ML4IoT-HW02

Hojjat Miryavifard s334026
Ali Behlooei Dolatsaraei s334033
Sadjad Zamani s331775

1.1 Methodology to Discover Hyper-Parameters

We chose MFCC over Mel Spectrogram due to its compact representation, noise robustness, and suitability for speech recognition. Our test showed that using MFCC as a preprocessor showed silghtly better results.

To optimize MFCC, we tested num-coefficients values between 10 and 20. These variations did not show significant performance differences, so we kept the number low to avoid overfitting.

Initially, a batch size of 16 led to early stopping, likely due to overfitting. Increasing the batch size to 32 resolved this, providing better generalization and stability during training.

Our model incorporates a width multiplier parameter (wm) used for structured pruning, which helps reduce the model size. In the CNN architecture, increasing wm increases the model's size and can sometimes improve performance, though it may also lead to overfitting. Through experimentation, we found that a value of 0.6 struck the right balance between model size and performance.

For the number of epochs, we slightly increased it, aligning this change with the increase in batch size to potentially enhance training accuracy.

For the remaining hyperparameters, we stick to the predefined values introduced in the lab notebooks as they were already meeting the constraints.

1.2 Preprocessing and Hyper-Parameters

The table below summarizes the pre-processing type and hyper-parameters used in the final solution:

Table 1. Preprocessing Type and Hyper-Parameters

Parameter	Value
Preprocessing Type	MFCC
Sampling Rate (Hz)	16000
Frame Length (s)	0.04
Frame Step (s)	0.02
Number of Mel Bins	40
Lower Frequency (Hz)	20
Upper Frequency (Hz)	4000
Number of MFCC Coefficients	10

1.3 Training Hyper-Parameters

The table below summarizes the training hyper-parameters used in the final solution:

Table 2. Training Hyper-Parameters

Parameter	Value
Batch Size	32
Initial Learning Rate	0.01
End Learning Rate	1e-5
Number of Epochs	25
Width Multiplier (wm)	0.6

1.4 Model Architecture and Optimizations

We used a similar model like in the Figure 1 which is the one that we used in Homework 4.2 but adjusted the width multipliers and number of filters. Specifically, instead of keeping all filters at 128 as, we reduced the last layer to 64 filters and applied different width multipliers to decrease the model size.

The model architecture consists of an initial Conv2D layer followed by batch normalization and ReLU activation. This is followed by two separable depthwise convolution blocks, each comprising a DepthwiseConv2D layer, a Conv2D layer, batch normalization, and ReLU activation. The model concludes with a global average pooling layer and a dense output layer. This design ensures compactness while maintaining accuracy.

As we obtained different results every time we trained the model, we set a seed to ensure reproducibility in our results (we did the same in other courses like ML-DL) and thought it would be appropriate here as well, but we are not entirely sure if this is acceptable.

Structured pruning was applied through the width multiplier parameter, wm, to control the number of filters. A value of 0.6 provided an optimal balance between model size and performance.

1.5 Table Reporting Final Results

The model's performance is summarized below:

- Accuracy: 99.50% (Requirement: $> 99.4\%$)
- TFLite Size: 47.0 KB (Requirement: < 50 KB)
- Latency: ≈ 35 ms (Requirement: < 40 ms)

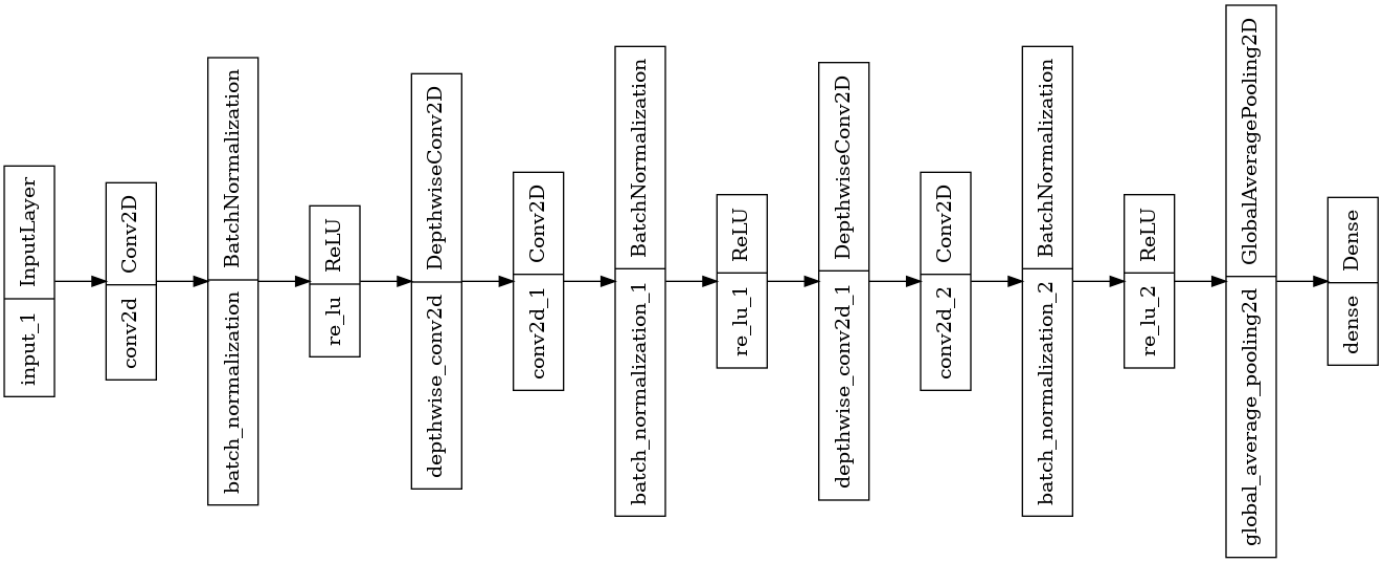

Figure 1. Final Model Architecture

1.6 Comments on Results

The model met all constraints. The high accuracy and low latency confirm the effectiveness of depthwise separable CNN and MFCC. The small size ensures the feasibility of deployment in resources-constrained devices.

We also would like to point out an issue with this model. This model, which focuses on just detecting ”up” and ”down,” faces some real-world challenges. Since it only assigns probabilities to these two words, it can mistakenly react to irrelevant sounds because it does not have the option to simply detect neither up nor down. We understand that this is a limitation which we should overcome to make it more practical for use in a real world scenario, where the sounds around us are much more varied.