# Deep Learning for Speech Signals, Assignment 4

## Submitted by:

Aylon Feraru, 325214492

1. GCommands is a dataset composed of 30 different words, where each word is pronounced by 1000 speakers. Compare CNN trained on this dataset to the DTW algorithm in terms of training time, inference time, and memory.

**Training time**:
*The DTW algorithm's* training time is comprised of computing the MFCC of each sample, and (at least in my implementation) associating them with the appropriate class. Training time grows with the dataset, but is negligible.
In comparison *The CNN's* training time is more involved, requiring the model to optimize the weights of multiple parameter layers through a stochastic gradient descent optimizer - requires multiplying, subtracting and updating lots of tensors - results in a significantly longer training time for adequate results.

**Inference Time**:
*The DTW algorithm's Inference time* grows linearly with the size of the training set, and can be quite significant. Since the algorithm is 1 Nearest Neighbor it requires us to compute the DTW distance of the sample we want to classify to every other sample in the dataset so we can choose the smallest one.

*The CNN's Inference time* on the other hand is relatively steady, requiring us to compute the output of the CNN only once on the sample we wish to classify with a sequence of convolutions dependent on the network's structure.

The bigger the dataset is, the more time it will take to the DTW algorithm to classify the model, whereas the CNN's inference time is fixed.

**Memory**:
*The DTW algorithm's Memory requirement* also grows with the training set, as each labeled example's MFCCs are stored for the 1NN inference. While they don't have a fixed size in the case of the 1-sec dataset samples we've had a (20,32) matrix of floats for each one, which is fairly large.

*The CNN algorithm's Memory requirement* is dependent on the structure of the network, and the parameters it contains. The CNN's memory requirements contain both the values of the current computation (a multiple channel tensor), as well as storing the kernel parameters.

So to sum up, the CNN's inference time is much more significant than the DTW 1NN's, but the CNN's inference time and memory requirements are fixed and often shorter than the

DTW's which grows with the size of the dataset.

2. Given a dataset A consists of 50 samples and dataset B consists of 1,000,000 samples, which approach, CNN or DTW, would you choose for each dataset? Explain shortly.

I would choose the *DTW for dataset A*, since 50 samples seems too little for the CNN to train properly and generalize the data. In contrast the DTW would train fast, not take too much space and have a relatively fast inference.

I would choose the *CNN for dataset B*, since with so many samples the DTW will have massive memory requirements (lots of stored MFCSS), and inference time (lots of comparisons). The CNN would have a good generalization, inference and memory complexity relative to DTW.

3. Assume that the input consists of speech signals of 7 concatenated digits (e.g., zip codes). Suggest an adaptation for the CNN model to support the prediction of multiple digits at once (the entire zip code).

I would change the output format of the CNN from a one hot vector of size one to a matrix comprised of seven concatenated one-hot vectors, one per digit. During training, I'd compute the loss of each digit with the cross-entropy loss and average out all the losses.

## part 3.2 - speaker classification:

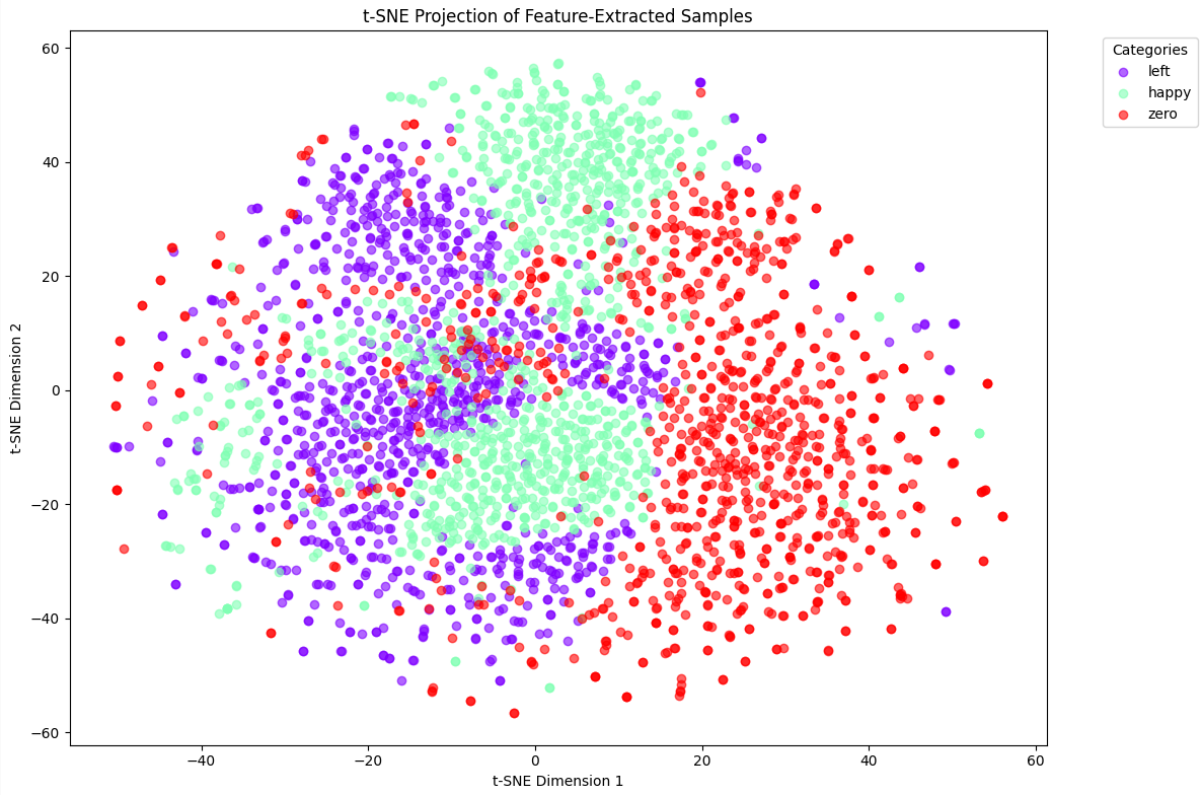
applying the classification algorithms to the dataset we got the following table:

| | Mean Accuracy | Std Dev |
|---|---|---|
| SVM | 0.986694 | 0.005628 |
| Decision Tree | 0.493794 | 0.021195 |
| Random Forest | 0.941499 | 0.008985 |


We can see how the SVM performs well against the random forest, and significantly better than the Decision Tree. The t-SNE shows us that our speaker x-vector embeddings are clumped together per-speaker and generally well separated. The SVM is adept at classification using the properties of multiple dimensions to its advantage, so it gets a really high mean-accuracy, and it does so relatively consistently and hence a low standard deviation. In comparison a decision tree is a relatively simple classifier that makes choices based on a single feature each time, minimizing entropy in its separation, but isn't too suited to high dimensional data that depends on a lot of features and as such has a very low accuracy and a relatively high standard deviation due to being noisy and making initial suboptimal splits. Random Forests work by aggregating decisions from multiple decision trees. It's a more robust classifier for this high dimensional data, and less prone to bias, but it may struggle still due to the curse of dimensionality.

**x-vectors seem to be really good for the speaker recognition task**, as they present via the t-SNE a really neatly separated feature space where they're clamped together, and high accuracy results (98%) are achievable with really simple classifiers like an SVM.
The reason for that is that they were created by extracting the embedding space from a DNN that was trained to classify speakers and as a result the embeddings are oriented towards extracting the best features for differentiating between different speakers.

## part 3.3 - command classification:




We can see that while there still is some clumping, our samples are spread out, there are more instances of outliers and our clumps seem to mix up with one another more so than before where we had some minor mixing but a mostly distinct clump. This feature space would be harder to use for command classification.

| | Mean Accuracy | Std Dev |
|---|---|---|
| SVM | 0.882000 | 0.014040 |
| Decision Tree | 0.557000 | 0.021119 |
| Random Forest | 0.863667 | 0.007846 |

we can see how the SVM achieves lower scores, with a lower standard deviation as well. The decision tree appears to fare slightly better but still perform poorly, and the random forest seems to perform worse as well. Linear classifiers are set to do poorly in tasks where there isn't a clear feature based linear separation, but the SVM and the Random forest are more robust (hyperplane and a composition of linear classifiers) and as a result can capture more complex relationships, but the features still aren't well separated.

**X-vectors seem to be worse at command recognition** which makes sense, since the speaker recognition task places no importance on text spoken and the features extracted from the DNN in the creation of the x-vectors are from a neural net trained on speaker recognition and not on command recognition - so the features aren't oriented towards text recognition. We can also see relatively worse results in that task.