MLPC Report Task 3

Team IMPORTED

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

Contributions

Mahmut Öztürk: Data Preperation and KNN
Michele Giambelli: Data Generation, Preparation and Random Forest
Barış Bakırdöven: CNN and Hyperparameter Tuning
Korhan Erdoğdu: Data Generation, Report and Presentation

1 Data Split

1.1 Describe the data split you used for model selection and hyperparameter tuning and the reason for your choice.

The data split is done using the speakers as reference. In the dataset, there are 172 speakers with an average of 263 audio recordings for each speaker. We split 70% of the speakers for training and 30% for validation. For each speaker, their recordings are used to create the respective training and validation sets. This results in 31,412 samples for training and 13,884 for validation, which is approximately 70% and 30% respectively of the total data (45,296 samples). Furthermore, we ensured that both training and validation sets contain all necessary words for classification. This approach minimizes the risk of omitting device or command words during training, thus ensuring their recognition during inference.

1.2 How did you avoid information leakage between the sets?

To prevent data leakage, we ensure that different samples of a speaker are exclusively allocated to either the training or validation set. For instance, if speaker 1 has at least one audio sample in the training set, all of their audio samples will be exclusively in the training set. This precautionary measure mitigates the risk of providing the model with future information. We identified that data leakage could potentially occur if the same words spoken by a single speaker were present in both training and validation sets, as each speaker with a unique ID utters all the words more than once. In real-world scenarios, the classifier should accurately predict audio from unknown speakers. Our stratified splitting process addresses these concerns, which could arise with random splitting, thereby aiming to produce a model that generalizes better to new speakers.

1.3 How did you derive final, unbiased performance estimates?

Final performance estimates are derived by the model checking itself against the validation data and applying k-fold cross-validation to compensate for possible biases caused by the manner in which validation data is split from the training data.

2 Classes and Features

2.1 How the 20 words and the audio labeled as 'other' are grouped

In the dataset, there are 20 different labels plus "other". Of the 20 labels, 10 words are key words for the classification task, and the other 10 are only similar words in sound to the key words. We decided to not aggregate the 10 words that are similar to the important one. This is because we want our model to be able to recognize the "similar" words and not

confuse them with the key words. We recognize that teaching the model to identify unimportant words increases the risk of mistakenly labeling other sounds as those unimportant words. However, we accept this risk because it is more important for our model to accurately distinguish key words. By teaching the model these distinctions, we can discard both wrongly or correctly labeled unimportant words, allowing us to focus more reliably on the presence of important words. So at the end, there are 21 classes; 20 for the different words and 1 for "other".

2.2 Which subset of features was selected? How did you select the subset?

Between all the 175 features we take into consideration for our model, only the 64 features, called melspect, are used. We discovered that melspect features are the raw details for the mel spectrogram image of a sample. So we decided to approach the classification problem as an image classification task, instead of audio. We selected the features that cause the most distinction between labels when converted to spectrograms and contain the most information. They are the 64 melspect features. We realize the possible existence of additional noise caused by this, but we trust that our classification model can deal with this. From these features, we are able to recreate the mel spectrogram of each sample. For example, Figure 1 is the mel spectrogram of the word 'Staubsauger' generated from the melspect features of all 44 frames.

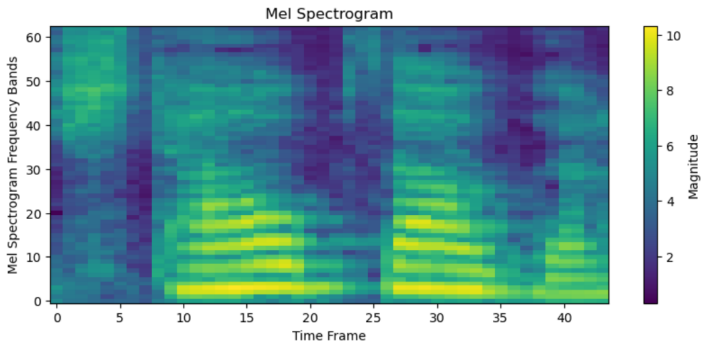


Figure 1: Mel spectrogram word Staubsauger

The spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. Mel spectrogram is the spectrogram remap for 64 pixels high, instead of 201, with high resolution on the bottom, and progressively lower resolution on the top. So, it contains all the important information for detecting a word.

We also tried features such as "mfcc", "mfcc_d", and "mfcc_d2", that come from melspect, but the best results were still obtained by melspect features.

2.3 Preprocessing of the data.

The preprocessing process of data is different from model to model. For the project, we chose to use k-Nearest Neighbor, Random Forest, and Convolutional Neural Network.

Regarding k-NN and Random Forest, we took into consideration the 64 melspect features. We reshaped the dataset from three dimensions to two dimensions by multiplying the 64 features by their 44 frames. In this way, the new data are in the correct size for the models. So, the training set has an initial shape of $(31412, 175, 44)$. We took into consideration only melspect features, so the shape becomes $(31412, 64, 44)$. Now we reshaped the dataset, and the result is $(31412, 2816)$, where $2816 = 64 \times 44$. We did the same on the validation set. So the model will use 2816 features for training.

On the other hand, the preprocessing for the CNN is similar but not the same. The melspect features of the data for 44 frames are converted to spectrograms. The obtained images are saved into different folders named as the labels. Then each image will be saved in the folder with the name of its label. This gives 21 folders (20 labels and "other"), each containing spectrograms of a single label. This is done for both the validation and training set. We obtained two directories, training and validation folder, in each of them 21 subfolders named as the 21 labels. We opted to implement resizing and normalization techniques. To feed the images into the CNN, they are resized to 128x128. Additionally, pixel values are scaled between 0 and 1 by dividing them by 255. These steps were essential for efficient resource management and to promote the model's robust performance, particularly post-normalization.

3 Evaluation

3.1 Evaluation criterion for evaluating hyperparameters and algorithms.

For evaluating the hyperparameter settings and models, we prioritized validation accuracy over validation recall or precision. This decision was based on our goal to correctly classify every single word, rather than solely minimizing false negatives. However, we still monitored metrics such as recall, precision, and F1 score, as they provide valuable insights into the model's performance and its ability to handle class imbalances. Our primary objective was to maximize validation accuracy, ensuring that our model would perform optimally on new, unseen data.

3.2 Baseline performance.

A baseline classifier is a simple model used as a reference point for comparing the performance of more complex models. In our project the baseline classifier is represented by a Naive Bayes classifier. The Naive Bayes classifier has a validation accuracy of 46%. This result tells that this classifier is slightly better than random performance and so the features are informative for the model. Obviously there is space for improvement. We will compare the results of our models with this one to know if our models are better than the baseline performance. In general, we will expect that the performance of more complex models with optimal tuning of parameters will be much better than the Naive classifier. For example, we will expect an accuracy of around 90% from a CNN.

4 Experiments

We decided to implement three different models: K-nearest neighbor, Random Forest and Convolutional Neural Network.

4.1 How does classification performance change with varying hyperparameter values?

In K-NN to analyze how the classification performance changes with varying hyperparameter values, we use Grid Search Cross Validation (GridSearchCV). We specified as hyperparameters 'n_neighbors', 'weights', and 'metric'. In Figure 2 it is possible to see the performance metrics for different hyperparameter combinations. When we use the best hyperparameters combination, found from GridSearchCV: 'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'distance', the model performs better with a validation accuracy of 0.7796. Without the grid search, the validation accuracy was 0.70.

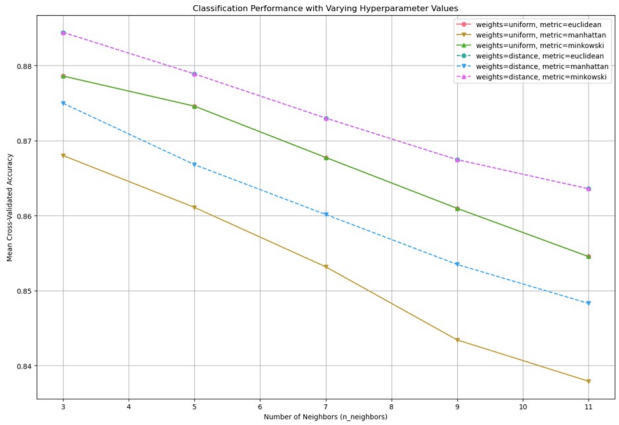


Figure 2: Performance metrics for k-NN of different hyperparameter combinations

Regarding the Random Forest, we started with a model with 100 trees and a maximum depth of 10. The model seems to perform very well on the training data with an accuracy of 88%. However, when testing the model on the validation set, the results decrease to 77%. Then we tried hyperparameter tuning with a GridSearchCV approach, as in K-NN. We specified a grid of hyperparameters: 'n_estimators': [100, 200, 300], 'max_depth': [None, 10, 20], 'min_samples_split': [2, 5, 10]. In this way, we tried different hyperparameters and also used cross-validation to avoid the problem of overfitting. The best model is the one with 300 trees with the maximum depth of the trees set to None, meaning the trees are expanded until all leaves are pure. It is possible to see in Figure **??** that the growth of the accuracy is linear

and it will increase even after ‘max_depth‘ = 20. The minimum number of samples required to split an internal node is set to 2. The Random Forest with these hyperparameters performs better both on the training and validation sets. The training accuracy grows to 99%, and the validation accuracy increases from 77% to 83%.

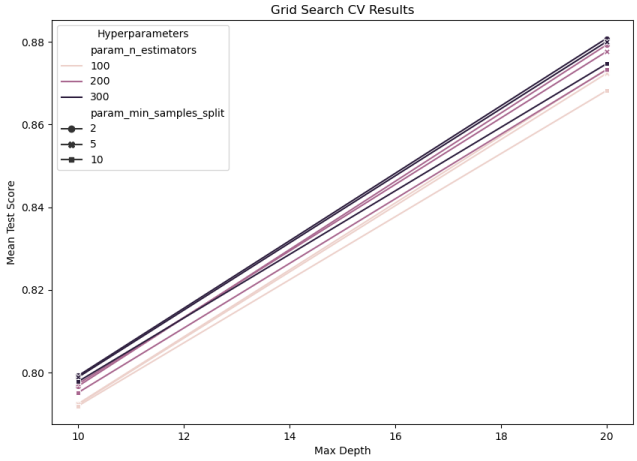


Figure 3: Performance metrics for Random Forest of different hyperparameter combinations

For CNNs, classification performance improved with proper tuning of hyperparameters like learning rate, batch size, and regularization. Given the high dimensionality of these parameters and the computational demands of training a CNN, we utilized the Weights and Biases tool. This tool allowed for an easier hyperparameter sweep process and provided high-quality visualizations by logging numerous metrics at each epoch, which can be seen in Figure 4.

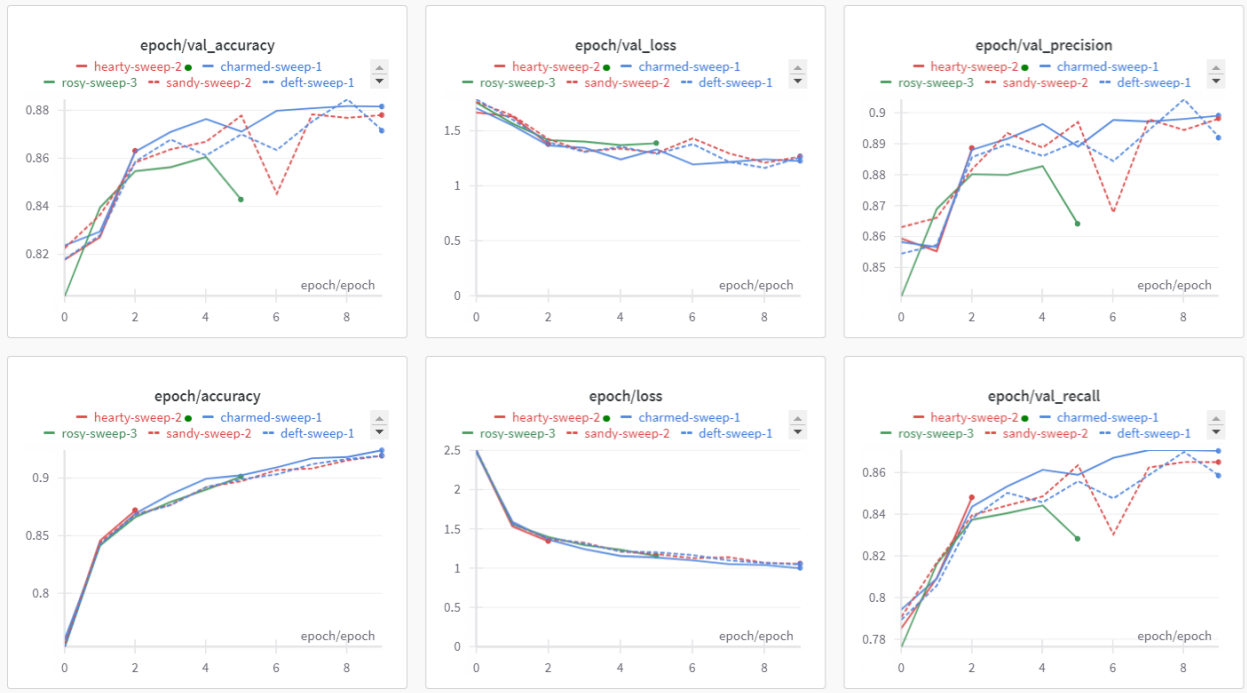


Figure 4: Performance metrics for CNN of different hyperparameter combinations

4.2 Does overfitting or underfitting occur? and what does it depend on?

All the models demonstrated overfitting, indicated by very high training accuracy and significantly lower validation accuracy. This overfitting likely stems from the model's complexity or the high number of features. In the case of CNNs, insufficient regularization also contributed. Overfitting was evident from epoch 6 (see Figure 5), where training accuracy continued to rise while validation accuracy declined, and validation loss increased.

To overcome overfitting, we used cross-validation for k-NN and Random Forest models. For CNNs, we increased regularization, applied transfer learning, adjusted the number of trainable layers, and implemented early stopping. The figure also illustrates early stopping's effectiveness. It stops 3 epochs after the 6th because validation loss does not decrease anymore, preventing the model from memorizing the training data and saving computation time.

We never encountered underfitting.

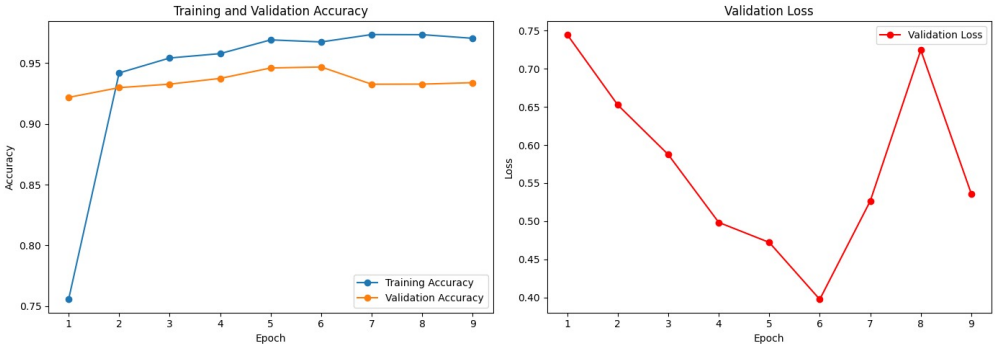


Figure 5: Training and validation accuracy/loss over epochs indicating overfitting and the effect of early stopping.

4.3 Compare the performance of the three classifiers.

Naive Bayes	K-NN	Random Forest	CNN
46.06%	77.96%	83.31%	94.60%

Table 1: validation Accuracy

As it is possible to see in Table 1 the results of the three models, we can see that all of them are better than the Naïve Bayes. Also the CNN has the best validation accuracy. The CNN outperformed the other models due to its ability to capture complex patterns in the spectrograms. For this reason, we applied the CNN to the next step, i.e, 4 realistic scene.

5 Analysis of Realistic Scenes

5.1 Listen to the scenes and inspect the corresponding predictions of the classifier. How well does the classifier recognize keywords?

After listening to the four scenes, we identified the key speech commands for classification. The data from these scenes were processed into spectrogram images using a sliding window approach, with a window size of 44 frames (25ms) and a step size of 3 frames. From the "Florian_Heizung_aus" scene, this method produced 271 spectrograms. These spectrograms were then preprocessed similarly to our training data and classified using a pretrained EfficientNetV2B3 model with transfer learning. The results, saved in an Excel file, displayed around 95% accuracy. To verify correct classifications, we correlated predicted labels with the timestamps in the original recordings. For instance, "Heizung" starts at frame 562. By dividing this by 44 and multiplying by 1.1, we pinpointed the exact second in the recording for verification. The same method was used for "aus.". The resulting table can be seen in Figure 6.

We understand that the *Heizung* is correctly labelled by checking the time it is spoken, which is 562 (start of Heizung) divided by 44 and then multiplied by 1.1. In this way, we know the exact second that we have to check in the original recording to know if the classification is correct. The same applies for *aus.*

The predictions have, as expected, around 95% correct, and there are some single wrongly labelled images. We can note that the smaller the sliding window step size, more images there are. This means less amount of false negatives, but with the risk of more false positives. We found that either 3-4-5 as step sizes are the most efficient in this sense.

Spectrogram ID	Label
2_Florian_Heizung_aus\559-602.png	other
2_Florian_Heizung_aus\562-605.png	Heizung
2_Florian_Heizung_aus\565-608.png	Heizung
2_Florian_Heizung_aus\568-611.png	Heizung
2_Florian_Heizung_aus\571-614.png	Heizung
2_Florian_Heizung_aus\574-617.png	Heizung
2_Florian_Heizung_aus\577-620.png	Heizung
2_Florian_Heizung_aus\580-623.png	Heizung
2_Florian_Heizung_aus\583-626.png	Heizung
2_Florian_Heizung_aus\586-629.png	Heizung
2_Florian_Heizung_aus\589-632.png	Heizung
2_Florian_Heizung_aus\592-635.png	other
2_Florian_Heizung_aus\595-638.png	other
2_Florian_Heizung_aus\598-641.png	other
2_Florian_Heizung_aus\601-644.png	other
2_Florian_Heizung_aus\604-647.png	other
2_Florian_Heizung_aus\607-650.png	other
2_Florian_Heizung_aus\610-653.png	other
2_Florian_Heizung_aus\613-656.png	aus
2_Florian_Heizung_aus\616-659.png	aus
2_Florian_Heizung_aus\619-662.png	aus
2_Florian_Heizung_aus\622-665.png	aus
2_Florian_Heizung_aus\625-668.png	aus
2_Florian_Heizung_aus\628-671.png	aus
2_Florian_Heizung_aus\631-674.png	aus
2_Florian_Heizung_aus\634-677.png	other
2_Florian_Heizung_aus\637-680.png	other

Figure 6: Prediction CNN on scene "Florian_Heizung_aus"

5.2 What are particular problematic conditions that cause the classifier to miss or mispredict keywords? How could you alleviate this problem in the challenge phase of the project?

The main challenge is loud background noise, like a hairdryer, which can obscure short words such as "an." The model can handle noise or short words separately but struggles when both are present. To address this, we can smooth the spectrogram using filtering techniques. Median filtering can reduce constant background noise, while Gaussian filtering provides smoothness. This combined approach can help the model better detect short words even when there is loud background noise.

5.3 Use the spectrogram and the sequence of predictions to visualize interesting (positive or negative) findings.

One interesting observation is that in analyzing the word "an" spoken with a hairdryer running in the background, we saw a constant line of intense pixels at lower frequencies, representing the hairdryer noise. This combination of short words and loud background noise posed a challenge for the model, and its representation can be seen in Figure 7.

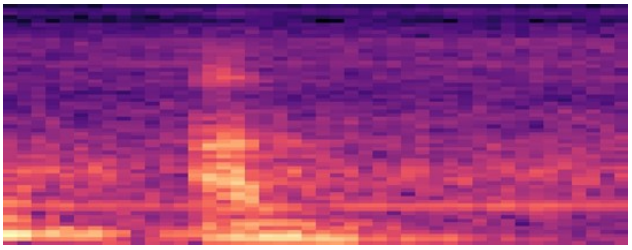
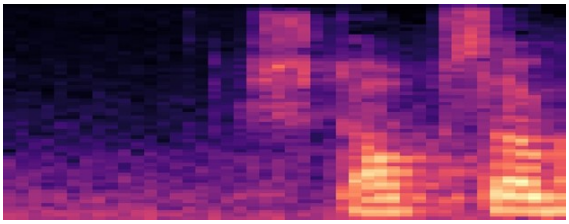


Figure 7: Representation of the word "an" with hairdryer noise in the background.

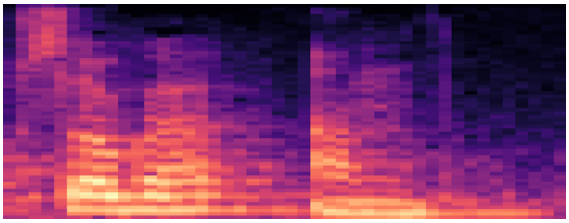
Another finding was regarding at what point the model starts and stops understanding the words. In the "Verena_Staubsauger_an_Alarm_an" scene, the model successfully detected the word "Staubsauger" within the specific frames of 268-311 and 301-344. The labelings can be seen in Figure 8 and the detected "Staubsauger" spectrograms can be seen in Figures 9a and 9b.

Spectrogram ID	Label
3_Verena_Staubsauger_an_Alarm_an\259-302.png	other
3_Verena_Staubsauger_an_Alarm_an\262-305.png	other
3_Verena_Staubsauger_an_Alarm_an\265-308.png	other
3_Verena_Staubsauger_an_Alarm_an\268-311.png	Staubsauger
3_Verena_Staubsauger_an_Alarm_an\271-314.png	Staubsauger
3_Verena_Staubsauger_an_Alarm_an\274-317.png	Staubsauger
3_Verena_Staubsauger_an_Alarm_an\277-320.png	Staubsauger
3_Verena_Staubsauger_an_Alarm_an\280-323.png	Staubsauger
3_Verena_Staubsauger_an_Alarm_an\283-326.png	Staubsauger
3_Verena_Staubsauger_an_Alarm_an\286-329.png	Staubsauger
3_Verena_Staubsauger_an_Alarm_an\289-332.png	Staubsauger
3_Verena_Staubsauger_an_Alarm_an\292-335.png	Staubsauger
3_Verena_Staubsauger_an_Alarm_an\295-338.png	Staubsauger
3_Verena_Staubsauger_an_Alarm_an\298-341.png	Staubsauger
3_Verena_Staubsauger_an_Alarm_an\301-344.png	Staubsauger
3_Verena_Staubsauger_an_Alarm_an\304-347.png	other
3_Verena_Staubsauger_an_Alarm_an\307-350.png	other

Figure 8: Labeling of Staubsauger.



(a) Frame 268-311, first spectrogram to be labeled as "Staubsauger".



(b) Frame 301-344, last spectrogram to be labeled as "Staubsauger".

Figure 9: Spectrograms for the "Staubsauger" detection.