MLPC Report Task 4

Team IMPORTED

Mahmut Öztürk **Michele Giambelli** **Barış Bakırdöven** **Korhan Erdoğdu**

Contributions

Mahmut Öztürk: Established a naive Baseline system, Presentation.

Michele Giambelli: Built a simple speech command detection system, Investigated at least three diverse strategies for improving the starting point.

Barış Bakırdöven: Built a simple speech command detection system, Investigated at least three diverse strategies for improving the starting point.

Korhan Erdoğdu: Established a naive Baseline system, Report.

1 Establish a naive Baseline system

Objective: To establish naive baseline systems for detecting speech commands in audio scenes and evaluate their performance in terms of cost.

Establishing baseline systems is crucial for evaluating the effectiveness of more sophisticated speech command recognition models. This report explores various naive baseline methods and provides an analysis of their expected costs and performance.

The baseline systems that we investigated are:

Most Frequent Command Baseline: Predicts the most frequently occurring command in the dataset for all instances. Leads to lower overall cost if the most frequent command is common, but results in high false negatives for less frequent commands.

Stratified Random Baseline: Randomly guesses commands while maintaining the dataset's command distribution. Performs better than uniform random guessing but incurs costs due to random errors.

Uniform Random Baseline: Randomly guesses commands with equal probability for each command. Results in high costs due to many incorrect predictions.

Last Command Baseline: Predicts the last observed command in the dataset for all instances. Performance varies depending on the representativeness of the last command.

Sequential Random Baseline: Randomly guesses a command based on the last three observed commands. Attempts to incorporate recent context but still results in substantial costs.

Command Frequency Baseline: Predicts commands based on their overall frequency distribution in the training data. Balances predictions between common and less common commands, reflecting the actual distribution.

Energy-Based Baseline: Predicts the presence of speech commands based on the energy levels of audio signals. Steps include energy calculation (sum of the squares of the signal values), thresholding (using a predefined threshold to determine if the energy level corresponds to a speech command), and prediction (if energy exceeds the threshold, predict the most frequent command; otherwise, predict no command). Focuses on distinguishing speech from background noise but may not accurately identify specific commands.

Evaluation of Methods: The Most Frequent Command Baseline is often the most useful naive baseline due to its simplicity and lower overall cost if the most frequent command is prevalent.

Expected Costs: Most Frequent Command Baseline has lower cost for common commands but higher false negatives; Stratified Random Baseline has moderate cost, maintaining distribution but still random; Uniform Random Baseline has high cost due to random predictions; Last Command Baseline has variable cost, dependent on the representativeness of the last command; Sequential Random Baseline has moderate cost, incorporating recent context but still random; Command Frequency Baseline has balanced cost, reflecting actual distribution; Energy-Based Baseline's cost depends on the chosen threshold, good at distinguishing speech from noise.

Conclusion: Naive baseline systems provide essential benchmarks for evaluating more advanced speech command detection models. Each method has unique strengths and weaknesses, with costs varying by approach. Comparing these baselines helps set realistic performance targets for sophisticated models. As a result, the one with the lowest loss function is the 'most frequent command baseline,' which has a loss of 430 and has an accuracy of 0.092437, i.e., 9 percent.

	Method	Cost	Accuracy
0	Most Frequent Command	430.0	0.092437
1	Stratified Random	2880.5	0.053782
2	Uniform Random	2964.5	0.055462
3	Last Command	2113.0	0.074790
4	Sequential Random	2952.5	0.032773
5	Command Frequency	2823.0	0.059664

Figure 1: "Baseline Results with Costs and Accuracy

2 Starting point: CNN model

2.1 Describe how your system was used to detect keywords in the longer domestic recordings (windowing, hop size, etc.)

We utilized our Convolutional Neural Network (CNN) model from Task 3 as the starting point for this final challenge. The model was originally trained on 1.1-second mel spectrograms of the recordings. Since the model is limited to processing inputs of this length, we employed a windowing approach to handle longer audio recordings, as the one of the new development_scenes.

To begin, we focused exclusively on the mel spectrogram features extracted from the domestic recordings numpy array. This allowed us to construct the mel spectrogram representation for each domestic recording. Next, we implemented a sliding window technique. We created a window of 44 frames, equivalent to 1.1 seconds, and slide this window across the entire audio recording to generate mel spectrograms for each windowed segment. For each windowed segment, we applied our CNN classifier to detect keywords. The window was moved with a step size of 3 frames, that is a hyperparameter. Adjusting the step size can impact the accuracy and efficiency of predictions. Through our experiments, we found that a step size of 3 frames provided the best balance for accurate predictions.

2.2 How did you threshold and combine keyword predictions to detect full speech commands? Describe any heuristics or post-processing strategies used.

Our model generates predictions for each window, resulting in multiple predictions where a word may be detected more than once. For example, in a 15-second audio clip, we produce around 234 mel spectrogram images, which means we get 234 predictions. A keyword may be detected multiple times within these predictions, as shown in 2.

To identify the final speech command, we check if a word is detected at least three consecutive times. If this condition is met, we consider the prediction to be accurate. This approach strengthens the predictions against possible misclassifications. After confirming the keyword, we go backwards through the predictions and check if there is a device word preceding each detected 'command' keyword. If a device word is found, we combine the 'command' and

3_speech_true_Radio_an\844-887.png	other
3_speech_true_Radio_an\847-890.png	Radio
3_speech_true_Radio_an\850-893.png	Radio
3_speech_true_Radio_an\853-896.png	Radio
3_speech_true_Radio_an\856-899.png	Radio
3_speech_true_Radio_an\859-902.png	Radio
3_speech_true_Radio_an\862-905.png	Radio
3_speech_true_Radio_an\865-908.png	Radio
3_speech_true_Radio_an\868-911.png	Radio
3_speech_true_Radio_an\871-914.png	other
3_speech_true_Radio_an\874-917.png	other
3_speech_true_Radio_an\877-920.png	other
3_speech_true_Radio_an\880-923.png	other
3_speech_true_Radio_an\883-926.png	other
3_speech_true_Radio_an\886-929.png	other
3_speech_true_Radio_an\889-932.png	other
3_speech_true_Radio_an\892-935.png	an
3_speech_true_Radio_an\895-938.png	an
3_speech_true_Radio_an\898-941.png	an
3_speech_true_Radio_an\901-944.png	other

Figure 2: Prediction of the model

the ‘device’ word to form the full speech command. For example, the final result from 2 is “Radio on,” as illustrated in 3. Additionally, we define the timestamp at which the speech command is detected.

filename	command	timestamp
3_speech_true_Radio_an	Radio an	22.25

Figure 3: Prediction of the model

2.3 What strategies did you apply to minimize the task-specific cost function?

To minimize the custom cost function, we applied several strategies. We used post-processing techniques on the model’s output, such as smoothing the prediction sequence and applying filtering. Additionally, we aimed to generalize the model better on specific keywords like ‘an,’ ‘aus,’ ‘Ofen’ and ‘Licht’ by using data augmentation. Furthermore, we trained may models with different parameters to find the best hyperparameters.

2.4 Describe your evaluation setup and provide evaluation results on (parts of) the public, annotated scenes.

2.5 Does your simple speech command detection system achieve lower costs compared to the naive Baseline system?

As previously mentioned, our model achieves a loss function value of -210 on the "development scenes." Among the naive baseline systems, the one with the lowest loss function is the ’most frequent command baseline,’ which has a loss of 430. Therefore, our model performs significantly better than the baseline method.

3 Improvements

3.1 Post processing: Smoothing prediction sequences and Filtering

Based on our initial observations, we noticed that our predictions were not always smooth. To address this, we applied a smoothing prediction sequence. For each 44-frame prediction, we checked if its neighborhood had the same label classification. The neighborhood of a prediction x consists of the previous prediction of x and the two predictions following it. These three predictions “vote” on the class of the current prediction, assigning the most frequent class within the neighborhood. For example, if the prediction is “Other” but the model detected “Alarm” before and after it, the single 44-frame prediction is converted to “Alarm,” and vice versa. We used a majority vote for this process without any weighting, so it was necessary to have an odd number of neighbours. After smoothing, we also applied filtering. If, after the previous post-processing techniques, a keyword segment is shorter than a threshold (which was 2), we changed all the labels to “Other”.

3.2 Data agumentation

After applying the loss function for the first time, we realized that the most problematic part was the detection of some sensitive words. For example, “an” was not always detected, and a device word like “Often” was predicted as a False Positive, which in our custom loss function is one of the worst scenarios. For this reason, we chose to apply data augmentation to make our model more sensitive to these keywords. We investigated different augmentation techniques, but many of them were not appropriate for this kind of data because they alter the timeline, which is crucial in this classification task. For example, flipping an image vertically would mean that the time is going backward. Among various techniques, we opted for adding noise and dropping out pixels from the mel spectrogram images. This way, we hypothesised that the waveform of a keyword would be distorted, making it more similar to real scenarios where there is background noise.

3.3 Hyperparameter tuning

Hyperparameter tuning using weights and biases tool is applied over models trained with augmented data. 5 models are trained with the best one being with validation accuracy of 90% , recall of 87% and precision of 90%. The parameters we tried tuning are l1 regularization, learning rate and dropout rate.

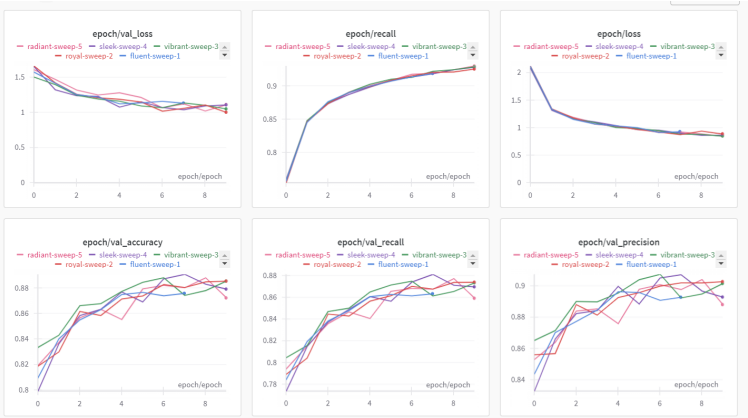

Figure 4: Visualization of hyperparameter tuning

The best and original model has an accuracy of 94.6%. Hyperparameter tuning and data augmentation did not yield any improvements compared to this model. The best model obtained through these approaches achieved only 90% accuracy. The reason for this is that hyperparameter tuning started at a random point of parameter space. Therefore, we chose to use this model to give predictions to be used for calculating task specific loss. On the other hand, the postprocessing techniques heavily helped with minimizing the loss function, achieving a great result of -221.3.

4 Critical Reflection

At the end of the project we believe that our model is appliable in a real-world environment. The results are encouraging to performs well also in a real scenario. For sure, to apply this model to a real world scenario, we need to define some predefined command to turn on the model, like “Ok Google”. At this point, we can set a fixed time during which the model will record some commands. After that we have to apply the steps already mentioned during the project. So, split the record in windows and create, for each of them, the mel spectrogram images from the 64 mel spectrogram features. All this assuming that the raw audio is converted into numpy arrays as provided during the project. Then applied the classification model to get a results. We tried to estimate the possible delay between the command pronunciation and the prediction of the model. On a 15 seconds audio we got that all the process require around 8 seconds. Generating the images from the audio is the longest part, around 7.5 seconds, and then the prediction is very fast, 0.5 seconds. So there is space to reduce this delay, improving the time to get the necessary information for the model from the raw data. To be a competitive audio detection system, like Siri or Ok Google, we need that the waiting time is decreased to 1 second. Also, there is still place for improvements on the predictions. For example, minimizing the false positive that are the most dangerous problem in a real word scenario.