Handwritten digit classification using Raspberry Pi Pico and Machine Learning

Abstract:
This report presents the implementation of a handwritten digit classification system on the Raspberry Pi Pico micro controller using machine learning techniques. The project leverages Tensor Flow Lite for Micro controllers to deploy a trained model for real-time digit recognition. The integration of a camera module enables the capture of handwritten digits, and the system provides predictions with minimal computational resources.

Background:
Digit recognition has witnessed significant advancements, primarily driven by deep learning models. However, deploying these models on resource-constrained devices like micro controllers remains a challenge. The Raspberry Pi Pico, a low-cost micro controller, provides a promising platform for edge computing applications.

Objectives:

- **Real-time Classification:** Develop a system capable of classifying handwritten digits in real-time.
- **Resource Efficiency:** Optimize the machine learning model and code to fit within the limited resources of the Raspberry Pi Pico.
- **Edge Computing:** Demonstrate the feasibility of running a machine learning model on a micro controller for edge applications.

1. Introduction:
Digit recognition has gained significant importance in various applications, from postal services to automatic form processing. This project focuses on implementing a lightweight digit classification system on the Raspberry Pi Pico, offering a cost-effective and energy-efficient solution for embedded systems.

2. Related Work:

Review existing literature and projects related to digit recognition and embedded systems. Highlight the significance of using Raspberry Pi Pico for edge computing applications.

3. Methodology:

<u>*3.1 Data Collection:*</u>
Describe the dataset used, mentioning the source (e.g., MNIST) and any preprocessing steps.

<u>*3.2 Model Training:*</u>
Explain the machine learning model used, including its architecture and training process on a separate machine (e.g., using TensorFlow on a computer).

<u>*3.3 Model Deployment on Raspberry Pi Pico:*</u>
Detail the steps involved in converting and deploying the trained model on the Raspberry Pi Pico using TensorFlow Lite for Microcontrollers.

<u>*3.4 Integration with Camera Module:*</u>
Discuss the setup and configuration of the camera module to capture images of handwritten digits.

<u>*3.5 Inference and Results:*</u>
Explain how the model performs inference on captured images and present the results, including accuracy and latency.

Required Hardware:

- Raspberry Pi Pico H
- 128x160 TFT LCD
- OV7670 Camera Module
- Full sized breadboard (
- Jumper Cables - May 20 each of M-F and M-M. There are lots of connections to be made !!

Required Software:
Any text editor for editing the code if you plan to make any chances. A full Python distribution and pip for training and exporting the machine learning model. And off-course, lots of patience.

Implementation:

The below Python script is designed for a microcontroller, specifically the Raspberry Pi Pico. Its purpose is to implement a real-time handwritten digit recognition system using an OV7670 camera, a TFT LCD display (ST7735R), and a logistic regression model. Here's a breakdown of the key components and functionalities:

1. Importing Libraries:

```
import gc

import sys

from time import sleep

import bitmaptools

import board

import busio

import digitalio

import displayio

import logistic_regression_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
```

2. Conversion Function:

```
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
```

3. 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)
```

4.Creating Display Elements:

```
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_SWAPPED
    ),
    x=30,
    y=30,
)
group.append(camera_image_tile)
camera_image_tile.transpose_xy = True

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

5. Setting up the OV7670 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
```

6. Main Loop:

```
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=logistic_regression_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)
```

7. Memory Management:

```
import gc
gc.collect()
```

8. Printing Python Version:

```
print(sys.version)
```

Training Model Code:

```
!pip install m2cgen
%matplotlib inline
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn import datasets, metrics
from sklearn.model_selection
import train_test_split
from sklearn.linear_model
import LogisticRegression
import numpy as np
import cv2
import m2cgen as m2c  # Import m2cgen library
# Load digits dataset
data = datasets.load_digits()
# Resize and normalize images
image_ds = []
for img in data.images:
    res = cv2.resize(img, dsize=(12, 12))
    res = res / 16
    image_ds.append(res)
plt.imshow((image_ds[78]), cmap='gray')
img_data = np.asarray(image_ds)
flattened_data = img_data.reshape((len(image_ds), -1))
clf = LogisticRegression()
X_train, X_test, y_train, y_test = train_test_split(
```

```
    flattened_data, data.target, test_size=0.4, shuffle=False
)
clf.fit(X_train, y_train)
predicted = clf.predict(X_test)
plt.imshow(X_test[12].reshape((12, 12)))
plt.show()
print("Predicted: " + str(predicted[12]))
print(
    f"Classification report for classifier {clf}:\n"
    f"{metrics.classification_report(y_test, predicted)}\n"
)
# Export the model to Python code
code = m2c.export_to_python(clf)
with open('logistic_regression.py', 'w') as f:
    f.write(code)

# Example input data for prediction
arr = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.648438, 0.820313, 0.828125, 0.75, 0.75, 0.75, 0.742188, 0, 0, 0, 0, 0, 0.65625, 0.757813, 0.648438, 0.5625, 0, 0, 0, 0, 0, 0, 0, 0, 0.65625, 0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.734375, 0.742188, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.75, 0.546875, 0.640625, 0.640625, 0, 0, 0, 0, 0, 0, 0, 0, 0.734375, 0.75, 0.742188, 0.742188, 0.75, 0.75, 0.546875, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.734375, 0.648438, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.742188, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.75, 0, 0, 0, 0, 0, 0, 0.539063, 0, 0, 0, 0.65625, 0.75, 0, 0, 0]
b = np.asarray(arr).reshape((12, 12))
plt.imshow(b, cmap='gray')
plt.show()
print('Prediction: ' + str(clf.predict([arr])[0]))
!pip install python-minimizer
!python-minimizer logistic_regression.py
-o logistic_regression_min.pya
!ls -alh logistic_regression_min.py
```

4.Challenges Faced:

During the implementation, some challenges were encountered and addressed:

Resource Constraints: Optimizing the model and code to fit within the limited resources of the Raspberry Pi Pico required careful consideration of memory and processing constraints.

Integration of Camera Module: Configuring the OV7670 camera module for capturing images of handwritten digits involved overcoming compatibility issues and ensuring smooth integration.

5.Implications:

The successful implementation of this handwritten digit classification system holds several implications:

Cost-Effective Solution: The Raspberry Pi Pico, being a low-cost microcontroller, provides a cost-effective solution for embedded systems requiring digit recognition.

Energy-Efficient Edge Computing: Edge computing on microcontrollers offers energy-efficient alternatives for applications where real-time processing is crucial.

6. Future Work:

As with any project, there are opportunities for further improvement and expansion:

Model Optimization: Continued efforts in optimizing the machine learning model can enhance its efficiency and reduce memory footprint.

Exploration of Additional Datasets: Testing the system with diverse datasets could improve the model's generalization to different styles of handwritten digits.

Advanced Features: Exploring advanced features such as multi-digit recognition or integration with other sensors could broaden the system's capabilities.

7**. Conclusion:**

In conclusion, this project sets the foundation for deploying machine learning applications on resource-constrained devices, showcasing the potential of edge computing for real-time tasks.