Keyword Spotting

- Mithilesh Vaidya

The problem

- Keyword Spotting (KWS) refers to the task of identifying a unique, pre-defined keyword in either an audio file or a continuous stream of audio. Since it is extensively used in mobile devices to detect a trigger word, the algorithm should be both power and computationally efficient
- It is often referred to as a ‘needle in a haystack’ problem since the utterance of the word we are searching for is generally very rare and hence the algorithm must learn to smartly ignore a wide-range of background noises, non-keyword human speech, etc. while remaining vigilant enough to detect the utterance of the pre-defined keyword

Datasets

Two datasets are used:

1. The TIMIT dataset contains about 6300 recordings of sentences, along with both frame-labelled phones and frame-labelled words. This dataset is used for training a neural network feature extractor, as described in the subsequent slides

2. The ‘Speech Commands’ (Speech) dataset by Google contains 65K one-second recordings of about 30 different words. Each recording has the corresponding word label. This dataset is used for testing the performance of the algorithm

Overview

The pipeline can be divided into the following steps:

1. Short audio clips (average of 6 words in each clip) are generated for testing from the Speech dataset. Keywords which are used as templates are also drawn from this dataset

2. Features are extracted (for both the clip and the template) in regions where speech is detected

3. Resulting features of the clip and the template (of the region containing speech) are compared by a variant of Dynamic Time Warping (DTW) and an adaptive threshold decides if the keyword is present in the clip

Generation of test data

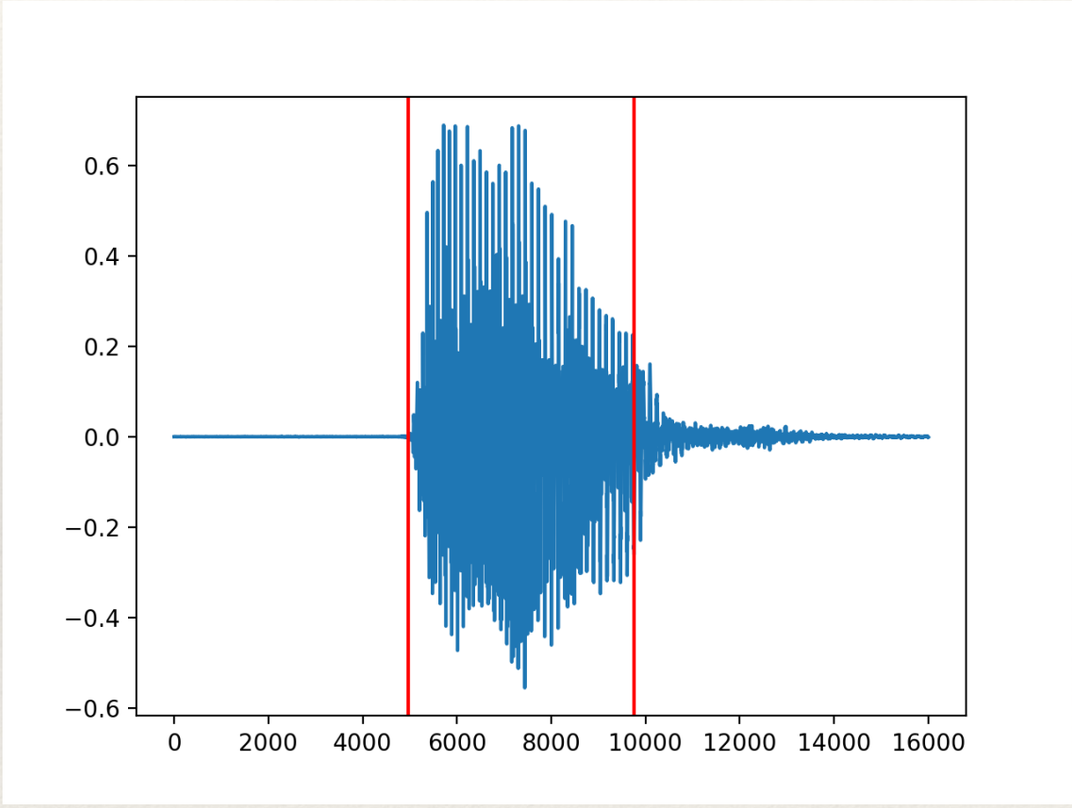
1. Audio clips for testing are generated by concatenating 2-3 sets of 2-3 randomly chosen words from the Speech dataset, each set being separated by a short randomly-chosen but constrained silence. 15 such clips are generated.
2. Currently, the following words are included in the dataset: 'bed','cat','dog','one','marvin','two','go','eight', 'wow', 'five', 'happy', 'sheila', 'zero', 'house', 'nine', 'three'
3. 5 templates are randomly chosen from the entire Speech dataset, which are compared with the above audio clips at test time for measuring the performance of the KWS model

Silence detection

The signal is first normalised to the range [0,1]. Then:

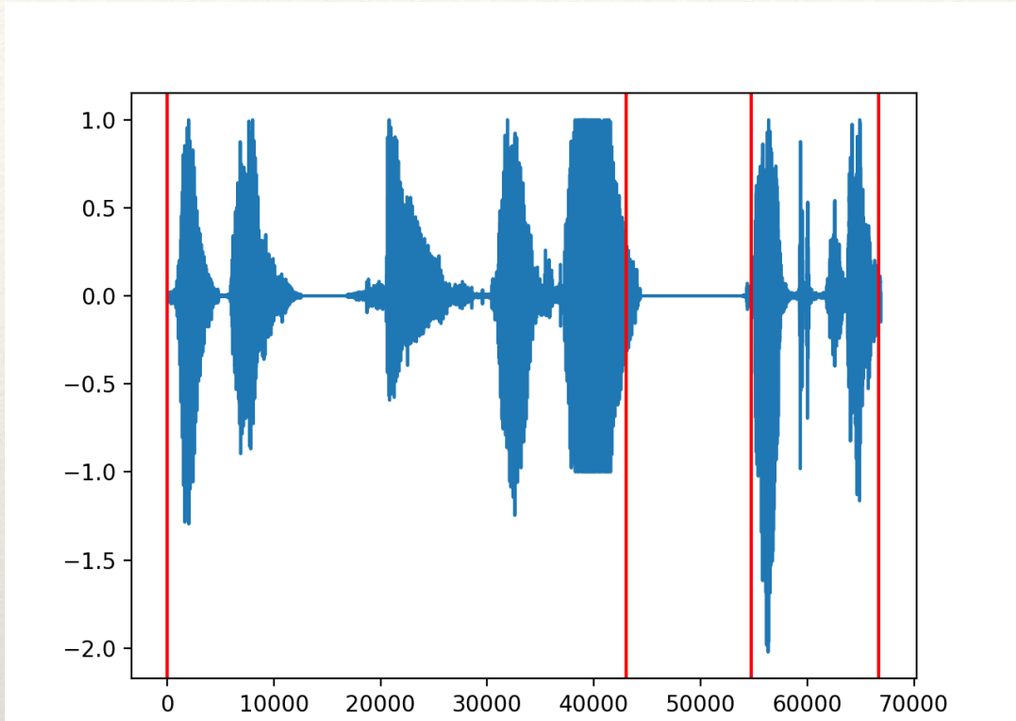
In case of a template, power spectra is calculated for the 1-second keyword and the contiguous frames where power*70 > max(power) is satisfied, are chosen, discarding the remaining as silence/noise

e.g. if the above condition is satisfied for frames: [43 … 46, 48, 50, 51, 52, 54 … 60, 63], we return the frames from 43 to 63.



Silence for audio clip

In case of the audio clip, the criterion described above is used with one minor change: there could be multiple such contiguous stretches of audio (since we had randomly inserted silences) and hence each such stretch is returned as elements of a list (e.g. if the frames satisfying the condition are [10…50, 65, 66, 67, 145 … 160], then we return features[[10 … 67], [145 … 160]] as the final output



Note that [10 … 50, 65, 66, 67] was returned as one segment because the number of frames of silence in between these two audio segments < 50(a threshold). This is because the frames in the above case correspond to one set of 2-3 words as described previously . Splitting them further is equivalent to splitting the clip into individual word segments which is not desirable as discussed in the later sections

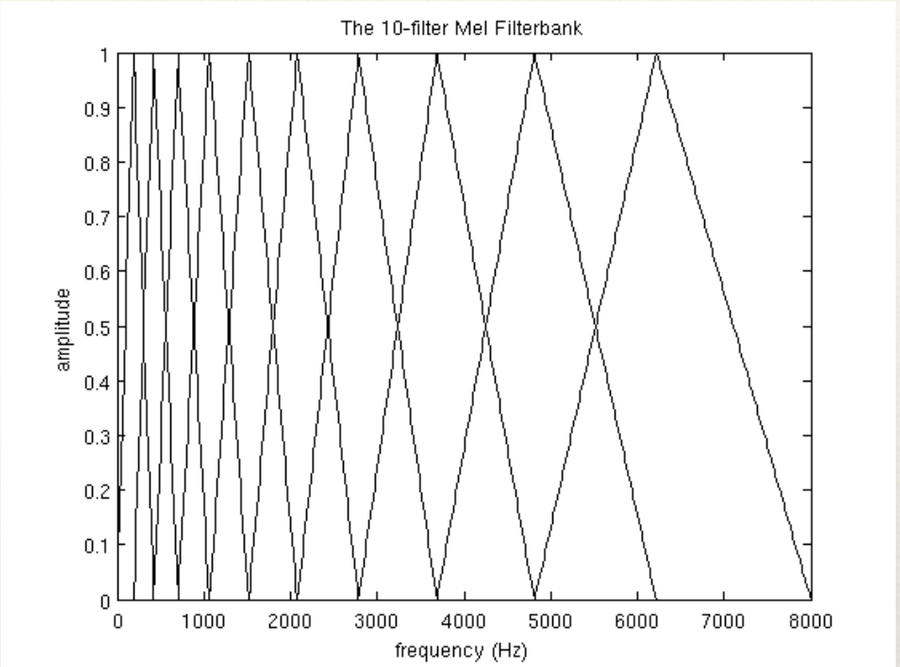
Feature extraction

- Feature extraction is a crucial step since it transforms the raw speech signal into a number of fixed-dimensional vectors. The aim is to find a mapping such that the distance between two set of vectors which contain the same word/phone is low
- Hence, the vectors encode the speech content by retaining the important distinguishing features of the word/phone while ignoring the variations found in the raw waveform from person-to-person
- As of now, 2 methods for feature extraction have been studied which are discussed in the subsequent slides.

MFCC features

MFCC features are widely used as they replicate the sensitivity of the human ear

1. Calculate power spectra of the audio signal (a window size of 25 ms and a hop of 10 ms is used since it is safe to assume that the frequency content of a signal is stationary in this window)
2. Apply the Mel Filterbank to the power spectra and sum the energy in each filter
3. Take logarithm of Filterbank Energies
4. Since the above coefficients are correlated, take DCT, keep coefficients 2-13, discard the rest



A set of 10 filters is shown above. It simulates the decreasing sensitivity of the human hear to increasing frequencies

Neural Network features

- The second approach is to use a neural network (NN) as a feature extractor. The input to the neural network is the log Filterbank Energies discussed in the previous step (We do not use MFCC since the NN can detect correlations and adjust it's parameters accordingly)
- We train the NN to predict the phone present in the central frame
- Also, we study the impact of providing a context of ±4 frames as input to the NN. This dramatically improves the performance since consecutive frames of speech are highly correlated and hence boosts the accuracy
- We exploit the phone-level annotations in the TIMIT database by constructing a database of (phone_id, features) pair where features are vectors (of dimension 26 when no context is provided and 26*(4+1+4) = 234 when a context of ±4 frames is provided) and phone_id is the phone present in the central frame of the features

Architecture

The output layer consists of 39 nodes, each corresponding to a unique phone-folded label

2 different NN architectures are studied:

1. The shallow model consists of 3 hidden layers with 512, 1024 and 512 nodes in the hidden layers
2. The deeper model consists of 5 hidden layers with 512, 1024, 2048, 1024 and 512 nodes in the hidden layers

   The decision for the number of nodes was arbitrary and chosen since increasing-decreasing powers of 2 is a common practice for choosing the number of nodes, especially when there is no particular feature of the dataset to exploit/drive the decision behind choosing the number of nodes

Phone folding

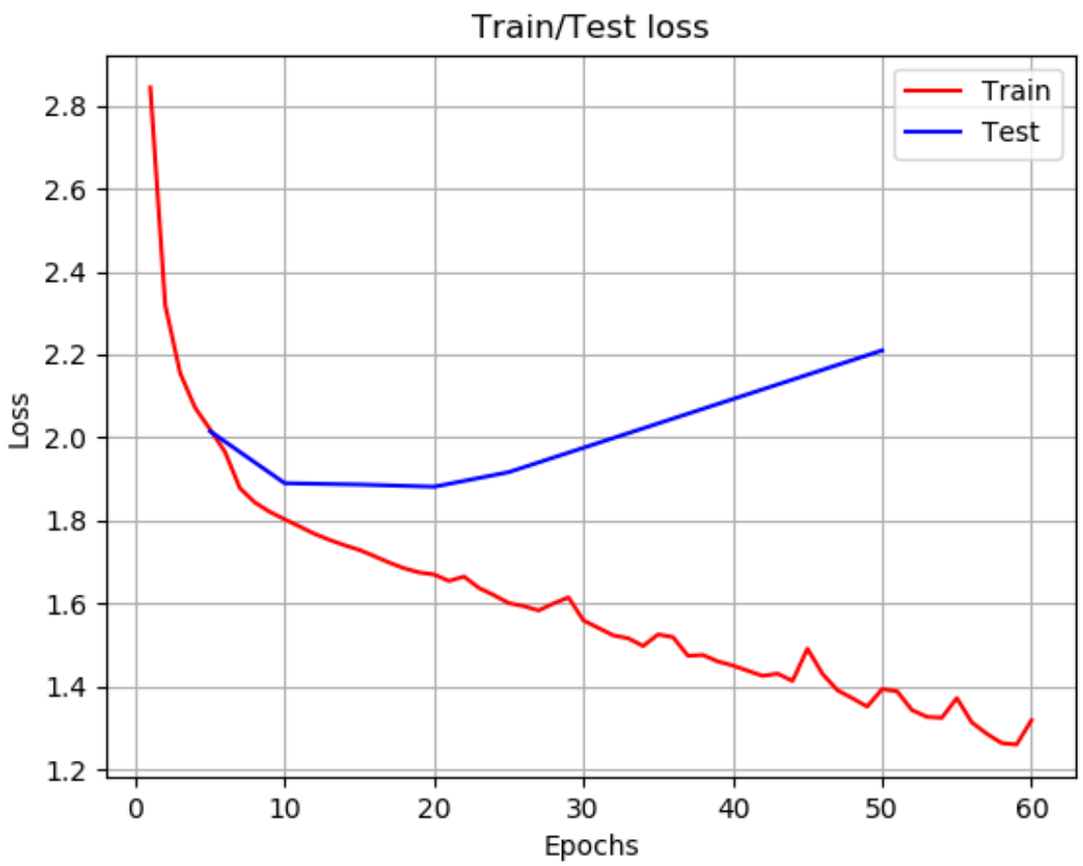
- The original TIMIT dataset contains 61 phones
- We combine some of them into a single category to ease the training for the NN model
- The table on the right indicates which phones (in the right column) are collapsed into a single category (in the left column)
- Note that the phone ‘q’ is not used for training since it marks a ‘global stop’

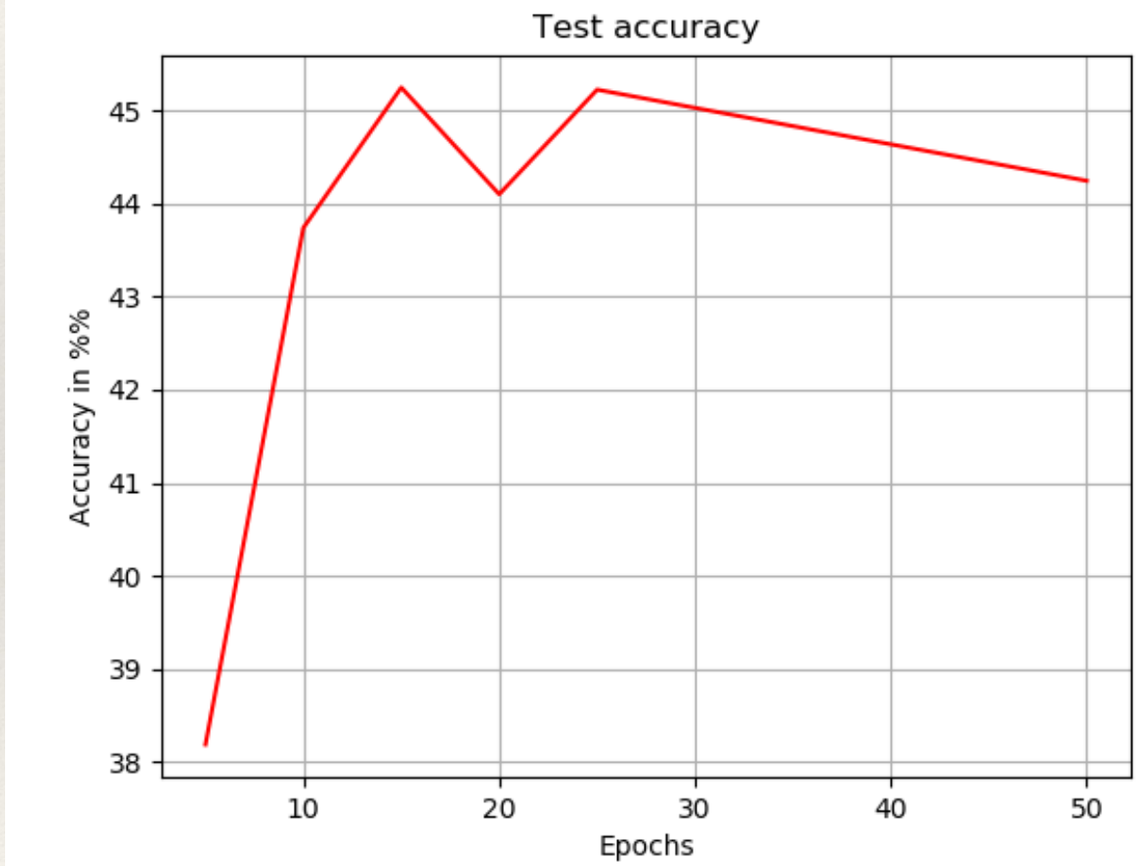
aa	aa, ao
ah	ah, ax, ax-h
er	er, axr
hh	hh, hv
ih	ih, ix
l	l, el
m	m, em
n	n, en, nx
ng	ng, eng
sh	sh, zh
sil	pcl, tcl, kcl, bcl, dcl, gcl, h#, pau, epi
uw	uw, ux
—	q

Training

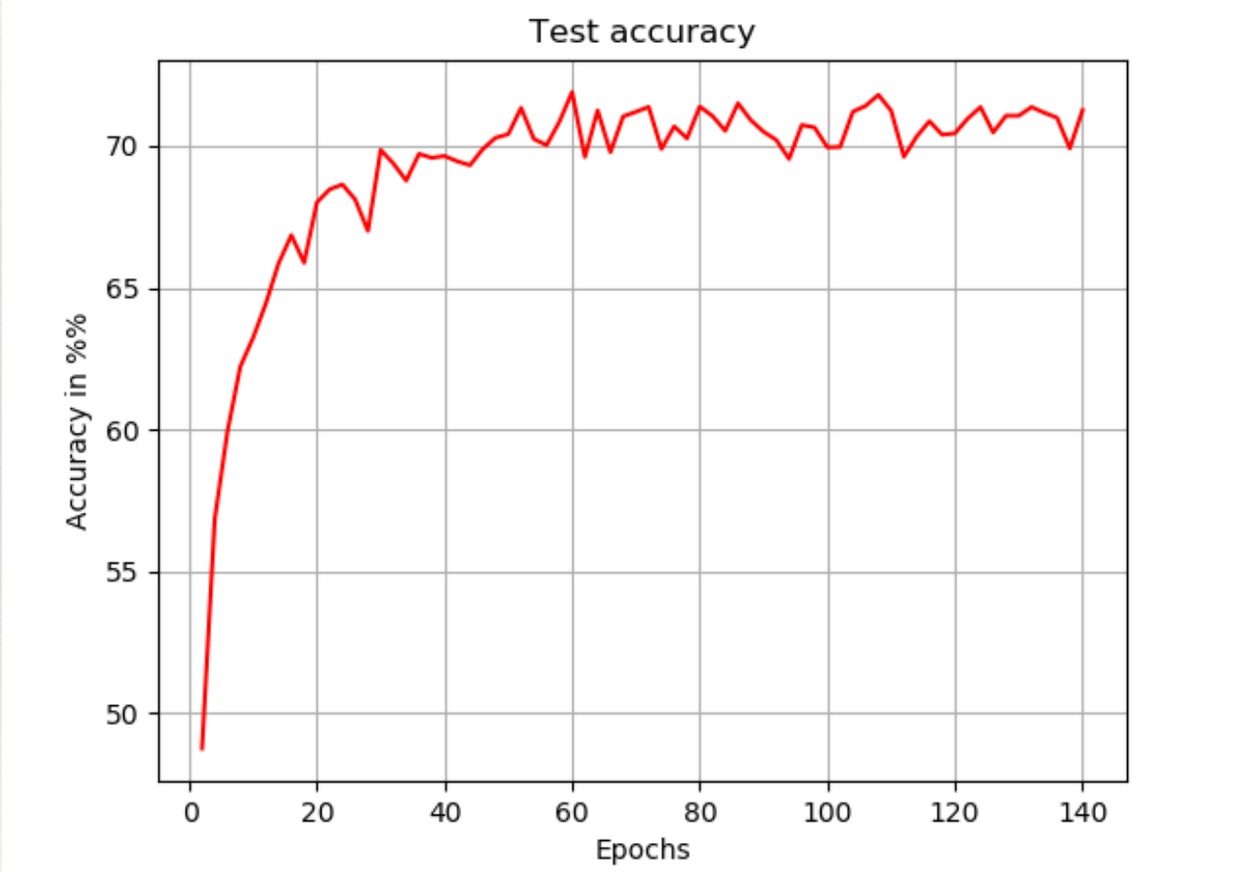
- A weighted cross-entropy loss function is used since the dataset is skewed towards some phones (especially after folding). Weights are inversely proportional to the number of training samples of the corresponding phone
- During testing, we take the argmax output and check if it matches the ground truth phone label. Accuracy is defined as (number of correctly identified phones)/(total phones tested)
- Dropout and ReLU are used to increase performance while BatchNorm and Adam optimiser are used to speed up the training
- The best model (which is used for feature extraction) is the one which gives minimum testing loss since on further training, the model overfits the training data. However, testing accuracy was found to remain almost constant despite overfitting. The skewed testing dataset could be a plausible reason for the anomaly

Deep NN with no context

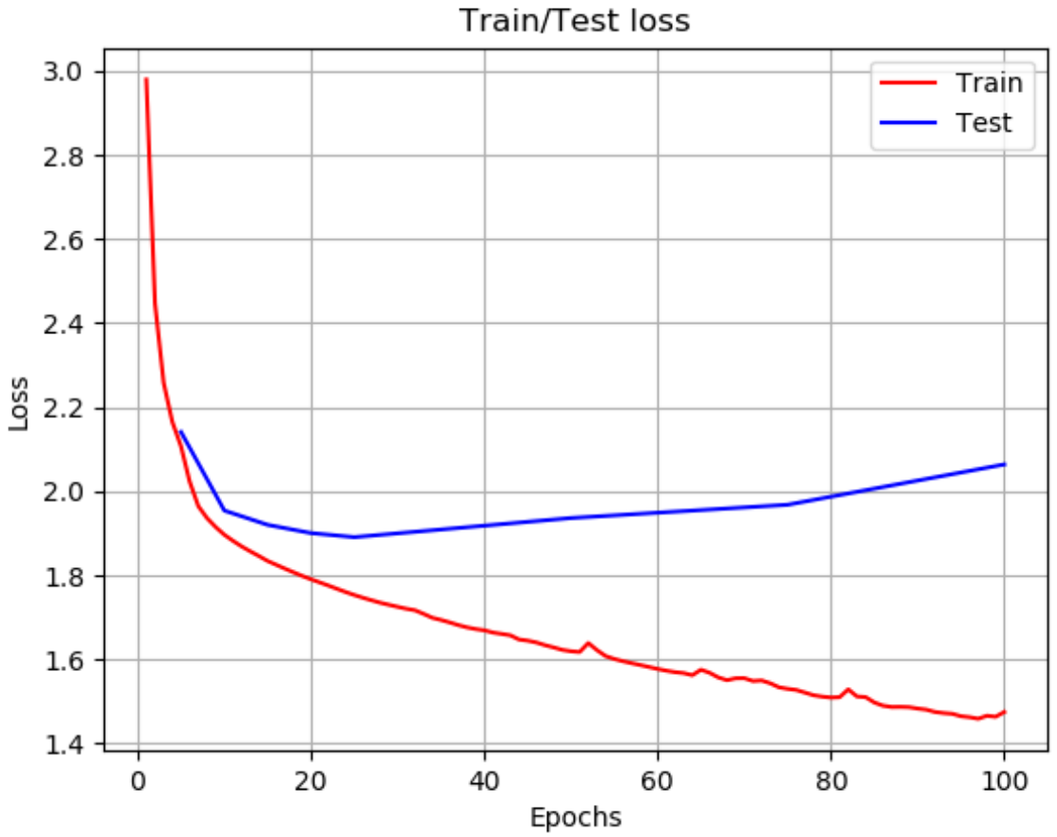


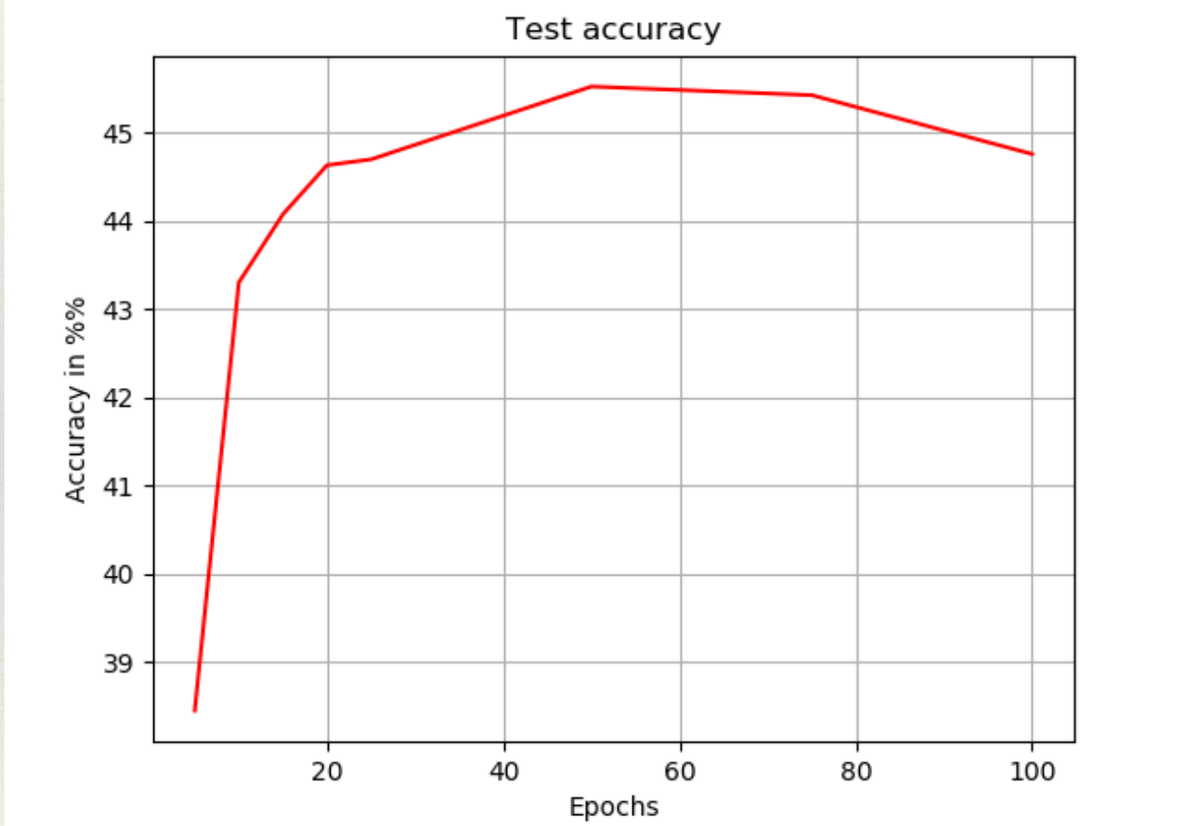


Deep NN with ±4 context

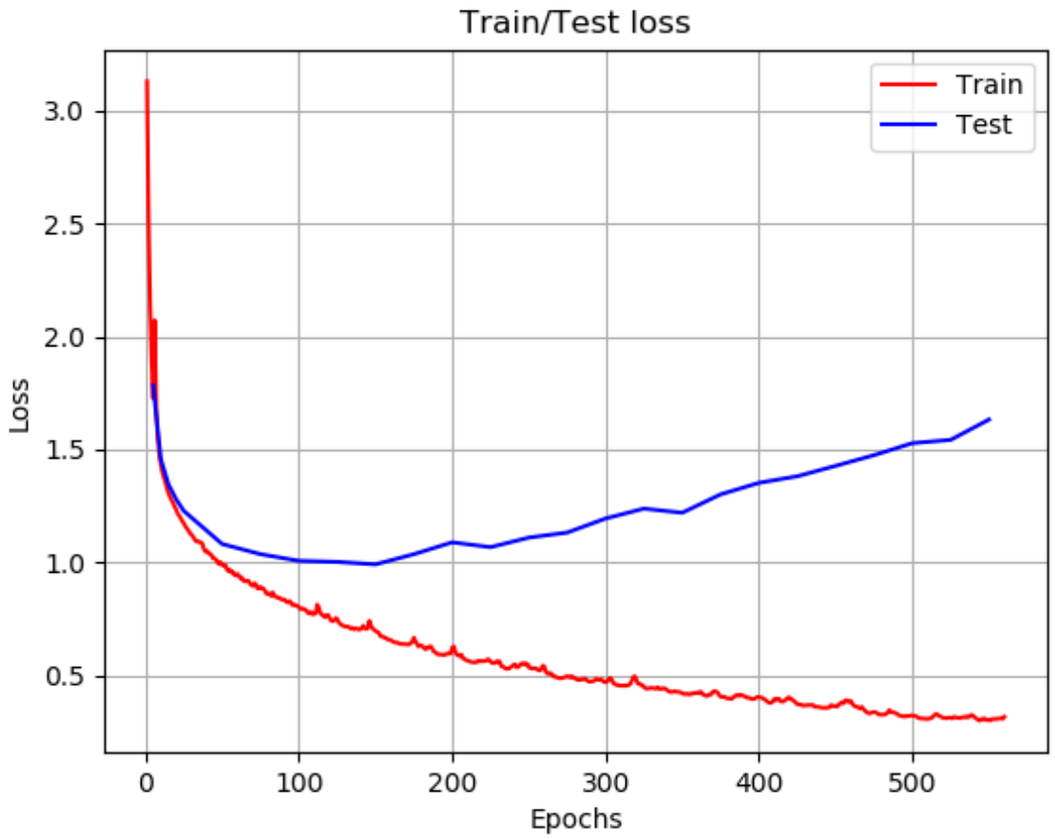


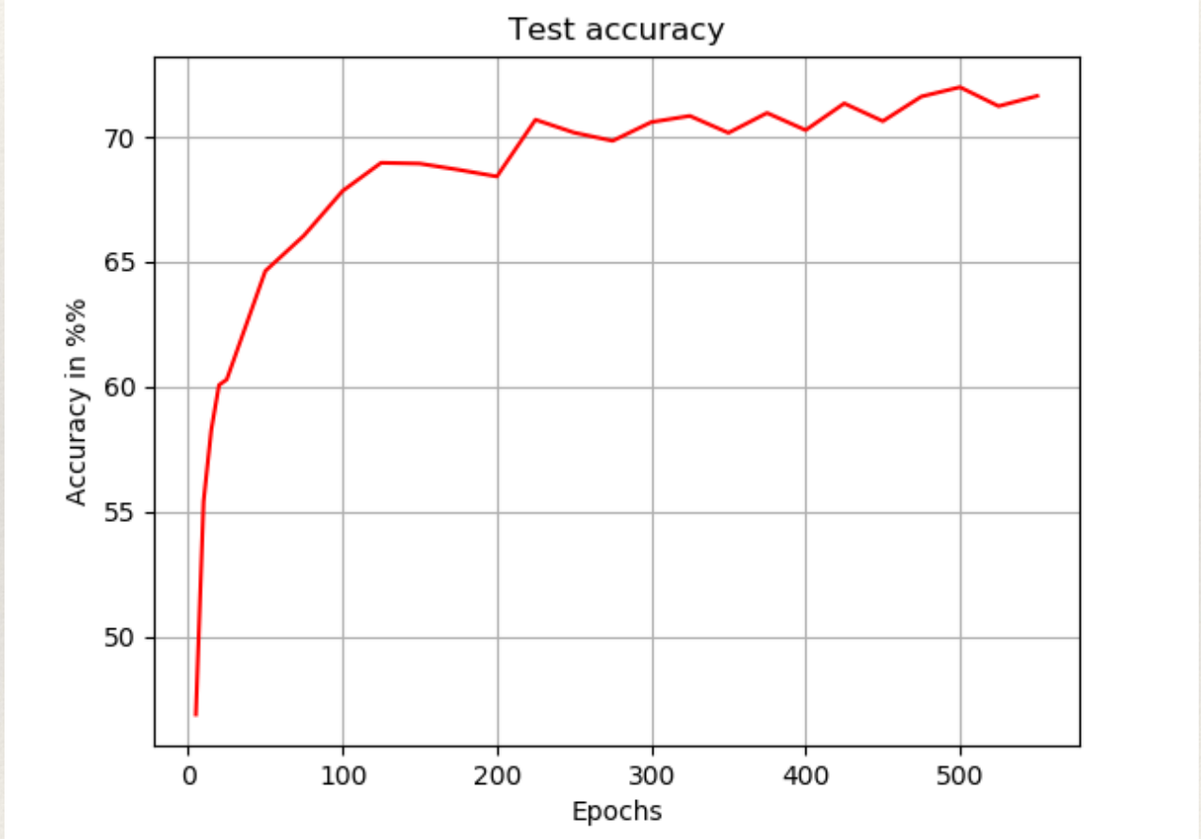
Shallow NN with no context





Shallow NN with ±4 context





Comments

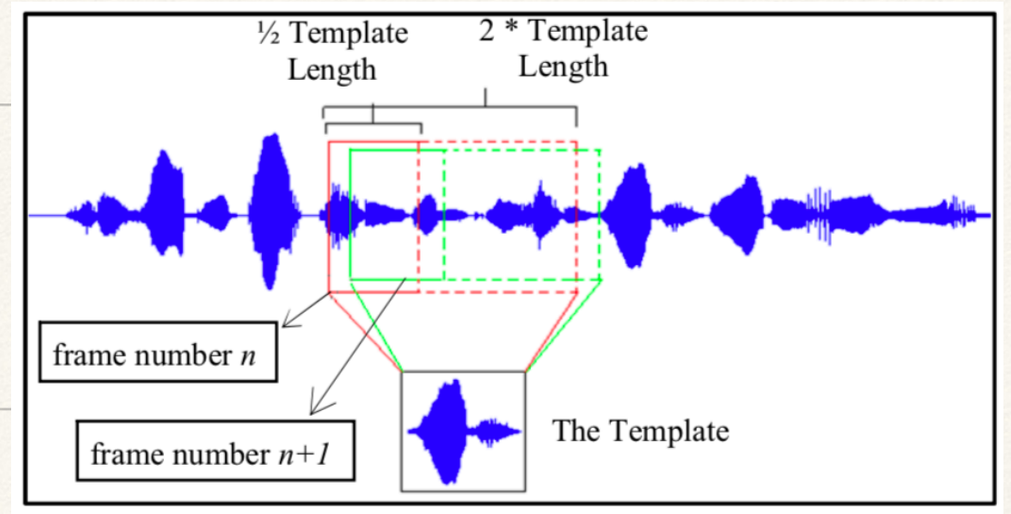
From the model loss/accuracy, we can conclude that:

- The shallow model with ±4 context is the best performing model since it has the lowest minimum test loss. The curves are smooth as opposed to the ones for the deep model (which indicates model instability)
- The models without context are much inferior to the ones which are provided context i.e. it fails to predict the phones when given one frame at a time

Dynamic Time Warping (DTW)

- Once the features of the audio clip and the template are extracted, we compare them using DTW
- The easiest option is to compute the DTW cost between the two and declare that the keyword is present if the cost is below a certain threshold. Euclidean distance is being used as a measure of distance between two vectors
- However, our clips contain 2-3 words of varying lengths (since it is closer to a real-world speech utterance) and hence an absolute threshold for the DTW cost is not effective
- An adaptive threshold is better because in general, absolute values in speech processing are meaningless. Comparing them with some baseline makes more sense

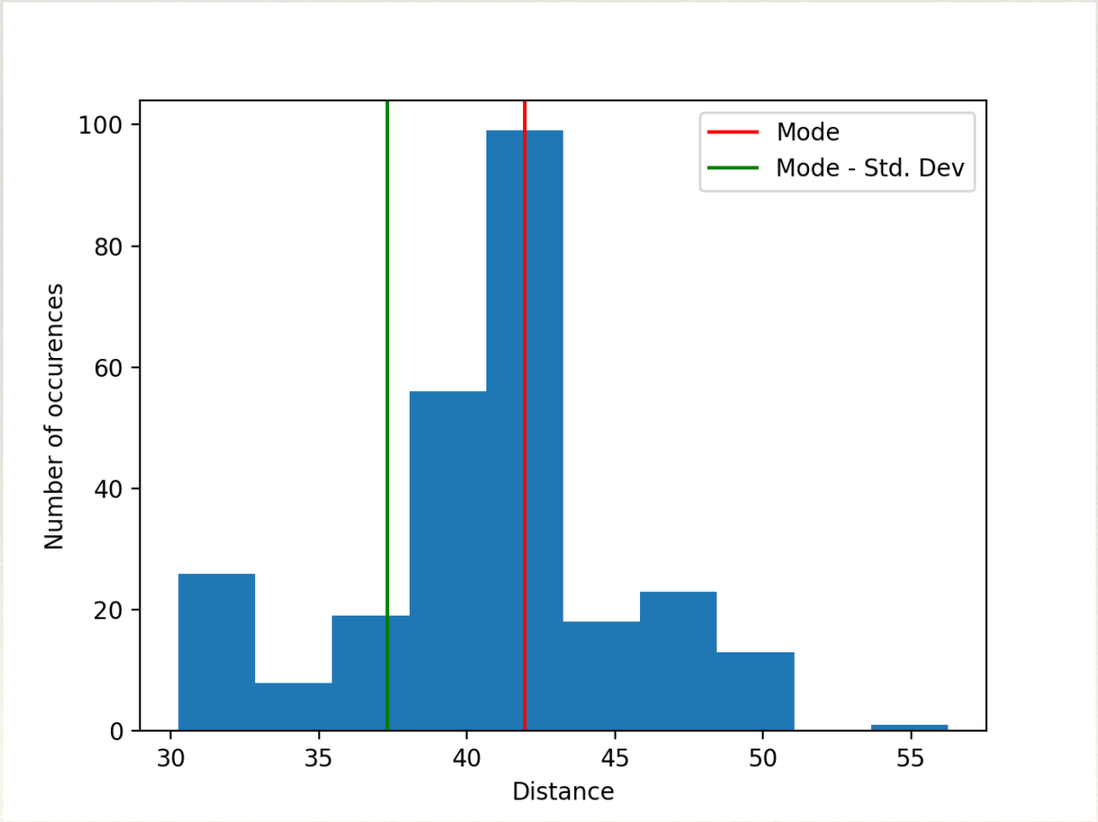
Sliding DTW



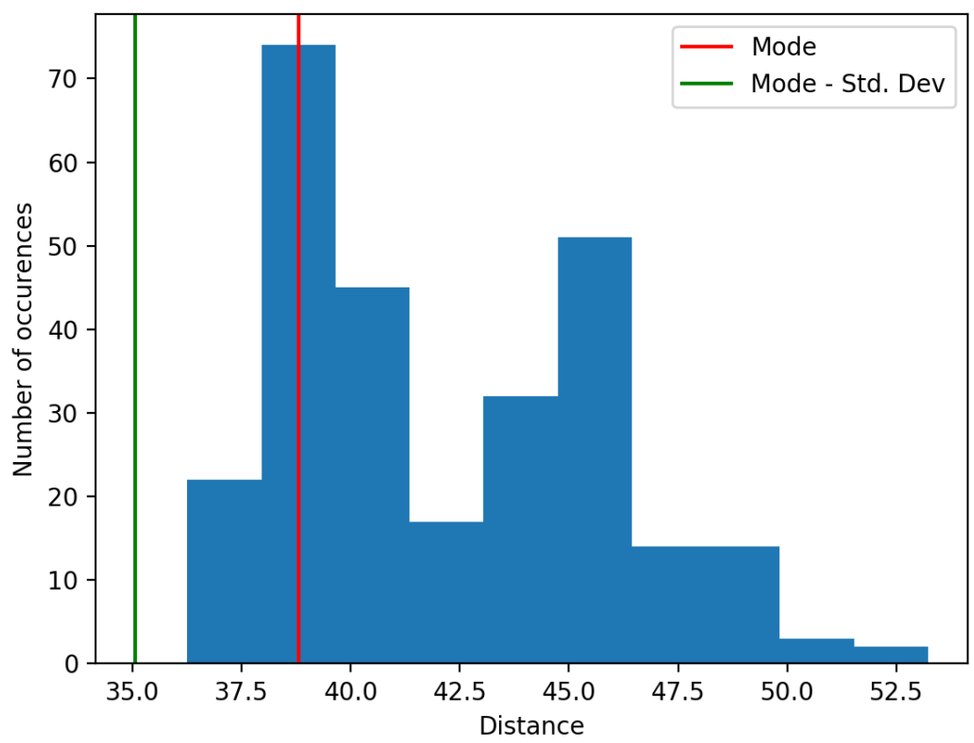
- Let the shape of the clip and the template be (m,d) and (n,d) (d is the dimension of our feature vector) . Due to differences in the rate of speech, m ≠ n. (In our case, m > n in almost all cases since the clip is a concatenation of 2-3 words)
- Assuming a range of (0.5, 2) when it comes to the human speech speed, we fix a starting frame (say 's') and compute the DTW cost between the template and a range of clip lengths (0.5*n to 2*n) and store the minimum of these distances in a table called LMD whose key is 's' and value is the minimum distance
  i.e. LMD[s] = min (DTW(template, clip[s:s+k])) for k in range(0.5*n to 2*n)
- For intuition, if the word we are looking for starts at the 10th frame in the clip, then LMD[10] will have the least distance among all other starting frames and the varying number of frames it is compared with helps overcome the speech rate differences among speakers

Histogram

- After obtaining the table of minimum distances, we compute a histogram of the distances obtained in the previous step
- This is because the histogram of a template with a clip containing the template word, has a different shape than the one which does not contain the word we are looking for



Clip with keyword



Clip without keyword

Hypotheses

- The algorithm works on the hypothesis that when the keyword is present in the clip, as compared to when it is absent, the histogram tends to have a central peak, larger variance of Euclidean distance and a higher occurrence of small distances. You can observe the 3 features in the histograms on the previous slide. Also note the absolute value of the distances
- Since we are interested in smaller distances, we consider the region of the histogram to the left of:
  Threshold (TH) = **Mode - c*σ**
- We look for values for starting frame 's' whose minimum distance
  i.e. LMD[s] <= TH and consider these values of 's' for further processing

Finding sequences

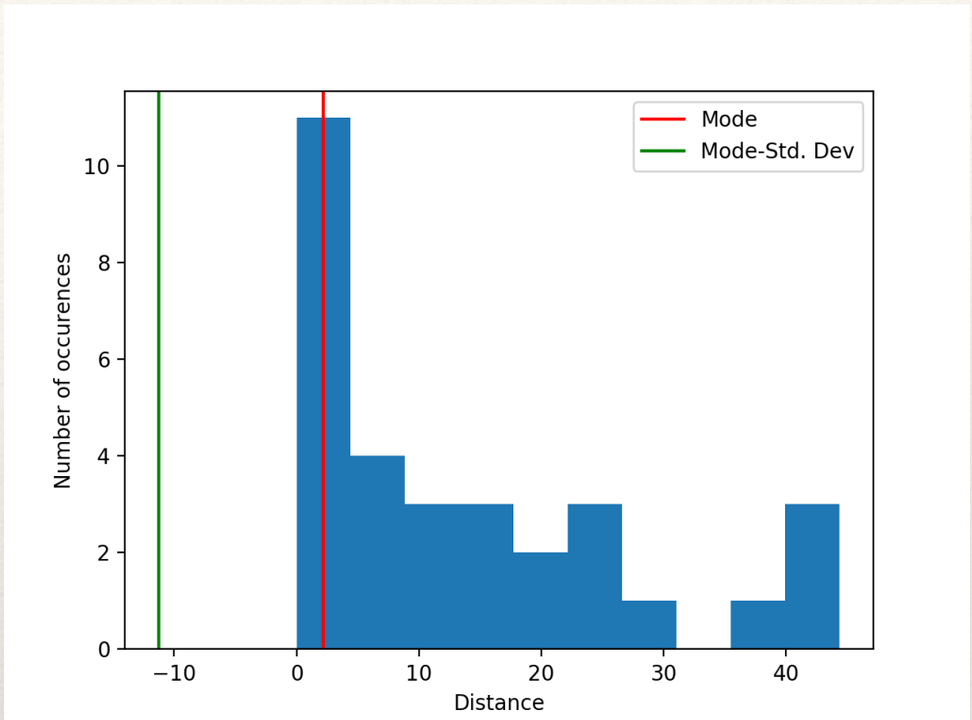
- For the e.g. given in the slides, the set of frames below threshold TH, when keyword is present are [222 223 224 225 226 227 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325]
- On the other hand, the frames below threshold when keyword is not present is an empty set! (Both for c = 1) This is obvious from the histogram)
- Then, we calculate the sequence of starting and ending contagious frames for the set. In our case, it is
  [(222, 227), (288, 325)] when the keyword is present and obviously an empty list for the other case since our starting set is anyways empty!

Finding sequences (cont.)

- Then, we calculate the length of these sequences i.e.[6, 38] and [] for the keyword and non-keyword clip
- We divide this by the number of template frames (34 in this case) which gives [0.17, 1.17] and [] respectively. This is done because for longer templates, more number of frames could be below the threshold and hence we need to normalise this by taking the ratio (call these values as **k_con**)
- If you listen carefully, the template-containing clip has two utterances of 'happy', the latter one being long and more clear. Hence, more number of starting frames match this utterance!
- We have a parameter k between (0,1) which decides if the keyword is present or not i.e. if **k_con >= k**, we declare that the keyword is present, else it isn't

Drawbacks of sliding DTW

- One drawback of the sliding DTW approach is that it fails when the clip and the template are of approximately the same length and have the same utterance of the word (even when the two are the exact same files)
- The issue is evident in the histogram to the right. Note that the absolute distance is close to 0 for majority of the frames, as a result of which the TH goes below 0!

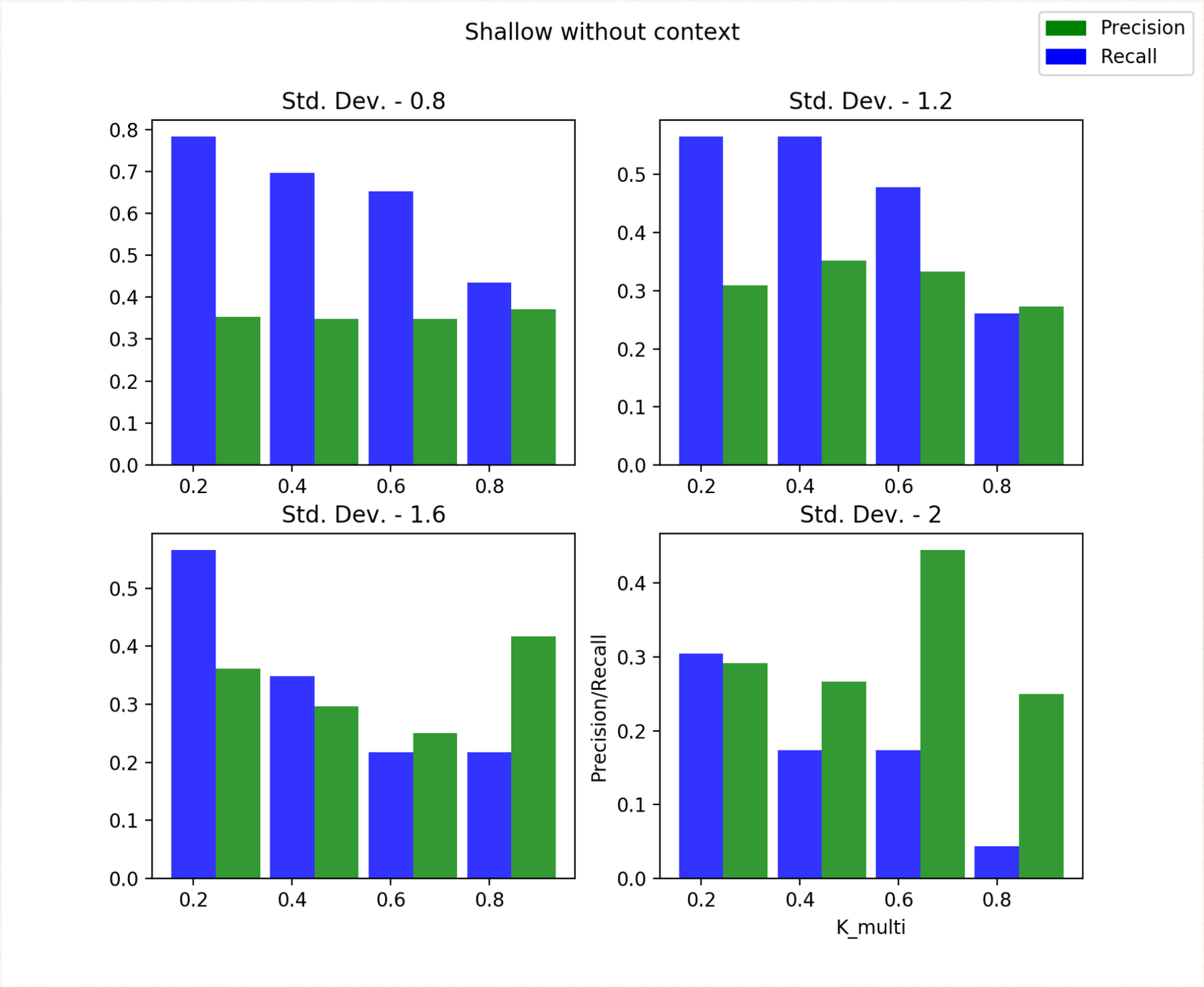


This explains why we split the frame into clips containing 2-3 words each and not further split them into individual word clips. The advantage is that we can use simple power-based analysis to detect the longer silences

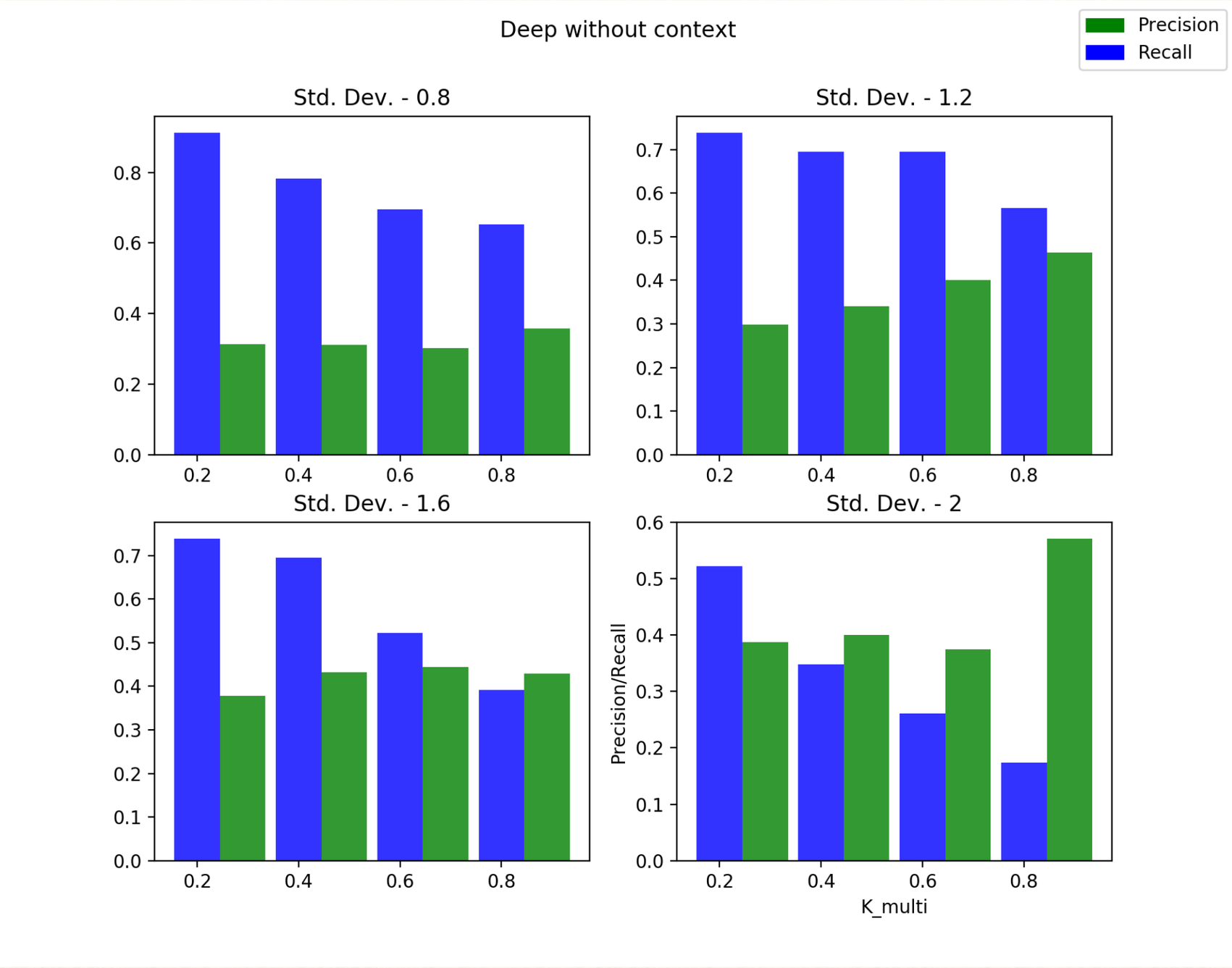
Testing

- We have two parameters to play with:
  c: controls the multiplier with standard deviation in the histogram. We carry out experiments for c in the list [0.8, 1.2, 1.6, 2]
  k: controls the adaptive threshold for taking the final decision. The values of k chosen for the experiments are [0.2, 0.4, 0.6, 0.8]
- Each pair of (c, k) values is considered for the experiment
- We generate 15 audio clips and 5 keyword templates; each template is compared with each audio clip and the result is categorised into either one of
  True +ve (TP)/True -ve (TN)/False +ve (FP) /False -ve (FN)
  Precision (TP/(TP+FP)) and Recall (TP/(TP+FN)) are recorded
- A good algorithm has both high Precision and high Recall
- The results are given in the next slide

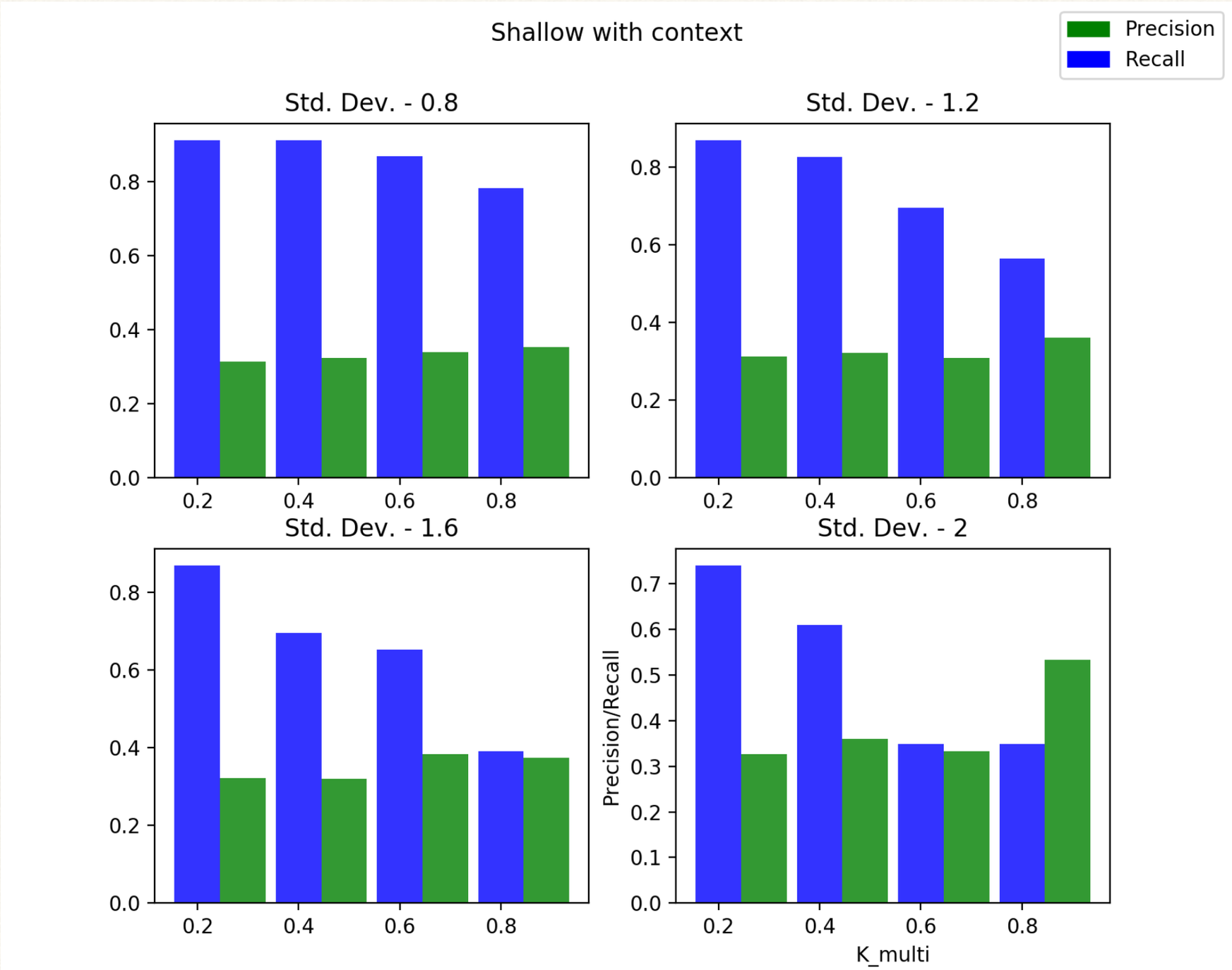
Shallow NN without context



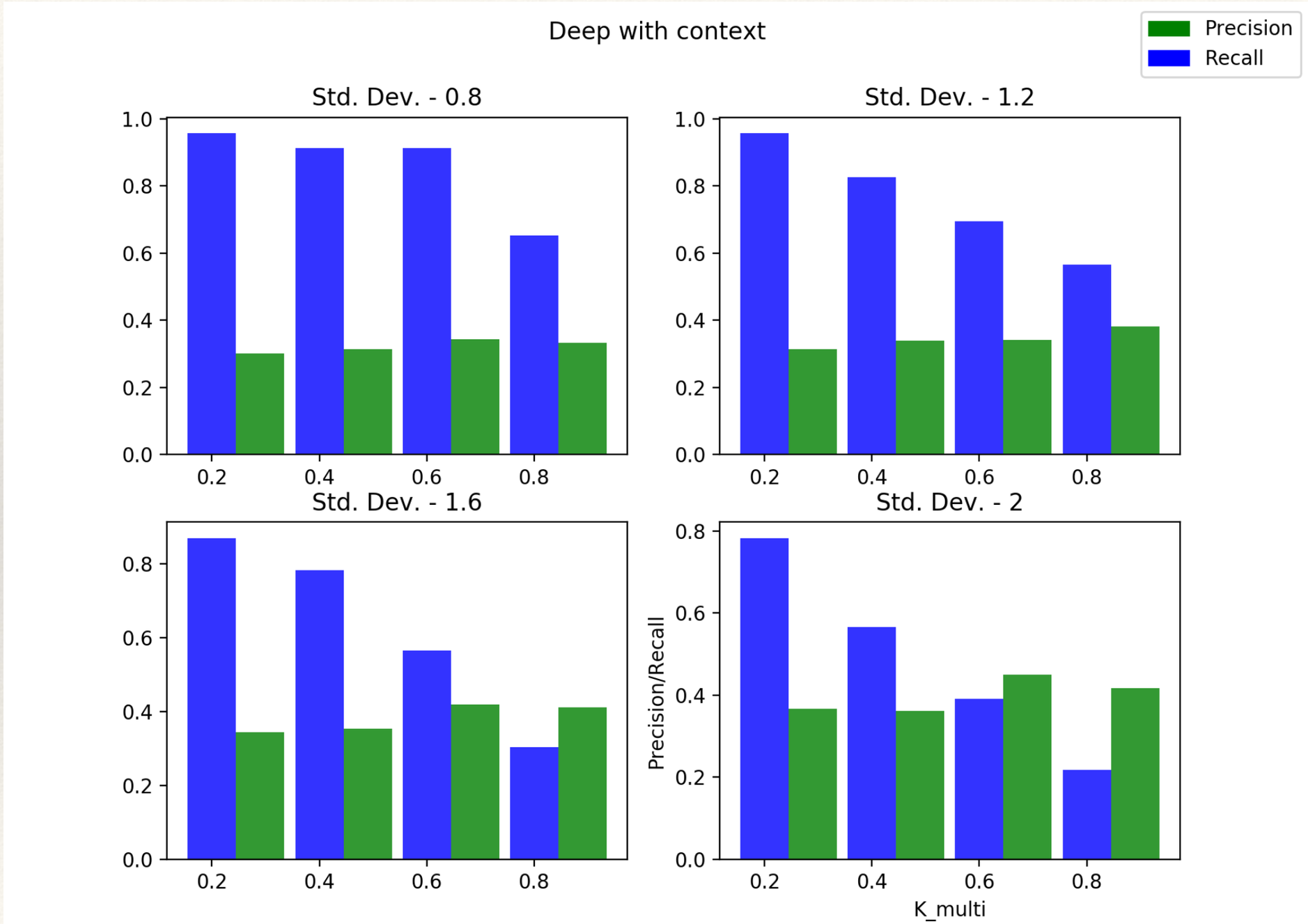
Deep NN without context



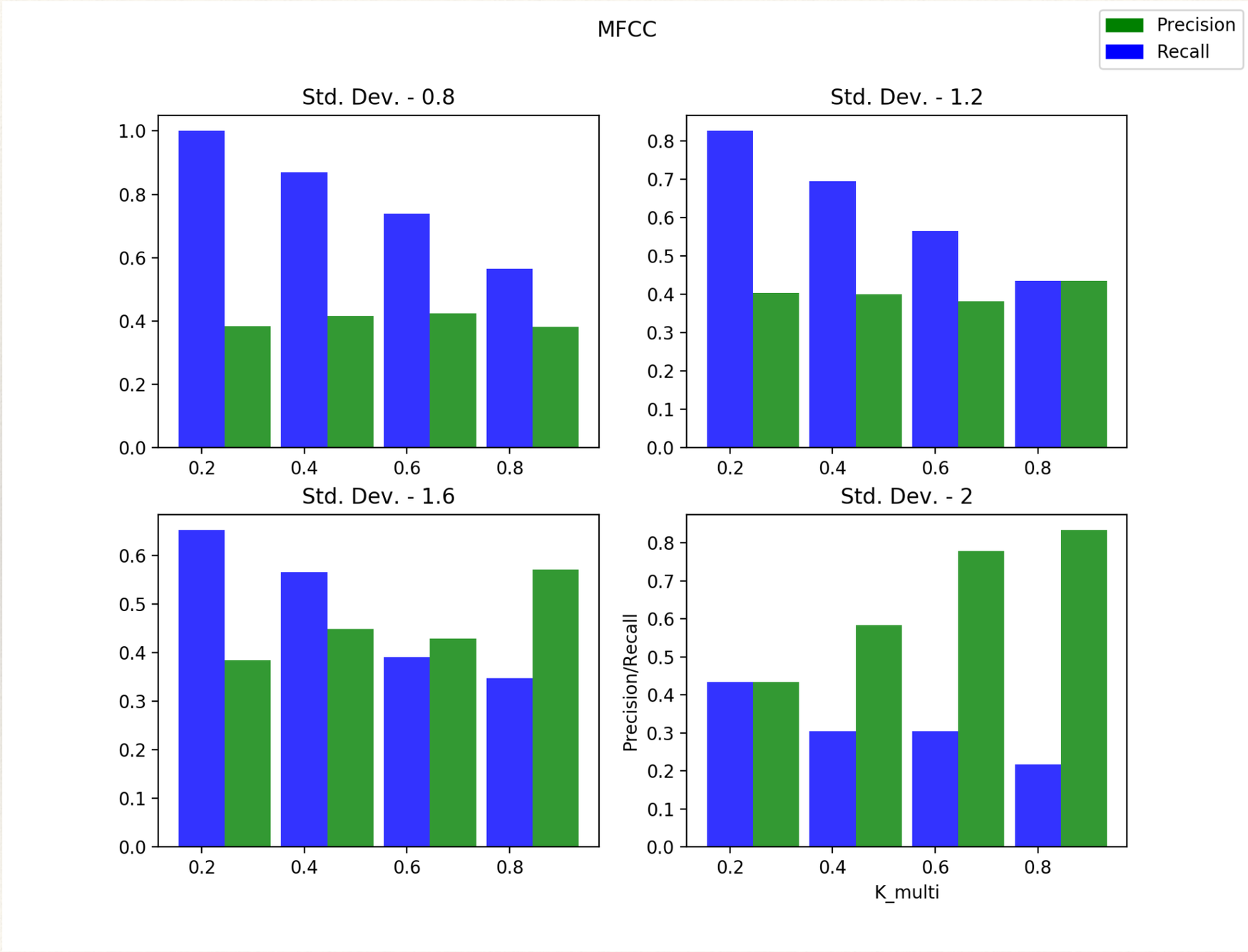
Shallow NN with ±4 context



Deep NN with ±4 context



MFCC


MFCC
Precision
Recall
Std. Dev. - 0.8
Std. Dev. - 1.2
Std. Dev. - 1.6
Std. Dev. - 2
Precision/Recall
K_multi

Comments

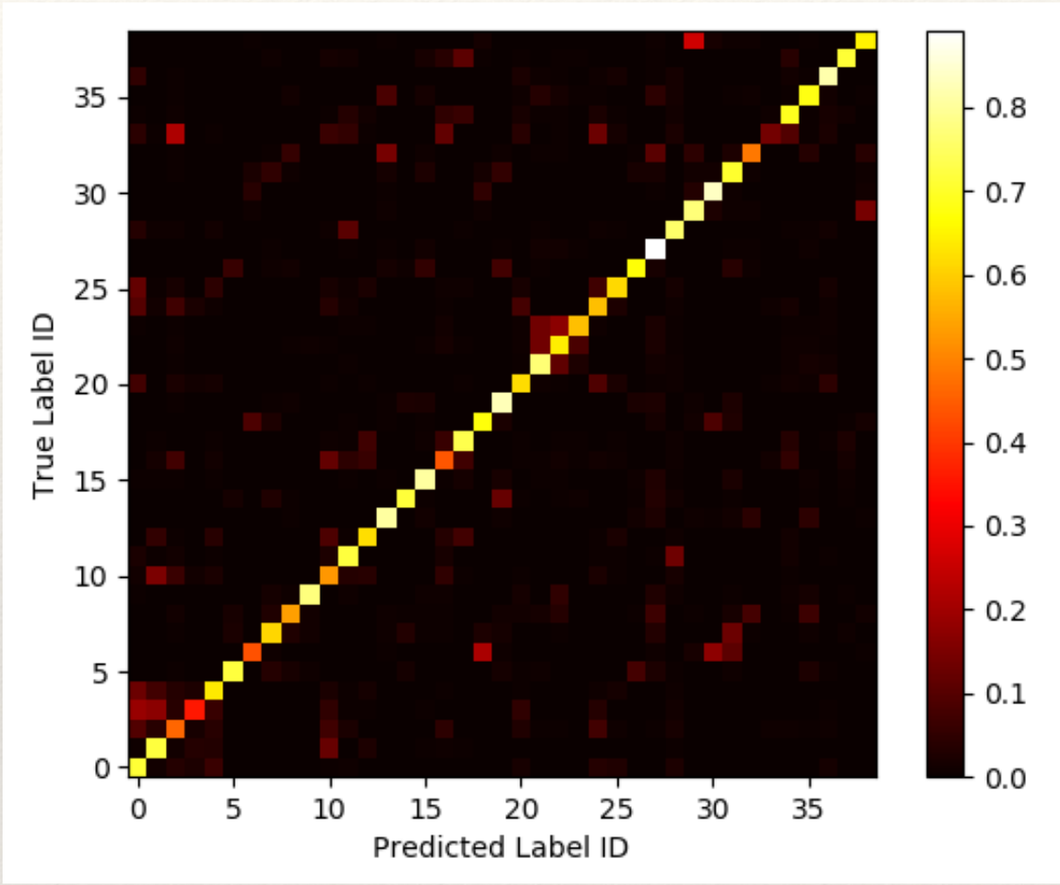
- From preliminary observations, we can conclude that MFCC features give superior performance as compared to the NN. Particularly, it is difficult to attain a good Precision value with NN features, despite sacrificing on the Recall.
- On observing the argmax outputs of the NN, I observed many labels which correspond to the ‘pau’ phone class. This class contains the maximum number of training and testing examples and hence the NN is highly skewed towards this class.

Future steps

- Inspired by the dramatic increase in the test accuracy on adding context, the next step would be to use a LSTM-based feature extractor since a LSTM can automatically learn which context to retain and ignore. A fixed number of context frames may be suitable for some phones and unsuitable for the others
- A ‘confusion matrix’ is shown on the next slide. The shade of square (i,j) denotes the proportion of testing samples such that the ground truth phone_id was *j* but phone_id predicted by the model was *i*. So ideally, we should have a perfect diagonal with all values 1 and all non-diagonal entries as 0.

Future steps (cont.)

- The plot tells us which two phones are difficult to distinguish for the model. For e.g. a lot of phones with id=6 ('ch') are being confused with id=30 ('sh') since the square (30,6) is pretty bright and off-diagonal. Makes sense since the sounds do sound similar!
- Armed with this knowledge, we could improve the model by giving more such examples for training/increasing the weight of the loss for such examples, etc.



Confusion matrix for the shallow model with ±4 context

Future steps (cont.)

- Separating the words in audio clips even when there is negligible silence between them can help boost the accuracy since it is much easier to compare a template word with another word, as against a set of words
- Another popular variant is the CRNN, which first uses a CNN on the spectrogram of the audio, followed by a RNN (generally LSTM). Such an architecture can detect complex frequency-time trends and may boost the accuracy

Thank you!