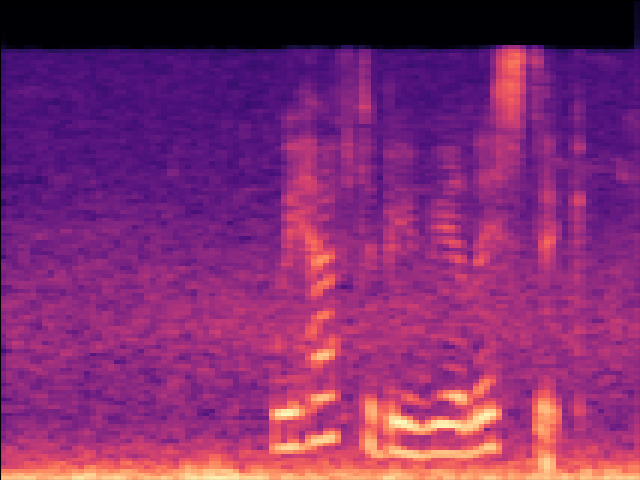
However, is something to consider after using Voice activation on the speech





Difference between wideband and narrowband





# Evaluating Different SNR Methods

## Testing parameters

### Spectrograms

* Spectogram\_Narrow
* Spectrogram\_Wide
* Spectrogram\_WideVAD

### Models

* Raw data narrow
* Raw data wide
* Data with VAD

## Results

### Spectrogram new SNR with VAD

With the new SNR and VAD the SNR level is considered much larger than the older SNR used from TensorFlow

* This resulted in anything higher than -6dB SNR to be considered as 9dB
* Clean to be considered as clean
* And -9dB to be considered as 6dB or 3dB

### Spectrogram old SNR with VAD

With the older SNR and VAD testing with the New SNR with VAD

* Clean is considered as clean
* Anything higher than -9dB SNR is considered as -9dB.

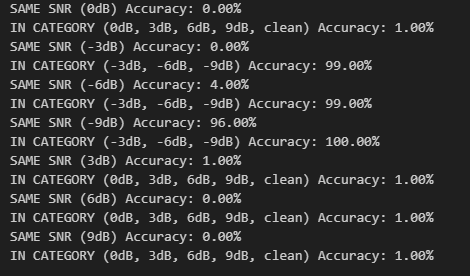
# Evaluating the GAN with the SNR classification

## Testing parameters

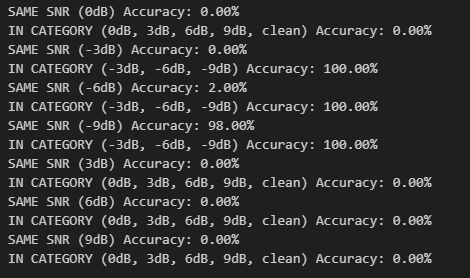
* 100 Data for each SNR
* Calculating the accuracy for both noisy and clean and the right SNR
* Using GAN, GAN with only noise, and the wiener filter
* The GAN uses a generator that is trained with an epoch size of 73 and batch size of 64.
* The testing uses unseen data to be generated with the GANS before putting it into the classifier
* The accuracy for the classifier model is 86% that trains with an epoch of 20 and batch size of 10 with 1500 data with a 80/20 split using a wide spectrogram with VAD

## Result

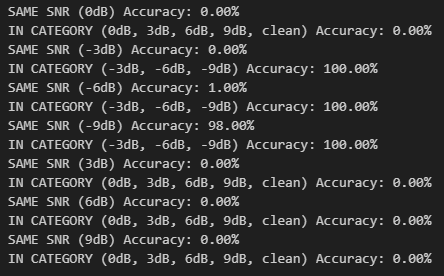
### With GAN



### GAN with noise only



### Wiener Filter



The results are pretty bad, with most predictions showing that they have been transformed into -9dB, even the clean sounds.

* The issue with using the clean noise as it used VAD while the rest of the data did not

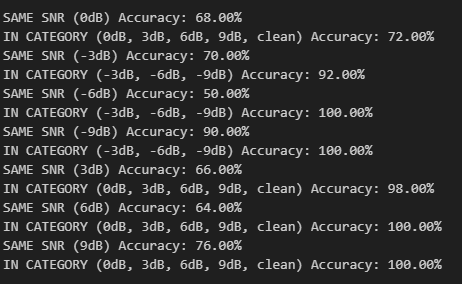
# Evaluate the GAN with the SNR classification part 2

## Parameters

* Using 50 unseen data

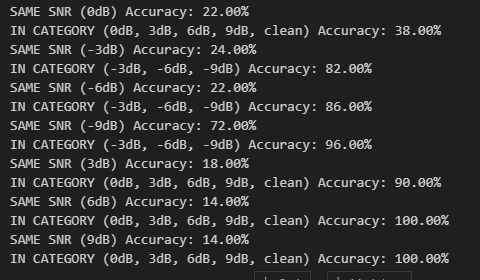
## Result

### GAN both generalisation of noise and speech using the GAN framework



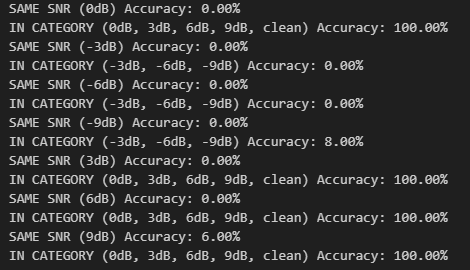
* This method doesn’t change much, a majority of the clean noise is still clean with some resulting in either higher or lower SNR
* While the same result for the lower SNR
* 0dB changes 42% to noisy data

### GAN for generalisation of noise with GMM speech



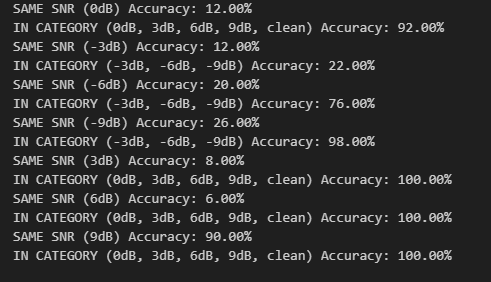
* More of the noisy data results in a higher SNR but in the cost of the cleaner data being considered lower SNR

### Perfect generalisation of noise and speech using both existing speech and noise with the wiener filter (Not real case)



* Perfect case is perfect

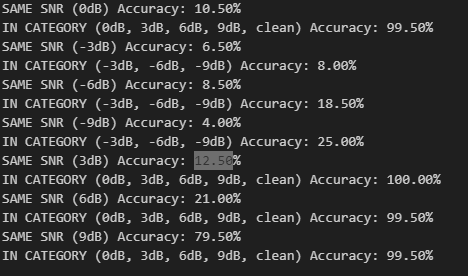
### Using Wiener filter with generalised noise and speech of the GMM



* Wiener filter currently works best as the method of making the lower SNR a higher SNR

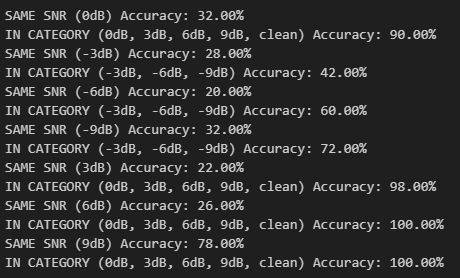
# Evaluate the GAN with the SNR classification using spectral subtraction of noise

## Alpha = 2.0



* The Classification does not care about what speech sounds like just if the noise is present within the audio

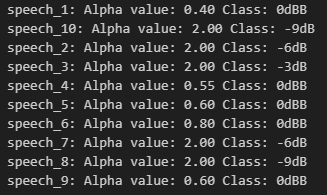
## Alpha = 0.8



* Some cases give a better result than others, as specific alpha gives better results for different alpha values.

# Producing better speech enhancement with the use of more aggressive noise spectral subtraction with higher SNR

* With the use of 10 -9dB files, increasing the alpha value for the spectral subtraction, which controls how aggressive it is, can be performed to attempt to get a clean speech without the loss of quality.

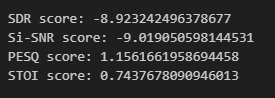


* In some certain cases, the SNR does not change or increase by enough to be considered “clean speech.”
* Another issue is if the alpha value increases too much, the audio quality diminishes.

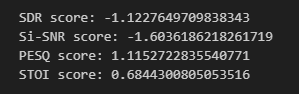
# Evaluating the quality and intelligibility of speech

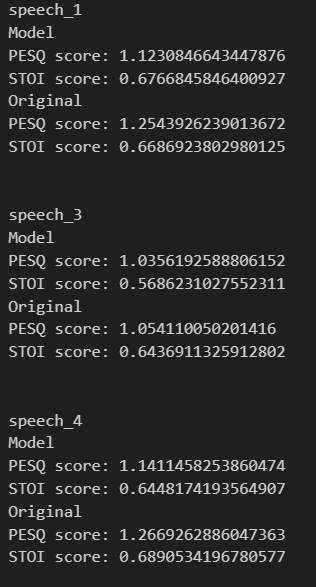
* PESQ
* STOI
* BSSEVAL

Original (-9dB)



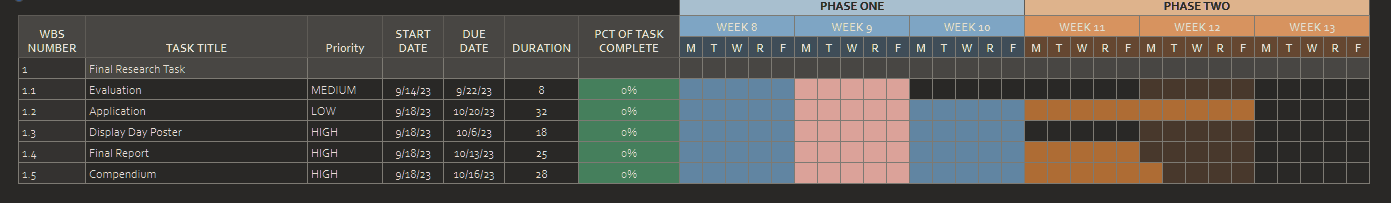
New (0dB)





## 

* The model reduced the eval by a bit for all the scores.



https://docs.google.com/spreadsheets/d/1KqBngcgfNJMJeRsQV0MAyNfu5IAkF6oMLtDXKy1hVLA/edit#gid=0

The STOI score is a value between 0 and 1, where 0 represents no intelligibility (completely unintelligible) and 1 represents perfect intelligibility (no degradation, the speech is fully understandable).

The estimated quality of a degraded speech signal compared to a reference (original) speech signal. PESQ scores typically fall within a specific range, and their interpretation can vary based on this range.

In the PESQ scale, the scores generally range from approximately -0.5 to 4.5, with specific meanings attributed to different score ranges:

Scores above 4.0: Excellent quality. The degraded speech is very close to the reference, and the quality is perceived as excellent.

Scores between 3.5 and 4.0: Good quality. The degraded speech is of high quality and retains most of the characteristics of the reference.

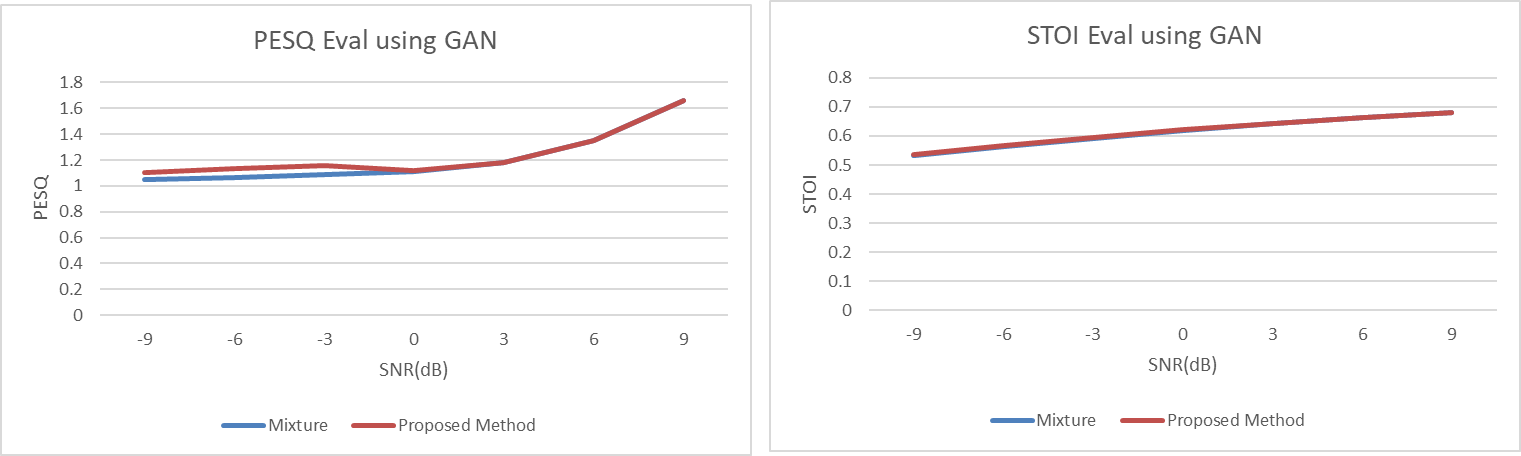
Scores between 2.0 and 3.5: Fair to acceptable quality. The degraded speech is intelligible but may have noticeable quality issues.

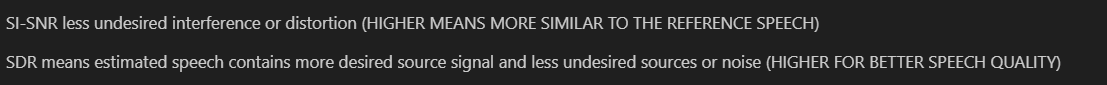
Scores below 2.0: Poor quality. The degraded speech has significant quality issues, and it may be challenging to understand.

* Most of the time, the method does increase both the quality and intelligibility of the speech according to both PESQ and STOI evaluation. But it would increase then decrease after a certain Alpha value. Once the classifier thinks that the speech is clean, the eval decreases.
* An issue I have thought about the classifier is that, unlike the eval tools, it cannot distinguish between audio that can detect speech and can not detect speech.
* OVERALL, according to the evaluation tools, speech that can be detected by the GAN can improve both the quality and intelligibility of speech, which can be heard in many cases.

Next steps

* Run the code til 3dB
* Test the PESQ and STOI at every ALPHA
* The MAXIMUM threshold ALPHA is 2
* If the Alpha increases by 0.05 every iteration, there will be a maximum of 40 iterations.





# Signal conditioning

* Most of the available speech and noise samples are in wav format and come in sampling rate between 16KHz and 48Khz with varying bit depth.
* To downsample the signals in order to get acceptable frequency and temporal resolution, a python script was developed.
* Code has been designed to handle both mono and stereo cases for generality.
* This python script first randomises the names of each file in a directory in order to ensure minimal chances of duplicate file names.
* The designed low pass filter uses an IIR butterworth lowpass filter which has a gain of 1 within the passband and a high attenuation at the specified corner frequency.
* The a and b coefficients are calculated from a pre-defined library/function.
* The filtering is done through a scipy function that can handle both FIR and IIR cases.
* The spectrogram generation uses 75% overlap with a 32ms Hann window.
* CHECK IIR FILTER IF non-linear phase is removed.

# GMM (Gaussian Mixture Model) integration with GAN

### GMM/MMSE

* The current code uses hardware acceleration (TensorFlow) to speed up the calculation results.
* Sieving is planned to be implemented to reach the global maximum before returning the mean and variance for each component.
* Currently testing GMM algorithm
* Currently, 18 components were selected as there were 18 categories of noise within the training set. This can be changed to accomplish generality.
* PSD estimator is still being worked on in order to calculate the necessary wiener filter gain.
* Using the codebook method that Yusuke has shown in an earlier email regarding single channel speech enhancement.
* Provided that the distribution of the input is Gaussian, the Wiener filter minimises the MSE.

GMM BASED MULTI-STAGE WIENER FILTERING FOR LOW SNR SPEECH ENHANCEMENT - THE PAPER I BASED THIS ON

For the Gaussian mixture model, an unsupervised method (GMM with sieving) was chosen to avoid issues with convergence due to the non-convex optimization problem. Basically, as there are many local minima,maxima, saddles and plateaus sieving ‘generates’ a map of the entire region and selects the global minima/maxima. In this case, the highest log likelihood is the performance criterion where more negative values represent higher probabilities. There are many points of convergence, therefore it is important to select the convergence point that results in the highest likelihood.

To get the mean PSD for each frequency bin of each data point, the following format has been used:

Frame1 Frame2 Frame3…… Framex

F

F2 = data vector, Mean\_PSD\_per\_frame = mean(F\_bin, All\_frames)

F3

.

.

.

Fn

Similarly to arrange the means PSD per data point

Data1 Data2 Data3…… Datax

F

F2 = data vector, Mean\_PSD\_per\_datapoint = mean(F\_bin, All\_data)

F3

.

.

.

Fn

For each frequency bin the EM(GMM) algorithm is called to return a mean PSD vector associated with the frequency bin of size K(which in this case is 9).

After the mean vector has been returned, it is assigned to one of the Nth row of the codebook which represents a frequency bin. This occurs until all 1024 frequency bins are populated.

The code used for the implementation: <https://youtu.be/Vj4b4xojPMw?list=PLISXH-iEM4JlFsAp7trKCWyxeO3M70QyJ>

To filter the signal, the Wiener filter coefficients are determined by estimating the PSD of the desired signal and the additive noise signal. GMM is used to extract the mean PSD at each frequency bin that is extracted from the data with the use of a sliding Hann window with 75% overlap. The mean PSD vector of noise and speech are then used as a codebook that is represented by an array with each row being the frequency bin and each column being a datapoint. To find the vector projection of speech and noise from the mixture, the individual pseudo-inverse of each codebook is taken with the current mixture frame to determine the projection of the mixture on the noise and speech vectors through linear overdetermined combinations.

Key thing to consider: From analysis, if the frequencies of the noise corrupting the speech are well-separated from the speech frequencies, then no matter the amount of suppression to the frequencies, intelligibility and quality will be mostly kept. HOWEVER, if there are many overlaps between the noise frequencies and speech frequencies, the level of noise suppression dictated by Wiener\_filter(w) = Speech(w)/(Suppression\_level \* Noise(w) + Speech(w)) will attenuate some frequencies of speech that could reduce the intelligibility and introduce distortions.

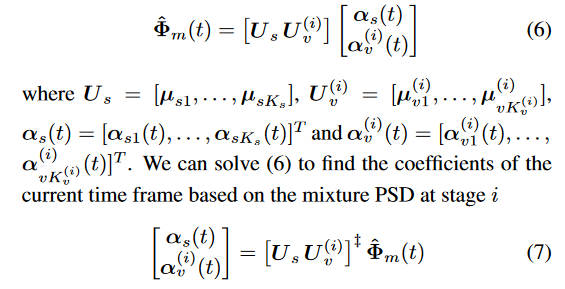
The method of reconstructing the original time domain signal is the overlap add method.



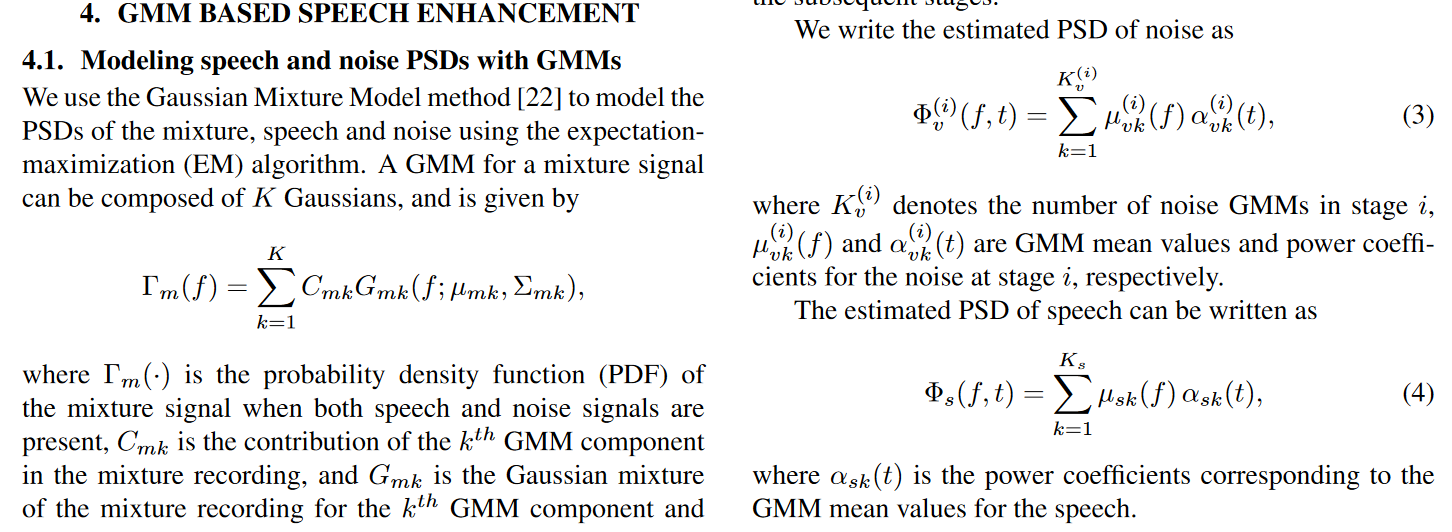
### GAN implementation & Integration.

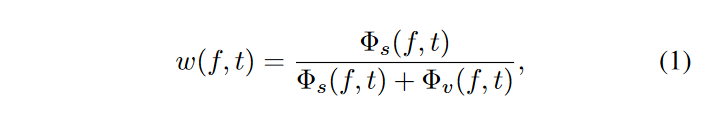
The first part consists of an adversarially trained GAN to gain a mapping according to the noisy input by pairing the mixture along with its real counterpart through the data pipeline.

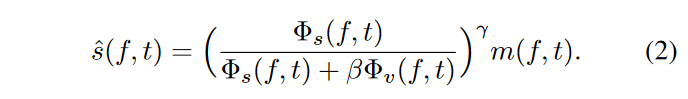
The generator in this case learns to take the PSD (a 1x1024 length array) of the mixture and turn it into a 1024x9 array like that of the GMM, where it learns a mapping between the generates 1024x9 array and the equation of:



The PSD is then calculated from this and compared with the discriminator trained on real data. Ideally this would generate a ‘realistic’ and ‘accurate’ mean vector that can be modelled as a gaussian mixture model through a modified EM algorithm where the covariances and component probabilities can be calculated for each row of the mean vector. This would result in an adversarially trained gaussian mixture model. What this accomplishes is a time-varying uni-variate Gaussian mixture model for each frequency bin.









Method: For this research project, the same method from the paper was used where the GMM derives the filter specification by clustering the resulting PSD from a 1024 point FFT. The negative frequencies can be dropped to reduce the dimensionality/training time of the GMM due to FFT symmetry but was kept to see the effects of sieving on the resulting PSD and performance of the filter. The window function chosen was a 512 point Hann window with zero-padding to avoid spectral leakage. The chosen filtering method was Wiener filtering where frequencies are suppressed according to the predicted PSD by the GMM.

To derive the mean vectors from the chosen dataset, the following approach was taken:

To get the mean PSD for each frequency bin of each data point, the following Pseudo-code has been used:

For each data calculate the PSD of each frequency per frame:

Frame1 Frame2 Frame3…… FrameN

Freq1 PSD PSD PSD …….

Freq2 PSD PSD PSD

Freq3 PSD PSD PSD

. .

. .

. .

FreqN

Per row, of the array above the mean spectral power is calculated through all analysis frames and is then assigned to one columns of an array of the format:

Data1 Data2 Data3…… DataN

F

F2

F3

.

.

.

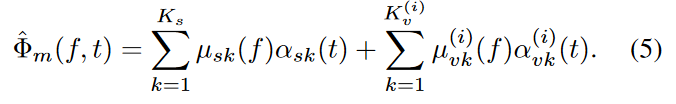
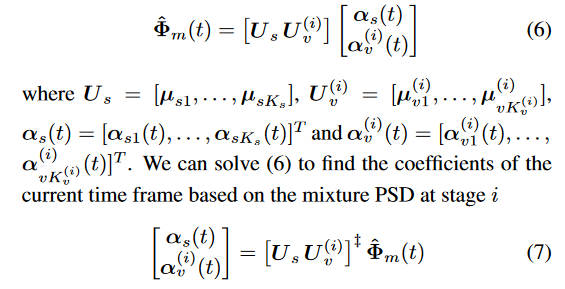
FN

We treat the PSD of the noise as the superposition of all the frequencies.

From this array, each row is fed into a univariate GMM algorithm with sieving through the parameter space to find the highest log likelihood statistics. This is a very expensive algorithm where many guesses of means, variance and components probabilities are calculated and small iterations of the EM algorithm are performed on each of the guesses in the parameter space. The M chosen ‘best candidates’ with the highest calculated log likelihood will then have the EM algorithm performed with higher iterations to gain a mapping of local maxima. This would result in the selection of the best candidates at the ‘seen’ global maxima that results in the highest log likelihood.

Note that the log likelihood for most cases working with much larger variances and means than our project would output a result from 0 to negative infinity where 0 is the highest chance and negative infinity is the lowest chance. In the context of this project the log likelihood is observed to be positive which is higher than 0. This is expected due to the low dispersion/variance of each gaussian component compared to most cases.

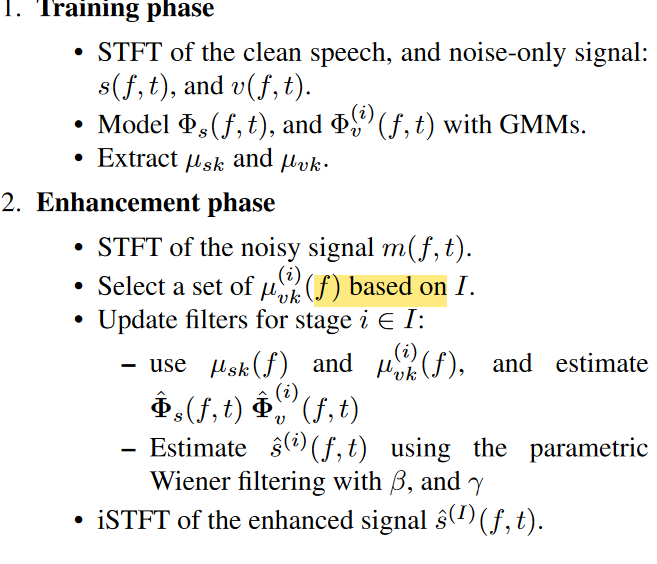
Once the univariate GMM for each frequency is calculated, a codebook of size 1024xK (In this case it was chosen to be 9 as per the previous study conducted) it is then used to derive the filter coe-efficients.



The codebook undergoes a pseudo-inverse which is then multiplied by the PSD of the mixture to derive the projection coefficients. These coefficients determine the projection of the mixture relative to the codebook. The coefficients are multiplied again with the codebook to derive the Mixture PSD as modelled by equation 5 from a previous study.

The interpretation of this model would be, how much do each of the latent classes contribute to the PSD of the noise within the mixture.

From inspection, the method seems to be a special case of singular value decomposition with overdetermined/underdetermined systems/matrices. Where the results of the projection is dependent on the pseudo-inverse of the A matrix (In this case, A is the 1024 x 9 codebook generated by the univariate GMM algorithm). Least squares regression and yada yada.



From their work, this seems to be the case as the mean vector (the 1024 x 9 codebook) undergoes iterations to further increase the PSD estimate. From this, we were motivated to make an adversarial GMM to change the mean vector (the 1024 x 9 codebook) based on the decision generated by a discriminator. The concept is similar to that of the GAN framework from where the generator and discriminator minmax their loss functions through unsupervised and simultaneous learning where the discriminator audits the results of the generator until it generates outputs that can fool the discriminator into thinking the generated output is real. By inspection, the only thing that determines the filter coefficients of the wiener filter and the generated PSD is the mean vector with no consideration to the overall structure of the distribution such as the covariance/variance and the component probabilities. This implies that for many guesses of the mean vector, there will also be many guesses of the covariance and component probabilities which would require retraining the algorithm many times to re-generate the structure of the GMM which would be very costly and time-consuming. Another problem that was considered about the discriminator was how it would have been implemented as even though it knows what the ground truth noise PSDs are, how would it then know what feedback to give to the GMM? A traditional discriminator outputs a probability that determines if the generator did a good job at replicating its preset distribution. This probability will then have to be translated to parameter guesses for the GMM.

A more simplified approach to this problem would then use a GAN framework consisting of a traditional generator and discriminator to change the generated 1024x9 codebook with the input to the generator being the mixture PSD. To train this, the mixture PSD and the noise PSD will be fed into the algorithm as a pair and through the same mathematical transform shown above the generated 1024x9 codebook will then turn into a PSD according to the PSD of the mixture. This PSD will be fed into a discriminator to determine how well the Generator is doing. Basically, this enables the generator to find a mapping between the 1024x9 mean vector and the mixture that is fed into it. After training the GAN, the mixture’s PSD per frame will be an input to the generator which results in a 1024x9 mean vector.

A modified EM algorithm which takes in each row of the mean vector as the ground truth and calculates the covariance and component probabilities that maximises the log-likelihood of the result. This is a proposal for the Adversarially trained GMM. This is motivated by the naive assumption that the increased accuracy of the PSD estimation results in higher objective scores and increased intelligibility of the Wiener filtered speech.

The previous paper hints at the usage of a time-varying Gaussian mixture model

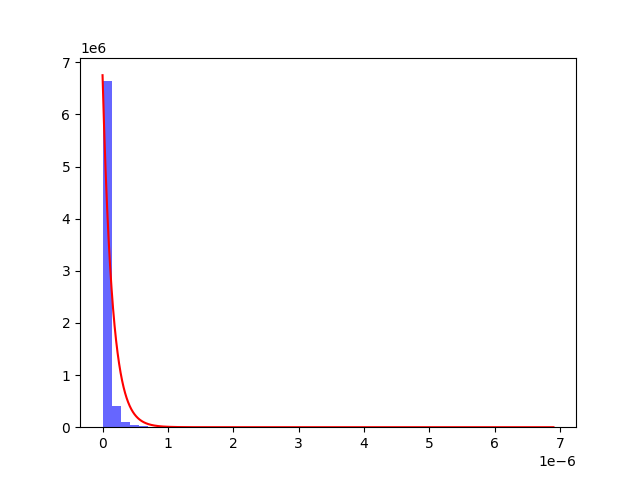
Last week's meeting with Yusuke regarding input SNR provided the following insights:

- Definition of input SNR is not too important but the improvement of post-enhancement metrics such as signal to distortion ratio (SDR), signal to artifact ratio (SAR) and signal to interference ratio (SIR) were more important.

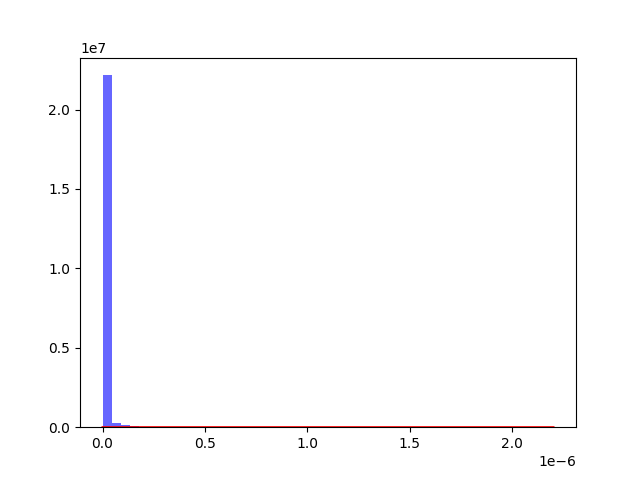
-As such, a python implementation (BSSeval) which evaluates these metrics for each different file was implemented and the filtered outputs of the GAN and GMM methods of Wiener filter coefficient derivation were evaluated using the specified metrics. The results show that the current GAN model (The one from the seminar) implementation scored lower SDR and SAR scores than that of the GMM due to the introduction of artifacts and distortion. An interesting result to see was that the signal to interference ratio for both GMM and GAN implementations were similar. It was found that the current GAN and GMM mathematical model/transform did not fully reflect the mathematical model defined in the research paper (clipping negative PSD values to 0 due to it not being possible). The mathematical model implementations were modelled accordingly for both GAN and GMM implementations and it was observed to slightly improve the intelligibility and noise reduction performance through an informal listening session.

With the current definition of SNR with the usage of the voice activation detector, Edward's classifier network has been retrained on the new dataset.

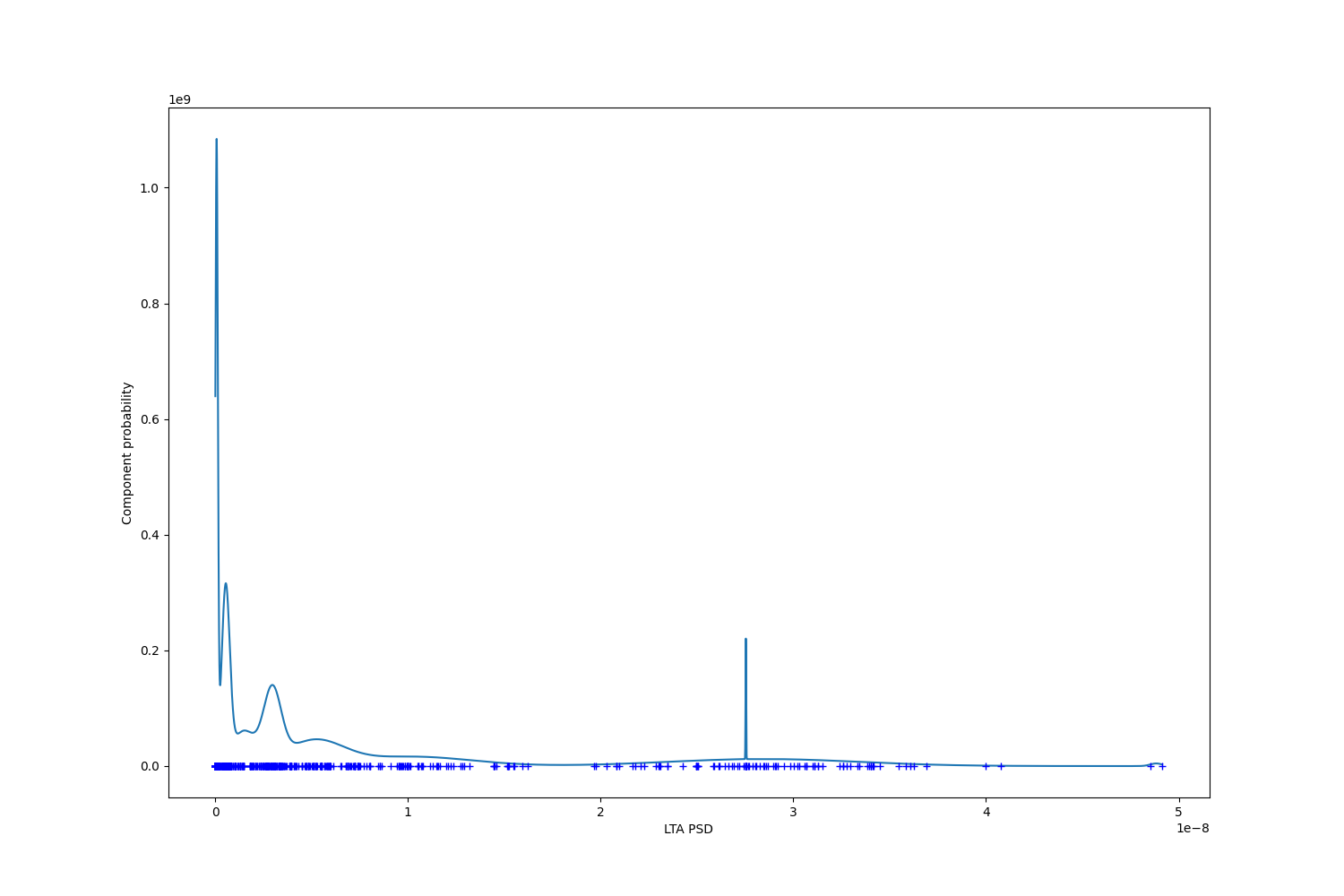
As per your advice regarding the non-converging frequency bins, the long term average of each file was plotted with particular interest on non-converging GMM inputs and it was seen that the resulting distribution of values were more akin to a Laplacian/one-sided negative exponential curve. I think that the GMM for certain frequencies did not converge as GMM also measures the 'gaussianness' of the distribution, in this case it is non-gaussian.



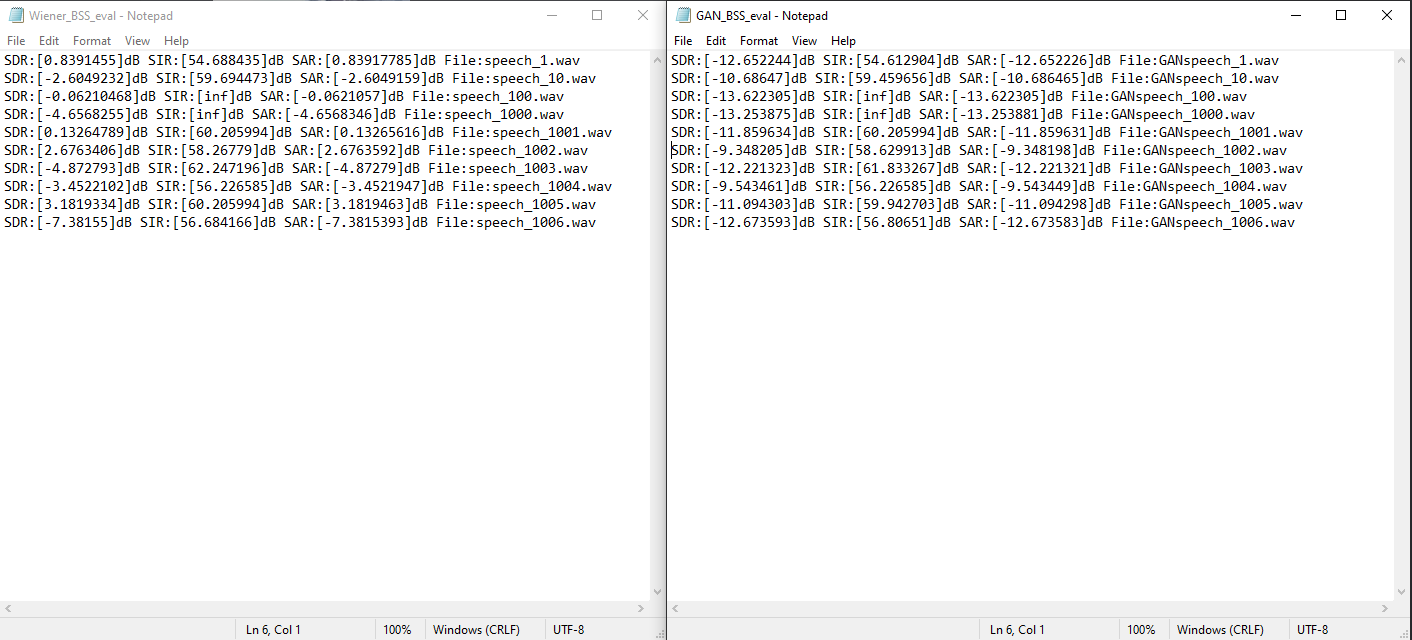
An even more extreme example:



As the distributions have now been approximated, the mean PSD of the non-converging frequency bins were calculated through a basic estimation of mean (sum of data/number of data). The mean PSD value is then forcefully added to the GMM derived Wiener coefficients. Evaluation of enhanced speech samples did not seem to improve the quality of speech compared to the past improvements (VAD truncation of silent sections).

From the evaluation of results, the distribution of the average power coefficient of noise (long term average) is skewed to the right like that of a negative exponential. This is an expected result as the noise used mostly has stationary noise with some outliers having non-stationary that directly influences the distribution. An example of which can be seen in the following plot: 

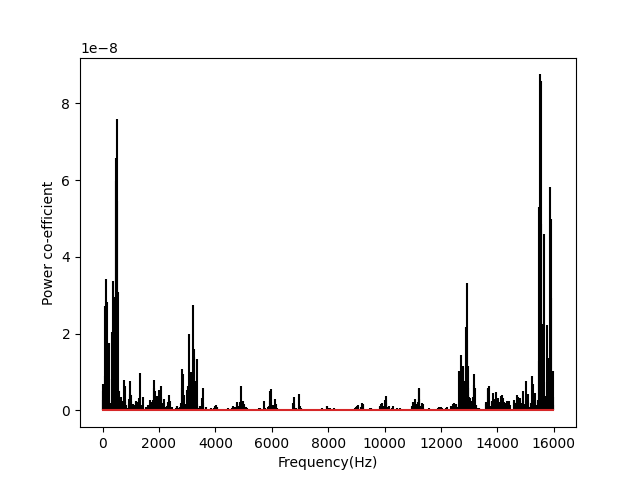
### Evaluation of Wiener vs GAN\_estimate (BSS\_eval)



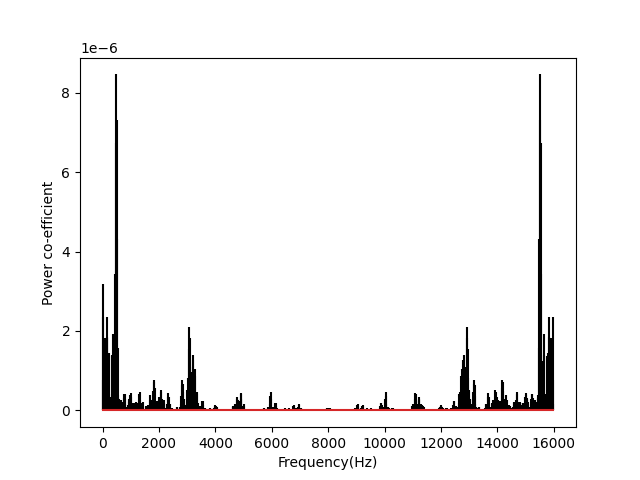
From continuous training of different models with hyperparameter and architecture adjustments, I think there is a good chance that the model is able to generate good matrices with less than 10 epochs.

### BIG UPDATE:

#### GAN Estimate - (undershoot)

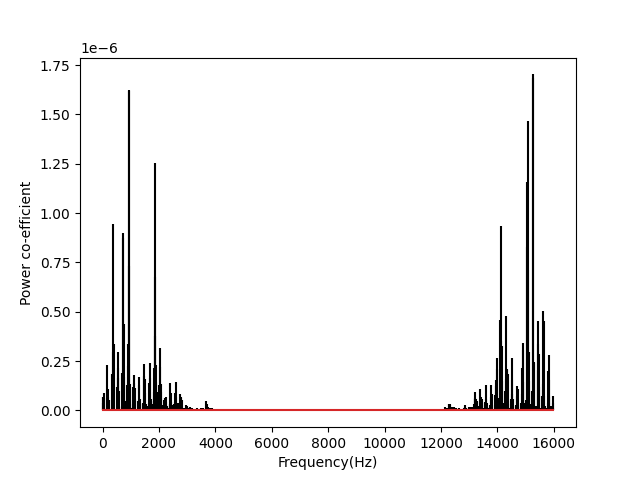


#### Real PSD

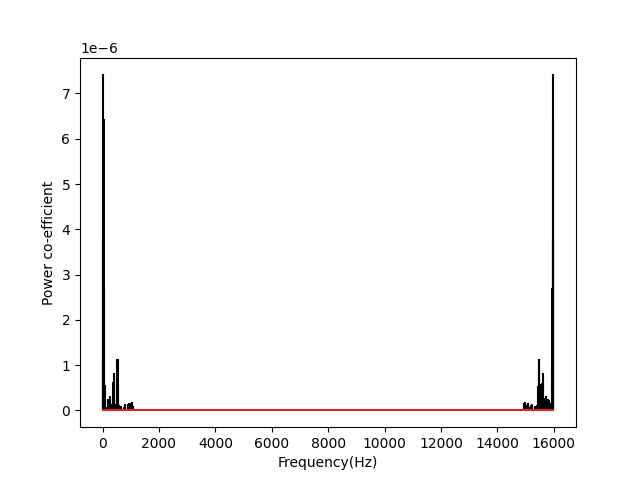


The current architecture shows very promising PSD results with impeccable PSD resemblance with that of its real counterpart. This is an interesting result due to the GAN clearly undershooting the power-coefficient at each frequency bin but it can still output a resembling shape. A possible explanation for this is that the machine is forced to adjust and learn even though the model in the middle is mathematically flawed (clipping to 0, reduces the overall energy of the matrix) the GAN learns how to get the right PSD shape with the difference being scaled down by a certain factor as seen from the undershoot. Still these are very good results for 7 epochs and a training set of 31.5k data.However for cases where it does not undershoot, the resulting PSD is very inaccurate as shown below:

#### GAN estimate - inaccurate



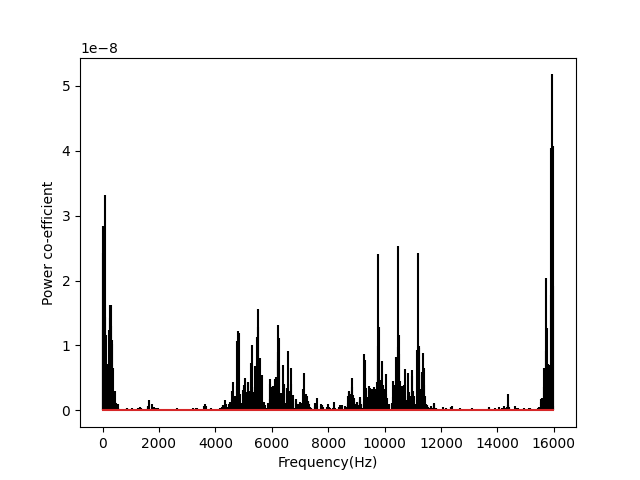
#### Real



#### Undershooting error

However, there are still estimation issues even with undershooting results:

#### GAN Estimate



#### 

#### 

#### 

#### 

#### 

#### 

#### Real



### The proposed fix:

To fix the training procedure of the GAN, it is very straightforward. The steps are as follows:

* Assuming that the GAN predicts/generates a structurally similar PSD to its real counterpart, all of the power coefficients of both the real and generated PSDs will be summed up and the following equation will be used to determine the scalar value that will need to be multiplied to the generated PSD before being evaluated by the discriminator. Sum of power real power coefficients / Sum of generated power coefficients = scalar value. Scalar value \* Generated PSD will ideally bring this to the same scale of the real PSD.

### Next steps

* If the fix above is applied, the GAN will be able to generate the required 1024x9 codebooks, but the resulting mathematical transforms (the mathematical model) will still result in undershooting but structurally correct PSDs. A proposed solution would be to use another machine learning model that will dynamically determine the required scalar value based on a mixture PSD, the generated PSD and the real PSD.
* A possible approach would be to use another GAN that takes in the input PSD (either image/data implementation) and follow an encoder-decoder approach. The generator will be structured like that of an encoder to ‘shrink’ the input mixture PSD to a scalar value. Another model in the middle will then ‘intercept’ the resulting scalar value and multiply it with the generated PSD based on the mixture PSD. This scaled up/down generated PSD will then be compared with the real PSD counterpart by the discriminator. NOTE THAT THE DATA WILL HAVE TO BE PAIRED IN THIS CASE AGAIN.

TO do: Prepare data (out of ram), ~~Half the PSD graph and plot in log scale, not just log amplitude.~~

#### Log graphs





### 29/08/23

Speech PSD estimation is a problem, A GAN that is trained to extract speech PSD from a mixture does a poor job and may need more investigation.

Subtract actual noise psd from mixture psd and see if the resulting normalised speech psd will do the trick.

Tried subtracting Noise PSD. doesn’t work spectral leakage and distortions make it sound bad

Tried Going from PSD to audio but seems like temporal domain subtraction

BIIIIIGGG -<https://vocal.com/noise-reduction/unifying-spectrum-subtraction-with-wiener-filtering/>

Last try: change the formula to having the set gmm weights matrix G

Mixture =B

Scale each row of G by the scale of each row of B

G\_scaled dot product GAN Vector = noise estimate.