Handwritten Digit Classification Using Raspberry Pi Pico and Machine Learning

***Abstract*—This paper details the creation of a machine learning-based digit classification system using the Raspberry Pi Pico. It integrates a Support Vector Machine (SVM) model with a Raspberry Pi Pico microcontroller, an OV7670 camera module, and a TFT LCD display. The project's focus was to develop a compact, efficient, and economical system capable of real-time digit recognition. Central to the project was adapting the SVM model to the Raspberry Pi Pico's limited resources and effectively preprocessing images from the OV7670 camera. The results demonstrate the viability of using low-cost microcontrollers in machine learning, particularly in resource-limited settings, and provide insights for educational and practical applications in image processing and pattern recognition. The project's success paves the way for future research in integrating machine learning models into similar constrained environments.**

***Index Terms*—Machine Learning, Raspberry Pi Pico, Digit Classification, Image Processing**

I. Introduction

In recent years, the intersection of machine learning and embedded systems has opened new avenues for technological innovation, particularly in the realm of image processing and digit recognition. Handwritten digit classification, a fundamental task in the field of machine learning, has extensive applications ranging from postal mail sorting to bank check processing. However, the implementation of such systems typically demands considerable computational resources and often relies on cloud computing or powerful processors. This study explores a novel approach to digit classification, utilizing the compact yet capable Raspberry Pi Pico.

The Raspberry Pi Pico, a microcontroller board based on the RP2040 microcontroller chip, offers a balance of affordability and computational ability, making it an ideal candidate for deploying machine learning models in a constrained environment. The core of this project revolves around using the Raspberry Pi Pico in conjunction with an OV7670 camera module and a 128x160 TFT LCD display to create a self-contained, efficient digit classification system.

This paper presents a comprehensive exploration of building a machine learning model suitable for the Raspberry Pi Pico's hardware limitations. The project underscores the challenges and solutions in miniaturizing machine learning models to fit within the constrained memory and processing power of microcontrollers. By leveraging CircuitPython, an accessible and versatile programming environment, we navigate the limitations inherent in microcontroller-based systems while harnessing their potential for educational and practical applications in the field of machine learning.

The project aims to demonstrate not just the technical feasibility of such an endeavor, but also its educational value, providing insights into the practical aspects of deploying machine learning models in resource-constrained environments. This work contributes to the growing body of knowledge in embedded machine learning, presenting a case study that showcases the potential of microcontrollers like the Raspberry Pi Pico in the realm of digital image processing and classification.

II. Methodology

A. Hardware Requirements

The implementation of the handwritten digit classification system using Raspberry Pi Pico and machine learning involves an assemblage of specific hardware components. Each component plays a crucial role in the functionality and efficiency of the system. The following are the key hardware components used in this project:

1) Raspberry Pi Pico: The Raspberry Pi Pico forms the core of this project. It is a microcontroller board built around the RP2040 microcontroller chip, known for its balance of performance and cost-effectiveness. The choice of Raspberry Pi Pico is driven by its compatibility with Python, ease of use, and sufficient computational power to handle machine learning tasks in a constrained environment.

2) OV7670 Camera Module: An essential component of the setup is the OV7670 camera module. This low-cost image sensor is capable of capturing 640x480 pixel images and is widely used in simple image processing projects. For this project, it is instrumental in capturing handwritten digits for classification.

3) 128x160 TFT LCD Display: The 128x160 TFT LCD display is used to output the results of the digit classification process. This display is chosen for its compact size, clarity, and ease of interfacing with the Raspberry Pi Pico. It plays a crucial role in providing real-time feedback and visual output of the system.

4) Breadboard and Jumper Cables: A full-sized breadboard and a set of jumper cables (male-to-female and male-to-male) are used to create the circuit connections. These facilitate a modular and non-permanent setup, allowing for ease of experimentation and adjustments.

5) Additional Components: Several additional components, such as power supplies, resistors, and connectors, are also necessary to ensure the smooth functioning of the system. These components are used to provide stable power, make secure connections, and ensure the overall integrity of the experimental setup.

Each of these components is integral to the project, ensuring that the system operates efficiently and effectively. The chosen hardware setup strikes a balance between cost, availability, and performance, making it suitable for educational purposes and hobbyist projects in the realm of embedded machine learning.

B. Software Requirements

The successful implementation of the handwritten digit classification system using the Raspberry Pi Pico requires specific software components and tools. These software elements facilitate programming the microcontroller, processing the image data, and implementing the machine learning model. The following are the key software requirements for this project:

1) CircuitPython: CircuitPython is a version of Python designed to simplify experimenting with and learning electronics. It is used as the primary programming environment for the Raspberry Pi Pico in this project. CircuitPython was chosen for its ease of use, readability, and strong community support. Its ability to interact seamlessly with hardware components makes it an ideal choice for microcontroller-based projects.

2) Python and Libraries for Machine Learning: For the development of the machine learning model, a full Python distribution is utilized. Key Python libraries used in this project include:

- **NumPy** - For numerical computing and handling arrays.
- **Scikit-learn** - This library is used for implementing the machine learning algorithm, particularly the Support Vector Machine (SVM) model for digit classification.

3) Machine Learning Model Conversion Tools: The project also utilizes specific tools for converting the trained machine learning model into a format compatible with CircuitPython running on the Raspberry Pi Pico. These tools include:

- **m2cgen** - A tool used to convert the Scikit-learn model into a form that can be executed on the Pi Pico.
- **Python-minimizer** - This tool helps in reducing the size of the Python code, ensuring that it fits within the limited memory resources of the Pi Pico.

4) Text Editor: Any text editor is suitable for writing and editing the Python code. Examples include Visual Studio Code, Atom, or even simpler editors like Notepad++.

5) Additional Software: Other necessary software components include:

- **Git** - For version control and managing updates to the code.
- **Serial Console** - For debugging and monitoring the output of the Raspberry Pi Pico.

Together, these software components form the backbone of the project, enabling the development, implementation, and operation of the machine learning model on the Raspberry Pi Pico. The selection of these specific tools and libraries is driven by their compatibility, ease of use, and efficiency in a microcontroller-based environment.

C. System Design

The design of the handwritten digit classification system using the Raspberry Pi Pico is a critical aspect of this project. It involves the integration of hardware components and the implementation of the software to create a cohesive and functional unit. The following subsections detail the system design, including the wiring setup and the software architecture.

1) Hardware Integration and Wiring: The Raspberry Pi Pico serves as the central processing unit of the system. It is connected to the OV7670 camera module and the 128x160 TFT LCD display. The connections are established as follows:

- The OV7670 camera module is connected to the Pi Pico via GPIO pins, enabling the Pico to capture images. The specific pin connections are designed to facilitate data transfer and control signals between the camera module and the Pico.
- The 128x160 TFT LCD display is interfaced with the Pi Pico to provide real-time visual feedback. It is connected via SPI (Serial Peripheral Interface) for efficient communication.
- Power supply connections are made to ensure stable operation, with considerations for the voltage requirements of each component.

This setup requires careful planning of the GPIO pin usage, as a significant number of pins are utilized for interfacing with the camera and display.

2) Software Architecture: The software architecture is built around CircuitPython, which runs on the Raspberry Pi Pico. The key components of the software architecture include:

- Image Capture and Processing: The software is designed to capture images using the OV7670 camera module, process these images (including converting them to grayscale), and prepare them for digit classification.
- Machine Learning Model Implementation: The SVM model, developed and trained using Scikit-learn, is converted into a CircuitPython-compatible format. This model is then integrated into the software running on the Pi Pico.
- User Interface and Feedback: The TFT LCD display is used to show the results of the digit classification process, providing an interactive user experience.

The software is structured to ensure efficient execution of these tasks within the memory and processing constraints of the Raspberry Pi Pico.

3) Circuit Design: The circuit design is meticulously laid out on a breadboard, with connections made using jumper cables. This design allows for easy modification and troubleshooting. The circuit includes:

- Pull-up resistors where necessary, especially in the I2C communication lines.
- Power distribution lines to supply the appropriate voltage to each component.

The design aims to be compact yet accessible for educational purposes, demonstrating the principles of embedded system design and machine learning implementation.

This system design encapsulates the integration of hardware and software components, forming the backbone of the digit classification project. The design considerations taken ensure a balance between functionality, efficiency, and educational value.

III. Implementation

A. Data Preprocessing

Data preprocessing is a critical step in the handwritten digit classification system, particularly in ensuring that the captured images are suitable for analysis by the machine learning model. The Raspberry Pi Pico, in conjunction with the OV7670 camera module, performs several key preprocessing tasks to convert raw image data into a format that can be efficiently processed. The following subsections detail the data preprocessing steps.

1) Capturing Images with the OV7670 Camera Module: The OV7670 camera module captures images at a resolution of 60x80 pixels. For this project, the camera is configured to use the RGB565_SWAPPED format, which is a common color format for image processing. The camera's configuration and initialization are handled by the software running on the Raspberry Pi Pico.

2) Image Format and Pixel Encoding: Each pixel captured by the OV7670 in the RGB565_SWAPPED format is encoded as a 16-bit integer. This encoding represents the red, green, and blue components of each pixel with 5, 6, and 5 bits respectively. The initial step in preprocessing involves 'unswapping' the pixel values. This is achieved through the following operation:

```
pixel_val = ((pixel_val & 0xFF00) >> 8) |
            ((pixel_val & 0x00FF) << 8)
```

After unswapping, the red, green, and blue components are extracted and used for further processing.

3) Converting to Grayscale: The machine learning model used in this project requires grayscale images for classification. Thus, the RGB images are converted to grayscale. The grayscale value is computed using a weighted sum of the red, green, and blue components. The conversion is performed using the following formula:

```
grayscale = 0.299 * r + 0.587 * g + 0.114 * b
```

4) Image Resizing and Cropping: The captured images are resized and cropped to meet the input requirements of the machine learning model. The original images of 60x80 pixels are first cropped to a square format (60x60) to maintain aspect ratio. These images are then resized to a 12x12 pixel format, which is the input size expected by the machine learning model.

5) Noise Reduction and Thresholding: During testing, it was observed that the captured images could contain considerable noise. To enhance the quality of the images for better digit recognition, a thresholding function is applied. This function helps in reducing noise and improving the clarity of the digits. The thresholding process involves setting a pixel value to zero if it is below a certain threshold, enhancing the distinction between the digit and the background.

Through these preprocessing steps, the images captured by the OV7670 camera module are effectively transformed into a suitable format for classification by the machine learning model. This preprocessing not only optimizes the image data for better recognition accuracy but also ensures compatibility with the limited processing capabilities of the Raspberry Pi Pico.

B. Machine Learning Model Training

The training of the machine learning model is a pivotal part of the handwritten digit classification system, involving the use of a Support Vector Machine (SVM) algorithm and the MNIST dataset. This section outlines the steps taken to train the model and prepare it for deployment on the Raspberry Pi Pico.

1) Choice of Machine Learning Algorithm: For this project, the SVM algorithm was chosen due to its effectiveness in high-dimensional data classification. SVMs are particularly well-suited for binary classification tasks and have proven to be robust in image classification problems. The LinearSVC implementation from the Scikit-learn library was used to create the SVM model.

2) Training Data: The MNIST Dataset: The MNIST dataset, a large database of handwritten digits commonly used in training and testing in the field of machine learning, was utilized as the training data for the model. This dataset contains 60,000 training images and 10,000 testing images, each of size 28x28 pixels, labeled with the corresponding digit they represent.

3) Preprocessing of Training Data: Before training, the images from the MNIST dataset were preprocessed to match the input format expected by the SVM model. This preprocessing involved:

- Resizing each image from 28x28 to 12x12 pixels to align with the input size used by the model deployed on the Pi Pico.
- Flattening the 2D image data into 1D arrays, as the SVM algorithm requires input data in a single vector format.

4) Model Training and Validation: The LinearSVC model was trained using the preprocessed images. The training process involved:

- Splitting the MNIST dataset into training and testing subsets, ensuring a representative distribution of data.
- Training the LinearSVC model on the training subset and validating its performance on the testing subset to evaluate its accuracy and effectiveness.

5) Model Evaluation: Post-training, the model was evaluated using standard metrics like accuracy, precision, and recall. The performance on the test dataset provided insights into the model's generalization capabilities and its effectiveness in classifying unseen data.

6) Exporting the Model: Once trained and evaluated, the model was exported into a format compatible with the CircuitPython environment of the Raspberry Pi Pico. This conversion process ensured that the SVM model, originally trained in a high-level Python environment, was translatable and operational within the resource-constrained environment of the microcontroller.

Through these steps, the machine learning model was trained, evaluated, and prepared for deployment, ensuring its readiness for real-time digit classification on the Raspberry Pi Pico platform.

IV. Implementation Details

The implementation of the handwritten digit classification system is a critical phase of the project, involving the integration of hardware components with the software. This section provides a detailed overview of the system's setup, including the initialization of the TFT LCD display and the OV7670 camera module, image preprocessing, and real-time digit classification.

A. System Setup and Configuration

The system's core is the Raspberry Pi Pico, which interfaces with the OV7670 camera module for image capture and a TFT LCD display for output. The following Python code demonstrates the setup and configuration process:

```
import gc
import sys
from time import sleep

import bitmaptools
import board
import busio
import digitalio
import displayio
import svm_min
import terminalio
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text import label
from adafruit_ov7670 import OV7670
from adafruit_st7735r import ST7735R

# Function to convert RGB565_SWAPPED
#to grayscale
def rgb565_to_1bit(pixel_val):
    pixel_val = ((pixel_val & 0x00FF)<<8) |
                ((25889 & 0xFF00) >> 8)
    r = (pixel_val & 0xF800)>>11
    g = (pixel_val & 0x7E0)>>5
    b = pixel_val & 0x1F
    return (r+g+b)/128


#Setting up the TFT LCD display
mosi_pin = board.GP11
clk_pin = board.GP10
reset_pin = board.GP17
cs_pin = board.GP18
dc_pin = board.GP16

displayio.release_displays()
spi = busio.SPI(clock=clk_pin, MOSI=mosi_pin)
display_bus = displayio.FourWire(
    spi, command=dc_pin, chip_select=cs_pin,
                        reset=reset_pin)

display = ST7735R(display_bus, width=128,
                  height=160, bgr=True)
group = displayio.Group( scale=1)
display.root_group = group

font = bitmap_font.load_font("./arial.bdf")
color = 0xffffff
text_area = label.Label(font, text='',color=color)
text_area.x = 10
text_area.y = 140
group.append(text_area)

cam_width = 80
cam_height = 60
cam_size = 3 #80x60 resolution

camera_image = displayio.Bitmap(cam_width,
                  cam_height, 65536)
camera_image_tile = displayio.TileGrid(
    camera_image ,
    pixel_shader=displayio.ColorConverter(
        input_colorspace=
        displayio.Colorspace.RGB565_SWAPPED),
    x=30,
    y=30,
)


group.append(camera_image_tile)
camera_image_tile.transpose_xy=True

inference_image = displayio.Bitmap(12,12,
                                  65536)

#Setting up the camera
cam_bus = busio.I2C(board.GP21, board.GP20)

cam = OV7670(
    cam_bus,
    data_pins=[
        board.GP0,
        board.GP1,
        board.GP2,
        board.GP3,
        board.GP4,
        board.GP5,
        board.GP6,
        board.GP7,
    ],
    clock=board.GP8,
    vsync=board.GP13,
    href=board.GP12,
    mclk=board.GP9,
    shutdown=board.GP15,
    reset=board.GP14,
)
cam.size =  cam_size
cam.flip_y = True

ctr = 0
while True:
    cam.capture(camera_image)
    sleep(0.1)
    temp_bmp = displayio.Bitmap(cam_height,
                        cam_height, 65536)
    for i in range(0,cam_height):
        for j in range(0,cam_height):
            temp_bmp[i,j]=camera_image[i,j]
    bitmaptools.rotozoom(inference_image,
        temp_bmp,scale=12/cam_height,
        ox=0,oy=0,px=0,py=0)
    del(temp_bmp)

    input_data = []
    for i in range(0,12):
        for j in range(0,12):
            gray_pixel = 1 -rgb565_to_1bit
                    (inference_image[i,j])
            if gray_pixel < 0.5:
                gray_pixel = 0
            input_data.append(gray_pixel)

    camera_image.dirty()
    display.refresh(minimum_frames
                _per_second=0)
    prediction = svm_min.score(input_data)
    #Uncomment these lines for debugging
    ctr = ctr + 1
    if ctr%50 == 0:
        print(input_data)
        print("------")
    res = prediction.index(max(prediction))
    #print(res)
    text_area.text = "Prediction : " +str(res)
    sleep(0.01)
```

This code snippet sets up the SPI bus and the TFT LCD display, initializing the necessary communication protocols and graphical interfaces for the project.

B. Image Preprocessing and Digit Classification

The captured images undergo preprocessing to convert them from the RGB565_SWAPPED format to grayscale, which is the required input format for the SVM model. The process includes resizing the images for compatibility with the model's expected input size. Here's the relevant section of the code:

```
# Function to convert RGB565_SWAPPED
to grayscale
def rgb565_to_1bit(pixel_val):
    pixel_val = ((pixel_val & 0x00FF)<<8) |
                ((25889 & 0xFF00) >> 8)
    r = (pixel_val & 0xF800)>>11
    g = (pixel_val & 0x7E0)>>5
    b = pixel_val & 0x1F
    return (r+g+b)/128
```

This function is crucial for converting the image data into a format that can be effectively analyzed by the machine learning model.

C. Real-time Digit Classification and Display

The system is designed to perform real-time digit classification. Once the images are preprocessed, they are fed into the SVM model for digit classification. The predicted digit is then displayed on the TFT LCD screen. The loop in the code continuously captures images, processes them, and updates the display with the classification results:

```
while True:
    cam.capture(camera_image)
    sleep(0.1)
    temp_bmp = displayio.Bitmap(cam_height,
                        cam_height, 65536)
    for i in range(0,cam_height):
        for j in range(0,cam_height):
            temp_bmp[i,j]=camera_image[i,j]
    bitmaptools.rotozoom(inference_image,
        temp_bmp,scale=12/cam_height,
        ox=0,oy=0,px=0,py=0)
    del(temp_bmp)

    input_data = []
    for i in range(0,12):
        for j in range(0,12):
            gray_pixel = 1 -rgb565_to_1bit
                    (inference_image[i,j])
            if gray_pixel < 0.5:
                gray_pixel = 0
            input_data.append(gray_pixel)

    camera_image.dirty()
    display.refresh(minimum_frames
                _per_second=0)
    prediction = svm_min.score(input_data)
    #Uncomment these lines for debugging
    ctr = ctr + 1
    if ctr%50 == 0:
        print(input_data)
        print("------")
    res = prediction.index(max(prediction))
    #print(res)
    text_area.text="Prediction : "+str(res)
    sleep(0.01)
```

This loop represents the real-time operational capability of the system, showcasing the integration of image capture, processing, and output display.

D. Code Execution and Performance Monitoring

Throughout the execution, the system's performance is monitored to ensure efficient processing and accurate digit classification. The code includes provisions for debugging and performance analysis, allowing for a deeper understanding of the system's operation under various conditions.

This implementation showcases the practical application of machine learning in an embedded system, demonstrating the feasibility of real-time image processing and classification using the Raspberry Pi Pico.

E. Exporting the Model

The final stage in preparing the machine learning model for deployment on the Raspberry Pi Pico involves exporting the trained model into a format that is compatible with CircuitPython. This process is crucial for integrating the model with the microcontroller environment. The following steps detail the model exporting procedure.

1) Model Conversion: The Support Vector Machine (SVM) model, initially trained in a standard Python environment using the Scikit-learn library, needs to be converted into a format that the Raspberry Pi Pico can interpret and execute. For this purpose, the m2cgen (Model to Code Generator) tool is utilized. This tool translates the trained SVM model into a pure Python code representation, making it compatible with the CircuitPython environment.

2) Code Minimization: Given the limited memory resources of the Raspberry Pi Pico, it is essential to minimize the size of the generated Python code. The python-minimizer tool is employed for this purpose. It optimizes the Python code, reducing its size while retaining the functional integrity of the model. This minimization ensures that the model fits within the memory constraints of the Pi Pico without compromising its performance.

3) Testing the Converted Model: After conversion and minimization, the model is tested in a CircuitPython environment to ensure its operational integrity. This testing involves:

- Loading the minimized model onto the Raspberry Pi Pico.
- Running inference tests using sample digit images to verify the model's prediction accuracy.
- Monitoring the resource usage to ensure that the model operates within the Pi Pico's computational constraints.

4) Integration with the Main System: Once the model has been successfully converted, minimized, and tested, it is integrated into the main system. This integration involves embedding the model code within the broader CircuitPython application that controls the Raspberry Pi Pico, ensuring seamless interaction between the image capture, preprocessing, and digit classification functionalities.

Through this exporting process, the SVM model is effectively transformed into a lean and efficient version suitable for deployment on the Raspberry Pi Pico. This step marks the transition of the project from a machine learning development environment to a real-world, embedded system application.

V. Results and Discussion

This section presents the outcomes of the handwritten digit classification project and discusses the key findings, challenges encountered, and the implications of the results.

1) Model Performance: The performance of the machine learning model, post-export and integration into the Raspberry Pi Pico, showed promising results. The Support Vector Machine (SVM) model demonstrated a high degree of accuracy in classifying handwritten digits. Key performance metrics included:

- **Accuracy**: The percentage of correctly classified digits out of the total number of images tested.
- **Precision and Recall**: These metrics provided insights into the model's ability to correctly identify each digit while minimizing false positives.

2) System Functionality: The overall system, comprising the Raspberry Pi Pico, the OV7670 camera module, and the TFT LCD display, operated effectively. The system was able to capture images, preprocess them, and display the classification results in real-time. The integration of hardware and software components proved to be robust and reliable for the intended application.

3) Challenges Encountered: Several challenges were encountered during the project:

- **Memory Constraints**: The Raspberry Pi Pico has limited memory, which necessitated careful optimization of the machine learning model and the CircuitPython code.
- **Image Quality**: The quality of images captured by the OV7670 camera module affected the model's performance. Achieving the right balance between image resolution and processing requirements was crucial.
- **Real-time Processing**: Ensuring that the system could process images and display results in real-time presented challenges in terms of computational efficiency and speed.

4) Educational and Practical Implications: The project demonstrated the feasibility of implementing a machine learning model on a low-cost, resource-constrained platform like the Raspberry Pi Pico. This has significant educational implications, offering a practical example of embedded machine learning that can be used in academic settings. Additionally, the project showcases the potential for developing cost-effective, real-world applications in the field of image processing and digit recognition.

5) Future Work: Future enhancements to this project could include:

- **Model Optimization**: Further refining the machine learning model to improve accuracy and efficiency.
- **Hardware Upgrades**: Exploring the use of more advanced camera modules or displays to enhance system capabilities.
- **Extended Applications**: Expanding the system's functionality to recognize more complex patterns or characters.

In conclusion, the project successfully demonstrated the application of machine learning in an embedded system context, highlighting both the potential and the challenges of such endeavors. The results are encouraging and open up avenues for further research and development in the field.

VI. Conclusion

The project "Handwritten Digit Classification Using Raspberry Pi Pico and Machine Learning" successfully demonstrates the integration of a machine learning model with an embedded system for the purpose of digit recognition. This endeavor highlights the feasibility of deploying sophisticated computational models on low-cost, resource-constrained hardware like the Raspberry Pi Pico.

Through the utilization of the Raspberry Pi Pico, along with the OV7670 camera module and a TFT LCD display, this project showcases an effective implementation of real-time digit classification. The successful adaptation of a Support Vector Machine (SVM) model for this purpose, optimized to fit within the Pi Pico's limited memory and processing capabilities, stands as a significant achievement in the realm of embedded machine learning.

Key accomplishments of the project include:

- Demonstrating the practical application of machine learning on a microcontroller.
- Achieving a balance between system performance and resource constraints.
- Providing a valuable educational tool for understanding embedded systems and machine learning integration.

Despite facing challenges related to memory constraints and real-time processing requirements, the project outcomes reinforce the potential of microcontrollers in machine learning applications. The experience gained from overcoming these challenges contributes to the growing knowledge base in the field, offering insights into optimization techniques and system design for constrained environments.

Future enhancements to this project could focus on refining the model for greater accuracy, exploring more advanced hardware components, and expanding the system's capabilities to recognize a wider array of patterns or characters.

In conclusion, this project not only serves as a proof of concept for embedding machine learning models in microcontroller-based systems but also paves the way for future innovations in the field of low-cost, efficient image processing and pattern recognition technologies.