EMBEDDED SYSTEMS COURSE PROJECT REPORT

Implementation of SVM algorithm on STM32 discovery board

Team members: Gandyadapu Sriharsha (B21CS029) Gavva Sathwika (B21CS030)

Introduction

The project aims to implement the Support Vector Machine (SVM) algorithm on an STM32 Discovery board for classification tasks using the MNIST dataset. The STM32F429I DISC1 is a powerful development board based on the ARM Cortex-M4 processor, providing a range of peripherals and features suitable for embedded applications. It offers sufficient computational power, memory resources, and GPIO pins for implementing machine learning algorithms like SVM. Additionally, it supports various communication protocols, making it suitable for interfacing with sensors and other external devices.

SVM algorithm for MNIST dataset

Support Vector Machines (SVMs) are a powerful class of supervised learning algorithms used for classification and regression tasks. In classification, SVM finds the hyperplane that best separates different classes in the feature space, maximizing the margin between classes. When applied to the MNIST dataset, which consists of handwritten digits, SVMs can effectively recognize and classify these digits into their respective categories (0-9). It works by finding the hyperplane that best separates the classes in the input space. *Kernel Trick:* One of the key strengths of SVMs is their ability to use the kernel trick. This allows SVMs to implicitly map the input features into a higher-dimensional space, where the data might be more easily separable. Common kernel functions used with SVMs include the linear kernel, polynomial kernel, radial basis function (RBF) kernel, and sigmoid kernel. *Margin Maximization:* SVMs aim to find the hyperplane that maximizes the margin between classes. The margin is the distance between the hyperplane and the nearest data point from either class. By maximizing this margin, SVMs can achieve better generalization and robustness to noise in the data.

Implementation

Dataset Preparation: The MNIST dataset consists of 28x28 grayscale images of handwritten digits (0-9). Each image is represented as a 784-dimensional feature vector (28x28 pixels). We preprocess the dataset by normalizing pixel values to the range [0,1] and splitting it into training and testing sets. As SVM operates on feature vectors, no specific feature extraction is required for this project.

SVM Training: We use the training set to train the SVM model. The SVM algorithm learns the optimal hyperplane that best separates the different digit classes while maximizing the margin between them.

SVM Inference on STM32: Once the SVM model is trained and optimized, we deploy it onto the STM32 discovery board. We write firmware code in C/C++ to load the trained model parameters onto the board and implement the necessary algorithms for inference. We ensure efficient memory management and optimize code for real-time performance.

Integration with Peripherals: Depending on the application, we integrate the SVM implementation with peripherals such as lcd for real-time digit recognition tasks.

Accuracy Measurement: To evaluate the performance of the SVM model implemented on the STM32 discovery board, we measure its accuracy on the testing dataset.

Steps to run

Model Training

1) We will use a svm model trained using tensorflow (Refer from svm_model.ipynb).

2) Now using the model that has been trained, we need to convert it to the TensorFlow Lite Model that our microcontroller can use. TensorFlow has a built-in converter function that will save the model as a TensorFlow Lite model file. (Download svm_mnist.tflite)

3) To get our tflite file to run on a microcontroller, we need to save it as a constant byte array in svm_model.h file.

Project Configuration

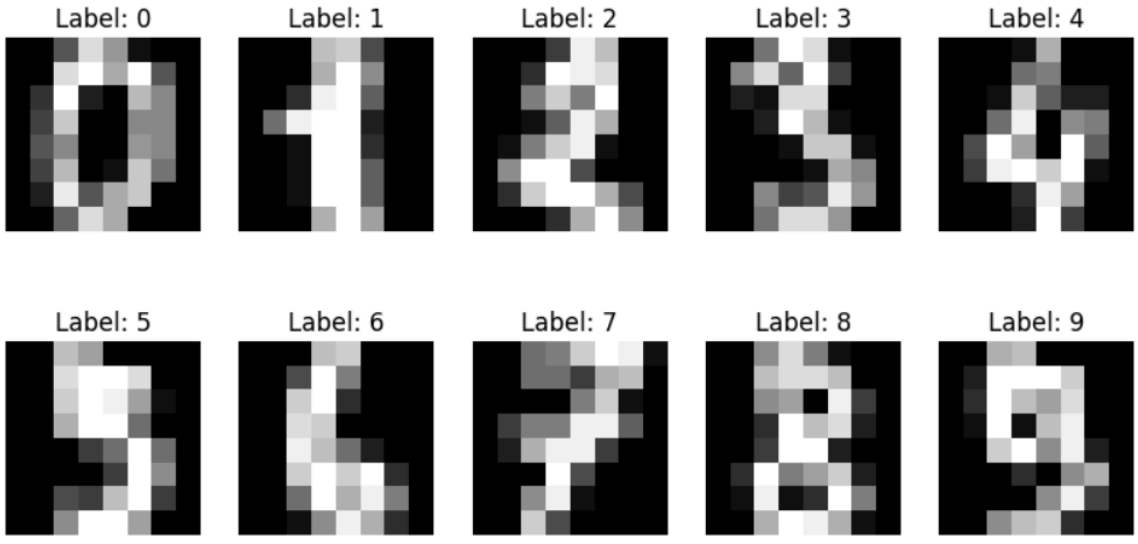
1) In STM32CubeIDE, Install X-CUBE-AI latest version.

2) Create a new project: Click File > New > STM32 Project. And In the Target Selection window, click on the Board Selector tab and select the board (“STM2F429I-DISC1” for me).

3) In CubeMX, enable Timer 16, and set the prescaler and AutoReload register with the maximum value and also select the Artificial Intelligence component class in Additional Software sub-tab.
4) Select STMicroelectronics.X-CUBE_AI and ensure that it is activated in Mode pane. In the Configuration pane, add network and Select TFLite for the network type and browse the svm_mnist.tflite downloaded earlier.
5) Click the Analyze button, we can get an overview of the model. This can be helpful for figuring out memory required to store it and how many multiply-and-accumulate (MAC) operations are needed.
6) Validate on Desktop runs some tests with models to make sure that inference works.
7) Head to the Clock Configuration tab. Under PLL Source Mux, change the input to the high speed internal clock, labeled HSI. For HCLK (MHz), enter the value “80” and the CubeMX software should adjust all of the prescalers to give you an 80 MHz system clock.
8) And then generate the code.We can now see a new X-CUBE-AI library appear in the project. In the App folder, we can find a set of source code files with <model_name>.
9) Open main.c. Take a look at the main.c code. After that Add printf Float Support.
10) Save the code. Click Project > Build Project. And Run in Debug Mode. Then Click the Play button to start running code on the microcontroller.

Results

Data Visualization (MNIST- Handwritten digit)



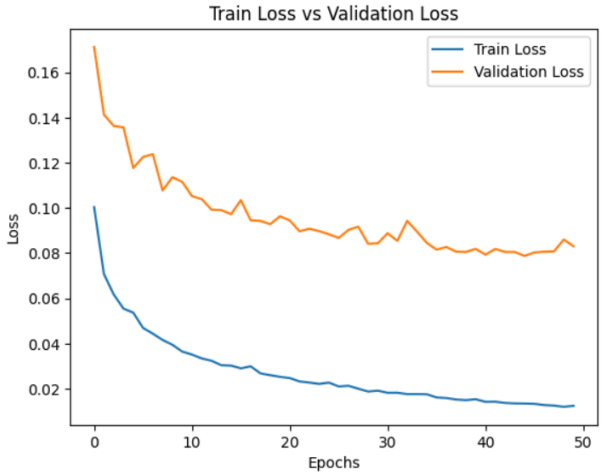
Accuracy and loss Values

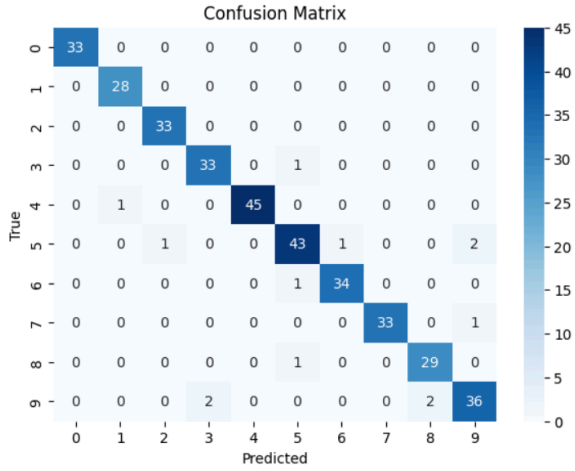
```
34/34 [==============================] - 1s 2ms/step - loss: 0.2470 - accuracy: 0.9109
Train loss: 0.24698783457279205
Train accuracy: 0.9108635187149048
```

```
12/12 [==============================] - 0s 5ms/step - loss: 0.3909 - accuracy: 0.8667
Validation loss: 0.3909262716770172
Validation accuracy: 0.8666666746139526
```

```
12/12 [==============================] - 0s 2ms/step - loss: 0.1505 - accuracy: 0.9639
Test loss: 0.15052752196788788
Test accuracy: 0.9638888835906982
```

We have plotted the train vs validation loss and also the confusion matrix





References

https://medium.com/@thommaskevin/tinyml-support-vector-machines-classifier-c391b54f3ab8#:~:text=Fundamentally%2C%20the%20SVM's%20role%20is,considered%20the%20potential%20separation%20boundaries.

https://www.tensorflow.org/lite/microcontrollers

https://www.digikey.in/en/maker/projects/tinyml-getting-started-with-stm32-x-cube-ai/f94e1c8bfc1e4b6291d0f672d780d2c0