



Marmara University
Faculty of Engineering
Electrical and Electronics Engineering

EE4065.1: INTRODUCTION TO EMBEDDED IMAGE PROCESSING#

Homework #4 Report & Results

Name Surname	Student ID
Yusuf Yiğit Söyleyici	150723524
Mehmet Açar	150719020

Introduction

The objective of this assignment was to implement embedded machine learning applications for handwritten digit recognition on the STM32 Nucleo-F446RE microcontroller. The work is based on Sections 10.9 and 11.8 of the course textbook (*Embedded Machine Learning with Microcontrollers*).

The assignment was divided into two main tasks:

1. **Binary Classification (Q1):** Distinguishing the digit "0" from "non-zero" digits using a Single Neuron model.
2. **Multi-Class Classification (Q2):** Recognizing all ten digits (0-9) using a Multi-Layer Perceptron (MLP) Neural Network.

Both implementations utilized a hybrid workflow: model training was performed offline using Python (TensorFlow/Keras) on the MNIST dataset, while inference—including the manual calculation of Hu Moments—was implemented in C on the STM32 microcontroller.

Q1 Single Neuron Application (Section 10.9)

2.1 Methodology

The goal was to classify images as "Zero" (Class 0) or "Not Zero" (Class 1).

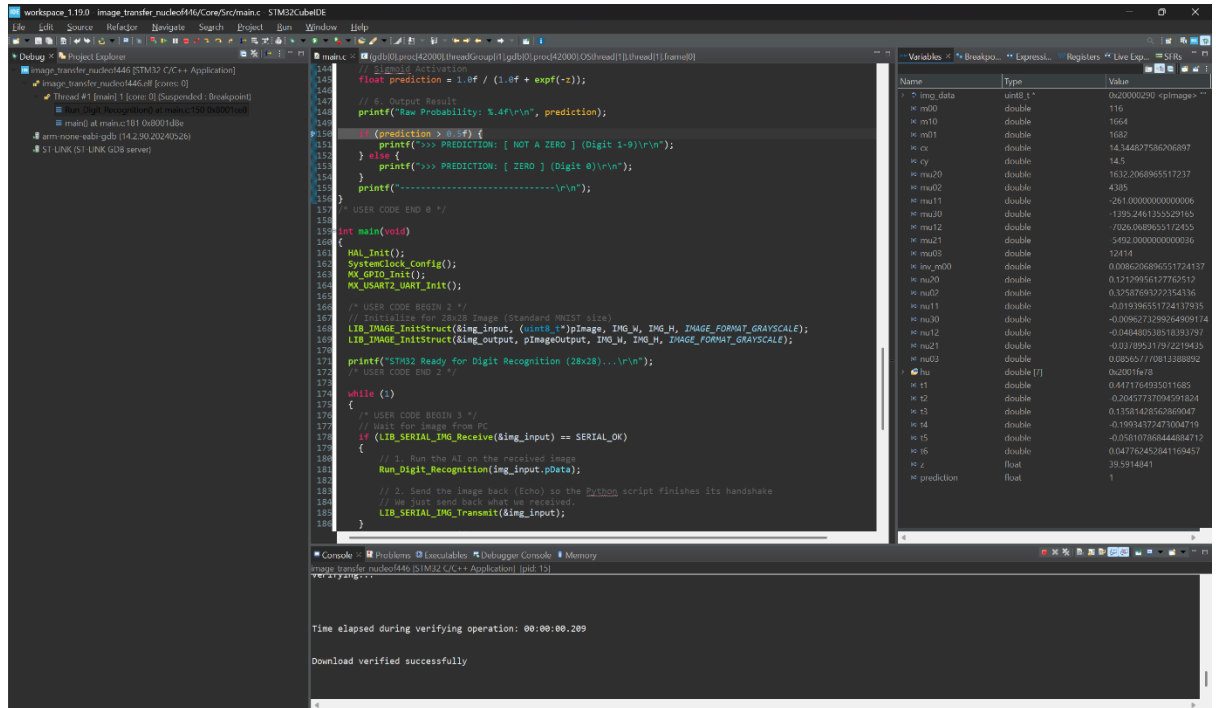
- Training (Python): A single neuron with a sigmoid activation function was trained on the MNIST dataset. The 784-pixel images were reduced to 7 Hu Moments before training. The learned weights and bias were exported to C.
- Embedded Implementation (STM32):
 - Input: 28x28 grayscale image received via UART.
 - Feature Extraction: A custom C function was implemented to calculate the 7 Hu Moments from raw pixels without using external libraries like OpenCV.
 - Inference: The neuron's output was calculated using the dot product of the normalized features and weights, followed by the sigmoid function:

$$P(y = 1|x) = \frac{1}{1 + e^{-(W*x+b)}}$$

2.2 Results

The system was verified using sample images sent from a PC via UART.

- Test Case A (Digit '7'): The model predicted a probability of 1.0, correctly classifying the image as "NOT A ZERO".



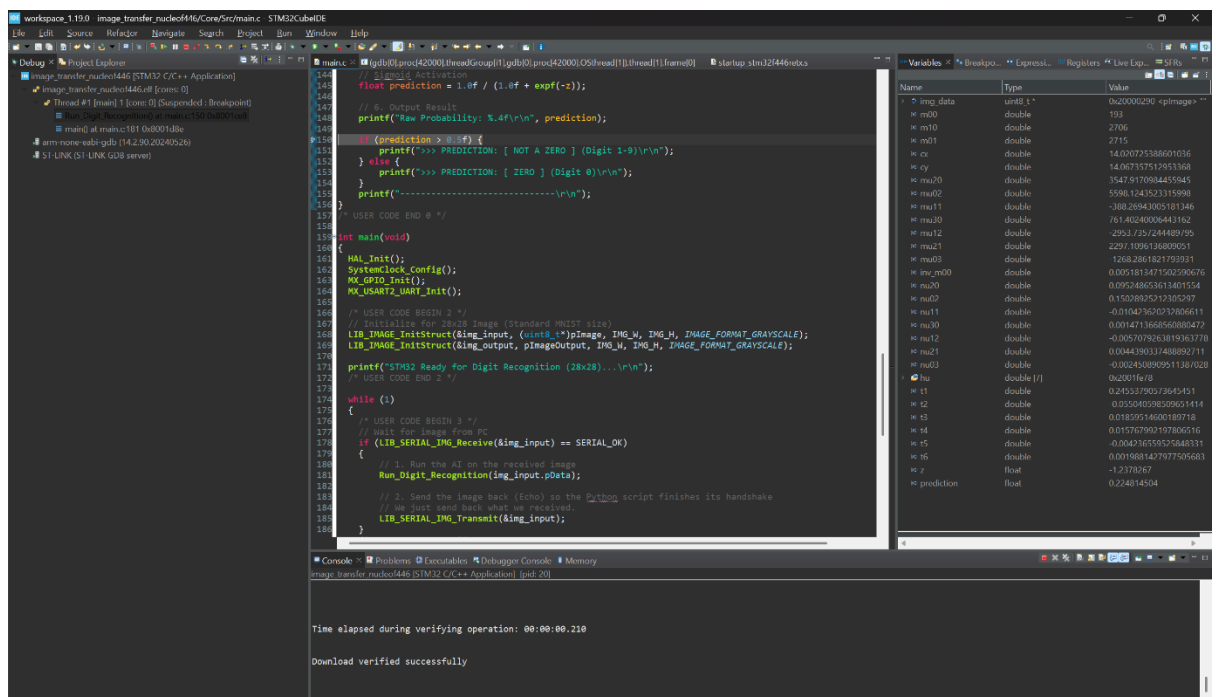
```
main.c:144 // sigmoid activation
float prediction = 1.0f / (1.0f + exp(-z));
// 6. Output Result
printf("New Probability: %.4f\r\n", prediction);
// 7. Prediction
if (prediction > 0.5f) {
    printf("PREDICTION: [ NOT A ZERO ] (Digit 1-9)\r\n");
} else {
    printf("PREDICTION: [ ZERO ] (Digit 0)\r\n");
}
printf("-----\r\n");
// USER CODE END 0 */
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    /* USER CODE BEGIN 0 */
    // Initialize for 28x28 Image (Standard MNIST size)
    LIB_IMAGE_InitStruct(&img_input, (uint8_t*)pImage, IMG_W, IMG_H, IMAGE_FORMAT_GRAYSCALE);
    LIB_IMAGE_InitStruct(&img_output, pImageOutput, IMG_W, IMG_H, IMAGE_FORMAT_GRAYSCALE);
    printf("STM32 Ready for Digit Recognition (28x28)...\r\n");
    /* USER CODE END 0 */
    while (1)
    {
        /* USER CODE BEGIN 3 */
        // Wait for image from PC
        if (LIB_SERIAL_IMG_Receive(&img_input) == SERIAL_OK)
        {
            // 1. Run the AI on the received image
            Run_Digit_Recognition(img_input.pData);
            // 2. Send the image back (Echo) so the Python script finishes its handshake
            // We just send back what we received
            LIB_SERIAL_IMG_Transmit(&img_input);
        }
    }
}
```

Name	Type	Value
img_data	uint8_t*	0x00000290 <pImage>
m00	double	116
m10	double	1664
m01	double	1682
cx	double	14.34407586206897
cy	double	145
mu20	double	1632.206865517237
mu02	double	4385
mu11	double	-261.00000000000006
mu20	double	-1395.2461355297165
mu12	double	-7020.0889655172455
mu21	double	-5022.00000000000036
mu03	double	12114
inv_mu00	double	0.0086206896551724137
mu20	double	0.12129956127762512
mu02	double	0.32587693227154336
mu11	double	-0.019366551724137935
mu30	double	-0.096273288064809124
mu12	double	-0.04848033851033797
mu21	double	-0.07395317972219435
mu03	double	0.08365777081338892
hu	double [1]	0x2011670
tx	double	0.471764932011685
ty	double	0.20407737004291824
tx	double	0.1358142656260047
ty	double	-0.19934372747004719
tx	double	-0.0581078644484712
ty	double	0.447762452041169457
z	float	39.514341
prediction	float	1

Time elapsed during verifying operation: 00:00:00.209

Download verified successfully

- Test Case B (Digit '0'): The model predicted a probability of 0.22, correctly classifying the image as "ZERO".



```
main.c:144 // sigmoid activation
float prediction = 1.0f / (1.0f + exp(-z));
// 6. Output Result
printf("New Probability: %.4f\r\n", prediction);
// 7. Prediction
if (prediction > 0.5f) {
    printf("PREDICTION: [ NOT A ZERO ] (Digit 1-9)\r\n");
} else {
    printf("PREDICTION: [ ZERO ] (Digit 0)\r\n");
}
printf("-----\r\n");
// USER CODE END 0 */
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    /* USER CODE BEGIN 0 */
    // Initialize for 28x28 Image (Standard MNIST size)
    LIB_IMAGE_InitStruct(&img_input, (uint8_t*)pImage, IMG_W, IMG_H, IMAGE_FORMAT_GRAYSCALE);
    LIB_IMAGE_InitStruct(&img_output, pImageOutput, IMG_W, IMG_H, IMAGE_FORMAT_GRAYSCALE);
    printf("STM32 Ready for Digit Recognition (28x28)...\r\n");
    /* USER CODE END 0 */
    while (1)
    {
        /* USER CODE BEGIN 3 */
        // Wait for image from PC
        if (LIB_SERIAL_IMG_Receive(&img_input) == SERIAL_OK)
        {
            // 1. Run the AI on the received image
            Run_Digit_Recognition(img_input.pData);
            // 2. Send the image back (Echo) so the Python script finishes its handshake
            // We just send back what we received
            LIB_SERIAL_IMG_Transmit(&img_input);
        }
    }
}
```

Name	Type	Value
img_data	uint8_t*	0x00000290 <pImage>
m00	double	193
m10	double	2706
m01	double	2715
cx	double	14.00072538091036
cy	double	14.067575119653368
mu20	double	3547.917084455945
mu02	double	5598.124323315998
mu11	double	-388.26943005181346
mu20	double	781.4024000443162
mu12	double	-2293.23244489399
mu21	double	2297.1096138800051
mu03	double	1268.2861821793931
inv_mu00	double	0.0051813471502290676
mu20	double	0.095248623613401554
mu02	double	0.150889291280597
mu11	double	-0.01043529237806611
mu12	double	-0.0057079263819363778
mu21	double	0.004439037488892711
mu03	double	-0.0024508909511387028
hu	double [1]	0x2011670
tx	double	0.24522792928645151
ty	double	0.055040298209651414
tx	double	0.01829514600180778
ty	double	0.015767902197806536
tx	double	-0.04076165955048181
ty	double	0.001968142977595683
z	float	-12.378267
prediction	float	0.224014504

Time elapsed during verifying operation: 00:00:00.210

Download verified successfully

Q2 Multi-Layer Neural Network (Section 11.8)

3.1 Methodology

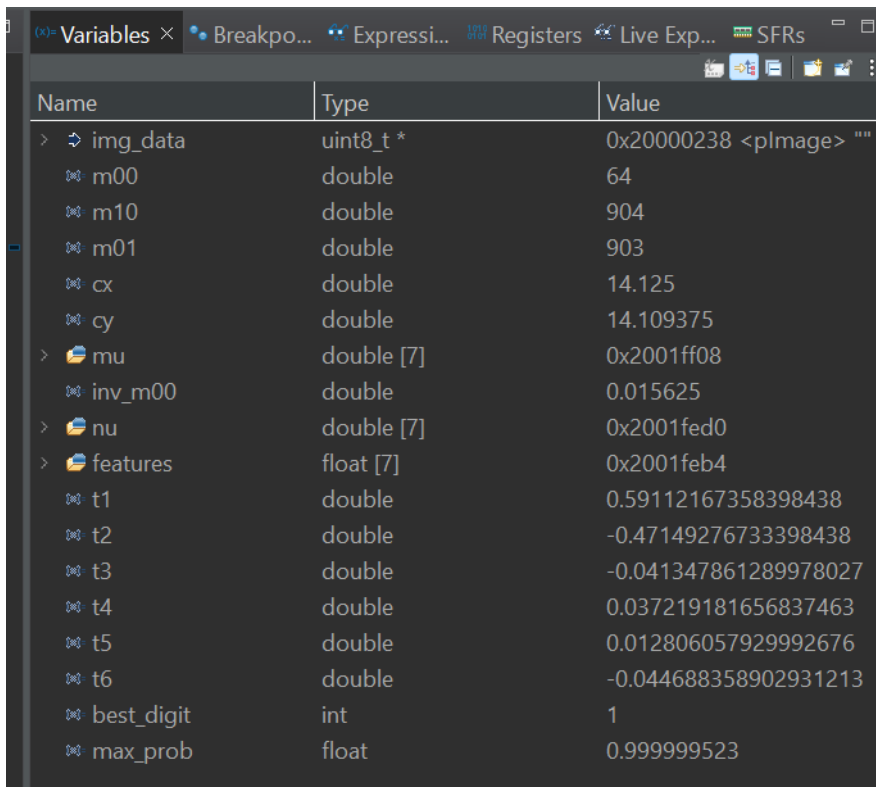
The goal was to classify digits 0-9 using a deep neural network.

- Architecture: A Multi-Layer Perceptron (MLP) was constructed with:
 - Input Layer: 7 neurons (Normalized Hu Moments).
 - Hidden Layer 1: 100 neurons (ReLU activation).
 - Hidden Layer 2: 100 neurons (ReLU activation).
 - Output Layer: 10 neurons (Softmax activation).
- Embedded Implementation: A general-purpose dense_layer function was implemented in C to handle matrix multiplication for layers of arbitrary size. The weights for all three layers were included via a generated header file (weights_data.h).

3.2 Results & Analysis

The model was tested with various digits. Distinct shapes were classified correctly, confirming the validity of the C implementation.

- Success Case: The digit '1' was correctly identified with high confidence.



Name	Type	Value
> ⇨ img_data	uint8_t *	0x20000238 <plImage> ""
m00	double	64
m10	double	904
m01	double	903
cx	double	14.125
cy	double	14.109375
> mu	double [7]	0x2001ff08
inv_m00	double	0.015625
> nu	double [7]	0x2001fed0
> features	float [7]	0x2001feb4
t1	double	0.59112167358398438
t2	double	-0.47149276733398438
t3	double	-0.041347861289978027
t4	double	0.037219181656837463
t5	double	0.012806057929992676
t6	double	-0.044688358902931213
best_digit	int	1
max_prob	float	0.999999523

Screenshots of test results for all of the digits can be found in the repository

3.3 Performance Analysis

While the code successfully executes the neural network logic, the model struggled to correctly distinguish certain digits. Specifically, digits '3' and '8' were frequently misclassified as '0', while the digit '5' was consistently misclassified as '7'.

Reasoning: These errors highlight the limitations of using Hu Moments as the sole feature extractor. Hu Moments describe global shape invariants—such as "roundness," "center of mass," and "skewness"—but they discard local topological details (like the number of holes or specific stroke angles).

- 3 & 8 identified as 0: Mathematically, '0', '3', and '8' share very similar mass distributions. They are all roughly symmetrical shapes where the pixel mass is distributed around a central empty space.
 - The Topology Problem: Hu Moments cannot "count" holes. They cannot distinguish between the single closed loop of a '0', the two stacked loops of an '8', or the open loops of a '3'. To the algorithm, all three appear as generic "round blobs" with similar rotational inertia.
 - Result: Since '0' is the simplest of these shapes, the network converges on '0' as the most likely prediction for all three.
- 5 identified as 7: This specific error arises from the "Top-Heavy" nature of both digits.
 - Mass Distribution: Both '5' and '7' are characterized by a strong horizontal bar at the top of the image.
 - Skewness: Unlike the symmetric '0' or '8', both '5' and '7' are asymmetrical. If the bottom loop of the handwritten '5' is open or angular, its overall center of gravity and skewness become statistically almost identical to a '7'.
 - Result: The feature vector for '5' overlaps significantly with '7', leading the model to predict the class it is more confident in (Class 7).

Conclusion

This homework successfully demonstrated the deployment of machine learning models onto a microcontroller.

1. Feature Extraction: Complex mathematical features (Hu Moments) were successfully implemented in pure C on the STM32F446RE.
2. Model Porting: Weights from a Python-trained model were successfully exported and used for inference in C.
3. Hardware-in-the-Loop: A robust testing framework was established using UART to transfer image data from a PC to the microcontroller for real-time verification.

The project highlights the trade-off in embedded ML: reducing dimensionality (using Hu Moments) lowers computational cost but introduces accuracy limitations for complex shapes.