Handwritten Digit,Alphabets and Shapes Classification Using Raspberry Pi Pico and Machine Learning

Abstract—**In recent years, the convergence of machine learning and embedded systems has paved the way for groundbreaking technological advancements, particularly in the domain of image processing and recognition of diverse elements such as shapes and alphabets. Handwritten character classification, a foundational task in the realm of machine learning, holds immense significance across various applications, spanning from postal mail sorting to bank check processing. Traditional implementations of these systems often require substantial computational resources, frequently relying on cloud computing or powerful processors. This study delves into an innovative approach to character classification, leveraging the capabilities of the compact yet powerful Raspberry Pi Pico. Beyond digit recognition, the research extends its focus to encompass shapes and alphabets, exploring their classification using the Raspberry Pi Pico. This versatile methodology opens up new possibilities for applications such as shape-based object identification and handwritten alphabet analysis, showcasing the potential of embedded systems in diverse and resource-efficient character recognition tasks.**

I. Introduction

The RP2040 microcontroller-based Raspberry Pi Pico stands out for its cost-effectiveness and computational capabilities, making it an optimal choice for implementing machine learning models in resource-constrained environments. This project revolves around the integration of the Raspberry Pi Pico with an OV7670 camera module and a 128x160 TFT LCD display to create an independent and efficient system for classifying digits.

This research paper provides an in-depth exploration of developing a machine learning model tailored to the hardware limitations of the Raspberry Pi Pico. The project highlights the intricacies of scaling down machine learning models to fit the constrained memory and processing power of microcontrollers. Through the utilisation of CircuitPython, a versatile programming environment, we address the challenges posed by microcontroller-based systems, maximising their potential for both educational and practical applications in the realm of machine learning.

The primary objective of this project is not only to showcase the technical feasibility of such an undertaking but also to emphasise its educational significance by offering insights into the practical aspects of deploying machine learning models in environments with limited resources. This work contributes to the evolving knowledge base in embedded machine learning, presenting a compelling case study that illustrates the capabilities of microcontrollers like the Raspberry Pi Pico in the domain of digital image processing and classification.built-in; examples of the type styles are provided throughout this document and are identified in italic type, within parentheses, following the example. Some components, such as multi-levered equations, graphics, and tables are not prescribed, although the various table text styles are provided. The formatter will need to create these components, incorporating the applicable criteria that follow.

II. METHODOLGY

A. Hardware Requirements

1) Raspberry Pi Pico: The core microcontroller that drives the project. The Raspberry Pi Pico serves as the main processing unit, executing machine learning algorithms and managing the integration of input from the camera module.

2) OV7670 Camera Module: Captures visual data for digit recognition. The OV7670 camera module is

essential for collecting image data, enabling the machine learning model to analyse and classify handwritten digits. It provides the visual input necessary for the digit classification system.
3) 128x160 TFT LCD Display: Visual output for displaying results and feedback.The TFT LCD display is crucial for showcasing the results of the digit classification process. It allows users to observe the system's predictions or any relevant feedback, enhancing the project's usability and practicality.
4) Breadboard and Jumper Cables: Facilitates connections and prototyping. The breadboard and jumper cables are essential for creating a temporary and flexible circuit. They enable easy and adjustable connections between the Raspberry Pi Pico, camera module, LCD display, and additional components during the prototyping and development phases.
5) *Additional Components:* Several additional components, such as power supplies, resistors, and connectors These additional components contribute to the overall functionality, reliability, and ease of use in your digit classification system based on the Raspberry Pi Pico. They enable proper power distribution, signal conditioning, and interconnection between different parts of the system.

B. Software Requirements

The software requirements for classification system based on the Raspberry Pi Pico and machine learning include:

1) CircuitPython: Provides an accessible and versatile programming environment for microcontrollers. CircuitPython simplifies the development process, offering a Python-based platform tailored for microcontrollers like the Raspberry Pi Pico. It enhances ease of programming and facilitates interaction with various components.
2) Machine Learning Library: Efficient numerical operations and array manipulation in Python. NumPy is crucial for handling numerical data, especially matrices and arrays, which are fundamental in machine learning tasks. It provides essential functionality for numerical computations that are at the core of many machine learning algorithms.Machine learning library for various algorithms and tools. scikit-learn is a versatile library that offers a wide range of machine learning algorithms and tools for tasks such as classification, regression, clustering, and more. It simplifies the implementation of machine learning models and provides tools for data preprocessing, model evaluation, and feature selection.
3) OV7670 Camera Module Driver: Interfaces with the OV7670 camera module. To capture and process visual data, you'll need a driver that facilitates communication between the Raspberry Pi Pico and the OV7670 camera module. This ensures proper integration and functioning of the camera in the system.
4) TFT LCD Display Library: Enables communication with the 128x160 TFT LCD display. A dedicated library for the TFT LCD display is necessary to control and update the visual output. This library ensures seamless integration with the Raspberry Pi Pico, allowing you to display digit classification results on the screen.
5)Integrated Development Environment: Facilitates code development, debugging, and uploading. Choose a suitable IDE for Python or CircuitPython that supports microcontroller development. Examples include Thonny, Mu, or VSCode with appropriate extensions for microcontroller programming.
5) Libraries for Additional Components: Converts machine learning models into native code (Python, C, etc.). m2cgen is used to convert machine learning models trained in Python (e.g., scikit-learn models) into a form that can be executed on the Raspberry Pi Pico.

These software requirements form the foundation for developing, implementing, and running the digit classification system on the Raspberry Pi Pico, integrating machine learning algorithms, camera input, and visual output.

C. System Design

• Hardware Integration and Wiring:

1) Raspberry Pi Pico Connections: Connect the Raspberry Pi Pico to the breadboard using jumper cables. Ensure proper power supply to the Raspberry Pi Pico, connecting it to the power rail on the breadboard. Establish ground connections between the Raspberry Pi Pico, camera module, and TFT LCD display.
2) OV7670 Camera Module Integration: Connect the OV7670 camera module to the Raspberry Pi Pico using compatible pins. Provide the necessary power supply and ground connections to the camera module.Check and set the I2C or SPI communication protocol depending on the compatibility with the Raspberry Pi Pico.
3) 128x160 TFT LCD Display Connection: Connect the TFT LCD display to the Raspberry Pi Pico using designated pins. Ensure a stable power supply and ground connections for the TFT LCD display. Verify the communication protocol (SPI, I2C) and configure it accordingly.
4) Power Supply and Distribution: Connect the main power source to the power rail on the breadboard. Distribute power to the Raspberry Pi Pico, camera module, and TFT LCD display using appropriate voltage levels. Use resistors or voltage regulators as needed to maintain stable power across components.
5) System Grounding: Establish a common ground for all components to ensure a reference point for electrical signals. Avoid ground loops by connecting all ground points to a single ground reference.

By detailing the hardware integration and wiring in this section, you provide a clear roadmap for assembling and connecting the various components of your digit classification system based on the Raspberry Pi Pico.

• Software Architecture:

1) Machine Learning Model Development: Develop a machine learning model for classification using NumPy and scikit-learn, specifically utilizing an SVM model.

- Data Collection: Gather a dataset of handwritten digits for training and testing.
- Data Preprocessing: Prepare the dataset by normalising and flattening images.
- Model Training: Train the SVM machine learning model using scikit-learn's algorithms.
- Model Evaluation: Assess the model's accuracy and performance.

2) Model Conversion and Optimization: Convert the trained SVM machine learning model for compatibility with the Raspberry Pi Pico.

- m2cgen Integration: Utilise m2cgen to convert the SVM model into native code (e.g., Python).
- Code Optimization: Use pythonminimizer to minimise and optimise the generated code for microcontroller deployment.

3) Raspberry Pi Pico Integration: Integrate the optimised SVM machine learning model with the Raspberry Pi Pico.

- CircuitPython Setup: Configure the Raspberry Pi Pico to run CircuitPython.
- Driver Integration: Implement drivers for the OV7670 camera module and the TFT LCD display.
- Code Upload: Upload the optimised SVM machine learning model code and associated scripts to the Raspberry Pi Pico.

4) Real-time Digit Classification: Enable real-time digit classification using the integrated system.

- Camera Input: Capture live video feed from the OV7670 camera module.
- Image Processing: Preprocess the captured images to match the SVM model's input requirements.
- Digit Classification: Utilise the optimised SVM machine learning model to classify handwritten digits.
- Display Output: Showcase classification results on the 128x160 TFT LCD display in real-time.

By outlining the software architecture in this section, you articulate the key steps and interactions involved in your digit classification system. This provides a roadmap for both development and future enhancements.

III. Implementation

A. Data Preprocessing

1) Capturing Images with the Camera Module: To capture images using the OV7670 Camera Module, set the camera resolution to 60x80 pixels. The camera can capture images in RGB565_SWAPPED format, meaning each pixel is represented as a 16-bit integer with a swapped byte order. It is essential to unswap the pixels using the formula:

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

This step ensures correct interpretation of the RGB565_SWAPPED format.

2) Image Format and Pixel Encoding: The image format is RGB565_SWAPPED, where each pixel is encoded as a 16-bit integer. The red component is represented by 5 bits, the green component by 6 bits, and the blue component by 5 bits. To extract individual components, use the following code:

```
#Copy coder = (pixel_val & 0xF800) >> 11
#             g = (pixel_val & 0x7E0) >> 5
#             b = pixel_val & 0x1F
```

3)Converting to Grayscale: Convert the RGB components of each pixel to grayscale using a simple average method:

```
#Copy codegrayscale_value = (r + g + b) // 128
```

This ensures that the image is represented in a single-channel grayscale format, suitable for the model's expectations.

4) image Resizing and Cropping:

The captured image is initially 60x80 pixels, but the model expects input images of size 12x12. To preprocess the image, first, create a temporary bitmap of the original size and copy the image onto it. Then, use the rotozoom function to resize the image to 12x12 pixels while maintaining the aspect ratio.

5) Noise Reduction and Thresholding:

Implement a thresholding function to reduce noise in the image. Iterate through each pixel in the 12x12 image and convert the RGB565 value to a 1-bit value using rgb565_to_1bit. Apply a threshold, setting values below 0.5 to 0 and values above or equal to 0.5 to 1.

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. 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. Same model used for all three classifications.

2) Training Data:

- *Digit classification:* This dataset contains 60,000 training images and 10,000 testing images, each of size 28x28 pixels, labeled with the corresponding digit they represent.

- Alphabet *classification: Here only first 10 alphabets are considered*. This dataset contains 4000 training images and 1000 testing images, each of size 28x28 pixels, labeled with the corresponding alphabet they represent.

- Shapes *classification: Here only first 3 shapes(Circles, Square, Triangle) are considered*. This dataset contains 4000 training images and 1000 testing images, each of size 28x28 pixels, labeled with the corresponding shape they represent.

3) Preprocessing of Training Data: Before training, the images from the datasets 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 generalisation 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 Circuit- Python 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 initialisation of the TFT LCD display and the OV7670 camera module, image preprocessing, and real-time digit classification.

Code:

```
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.show(group)

font = bitmap_font.load_font("./Helvetica-
Bold-16.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_SW
APPED
    ),
    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)
    alphabet
={0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J'}
    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))
    val = alphabet[res]
    text_area.text = "Prediction : " + str(res)
    sleep(0.01)
```

1) System Setup and Configuration:

- Display Configuration: The ST7735R TFT LCD display is configured with the necessary pins (mosi_pin, clk_pin, reset_pin, cs_pin, dc_pin). Display group and text area for prediction output are set up. A 128x160 resolution display is initialized with a specific color format (BGR).
- Camera Configuration: OV7670 camera module is set up with I2C and GPIO pins. Camera parameters, including data pins, clock, vsync, href, mclk, shutdown, and reset, are configured. The camera size is set to 80x60 pixels, and flip_y is enabled to account for the camera orientation.
- Bitmaps and TileGrid: Bitmaps are created for camera_image (80x60) and inference_image (12x12). TileGrid is used to display the camera image on the LCD.

2) Image Preprocessing and Digit Classification:

- Image Capture and Conversion: The camera captures images in a loop with a 0.1-second delay. The captured image is copied to a temporary bitmap for further processing.

- Image Resizing and Grayscale Conversion: The temp_bmp is resized to 12x12 using rotozoom to match the model's input size. The resized image is then converted to grayscale using the rgb565_to_1bit function.
- Thresholding and Input Data Preparation: A thresholding function is applied to the grayscale image to reduce noise. The resulting binary values are appended to the input_data list.

3) Real-time Digit Classification and Display:

- Prediction and Display: The SVM model (svm_min) scores the input_data for digit classification. The predicted class index is determined, and its corresponding alphabet is retrieved. The prediction result is displayed on the LCD in real-time.
- Debugging Information: Optional debugging information is printed every 50 iterations, including the input_data values.

4) Code Execution and Performance Monitoring:

- Continuous Execution: The code runs in an infinite loop for continuous image capture, processing, and prediction. The system continually refreshes the display to show real-time predictions.
- Performance Monitoring: The system performance is monitored by printing debugging information, including input_data values. The prediction result is displayed on the LCD, providing a real-time output of the digit classification.

V. Results and discussion

The implemented system successfully captures real-time images from the OV7670 camera module, processes them through various preprocessing steps, and performs digit classification using the SVM model. Here are the key observations and discussions regarding the results:

1) Real-time Classification: The system demonstrates real-time digit classification based on the images captured by the camera.The SVM model efficiently scores the preprocessed input data, providing a quick and accurate prediction.

2) LCD Display Output: The digit classification results are displayed on the ST7735R TFT LCD in a clear and readable format. The display area is updated in real-time, allowing users to see the predicted digit as the system processes each frame.

3) Image Preprocessing: The image preprocessing pipeline, including resizing, grayscale conversion, and thresholding, effectively prepares the images for input to the SVM model. The implementation successfully addresses challenges such as the original image size (80x60) not matching the model's input size (12x12).

4) Debugging Information: The code includes optional debugging information that prints input_data values every 50 iterations. This information is valuable for monitoring and understanding the raw data being fed to the SVM model.

5) Performance Considerations: The system's performance is evaluated in terms of execution speed and continuous real-time processing. The implementation achieves a balance between processing speed and accuracy, with a 0.1-second delay between image captures.

6) Model Accuracy and Predictions: The accuracy of the SVM model is critical to the system's success. It's assumed that the SVM model (svm_min) is trained and performs well on the task of digit classification. Predictions are displayed in terms of class indices, and the corresponding alphabet is retrieved for user-friendly output.

7) Potential Improvements: Further optimization of the image preprocessing steps or exploration of alternative methods may enhance system efficiency. Incorporating user feedback, such as allowing users to interact with the system, could be considered for future iterations.

In summary, the results indicate a successful integration of hardware components, image preprocessing techniques, and machine learning for real-time digit classification on a microcontroller platform. Continued refinement and potential enhancements can be explored based on specific use cases and user requirements.

VI. Conclusion

In conclusion, the implemented system effectively combines hardware interfacing, image preprocessing, and machine learning to achieve real-time classification using the OV7670 camera module and an SVM model. The integration of the ST7735R TFT LCD allows users to observe digit predictions in real-time, providing a seamless and interactive experience. The preprocessing pipeline handles challenges such as image resizing and grayscale conversion, ensuring compatibility with the SVM model's requirements. The system's continuous execution, with periodic debugging information, enables performance monitoring and contributes to a deeper understanding of the input data processed by the SVM model. The results demonstrate a successful synergy between hardware components and software algorithms, offering potential applications in scenarios where real-time digit recognition is essential.

While the current implementation meets the specified requirements, there is room for further refinement and expansion. Future iterations may explore optimisations in image preprocessing, alternative machine learning models, or user interaction features to enhance the system's versatility and accuracy. Overall, the project lays a solid foundation for real-world applications that demand reliable and efficient digit classification on embedded platforms.