MARMARA UNIVERSITY ENGINEERING FACULTY

EE 4065

Introduction to Embedded Digital Image Processing

Homework 5 Report

NAME:	*Baran*	*Muhammet Yücel*
SURNAME:	*ORMAN*	*ÇELİK*
NUMBER:	*150721063*	*150721024*

1. QUESTION

Keyword Spotting from Audio Signals (Application 12.8)

1.1 Introduction

Keyword spotting (KWS) is a fundamental task in speech-based human–machine interaction systems, where a device identifies predefined spoken words from audio signals. Due to limited computational and memory resources, implementing keyword spotting on microcontrollers requires lightweight feature extraction and efficient machine learning models.

In this study, the **Keyword Spotting from Audio Signals** application described in **Section 12.8** of the reference book is implemented on an **STM32F4 microcontroller**. The system follows an **offline training – on-device inference** approach. A neural network model is trained on a PC using Python and TensorFlow, converted to a TensorFlow Lite format, and then embedded into the STM32F4 using **X-CUBE-AI**. The performance of the embedded model is validated using real audio input.

2. Dataset and Feature Extraction

2.1 Dataset

The **Free-Spoken Digit Dataset (FSDD)** is used in this application. The dataset consists of spoken digit recordings from **zero to nine**, sampled at **8 kHz**. Each audio file is named in the format, **digitLabel_speakerName_index.wav.**

To ensure **speaker-independent evaluation**, the dataset is split as follows:

- Recordings spoken by **"yweweler"** are used as the **test set**.
- All remaining recordings are used for **training**.

2.2 MFCC Feature Extraction

Mel-Frequency Cepstral Coefficients (MFCCs) are used to represent audio signals. MFCCs capture perceptually meaningful spectral characteristics of speech and are well-suited for keyword spotting tasks.

The feature extraction parameters are chosen in accordance with the reference book:

- FFT size: **1024**
- Sampling rate: **8000 Hz**
- Number of Mel filters: **20**

- Number of DCT outputs: **13**
- Window function: **Hamming**

For each audio signal:

1. The signal is framed and windowed.
2. FFT is applied to obtain the frequency spectrum.
3. Mel filter banks are applied.
4. Logarithmic Mel energies are computed.
5. Discrete Cosine Transform (DCT) is applied to obtain **13 MFCC coefficients**.

To generate a fixed-length feature vector for each recording, **statistical aggregation** is applied:

- Mean of MFCC coefficients (13 values)
- Standard deviation of MFCC coefficients (13 values)

As a result, each audio recording is represented by a **26-dimensional feature vector**.

3. Neural Network Model and Training

3.1 Model Architecture

A multilayer perceptron (MLP) neural network is used for keyword spotting. The model architecture is as follows:

- Input layer: **26 neurons** (MFCC mean and standard deviation)
- Hidden layer 1: **100 neurons**, ReLU activation
- Hidden layer 2: **100 neurons**, ReLU activation
- Output layer: **10 neurons**, Softmax activation

This structure enables multi-class classification for digits from zero to nine.

3.2 Training Configuration

The neural network is trained offline using Python and TensorFlow with the following settings:

- Optimizer: **Adam**
- Learning rate: **1e-3**

- Loss function: **Categorical Cross-Entropy**
- Number of epochs: **100**

Training labels are converted to **one-hot encoded format**. After training, the model performance is evaluated using the speaker-independent test set.

3.3 Training Results

The trained model achieves an overall test accuracy of approximately **72%** on the unseen speaker data. The confusion matrix shows that most digits are correctly classified, while certain digits with similar acoustic characteristics exhibit confusion. Despite these limitations, the model performance is sufficient for demonstrating keyword spotting on a resource-constrained embedded system.

The trained model was evaluated on the test dataset. The resulting Confusion Matrix is presented in Figure 1.

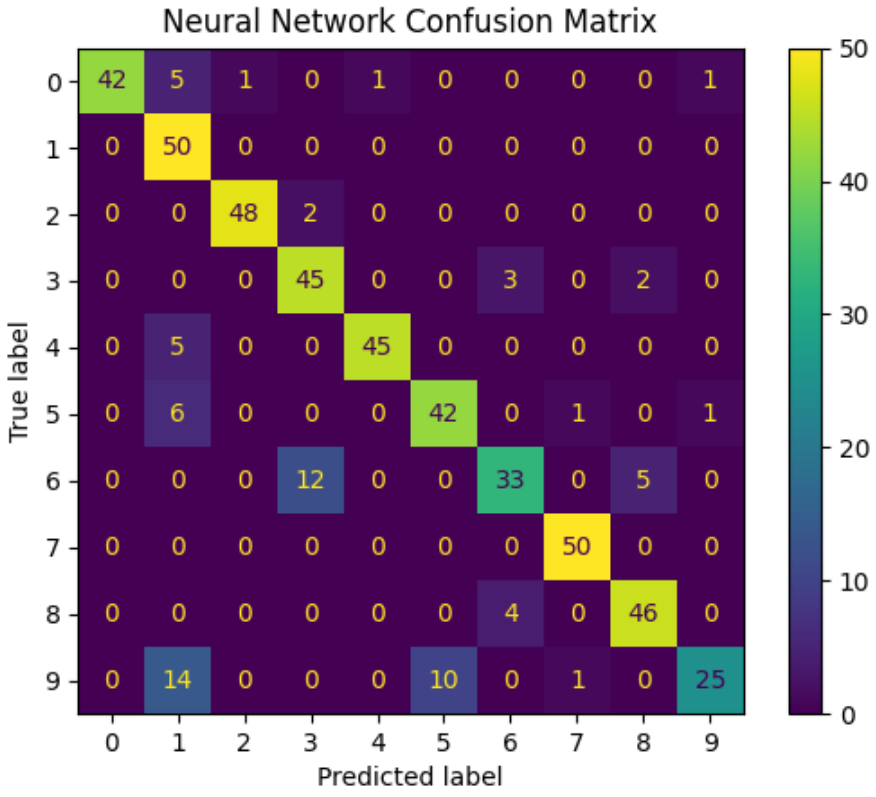

Figure 1 Confusion matrix of the neural network classifier evaluated on the speaker-independent test set.

The results confirm that the proposed MFCC-based feature representation combined with a lightweight multilayer neural network provides an effective solution for keyword spotting on resource-constrained embedded platforms.

4. Embedded Implementation on STM32F4

4.1 Model Conversion and Deployment

After training, the neural network model is converted to **TensorFlow Lite (.tflite)** format. The TFLite model is then imported into **STM32CubeIDE** using the **X-CUBE-AI** middleware.

X-CUBE-AI automatically generates the required C source and header files for neural network inference on the STM32F4 microcontroller. The generated code integrates seamlessly with the existing STM32 firmware.

4.2 Audio Acquisition and Inference Flow

In the embedded implementation, the audio signal is first captured using the STM32F4 audio acquisition peripherals. The acquired signal is then processed on the microcontroller, where Mel-Frequency Cepstral Coefficient (MFCC) features are extracted using the same parameter configuration as employed during the offline training stage. From these MFCC coefficients, a fixed-length feature vector consisting of 26 elements is constructed and provided as input to the embedded neural network. The neural network subsequently performs inference on this feature vector, and the final output of the network corresponds to the predicted spoken digit class.

5. Experimental Validation

To validate the functionality of the embedded system, a real audio sample (five.wav) is used. The audio file contains the spoken word *"five"*. The Python script five.py is used to transmit this audio data to the STM32F4-based system.

It is important to note that:

- The **model training** is performed entirely offline on a PC.
- The **actual inference** is executed on the STM32F4.
- The five.py script is used solely as a **functional validation tool** to verify that the embedded neural network correctly processes real audio input.

The STM32F4 successfully identifies the spoken digit, confirming that the embedded keyword spotting system operates as expected.

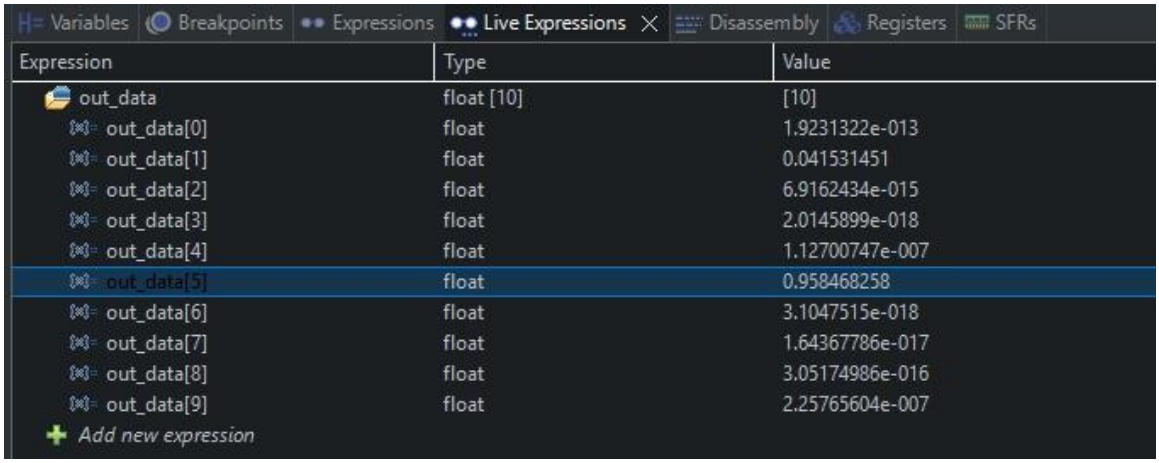

Figure 2 STM32F4 debugging view showing neural network output values during embedded inference.

6. Question 2

Handwritten Digit Recognition from Digital Images (Application 12.9)

Handwritten digit recognition is a classical pattern recognition problem and serves as a fundamental benchmark for evaluating machine learning algorithms in image-based classification tasks. Efficient implementation of such systems on microcontrollers is particularly important for embedded vision applications where computational resources and memory are limited.

In this work, the **Handwritten Digit Recognition** application described in **Section 12.9** of the reference book is implemented on an **STM32F4 microcontroller**. The system follows an offline training and embedded inference approach. A neural network model is trained using the MNIST dataset on a PC, converted into TensorFlow Lite format, and deployed on the STM32F4 using the **X-CUBE-AI** framework. The correctness of the embedded system is validated using real test inputs.

7. Dataset and Preprocessing

7.1 Dataset

The **MNIST handwritten digit dataset** is used in this application. MNIST consists of grayscale images of handwritten digits from **0 to 9**, each image having a resolution of **28 × 28 pixels**. The dataset includes many samples for training and testing, making it suitable for evaluating generalization performance.

7.2 Preprocessing

Before training, each image is converted to a floating-point representation and normalized to ensure numerical stability during training. The two-dimensional image data is reshaped into a one-dimensional feature vector, which is then provided as input to the neural network. This preprocessing step ensures compatibility with both the offline training environment and the embedded inference implementation.

8. Neural Network Model and Training

8.1 Model Architecture

A lightweight neural network architecture is used to balance classification accuracy and computational efficiency. The model takes the flattened image data as input and produces a probability distribution over ten output classes corresponding to digits from zero to nine.

8.2 Training Configuration

The neural network is trained offline using Python and TensorFlow. The training process employs commonly used optimization techniques and loss functions suitable for multi-class classification. After convergence, the trained model is evaluated using a held-out test set from the MNIST dataset.

8.3 Training Results

The performance of the trained neural network is evaluated using a confusion matrix, as shown in Figure X. The confusion matrix demonstrates that the model successfully classifies many handwritten digits, with strong diagonal dominance indicating high recognition accuracy across most classes.

Digits such as **0, 1, 6, and 7** are recognized with particularly high accuracy, showing minimal confusion with other classes. This indicates that the learned feature representations are highly discriminative for these digits. On the other hand, certain digits exhibit noticeable confusion due to similarities in handwriting styles. For instance, digits **2, 3, and 5** show mutual misclassifications, while digits **8 and 9** occasionally overlap with other rounded or loop-based digit shapes.

Overall, the confusion matrix confirms that the trained model generalizes well to unseen handwritten samples. The observed misclassifications are consistent with known ambiguities in handwritten digit recognition and reflect the intrinsic difficulty of distinguishing visually similar digit patterns.

The performance on the test set is visualized in the Confusion Matrix (Figure 3).

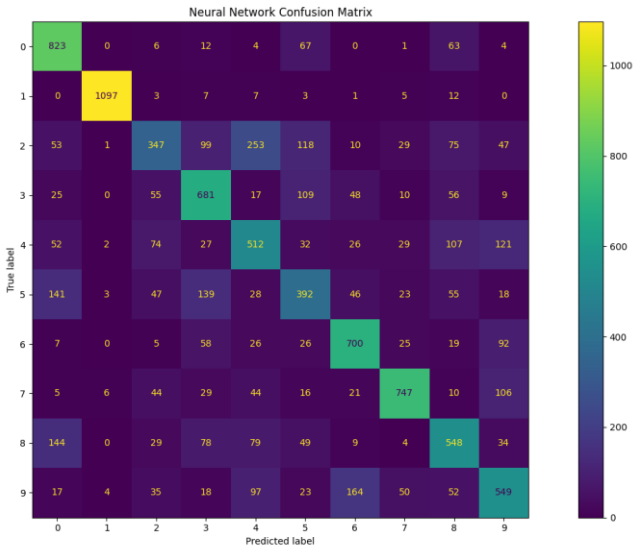

Figure 3 Confusion matrix of the handwritten digit recognition neural network evaluated on the MNIST test dataset.

9. Embedded Implementation on STM32F4

9.1 Model Conversion and Deployment

After offline training, the neural network model is converted to **TensorFlow Lite (.tflite)** format. The converted model is imported into **STM32CubeIDE** using the **X-CUBE-AI** middleware. X-CUBE-AI automatically generates optimized C source files that enable neural network inference on the STM32F4 microcontroller.

9.2 Inference Flow on the Microcontroller

In the embedded system, the preprocessed image data is provided as input to the neural network running on the STM32F4. The neural network performs inference using the embedded model, and the output layer produces class probabilities for digits zero through nine. The predicted digit corresponds to the class with the highest probability.

10. Experimental Validation

To validate the correctness of the embedded implementation, a test input representing the handwritten digit **"7"** is used. This test follows the same validation strategy applied in Q1, where a known input is provided to the embedded system to verify end-to-end functionality.

The STM32F4 successfully classifies the input digit as **7**, confirming that the trained model has been correctly deployed and that the embedded inference pipeline operates as intended. This result demonstrates consistency between the offline training environment and the on-device execution.

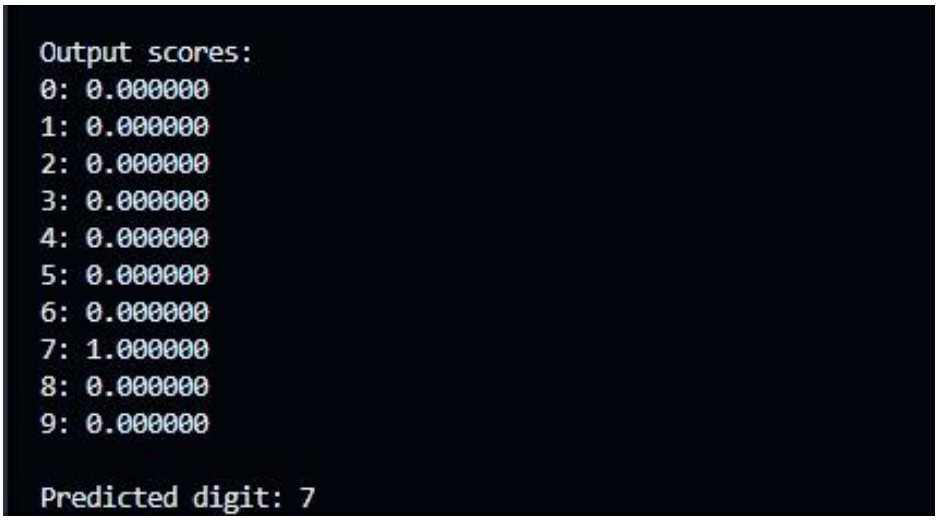

Figure 4 Embedded inference result on STM32F4 where the neural network correctly predicts digit 7

Variables | Breakpoints | Expressions | Live Expressions | Disassembly | Registers | SFRs

Expression	Type	Value
hu	float [7]	0x2001ffcc
in_data	float [7]	0x20000b28 <in_data>
out_data	float [10]	0x20000b44 <out_data>
out_data[0]	float	0
out_data[1]	float	0
out_data[2]	float	1.89127177e-014
out_data[3]	float	1.00893489e-043
out_data[4]	float	8.29012042e-019
out_data[5]	float	2.10816375e-026
out_data[6]	float	1.40129846e-045
out_data[7]	float	1
out_data[8]	float	2.49733067e-037
out_data[9]	float	1.9536657e-024
pred	volatile int	7
Add new expression		

Figure 5 Console output confirming the final prediction result of the embedded handwritten digit recognition system.

11. Conclusion

In this study, two embedded machine learning applications—keyword spotting from audio signals and handwritten digit recognition from digital images—were successfully implemented on an **STM32F4 microcontroller**. In both applications, neural network models were trained offline using Python, converted to TensorFlow Lite format, and deployed on the microcontroller using the **X-CUBE-AI** framework.

Experimental results demonstrate that the embedded models operate correctly and consistently with the offline training results. Real input tests, including spoken audio samples and handwritten digit inputs, confirm the correctness of the end-to-end inference pipeline on the STM32F4 platform. The confusion matrix analyses further indicate that the trained models achieve reliable classification performance, with most errors arising from intrinsically similar classes.

Overall, this work shows that lightweight neural network models can be effectively deployed on resource-constrained embedded systems, making STM32 microcontrollers a viable platform for real-time audio and image-based machine learning applications.