Automatic Speaker Recognition using Vector Quantization and Image Recognition

Bruno E. Gracia Villalobos

Independent Study

Professor Artyom Grigoryan

University of Texas at San Antonio

August 9, 2019

https://github.com/brunogracia

*All diagrams/content are created by the author unless referenced

INTRODUCTION

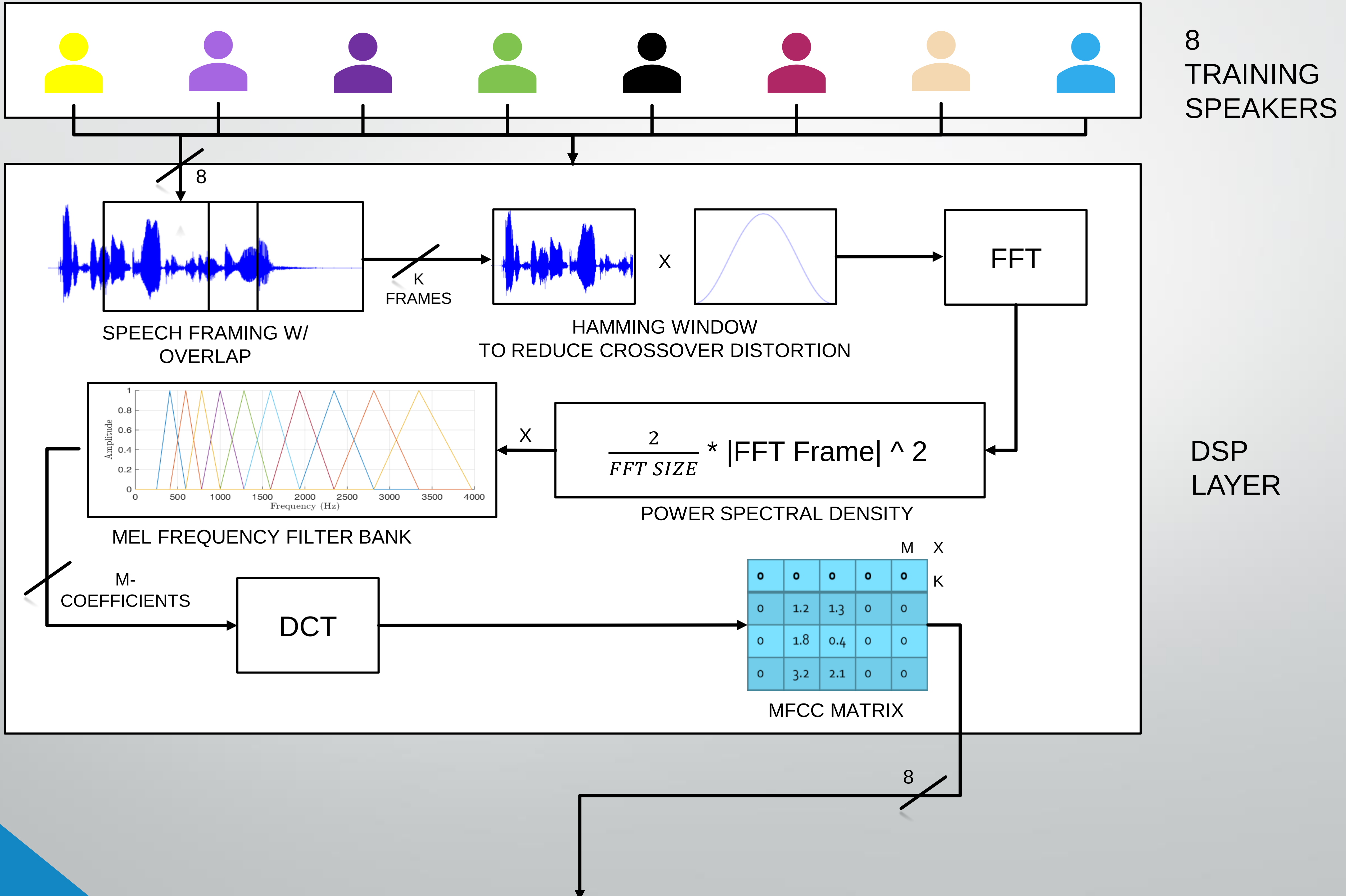
This presentation will cover two avenues for ASR using Mel Frequency Cepstrum Coefficients (MFCC) as *feature vectors*:

- Vector Quantization (VQ) (Reference 1)
- Image Recognition (IR)

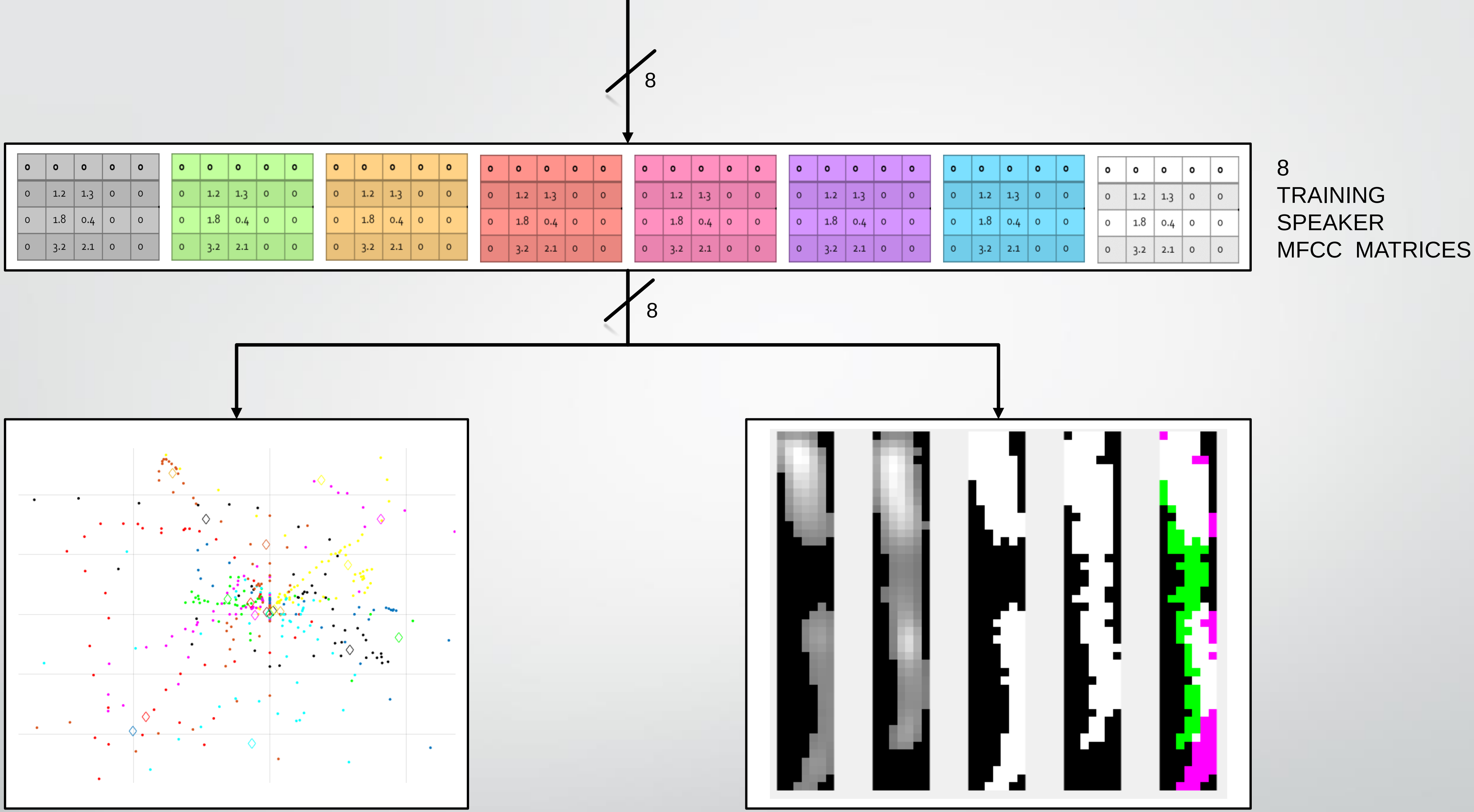
In a high level, an ASR system accomplishes the following:

- 1. *Training* speaker data is captured
- 2. Model is built around *Training* data using (VQ) and (IR)
- 3. *Testing* speaker data is fed to the model
- 4. Model matches *Test* speaker to *Training speaker*

STEP 1


8
TRAINING
SPEAKERS
8
SPEECH FRAMING W/
OVERLAP
K
FRAMES
X
HAMMING WINDOW
TO REDUCE CROSSOVER DISTORTION
FFT
X
Amplitude
Frequency (Hz)
0 0.2 0.4 0.6 0.8 1
0 500 1000 1500 2000 2500 3000 3500 4000
MEL FREQUENCY FILTER BANK
2 / FFT SIZE * |FFT Frame| ^ 2
POWER SPECTRAL DENSITY
DSP
LAYER
M-
COEFFICIENTS
DCT
M X K
0 0 0 0 0
0 1.2 1.3 0 0
0 1.8 0.4 0 0
0 3.2 2.1 0 0
MFCC MATRIX
8

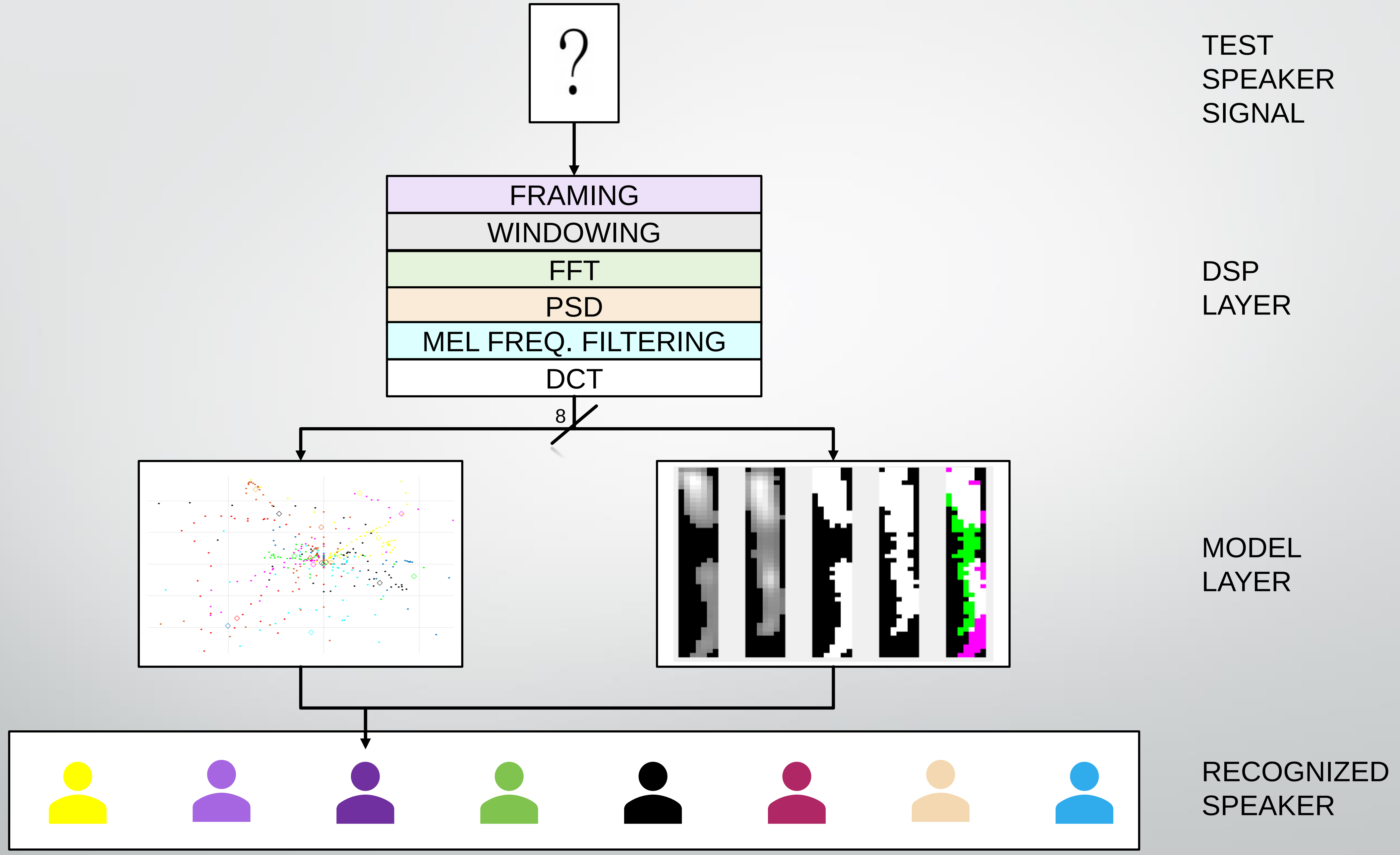
STEP 2



VECTOR QUANTIZATION
MODEL

IMAGE RECOGNITION
MODEL

STEPS 3 & 4



TEST SPEAKER SIGNAL
?
DSP LAYER
FRAMING
WINDOWING
FFT
PSD
MEL FREQ. FILTERING
DCT
8
MODEL LAYER
RECOGNIZED SPEAKER

MATLAB IMPLEMENTATION

- The ASR application is developed in MATLAB with a database of 8 training and testing speakers
- DSP and Model Application layers discussed in the diagrams are implemented from scratch for portability
- The following MATLAB Library functions are used:
  - fft
  - Ifft
  - dct
  - activecontour
  - dice
- The implementation will be discussed in the following sections
  - DSP Layer
  - Model Layer

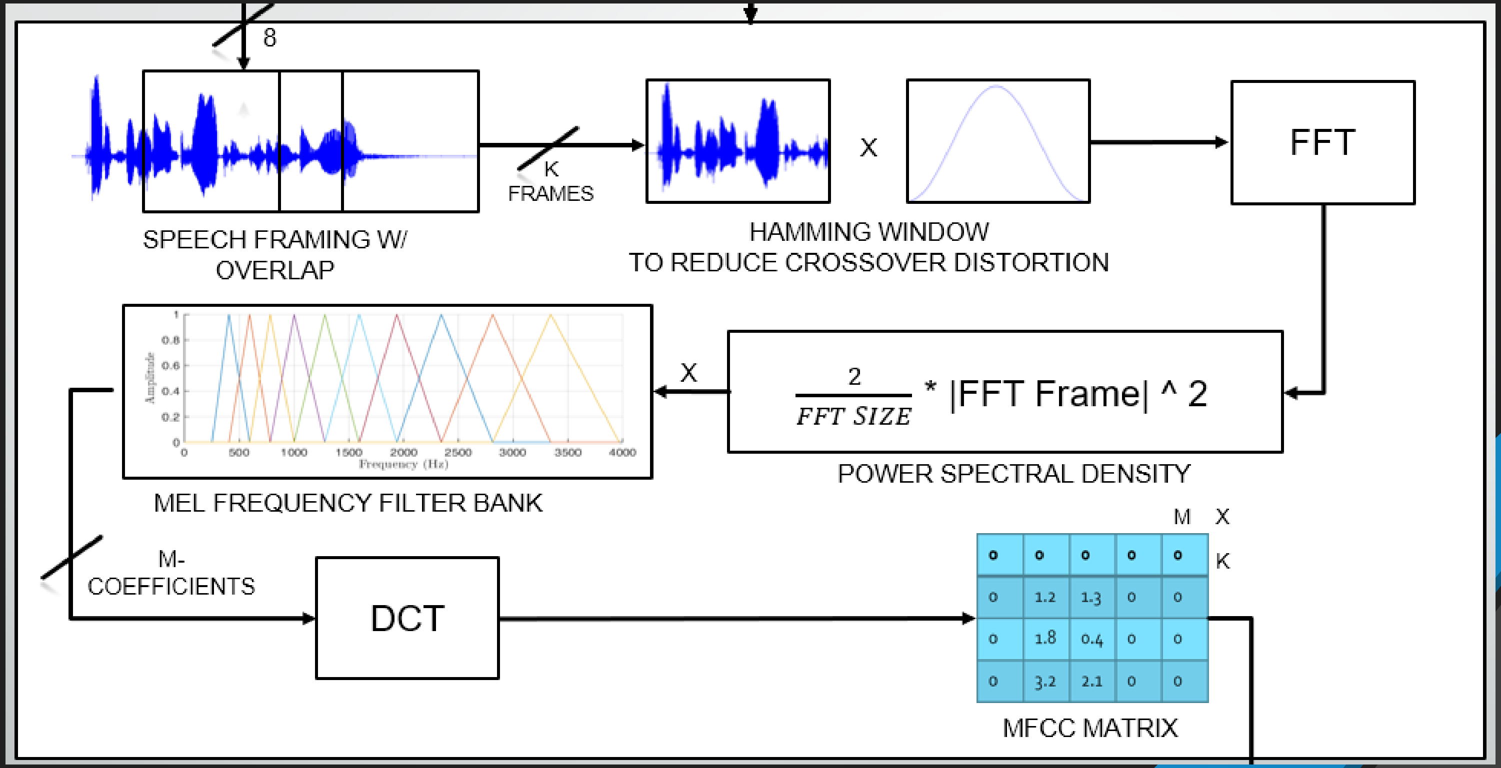
CONFIG

BRIEF INSIGHT

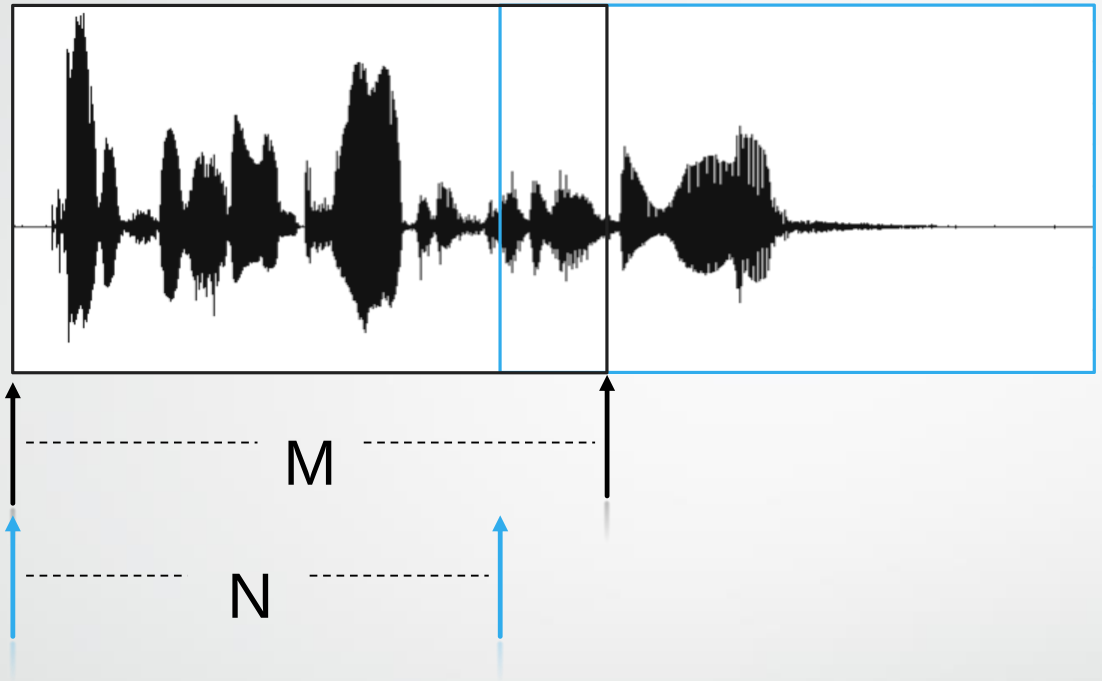
- IMPORTANT
- To use the database of *training* and *testing* speakers, the folder paths must be specified as well as the number of total speakers.
- Files should be named “s1.wav”, “s2.wav” or it won’t work
- The rest of the configuration can be left as default unless testing should be done

```
%~~~~~~~~~~~~~~~CONFIG PARAMETERS~~~~~~~~~~~~~~~~~~%
%   *************** Please rename speaker training voices as s1,s2,s3,s4......
trainingSamplesFolder = "database/train/";
testingSamplesFolder = "database/test/";
numSpeakers = 8;        %how many speakers are in each folder
numMelFilters = 40;     %default is 40

upperLim = 12500;       %this is best dependent on sampling freq of signals (in Hz)
lowerLim = 40;          %usally this low (in Hz)

N = 256;                %frame length
M = 100;                %num samples before overlap
overlap = N-M;          %frame overlap

VQDim = [30,4];         %which MFCC numbers to use for VQ section

VQThreshold = 0.005;    %MFCC's under this value will be set to 0
VQOffset = 0.00;        %quantity to add to MFCC's under VQThreshold

sizeCodeBook = 4; %Use POWERS OF 2. (2, 4, 8...)
%~~~~~~~~~~~~~~~CONFIG PARAMETERS~~~~~~~~~~~~~~~~~~%
```

DSP LAYER IN MATLAB



SPEECH FRAMING WITH OVERLAP



- Speaker Signals have the following characteristics:
  - Fs = 12.5kHz
  - Length ~ 1second
- Framing characteristics:
  - M = 256 (20 ms frame is standard for speech) $\frac{256}{12.5k} = 20ms$
  - N = 100

DSP LAYER

1) Read in WAV data for both training and testing speakers

2) Store in cell array’s

```
%cell array to store speaker data
[trainArr, trainFS] = getArr(numSpeakers, trainingSamplesFolder);
[testArr, testFS] = getArr(numSpeakers, testingSamplesFolder);
```

3) Frame the signals with M=256 and N=100

```
trainFrameArr = getFrames(trainArr, numSpeakers, N, M);
testFrameArr = getFrames(testArr, numSpeakers, N, M);
```

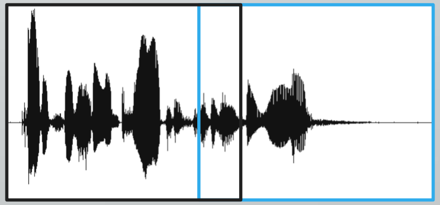
4) Result is a cell array of speakers like this:

trainMFCC | trainMFCC{1, 1} | trainArr | trainFrameArr

1x8 cell

	1	2	3	4	5	6	7	8
1	130x256 do...	135x256 do...	133x256 do...	148x256 do...	181x256 do...	148x256 do...	143x256 do...	145x256 do...

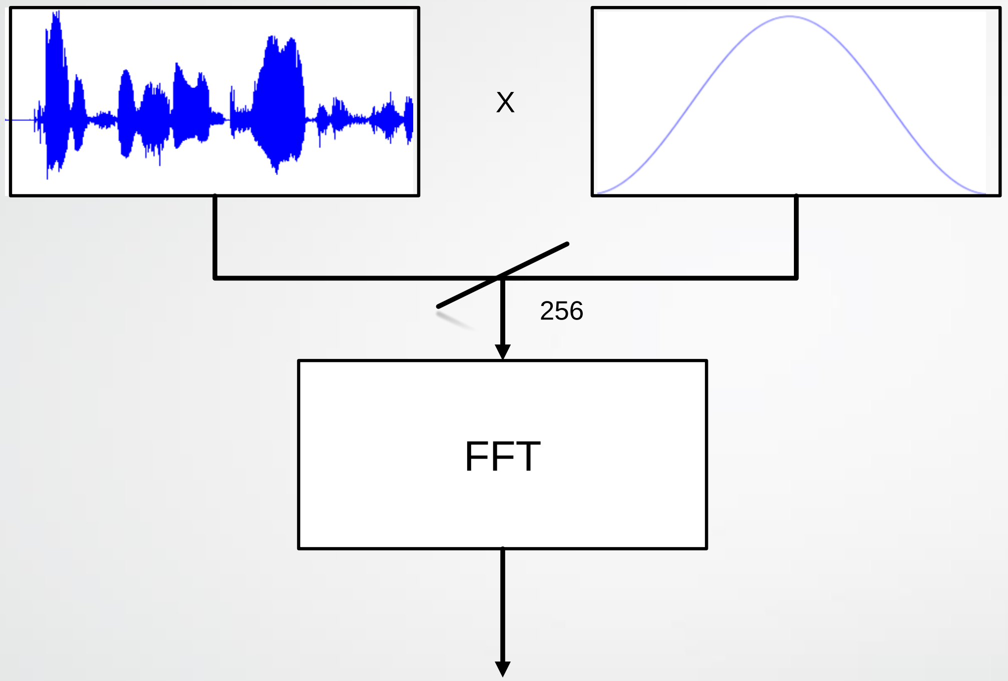
5) There are 256 columns for each speaker because frame size M=256



SPEECH FRAMING W/ OVERLAP

DSP LAYER

HAMMING WINDOW & FFT



- A 256 sample wide Hamming Window is used to reduce crossover distortion and error in the frequency domain
- The FFT outputs 256 bins

DSP LAYER

1) The getMFCC function processes the signals in the chain on the right

```
trainMFCC = getMFCC(trainFrameArr, numSpeakers, N, numMelFilters, trainFS);
testMFCC = getMFCC(testFrameArr, numSpeakers, N, numMelFilters, trainFS);
```

2) With all of the speaker's framed in 256 samples each, now each frame needs to have a Hamming window applied

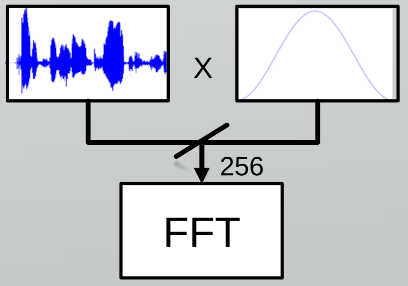
```
%create hamming windows for each chunk
window = hamming(N, 'symmetric'); %N point hamming window
```

3) Chunk3d_window is a copy of trainFrameArr and stores the windowed signal

```
%windowed signal = signal * window
chunk3d_window{m}(j, :) = chunk3d_window{m}(j, :) .* window;
```

4) FFT each chunk with a size of N=256

```
%FFT each chunk
chunk3d_FFT{m}(j, :) = fft( chunk3d_window{m}(j, :) );
```



HAMMING WINDOW W/ FFT

DSP LAYER

POWER SPECTRAL DENSITY

$$\frac{2}{FFT\ SIZE} * |\text{FFT Frame}| \wedge 2$$

1 : N/2 (NYQUIST)

- After the FFT of each frame, now the Power Spectral Density (PSD) of the frame is calculated
- The function outputs a frame of now 128 samples from bin 1 to bin 128 (NYQUIST)

DSP LAYER

1) Chunk3d_FFT{m}(j,:) holds the current frame FFT, where m is the current speaker, and j is the current frame.

```
% calculate PSD for each chunk using FFT chunk
chunk3d_PSD{m}(j, :) = getFullPSD(chunk3d_FFT{m}(j, :));
```

2) The getFullPSD function returns the Power Spectral Density

```
function fullPSD = getFullPSD(signal)
    N = length(signal);

    fullPSD = (1/(N)) * (abs(signal).^2);
```

3) plotPSD function can be used to view the periodogram of the frequency domain frame

4) plotTimeSignal function can visualize the time domain frame

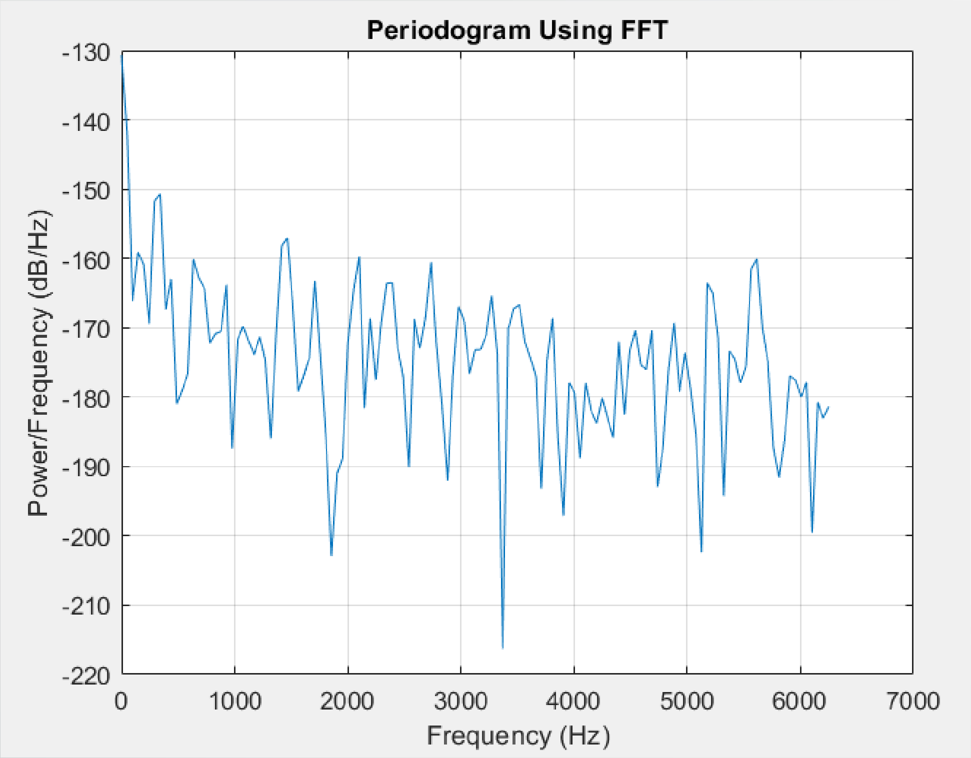
```
plotPSD(chunk3d_PSD{m}(j,:), fs);
plotTimeSignal(chunk3d_window{m}(j, :), fs);
```

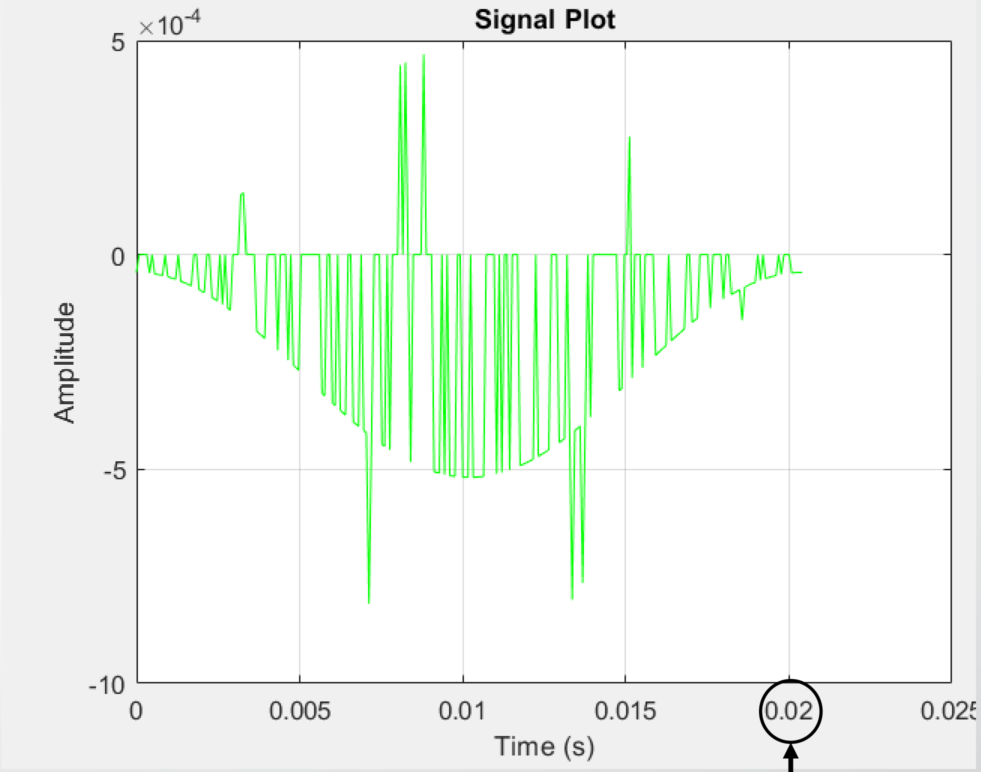
$\frac{2}{FFT\ SIZE}$ * |FFT Frame| ^ 2

1 : N/2 (NYQUIST)

POWER SPECTRAL DENSITY

DSP LAYER





M = 256 gives 20ms frame

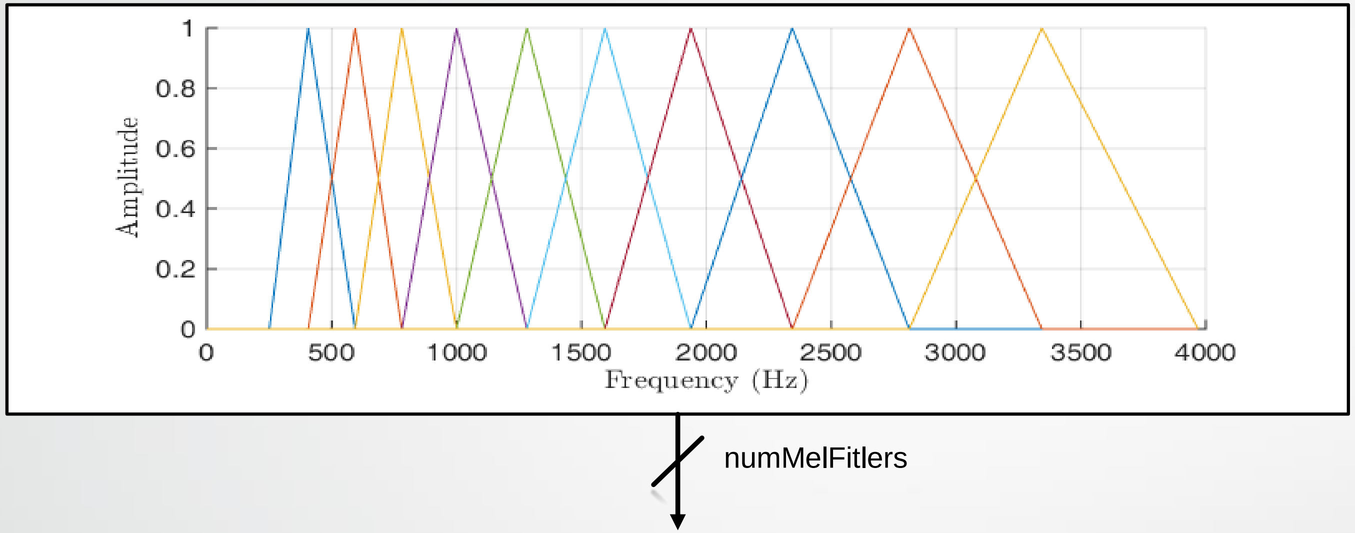
$$\frac{2}{FFT\ SIZE} * |\text{FFT Frame}| \wedge 2$$

1 : N/2 (NYQUIST)

POWER SPECTRAL DENSITY

DSP LAYER

MEL FREQUENCY FILTERS



- After calculating the PSD of each frame, the next step is to filter it with the Mel Frequency filter bank
- The Mel frequency scale effectively models the human ear as a set of filters, and to mimic our hearing capabilities with an application, a filter bank is composed of these frequencies to process the speech signal
- Depending on the configuration of *numMelFilters* in the code, the output will be a set of coefficients that represent the amplitudes at specific mel filters.

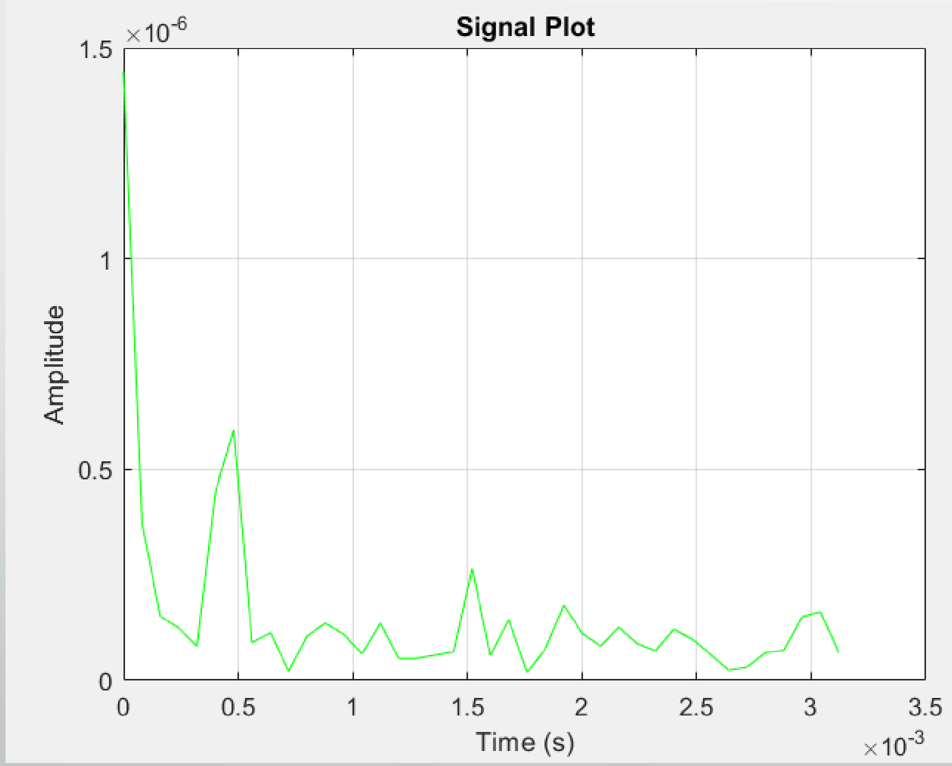
*See Reference 3 for Figure

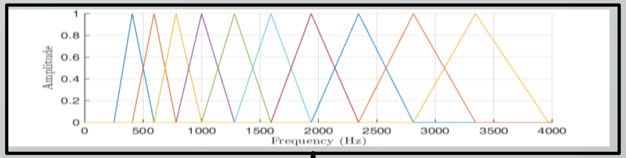
DSP LAYER

1) The PSD frame is now multiplied with a MEL filter bank *melFilters* of size *numMelFilters (Reference 1)*

```
%Take maximum value of each MEL FILTER BANK
chunk3d_MELMAX{m}{j} = max( full(chunk3d_PSD{m}(j, 1:(N/2)) .* melFilters), [], 2);
```

2) The max value is taken from the resultant matrix to remove excessive zeros from the triangle filters, and to obtain a vector of *numMelFilters* coefficients





MEL FREQUENCY FILTERS

MFCC MATRIX

- The last step in the DSP Layer is to do a Discrete Cosine Transform (DCT) to bring the log mel spectrum back into the time-domain (Reference 1).
- The resultant of the *getMFCC* function is a cell array of {8} speakers with M x K matrix each
  - M = numMelFitlers
  - K = # of frames from the first step in the DSP Layer signal chain

DSP LAYER

1) Using the MATLAB function DCT, each log spectrum mel filtered frame is transformed back to the time domain and stored into an array of frames for the speaker.

```
        %stores numMelFilters MFCC's per FRAME
        chunk3d_MFCC_TEST{m}{j} = dct(chunk3d_MELMAX{m}{j});

        %compile each frame into an MFCC array
        MFCC{m}(j,:) = chunk3d_MFCC_TEST{m}{j}';
```

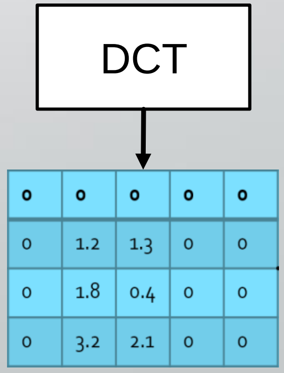
2) The array of frames for each speaker is stored into a cell array *MFCC{m}* where each cell corresponds to the MFCC's for that speaker.

+2 | trainMFCC | trainMFCC{1, 1} | trainArr | trainFrameArr | chunk3d_MELMAX

1x8 cell

	1	2	3	4	5	6	7	8
1	*130x40 dou...*	*135x40 dou...*	*133x40 dou...*	*148x40 dou...*	*181x40 dou...*	*148x40 dou...*	*143x40 dou...*	*145x40 dou...*

3) As shown here, the *trainMFCC* as well as *testMFCC* cell arrays contain the speaker's MFCC data. M columns by K rows represent (*numMelFilters* x # *speaker frames)*

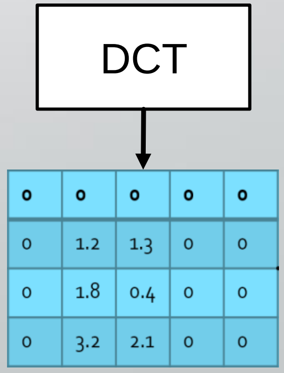


MFCC MATRIX

DSP LAYER

To recap, the *getMFCC* function returns a cell array of MFCC Matrices for all 8 speakers—*train* and *test*. The following slides demonstrate plots of the MFCC's for both *train* and *test* speakers.

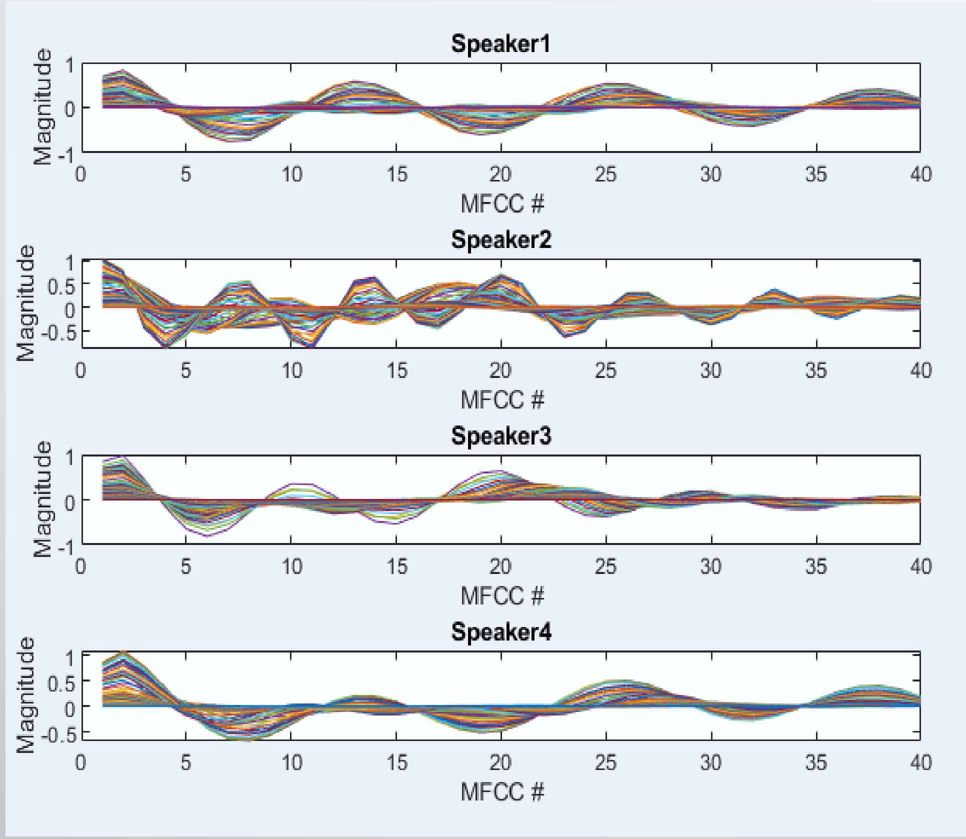
```
trainMFCC = getMFCC(trainFrameArr, numSpeakers, N, numMelFilters, trainFS);
testMFCC = getMFCC(testFrameArr, numSpeakers, N, numMelFilters, trainFS);

toc
fprintf('\n');

%Plot the MFCC's
plotMFCC(trainMFCC, numMelFilters, numSpeakers);
plotMFCC(testMFCC, numMelFilters, numSpeakers);
```



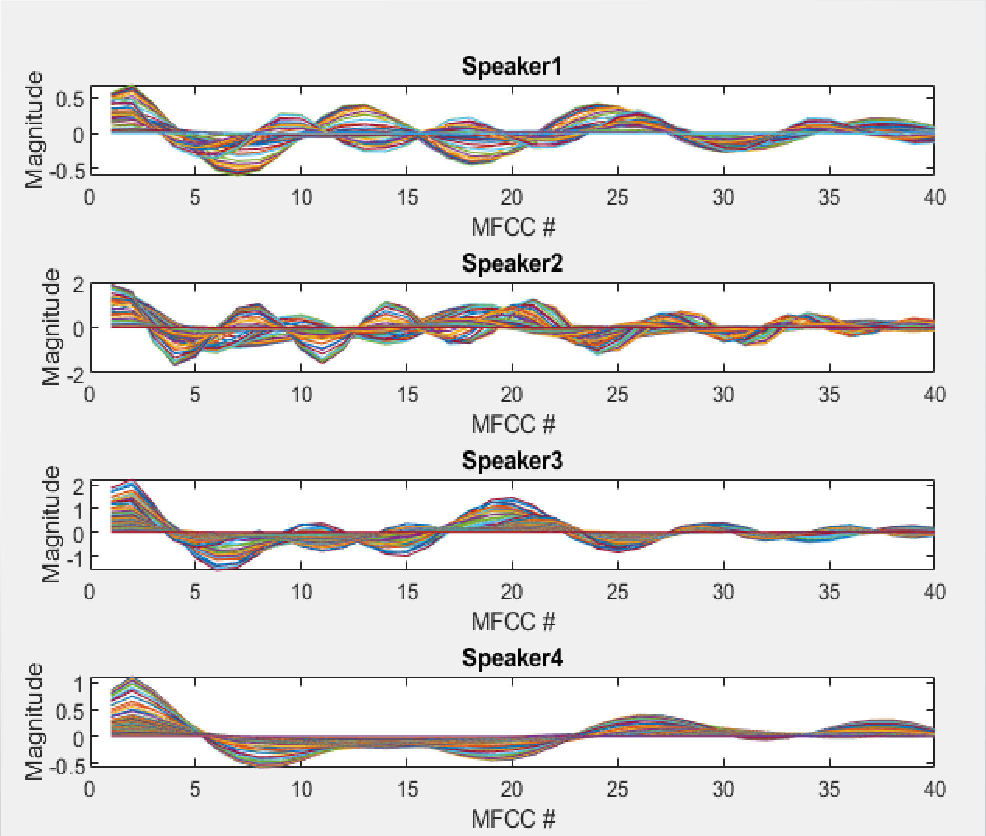
MFCC MATRIX

DSP LAYER

Train vs Test Speakers (1-4) MFCC's



TRAIN
SPEAKER
MFCC's

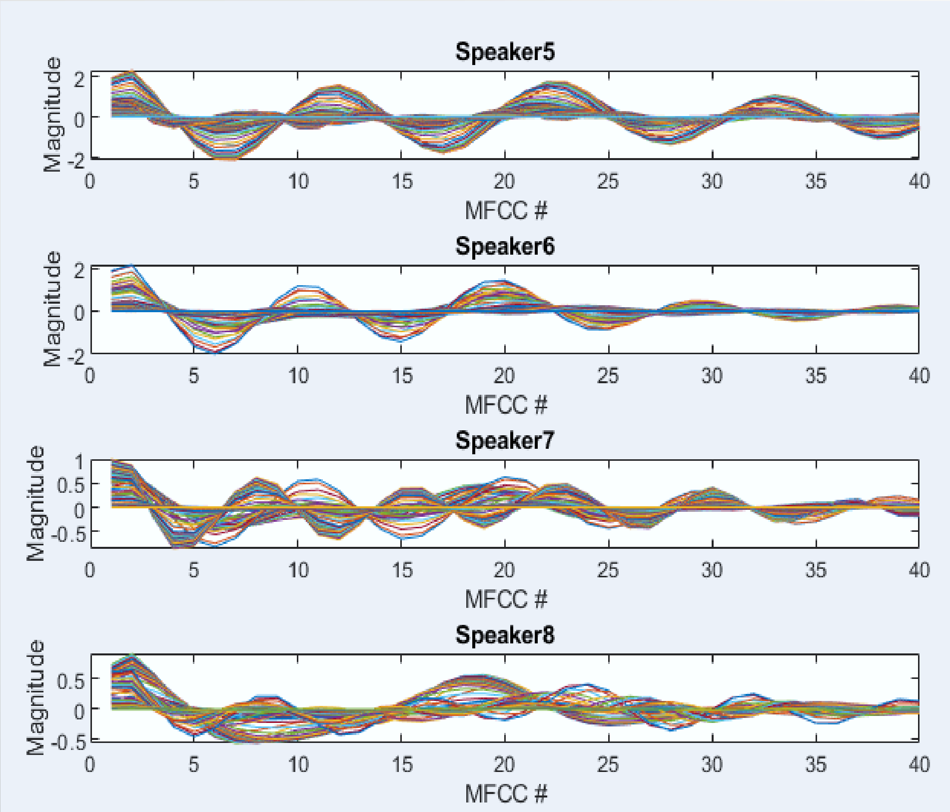


TEST
SPEAKER
MFCC's

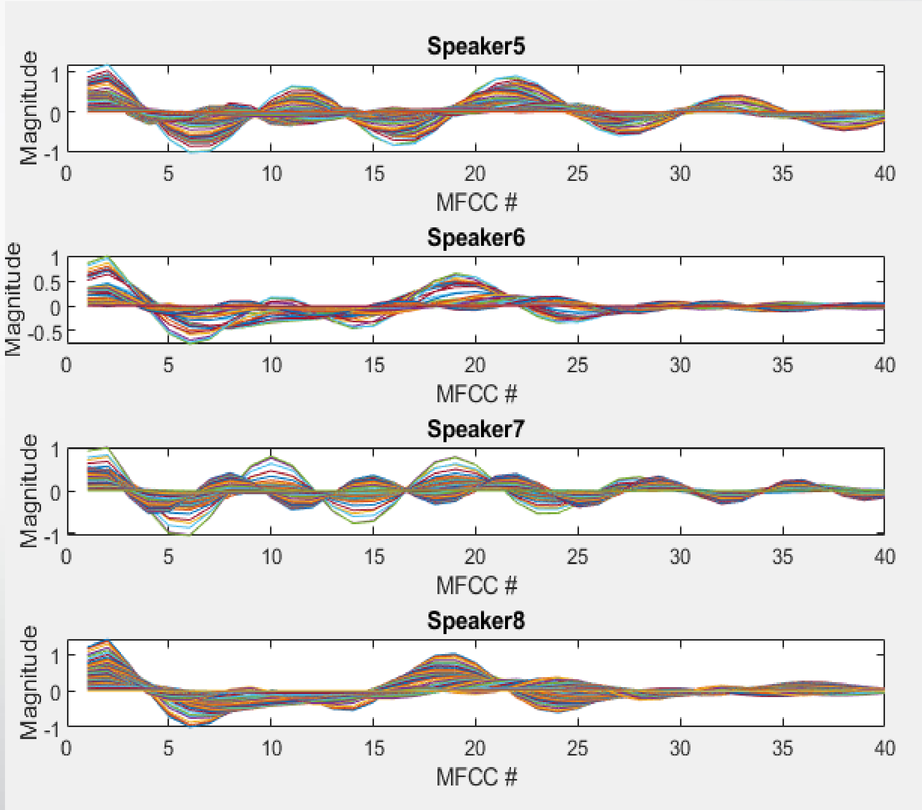
*Each speaker has multiple plots because there are K frames per MFCC array due to the input signal length.

DSP LAYER

Train vs Test Speakers (5-8) MFCC's



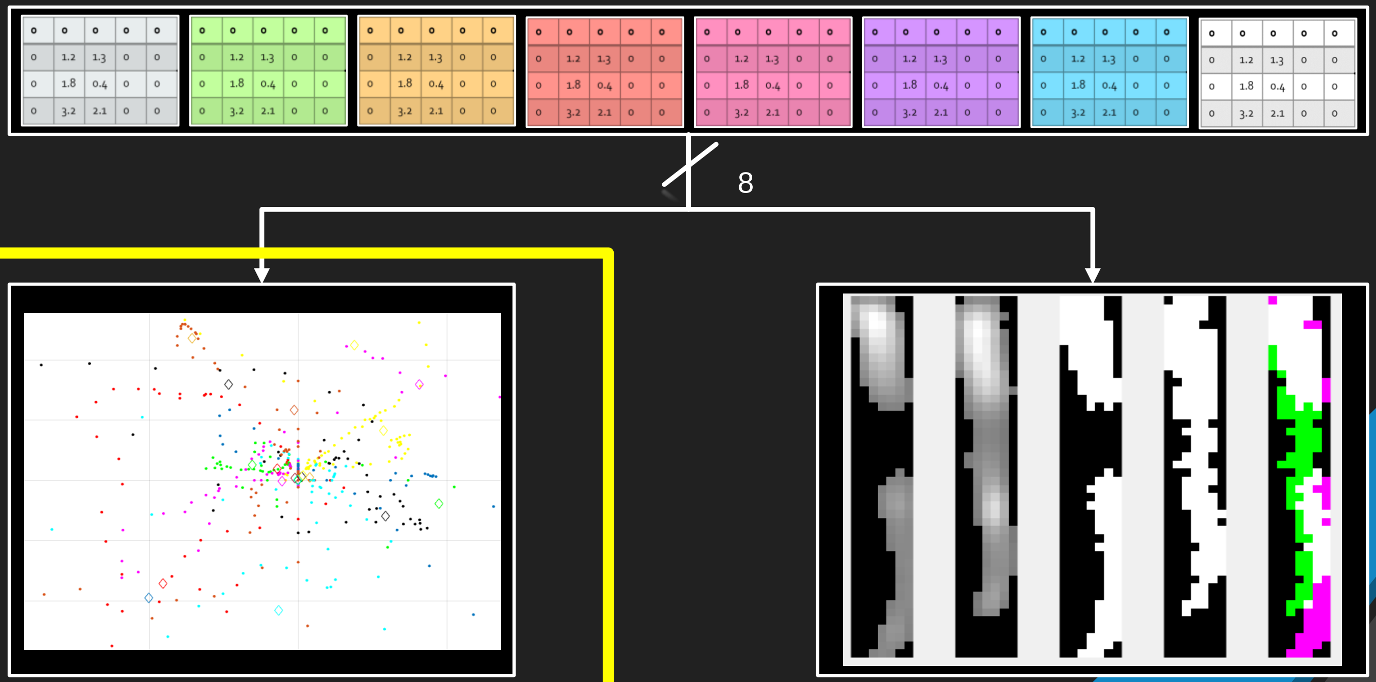
TRAIN
SPEAKER
MFCC's



TEST
SPEAKER
MFCC's

*Each speaker has multiple plots because there are K frames per MFCC array due to the input signal length.

MODEL LAYER IN MATLAB



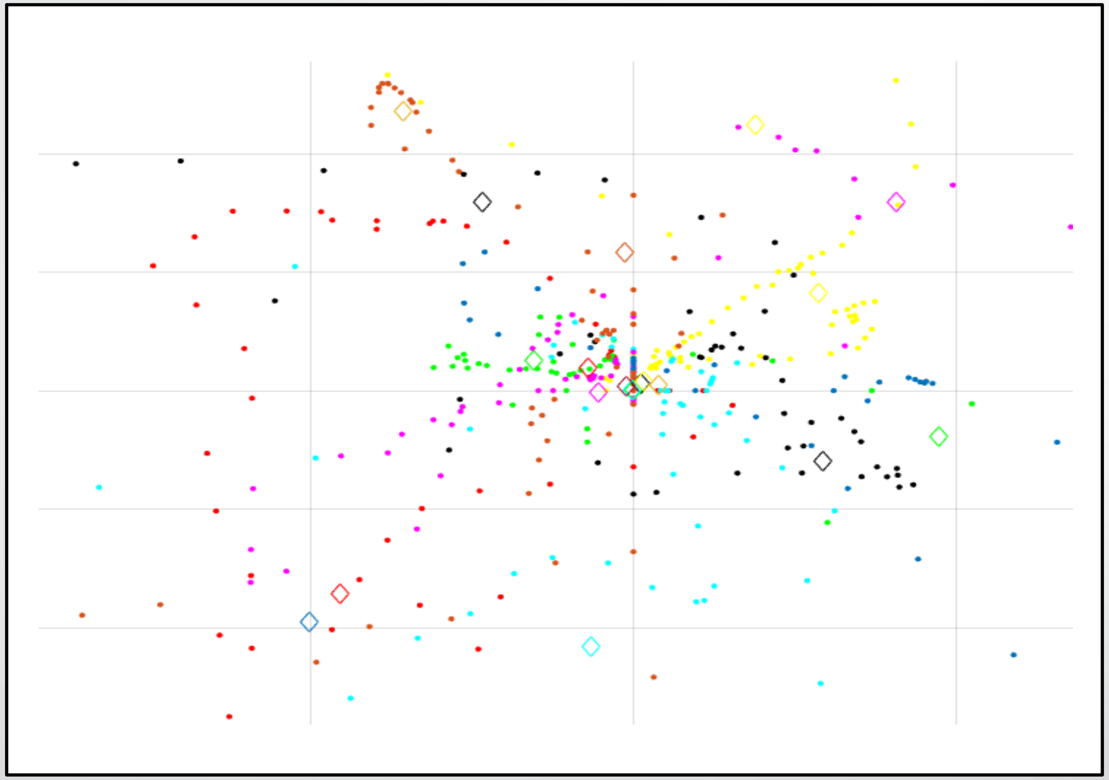
8
TRAINING SPEAKER
MFCC MATRICES

VECTOR QUANTIZATION
MODEL

IMAGE RECOGNITION
MODEL

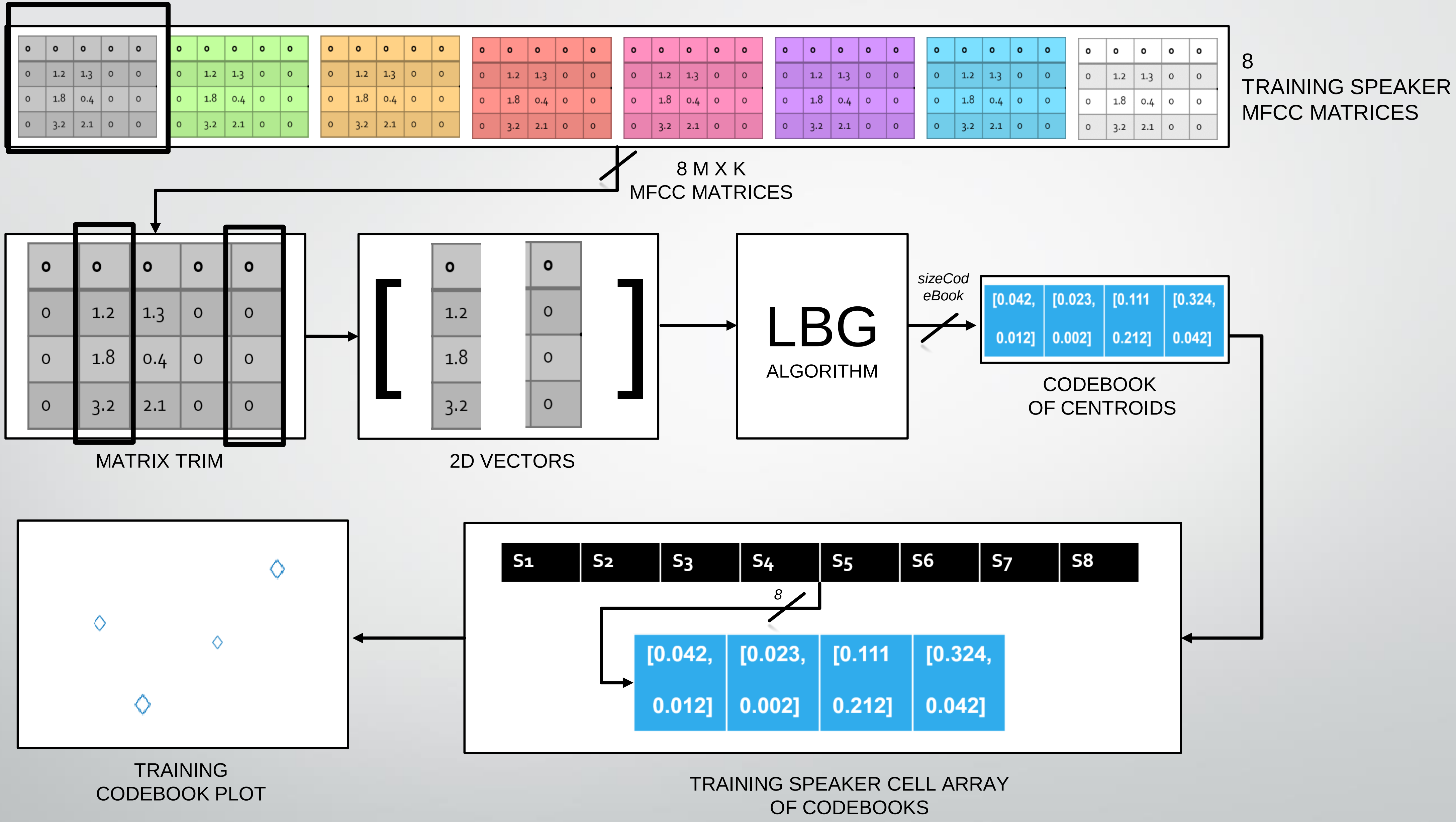
MODEL LAYER: VQ

VECTOR QUANTIZATION (VQ) MODEL



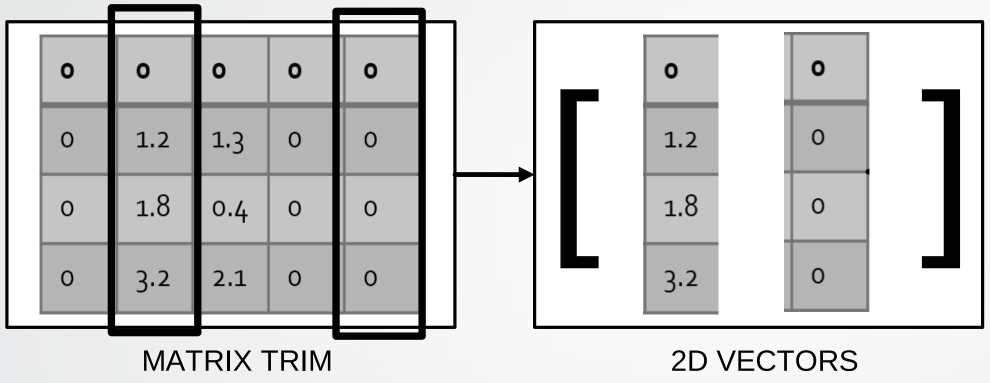
- The VQ model is based on the LBG algorithm. (Reference 1)
- The main idea is to take two columns from the *testing* MFCC array—say MFCC's [4,16]—and compare their Euclidean distance to the *training* MFCC array.
- The shortest Euclidean distance, or VQ Distortion, is the recognized speaker

MODEL LAYER: VQ



MODEL LAYER: VQ

TRAINING MATRIX TRIMMING



- Before creating the codebooks with the LBG algorithm, the KxM matrix must be trimmed to create 2D vectors.
- The LBG algorithm can be implemented in more than 2 dimensions, but this project is using two for simplicity.

MODEL LAYER: VQ

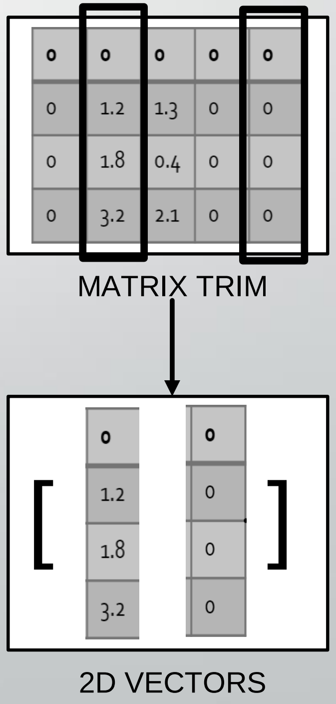
1) The configuration code block at the top of the file specifies which M column dimensions to choose from the MFCC array (K rows (frames) X M columns (MFCC’s)

```
VQDim = [30,4];            %which MFCC numbers to use for VQ section
```

2) VQDim must exist between 1 and *numMelFilters*. The order of the two dimensions is insignificant because it will be uniform across both *train* and *test* speakers

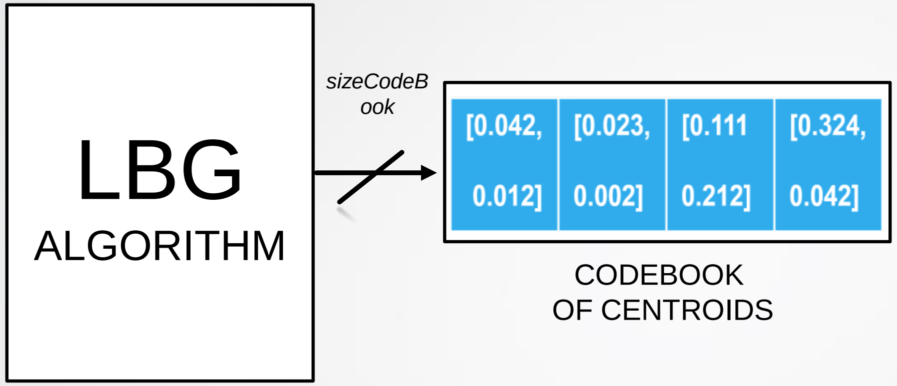
3) A simple loop is required to traverse the array and vectorize the output for all speakers in the given *train* or *test* array

```
for j=1:1:framesSpeaker
    %MFCCavg{i}(j) = mean(MFCCarray{i}(j,:));

    %create vectors using 4th and 16th dimensions of 40 MFCC's
    MFCCvectors{i}(j,:) = [MFCCarray{i}(j,VQDim(1)) MFCCarray{i}(j,VQDim(2))];
    %MFCCvectors{i}(j,:) = MFCCvectors{i}(j,:);
```



MODEL LAYER: VQ

CODEBOOK OF CENTROIDS

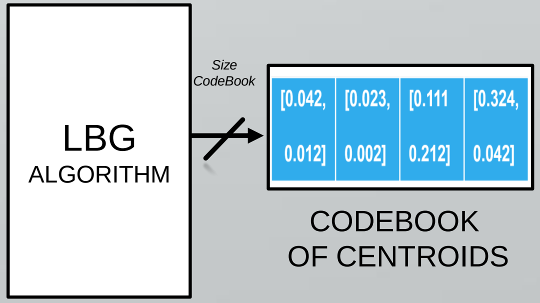


- With the 2D MFCC vectors for each speaker calculated, now the *codebook* for the vector quantizer is generated.
- The LBG algorithm is as follows (for one speaker):
  1) Calculate the MEAN of each dimension. The resultant is called a *centroid* and is a 2D vector. [x, y]
  2) Split the centroid into two using the following formula: *centroid**(1-0.01), *centroid**(1+0.01)
  3) Of the remaining MFCC vectors of the speaker, calculate the Euclidean distance between itself and each *centroid*, and assign it to the closest *centroid.*
  4) After all the MFCC vectors are assigned to a centroid, calculate the mean of each dimension to obtain new centroids again (list of *centroids* forms a *codebook*). Do steps 1-3 until codebook reaches *sizeCodeBook*.

MODEL LAYER: VQ

- *InitCentroid* on line 590 creates the first centroid. Then its split by * (1+-0.01) on lines 594-595.

```
581  eps = 0.01; %splitting parameter
582
583  % BEGIN SPLIT
584
585  for i=1:1:numSpeakers
586      sizeFrames = size(MFCCvectors{i},1);
587
588      %calculate the initial centroid of the currenet speaker's codebook
589      %centroid = [mean(x vals), mean (y vals)]
590      initCentroid = [mean(MFCCvectors{i}(:,1)) mean(MFCCvectors{i}(:,2))];
591      %initCentroid = initCentroid*2;
592
593      %how many times are we splitting the codebook
594      FullCodeBook{i}{1}(1,:) = initCentroid * (1-eps);
595      FullCodeBook{i}{2}(1,:) = initCentroid * (1+eps);
```



MODEL LAYER: VQ

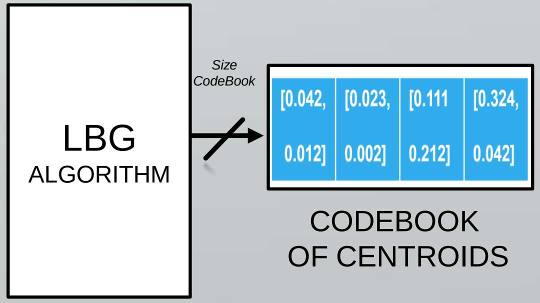
- Calculating Euclidean distance between MFCC vectors and *centroids*.

```
630
631         dist = zeros(1, sizeCB); %for holding the distances from training dat
632
633         for j=1:1:sizeCB
634             currentCBVec = FullCodeBook{i}{numCWUpdate, j}(1,:); %first row
635
636             dist(j) = ( currentCBVec(1) - currentVec(1) )^2 + ...
637                 ( currentCBVec(2) - currentVec(2) )^2;
638
639             %dist(j) = sqrt(dist(j));
640
641         end
642
643         %assign currentVec to closest codeword
644         closestCW = find(dist == min(dist)); %get index of closest codeword
```

- *Dist* is a vector that holds the relative distances between the *current MFCC vector* and the centroid of interest
- *Line 644* assigns the *MFCC* vector to the closest centroid in *Dist*

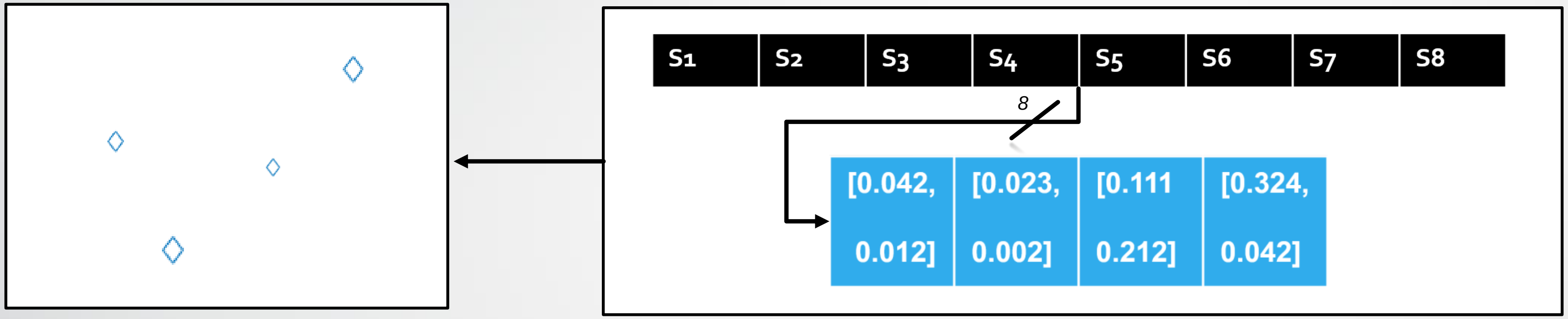
```
652     %now recalculate the centroids using assigned vectors
653     for l=1:1:sizeCB
654         FullCodeBook{i}{numSplits+1,l} = [mean(FullCodeBook{i}{numSplits,l}(:,1)) ...
655             mean(FullCodeBook{i}{numSplits,l}(:,2))];
656     end
```

- Lastly, lines 654-655 recalculate the centroid after assigning all *MFCC vectors* to it



MODEL LAYER: VQ

PLOT CODEBOOK



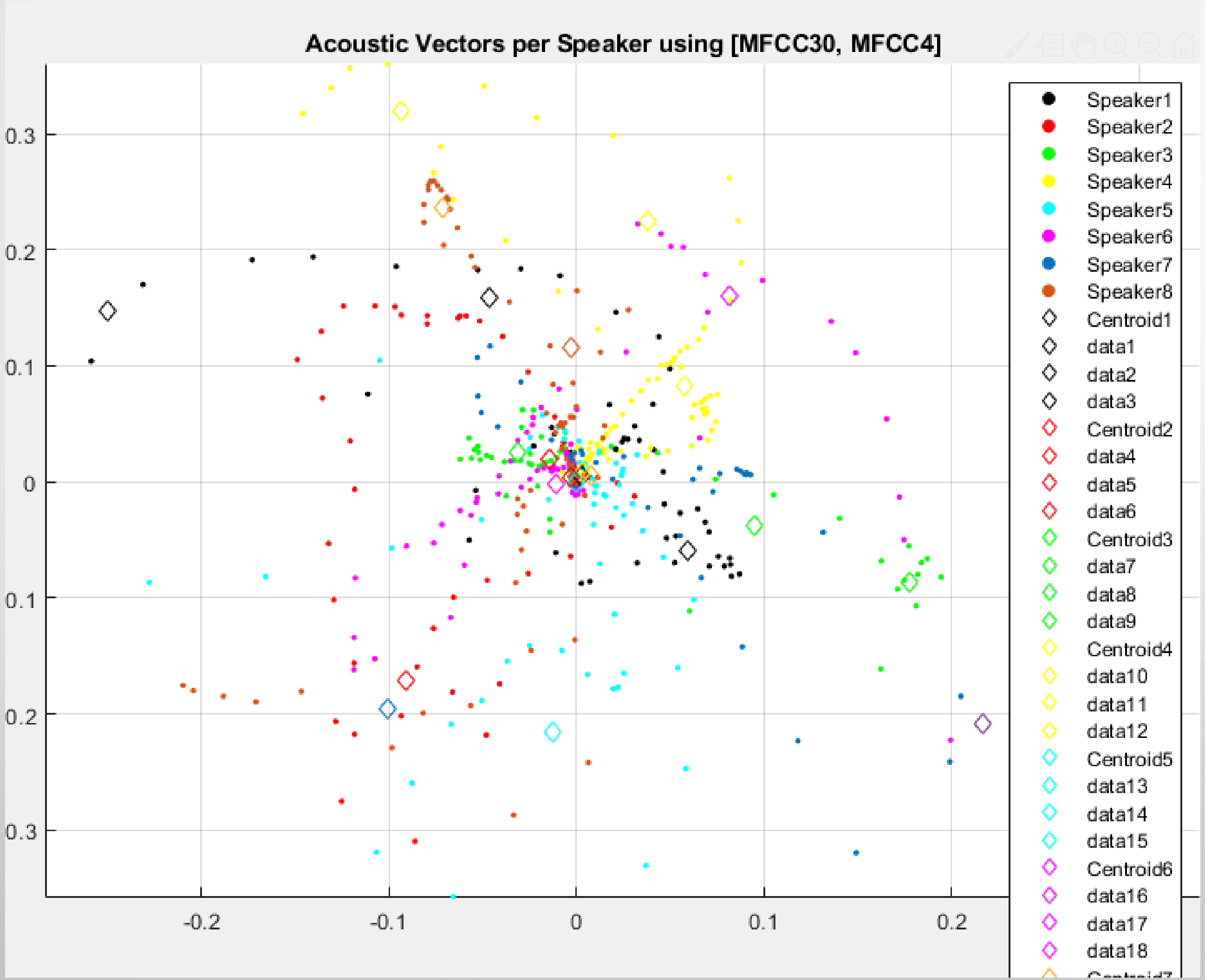
TRAINING CODEBOOK PLOT

TRAINING SPEAKER CELL ARRAY OF CODEBOOKS

- Next step is to plot the *codebooks* for each speaker so that we can see their similarities on the 2D plane
- When the *test* speaker is fed to the model, it will also go through the LBG algorithm, but it will be compared against the *training* codebooks.
- The sum of the *Euclidean Distance* for each of the *test* centroids *to* <u>each</u> of the *train* centroids is calculated
- The *test* speaker with the lowest sum is the recognized speaker

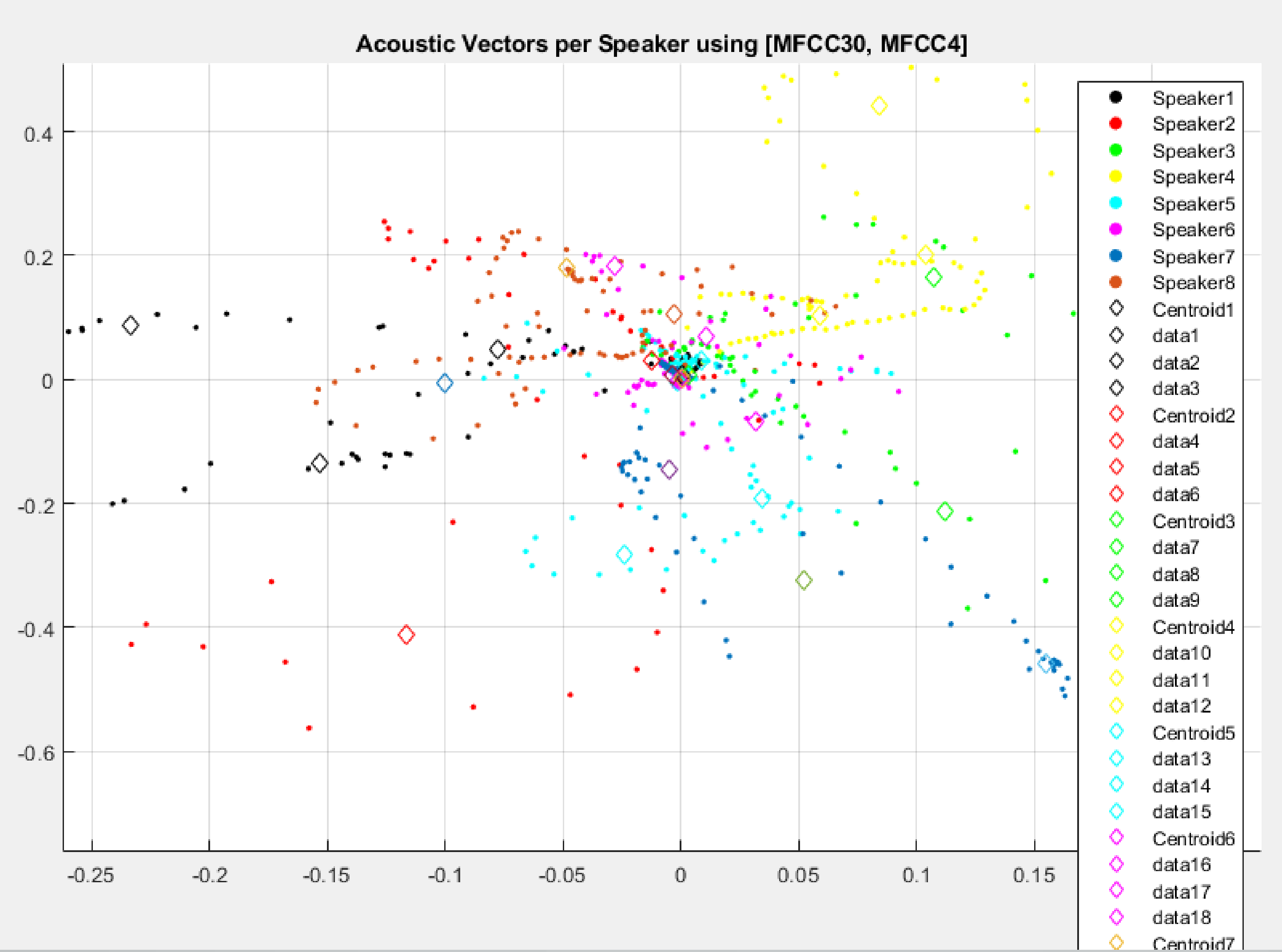
MODEL LAYER: VQ

TRAIN Speaker Centroids + MFCC vectors

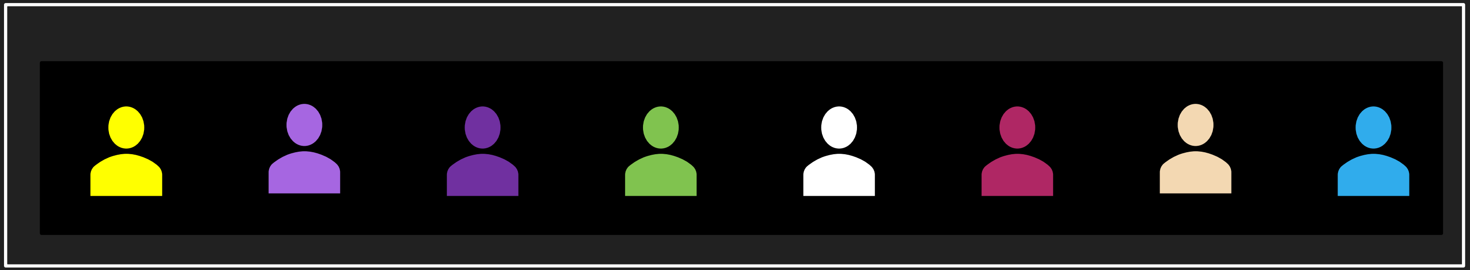


MODEL LAYER: VQ

TEST Speaker Centroids + MFCC vectors



RECOGNIZED SPEAKER

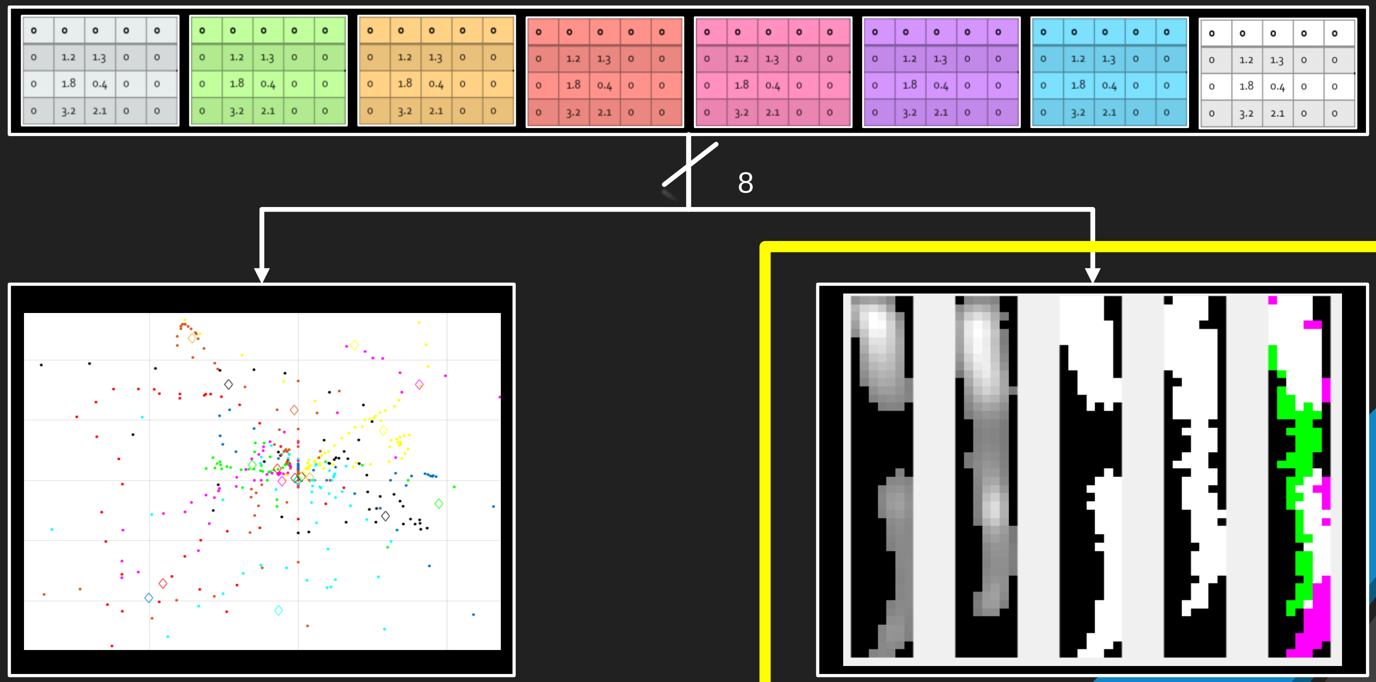


FULL RESULTS FOR VQ

```
Calculating MFCCs
Elapsed time is 1.076051 seconds.

Creating Codebooks for Vector Quantization
Elapsed time is 0.425464 seconds.

Matching Train Speakers to Test Speakers
Elapsed time is 0.005297 seconds.

Matches found.
Test:__1  2  3  4  5  6  7  8
Match:1  2  4  3  5  5  3  8
Recognition rate: 50.000000 %
```

- The total running time is 1.507 seconds
- With a 50% recognition rate, the Image Recognition avenue was motivated and is presented next

MODEL LAYER IN MATLAB



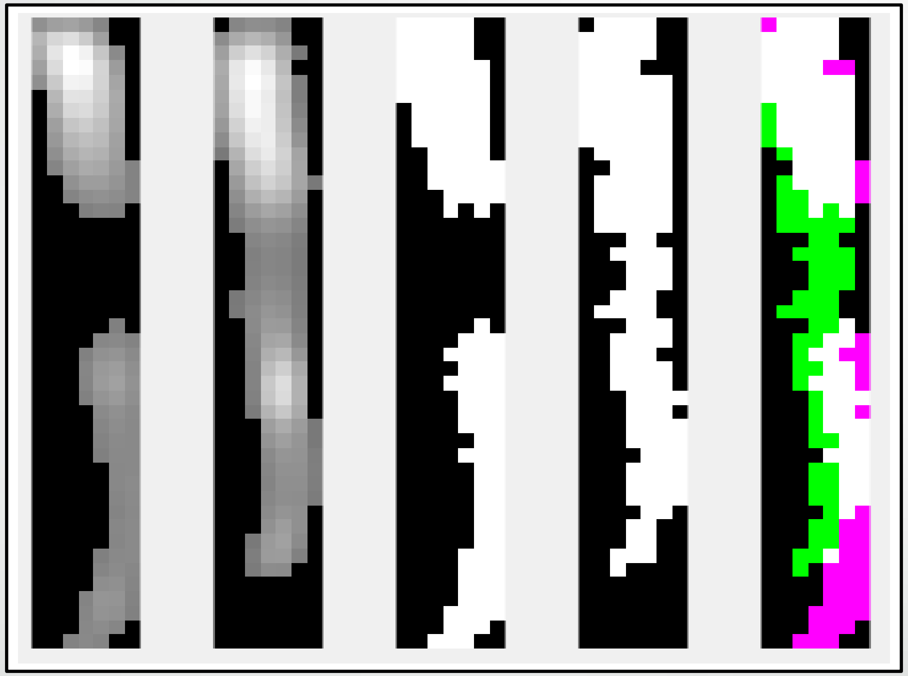
8
TRAINING SPEAKER
MFCC MATRICES

VECTOR QUANTIZATION
MODEL

IMAGE RECOGNITION
MODEL

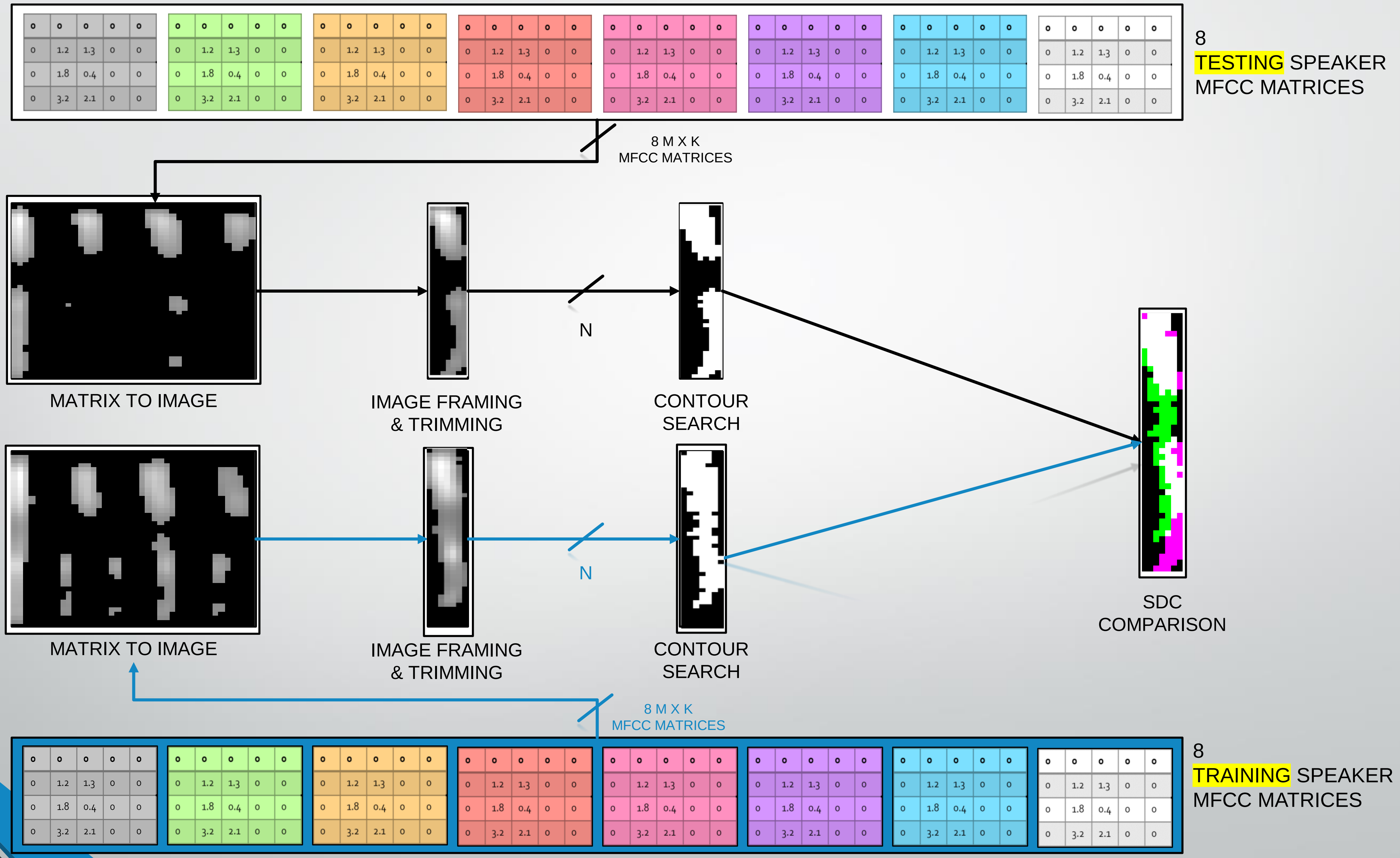
MODEL LAYER: IR

IMAGE RECOGNITION (IR) MODEL



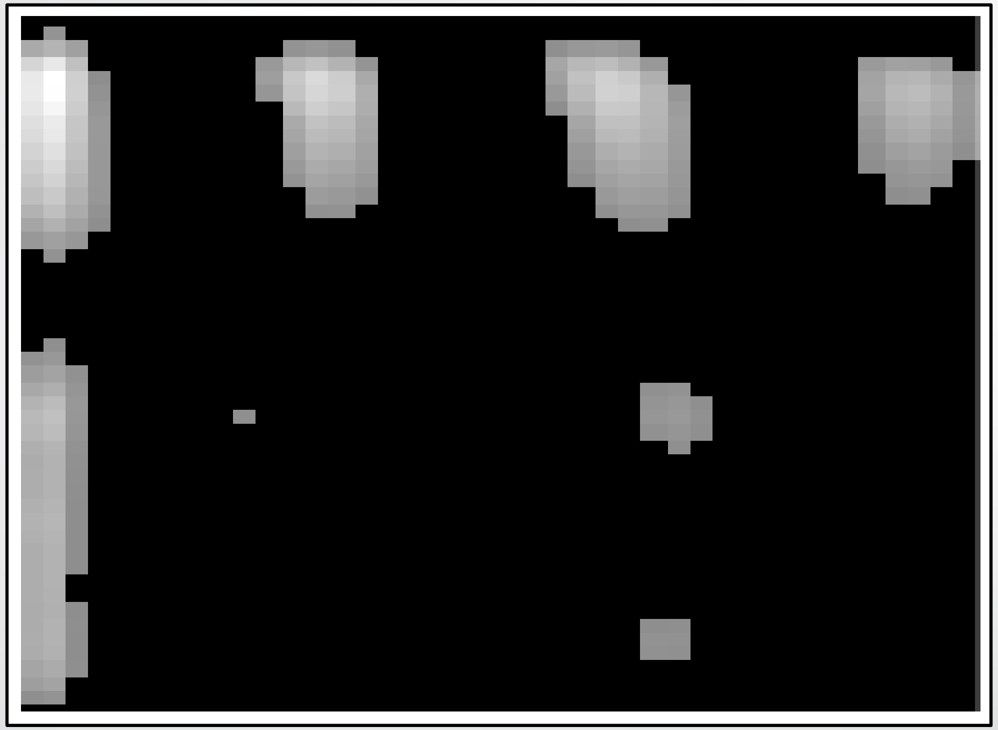
- The IR model is based on the author’s design.
- Using the Sorensen-Dice coefficient (SDC), the MFCC matrix of the *test* speaker is transformed into a greyscale image and then compared to the image of *train* speaker (ground truth)
- The higher the Sorensen-Dice coefficient, the closer the match

MODEL LAYER: IR



MODEL LAYER: IR

MATRIX TO IMAGE & FRAMING



- The MFCC matrices of both *train* and *test* are converted each to 16-bit grayscale images
- Given the MFCC matrices contain numbers in the real domain, the algorithm sweeps vectorizes the matrix into one row and normalizes all the data between 0 and 2^16-1.
- 8 and 32 bits were also tested but the former did not give as much accuracy and the latter did not provide an advantage over 16 bits

MODEL LAYER: IR

- The image is subdivided into four frames because of the image’s logical arrangement, but can be changed in the configuration section

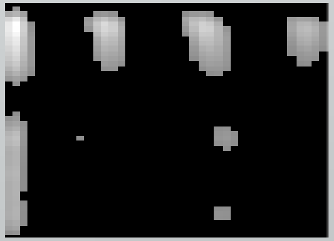
```
% IMAGE RECOGNITION
chunks = 4;                %How many image "frames" to use (2,4)
```

```
index = [1:floor(numMelFilters/chunks):numMelFilters numMelFilters];
```

- The iterator *i* signifies the current *test* speaker to analyze. The iterator *j* signifies the current *train* speaker to compare the *test* speaker to.

```
        %Convert MFCC for each speaker into a 16 bit image
        test = getMFCCimg(testMFCC{i}(:, index(j):index(j+1)), 16);
        train = getMFCCimg(trainMFCC{k}(:, index(j):index(j+1)), 16);
```

- The MFCC array is subdivided into the boundaries specified by the *index* vector, which contain the coordinates of the “frames” of the image. These are converted into images iteratively.
- The second parameter of the function specifies the number of bits to normalize the data to



MATRIX TO IMAGE

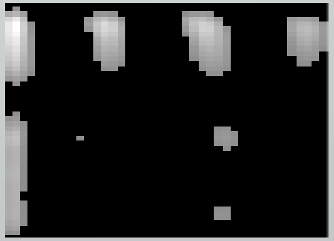
MODEL LAYER: IR

- Inside the *getMFCCImg* function, the array data is normalized into the # of bits specified by the caller using the following function (Reference 2)

```
for i=1:1:length(MFCCVector)
    %puts in range 0 to 2^bits
    normalMFCCVector(i) = floor(2^bits * (MFCCVector(i)-minval) ...
        / (maxval - minval) );
end
```

- The last step is to convert the matrix into uint(bits) datatype, and set data points under a certain threshold to 0 (black) so that the active contour can better detect the MFCC “islands”

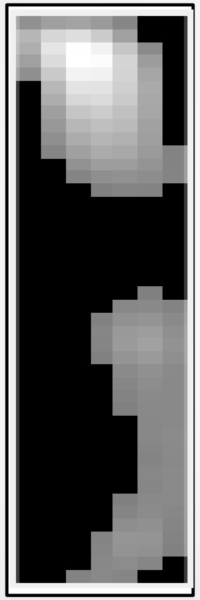
```
elseif(bits ==16)
    MFCCimg = uint16(normalMFCCArr);
    %MFCCimg( MFCCimg < (zeroVal+5000 )) = 0; %7/8 recognition
    MFCCimg( MFCCimg < (zeroVal+5000 )) = 0; %7/8 recognition
```



MATRIX TO IMAGE

MODEL LAYER: IR

TRIMMING



- The algorithm “captures” the “islands” in the MFCC image and makes frames out of the image by removing unnecessary black spaces (0 in value).

MODEL LAYER: IR

- The MFCC images *train* and *test* are sent to the *getSDC* function to calculate the SDC.

```
SDC = getSDC(train, test); %Get the SDC with Train and Test images
SDCBank{1,i}{1,k}(j) = SDC; %Store the SDC in the ith'speakers cellarray
```

- To trim the images, the *getSDC* function calculates the amount of non-zero spaces first

```
%Get values that are not zero
[trainx,trainy] = find(trainIMGMFCC > 0);
[testx,testy] = find(testIMGMFCC > 0);
```

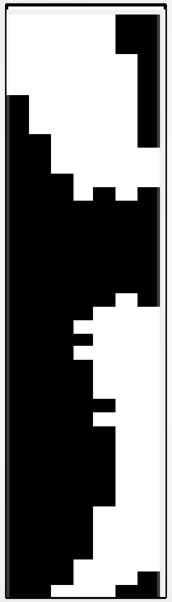
- Then the image is cropped by defining the boundaries of the non-zero values

```
%Get domain and range from training data
xone = min(trainx);
xtwo = max(trainx);
yone = min(trainy);
ytwo = max(trainy);

%Crop image for actual data, removing the black spaces
TRAINIMG = trainIMGMFCC(xone:xtwo, yone:ytwo);
```

TRIMMING

MODEL LAYER: IR

CONTOUR SEARCH



- With the image frame cropped out of unnecessary dark spaces, the light portion of the image is “contoured” using the “activecontour” MATLAB function from the Image Processing Toolbox

MODEL LAYER: IR

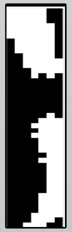
- To find the active contour of the trimmed image frame, the function needs to have a “mask” image first.

```
332  %Create contour mask for training image
333  maskTrain = false(size(newTrainIMG));
334  maskTrain(1:end, 1:end) = true;
```

- The mask is essentially a matrix of all zero’s (black spaces), and an initial contour is defined on line 334 (a portion of the image to begin the search) as all ones (white spaces)

```
336  %Create contour image for training data using the mask
337  BWTrain = activecontour(newTrainIMG, maskTrain, 500);
```

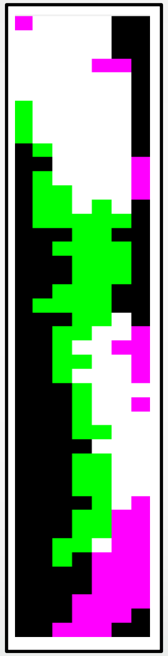
- Lastly, the contour of the image is found using the MATLAB function. The third parameter represents the number of iterations for the function to search for the contour of the image



ACTIVE CONTOUR

MODEL LAYER: IR

SDC COMPARISON



- The last step of the image processing section is to compare both *train* and *test* contours generated by the *getSDC* function using the Sorensen-Dice coefficient
- This is accomplished using the *dice* MATLAB function from the Image Processing Toolbox

MODEL LAYER: IR

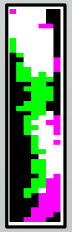
- The *activecontour* function is simply called and the SDC is returned

```
%Create contour image for training data using the mask
BWTrain = activecontour(newTrainIMG, maskTrain, 500);
```

- This application calculates the activecontour for all speakers, *train* and *test*. Using a chunk size of 4, this means the function is called

  (1 train+ 1 test) * 4 chunks * 8 different speakers = 64 times

- Further improvement to this algorithm can be made to take advantage of the previously calculated contours.



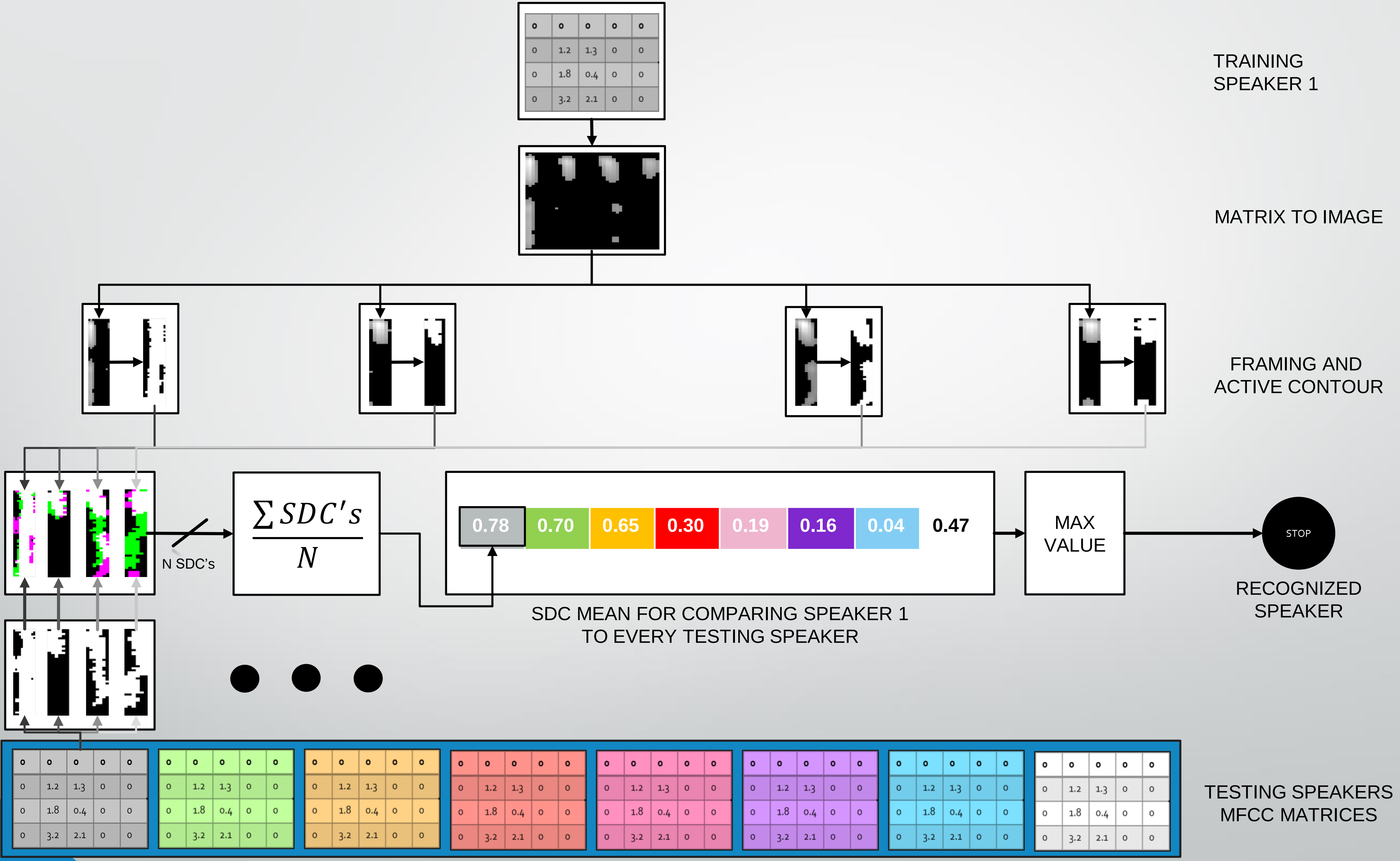
SDC COMPARISON

IMAGE RECOGNITION (IR) EXAMPLE

0	**0**	**0**	**0**	**0**
0	1.2	1.3	0	0
0	1.8	0.4	0	0
0	3.2	2.1	0	0

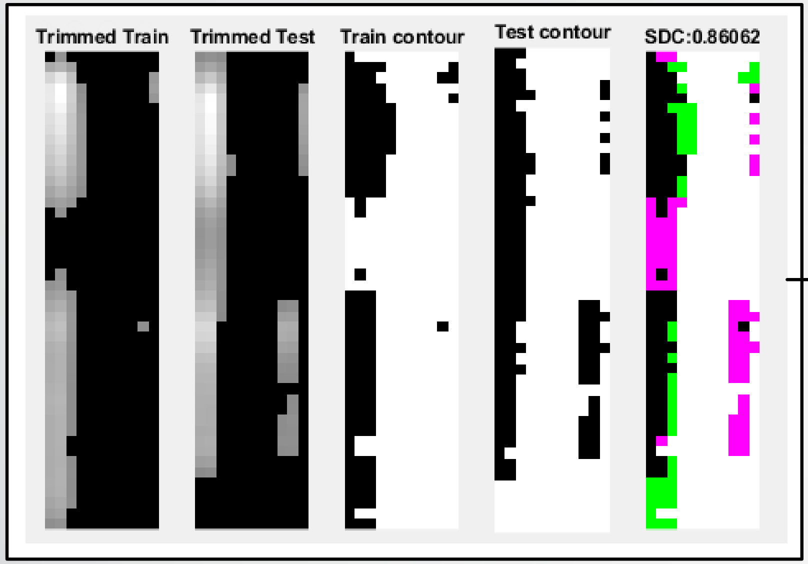
- The following slide showcases how the IR model performs the algorithm to recognize the *testing* speaker 1 (which is unknown to the model)
- To find the recognized speaker, the mean of the SDC's of each image frame of both *train* and *test* speakers are compiled for each *test* speaker
- The *train* speaker with the highest mean is the matched speaker
- The diagram shows the input as the *train speaker,* but since the algorithm is interchangeable and calculates all speakers at once, it has been left alone

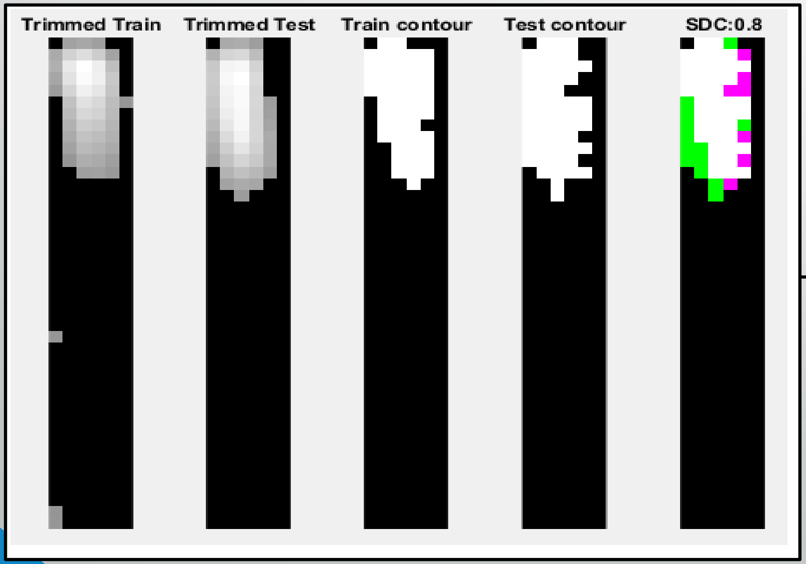
MODEL LAYER: IR



MODEL LAYER: IR

RECOGNIZING SPEAKER 1

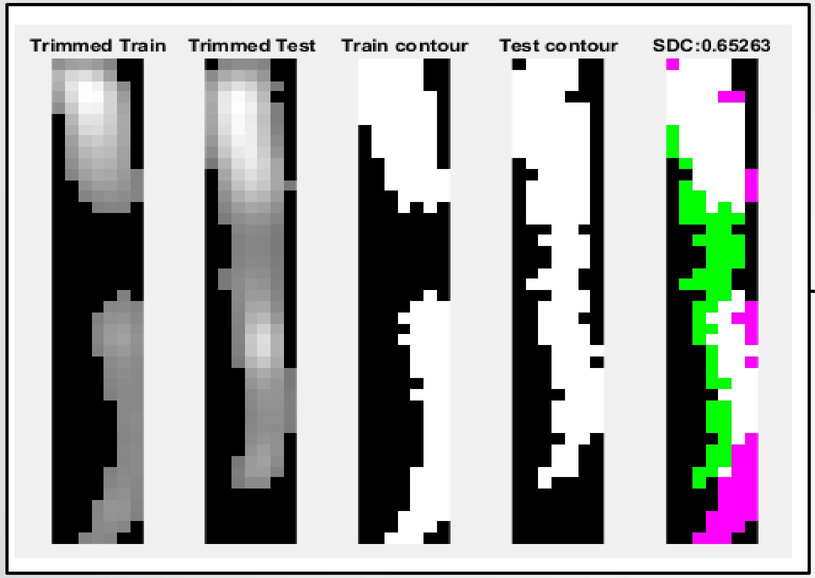


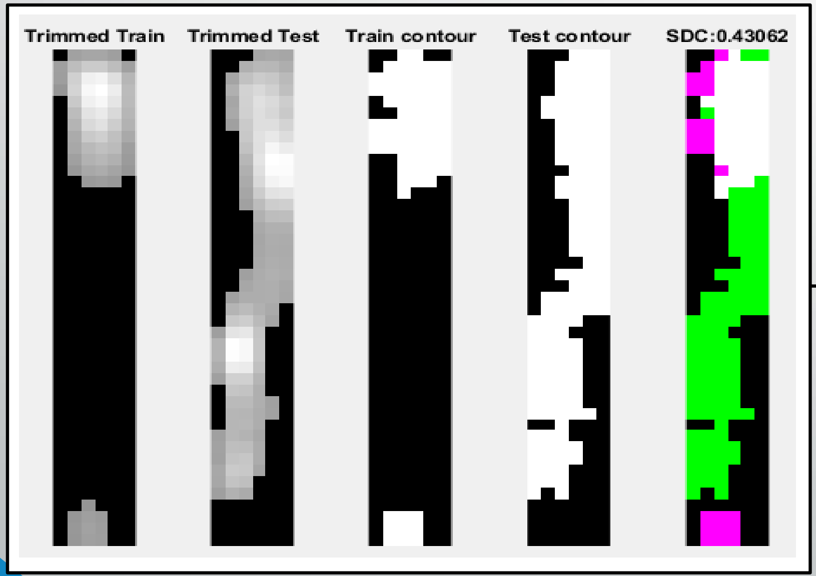


Testing: 1 Training: 1 Range: [1, 11]
Dice coeff: 0.860622
Testing: 1 Training: 1 Range: [11, 21]
Dice coeff: 0.800000
Testing: 1 Training: 1 Range: [21, 31]
Dice coeff: 0.652632
Testing: 1 Training: 1 Range: [31, 40]
Dice coeff: 0.430622

MODEL LAYER: IR

RECOGNIZING SPEAKER 1

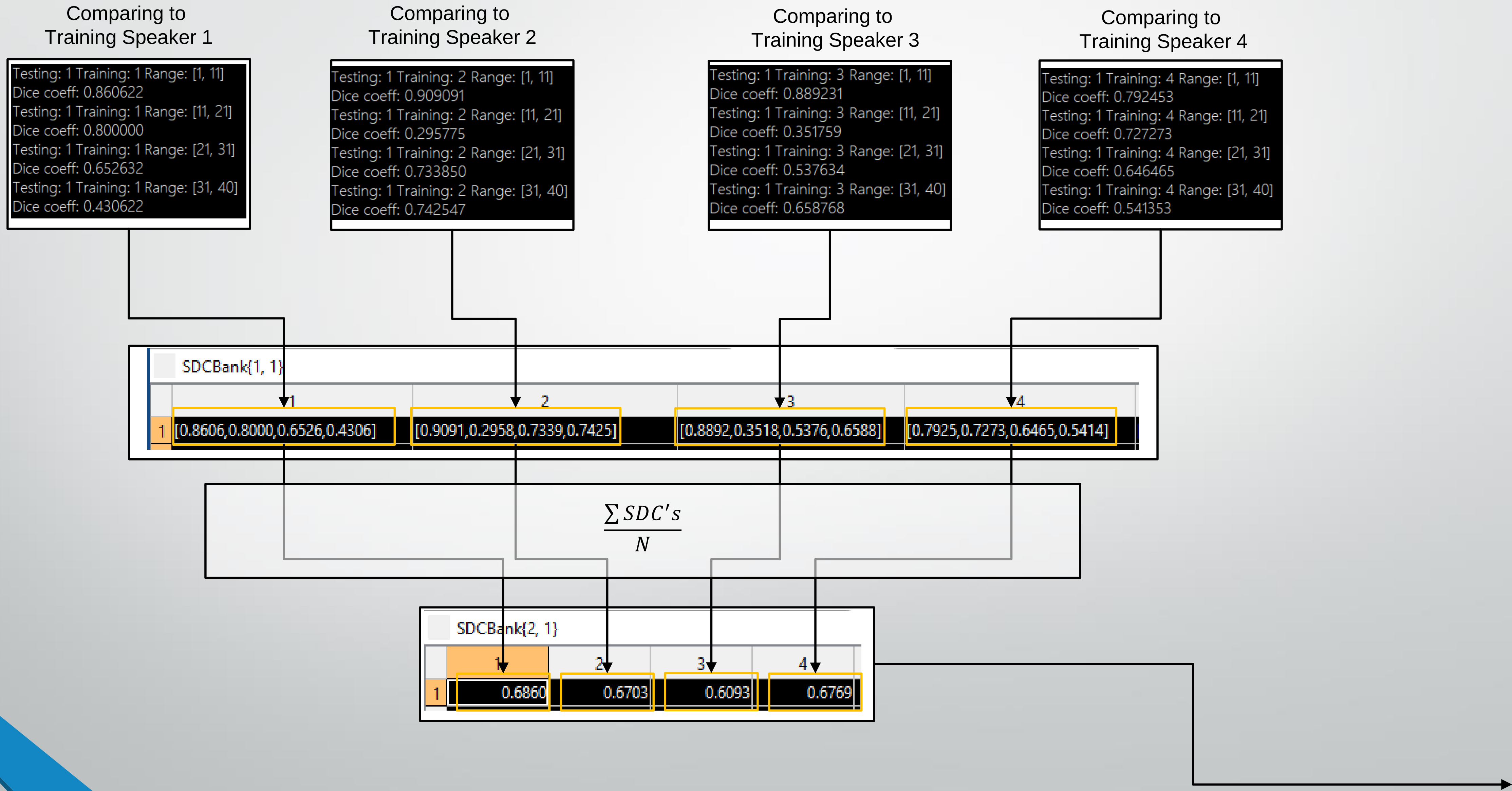




Testing: 1 Training: 1 Range: [1, 11]
Dice coeff: 0.860622
Testing: 1 Training: 1 Range: [11, 21]
Dice coeff: 0.800000
Testing: 1 Training: 1 Range: [21, 31]
Dice coeff: 0.652632
Testing: 1 Training: 1 Range: [31, 40]
Dice coeff: 0.430622

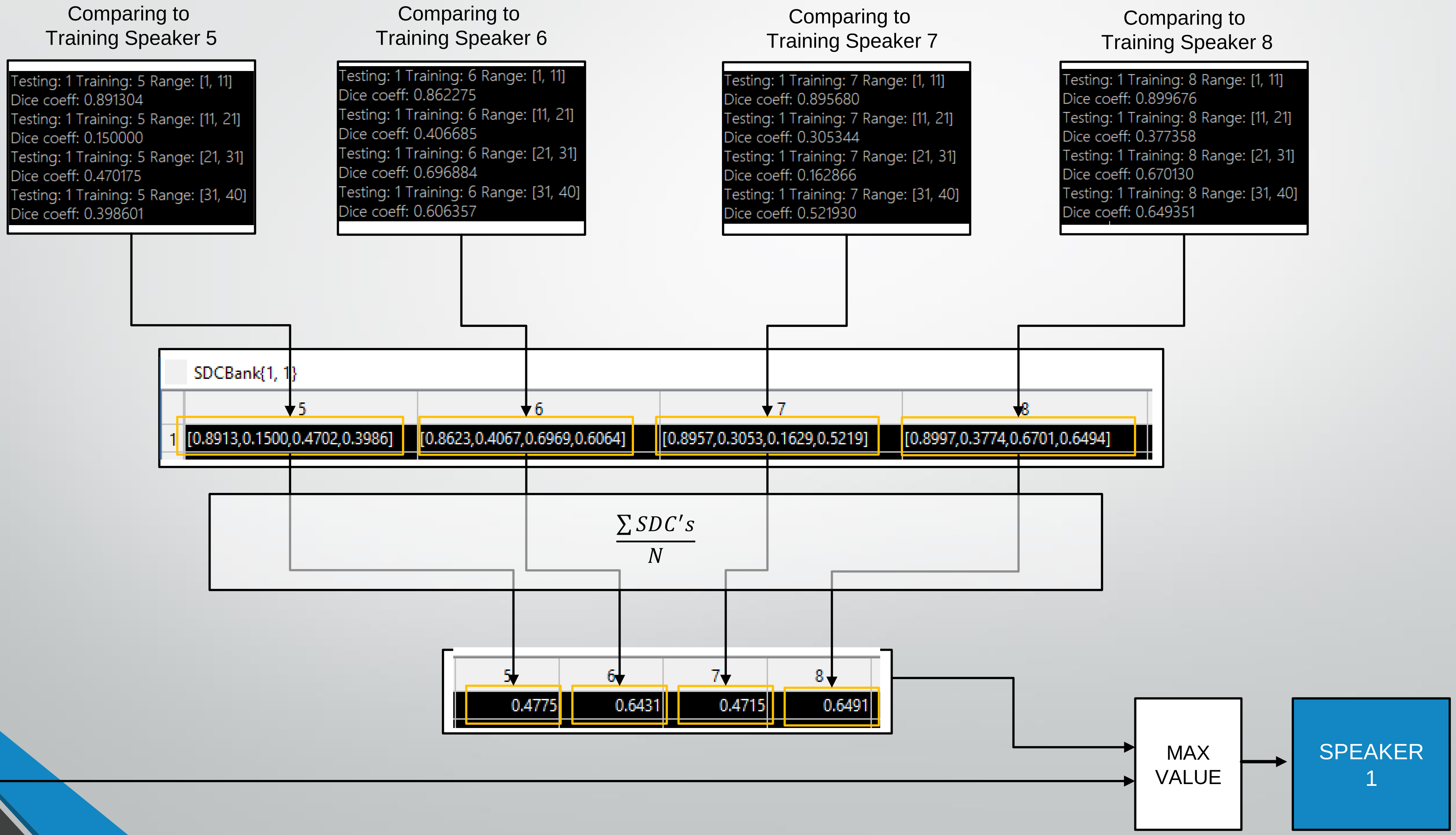
MODEL LAYER:
IR

RECOGNIZING SPEAKER 1



MODEL LAYER:
IR

RECOGNIZING SPEAKER 1



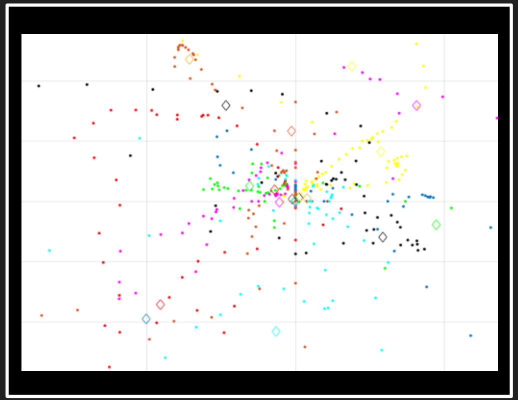
MODEL LAYER: IR

FULL RESULTS FOR IR

```
IR Model: Matching speakers
Elapsed time is 25.462397 seconds.

Matches found.
Test:__1  2  3  4  5  6  7  8
Match:1  2  3  6  5  6  7  8
Recognition rate: 87.500000 %
```

- Including the MFCC calculation time, the total running time is 26.538 seconds
- The recognition rate is 87.5%
- Although the elapsed time greatly increased over the VQ model, the recognition rate also increased dramatically

CONCLUSION



VECTOR QUANTIZATION MODEL

VS

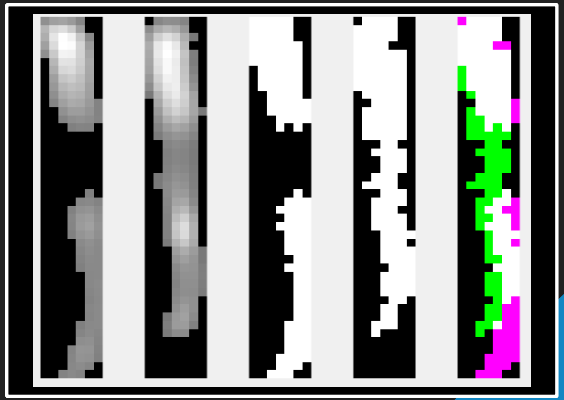


IMAGE RECOGNITION MODEL

CONCLUSION

COMPARISON CHART

	Vector Quantization	Image Recognition
Runtime	1.507 seconds	26.538 seconds
Recognition Rate	50%	87.5%
Code Complexity	High (A lot of cell arrays, loops, unique conditionals)	Medium (Straight forward transform of data to visual domain)
Future Value	Medium (Algorithm predates the 21st century)	High (Transferring auditory data to the visual domain can help recognition rates with the advancement of machine learning)

AUTHOR'S NOTES

- The MFCC's seem to cluster around 0. There might be an issue with the gain across the DSP layer. Decorrelating these MFCC's might aid the recognition rates.
- Building the VQ model was by far the most difficult task. No source code was used besides the melfb function from (Reference 1).
- The IR model stems from the author wanting to improve the VQ recognition rates. Although peer-reviewed literature is out there that claims VQ can deliver higher than 95% recognition rates, the author's implementation might be missing a crucial detail—perhaps in the clustering stage (splitting of centroids).
- The author could not find literature on transferring audio into the visual domain for recognition, so this might be a useful development.
- The code is more than 1000 lines long and can certainly be modularized.
- A big flaw in the code is that all functions are tailored to take in an array of speakers rather than a single speaker. This decreases the usability of the code and is one of the reasons for the complexity of the entire thing.
- The code file calculates the VQ model first, then reuses the MFCC's to calculate the IR model second.

REFERENCES

- (1) http://www.ifp.illinois.edu/~minhdo/teaching/speaker_recognition/
- (2) https://stats.stackexchange.com/questions/178626/how-to-normalize-data-between-1-and-1
- (3) Speech and Audio Signal Processing 2nd Edition. Gold, Morgan, Ellis. ISBN 978-0470195369. https://www.amazon.com/Speech-Audio-Signal-Processing-Perception/dp/0470195363
- (4) Mel Filter Bank Figure http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/f2015/jdl279_mfh65_yl2553/jdl279_mfh65_yl2553/jdl279_mfh65_yl2553/index.html
- Professor Artyom Grigoryan
- MATLAB Documentation
- StackExchange for MATLAB syntax