HW2 – Group 1

Mert Toprak (s336966) **Daghan Ufuk Leflef** (s338142) **Muhammed Emin Oral** (s343124)

1. Methodology

We implemented a Keyword Spotting (KWS) pipeline for "up" and "stop" commands using PyTorch and Torchaudio. The system is deployed on a Raspberry Pi using ONNX Runtime. Our approach focused on minimizing latency and size while maintaining high accuracy through valid architectural choices:

- **Data Pipeline:** We adhered to the official MSC split to ensure valid evaluation. To prevent overfitting on the small dataset, we implemented robust augmentations (Random Shift, Gain, Gaussian Noise) and Frequency Masking.
- **Architecture Selection:** A Depthwise Separable CNN (DSCNN) was chosen. This architecture separates spatial and channel-wise convolutions, significantly reducing parameter count and computational cost (MACs) compared to standard CNNs, making it ideal for edge deployment.
- **Quantization Strategy:** The final model was statically quantized to Int8 using a representative validation dataset, achieving a 4x size reduction with negligible accuracy loss.

2. Feature Extraction

We used Log-Mel Spectrograms as input features. A critical optimization was increasing the `frame_step` (hop length) to **20ms**. This reduces the total number of frames in the spectrogram by 50% (compared to the standard 10ms), which linearly reduces the inference load of the entire downstream neural network. To compensate for the reduced temporal resolution, we increased the frequency resolution (`n_mels`) to **46**.

Hyperparam.	**Value**
frame_length_in_s	0.025 (25 ms)
frame_step_in_s	0.020 (20 ms)
n_mels	46
f_min	0 Hz
f_max	8000 Hz
n_mfcc	- (Uses MelSpec)

Table 1: Feature Extraction hyperparameters.

3. Training Hyperparameters

We trained for 100 epochs using **OneCycleLR** and **AdamW**, enabling super-convergence and helping the lightweight model escape local minima to reach 100% accuracy.

Hyperparam.	**Value**
Epochs	100
Batch Size	32
Optimizer	AdamW
Learning Rate	0.005 (Max, OneCycleLR)
Weight Decay	0.001

Table 2: Training hyperparameters.

4. Model Design & Optimization

The architecture is a lightweight **Depthwise Separable CNN (DSCNN)** comprising a Stem (Standard Conv2d) and 5 Depthwise Separable Blocks. We iteratively tuned specific parameters to balance the tight Latency vs. Accuracy trade-off:

- **Hop Length (160 → 320):** Doubling the hop step reduced the temporal dimension of the input spectrogram by 50%. This was the most impactful change, reducing both feature extraction and model inference time by half (2.4ms saved).
- **n_mels (40 → 46):** We initially reduced n_mels to 40 to minimize latency, but accuracy dropped to 98.5%. Increasing it to 46 restored 100% accuracy with negligible latency cost.
- **Training Strategy:** Increasing epochs to 100 with **OneCycleLR** allowed the reduced-capacity model to fully converge. Augmentations prevented overfitting, pinning Test Accuracy at 100%.

5. Results & Discussion

The final solution meets all rigid constraints. As shown in Table 3, we achieved a perfect accuracy score on the test set while keeping latency under the 5ms threshold and size well below the 300KB limit.

Acc. (%)	**Size (KB)**	**Lat. (ms)**
100.00	117.48	4.8

Table 3: Evaluation results of the custom KWS pipeline.

The solution meets all project constraints. The total storage footprint is explicitly composed of **62.04 KB** for the feature extraction frontend and **55.44 KB** for the quantized classification backbone. The latency budget is effectively split between Feature Extraction (∼2.4ms) and Model Inference (∼2.4ms). This underscores the importance of the `hop_length` optimization; without it, feature extraction alone would have consumed nearly the entire 5ms budget.