Design Report
Processing Data from Neuromorphic Vision Sensors on Microcontrollers

Project Group 2:
Cristian Angheluș (s2680165)
Cezar Vintea (s2726467)
Emil Adam (s2811731)
Daniel Nicoară (s2467593)

Supervised by:
Dr. Yousefzadeh, Amirreza

Date
November 8, 2024

UNIVERSITY
OF TWENTE.

Contents

Abstract

This report documents the design and development of the “Processing Data from Neuromorphic Vision Sensors on Microcontroller” project, as part of the “Design Project” course at the University of Twente. The project aimed to leverage the neuromorphic vision sensors for real-time digit recognition, capitalizing on their unique advantages such as low power consumption, high dynamic range, and privacy preservation. Specifically, we used the Prophesee GENX320 neuromorphic vision sensor mounted on an STM32 microcontroller, which was chosen for its leading-edge performance in low-power, high-efficiency applications. Building on an existing demo application, which displays the camera output on the microcontroller screen, and the work of a previous student, we integrated a neural network enhanced with convolutional neural network (CNN) layers to enable and improve the digit recognition while ensuring the system met the constraints of embedded environments. The model was trained using the MNIST dataset and optimized for real-time predictions displayed directly on the STM32 board. This report outlines the full design trajectory, from requirement collection and system architecture to detailed implementation, testing, and performance evaluation. The report ends with a conclusion and a discussion regarding future work to expand and enhance the system's functionality.

Chapter 1 - Introduction

1.1 Neuromorphic Vision Technology

Neuromorphic vision sensors, also known as event-based or dynamic vision sensors, are a class of image sensors inspired by how human eyes and biological neural systems process visual information. Unlike traditional frame-based cameras that capture entire images at fixed time intervals, neuromorphic sensors only capture changes in the scene, such as movement or variations in light intensity, on a per-pixel basis. This enables them to record events as they happen, rather than producing a constant stream of redundant data, resulting in significantly lower data throughput and power consumption.

At the heart of this technology is an event-driven architecture. Each pixel in a neuromorphic sensor operates independently and asynchronously, detecting and recording changes in the scene as individual "events". This approach allows for ultra-fast response times and high dynamic range, making these sensors highly effective in challenging lighting conditions and fast-motion environments. In each event, the sensor records the pixel's location, the time the change occurred, and the polarity of the change (whether the light intensity increased or decreased). This output structure imitates the way neurons in biological visual systems act in response to changes in stimuli, making the sensor "neuromorphic" in nature.

This technology particularly benefits embedded systems with limited processing power and memory. Traditional image processing methods continuously capture and analyze full frames, consuming significant resources. In contrast, neuromorphic sensors filter out redundant information, making them ideal for applications where power efficiency and real-time processing are essential. This efficiency is especially valuable for edge devices, such as drones, robots, and wearables, which must function autonomously with minimal energy consumption.

1.2 Problem Statement

Embedded systems increasingly require efficient, real-time vision capabilities, especially in applications where low power consumption, high-speed processing, and data privacy are essential. Conventional frame-based cameras produce a continuous stream of redundant data, requiring significant processing power and energy, which can quickly drain embedded system resources. This makes traditional cameras and image-processing methods impractical for low-power, autonomous systems operating in real time.

Neuromorphic vision sensors present a promising solution to this challenge. However, the integration of neuromorphic technology with embedded hardware for real-time recognition tasks is still in the exploratory phase. Limited research exists on leveraging this technology with known machine learning models in constrained environments, such as microcontrollers, to deliver efficient on-device processing.

In this project, we will explore how well machine-learning techniques can be adapted to take advantage of neuromorphic vision sensors, by integrating a neural network in the microcontroller to process the data from the neuromorphic sensor and perform a simple task, such as recognizing hand-written digits shown to the camera.

1.3 Project Objective

Our project proposes to leverage neuromorphic vision sensors for digit recognition, capitalizing on their unique advantages. The objective is to develop a real-time digit recognition system using neuromorphic vision technology in an embedded environment. By harnessing the unique capabilities of the Prophesee GENX320 neuromorphic vision sensor on an STM32 microcontroller, this project aims to create a power-efficient, high-performance solution for digit detection, exploring the possibilities and limitations of using known machine learning techniques on the new type of neuromorphic vision. Specifically, the goal is to implement a more advanced neural network model capable of recognizing handwritten digits shown to the camera and displaying the predicted digit on the STM32 microcontroller's screen in real-time.

1.4 Tools Specifications

For the project, we used the Prophesee GenX320 module kit for STM32 devices with an S-Mount lens. The GenX320 module kit has been optimized for plug-and-play use with the STM32-F7 platform. It integrates the Prophesee GenX320 1/5" format 320x320 event-based vision sensor, with a 320x320 array of 6.3µm contrast detection pixels, and high-speed event data output (equivalent to >10kfps time resolution), which is suitable for the development of low-power embedded applications with hard real-time constraints [1].

The microcontroller used is the STM32F746-DISCO, which is used as a reference design for user application development before porting to the final product, thus simplifying the application development. The Discovery kit enables a wide variety of applications to benefit from audio, multi-sensor support, graphics, security, video, and high-speed connectivity features [2].

A more detailed specification of the tools can be found in Appendix A.

Chapter 2 - Requirements Specification

2.1 Client Overview

Our project supervisor, who also acts as our main client, is a member of the Computer Architecture for Embedded Systems (CAES) group of the Faculty of Electrical Engineering, Mathematics, and Computer Science at the University of Twente. The CAES group conducts research and education in computer architecture and computing systems with a particular emphasis on embedded systems, therefore this project aligns with the group's goals of exploring neuromorphic vision and machine learning applications on embedded hardware.

The client's main role is to define the key requirements that align with his expectations of the project and equip the team with the necessary tools to complete the project (the Prophesee GenX320 camera attached to the STM-32 microcontroller). Since our client is also our supervisor, he provided the necessary guidance to ensure that the project meets both functional and quality standards, while respecting the time constraints given.

The client's goal is to advance knowledge in neuromorphic vision applications and explore the practicality of real-time neuromorphic visual processing on embedded systems. This project aims to contribute to this knowledge and support the ongoing research and development in this area by providing a foundation for future enhancements and applications of the system.

2.2 Requirements Overview

A list of requirements was gathered and formulated from the meetings with the supervisor. We divided them into Functional and Quality requirements and classified them by their importance using the MoSCoW method, except for the Won't requirements, since the scope of the project is already limited, so we don't have a lot of requirements that need to be excluded or limited.

2.3 Functional Requirements

These requirements define the core functionality of the system.

Must-have - requirements that are critical to the project delivery:

- **Handwritten Digit Recognition**: The system must detect and recognize the handwritten digits shown to the neuromorphic camera with adequate accuracy.
- **Integration of CNN layers**: The system must integrate a neural network trained for handwritten digit recognition consisting of at least 2 Convolutional Neural Network (CNN) layers to improve the accuracy of the previous existing model.
- **Local Processing on the Microcontroller:** The system must be embedded in the microcontroller memory, relying only on the local resources to process the images and detect the right digit using the neural network.

Should-have - requirements that are important, but not crucial for the delivery of the project:

- **Latency Optimization -** The system should minimize the delay between capturing an image and displaying the recognition result.
- **On-screen Prediction -** The system should display the result of the digit prediction directly on the microcontroller's screen, improving the user experience.
- **Real-time Detection -** The system should recognize the digits in real time, ensuring immediate feedback for a practical user experience.

Could-have - requirements that are desirable but not necessary and could improve the user experience:

- **Robustness to Varying Angles and Surfaces -** The system could recognize the digits well under different angles and from digits written on both digital screens and on paper.

2.4 Non-Functional (Quality) Requirements

These requirements define the quality attributes of the system.

Must-have:

- **High Accuracy Prediction -** The system must achieve a minimum accuracy of 90% under typical conditions to ensure effective digit recognition.
- **Feasible Response Time -** The system must output the prediction of the digit within a feasible period of less than 1 second.
- **Resource Usage -** The entire system must be optimized to fully fit inside the microcontroller and not exceed its resources during the usage.

Should-have:

- **Minimized Latency -** The system should have minimal latency between input capture and output display, supporting the goal of real-time digit recognition.
- **Compact and Lightweight Model -** The neural network model should occupy minimal memory and processing resources to fit within the STM32's limited capacity.
- **User-Friendly Interface -** The system should have a user-friendly interface, displaying the predicted digit to the user clearly and understandably, requiring minimal intervention.
- **Detailed Documentation -** The system should have detailed documentation regarding the implementation process that could help future students/researchers with expanding the system's capabilities.

Could-have:

- **Consistent Performance Under Varying Conditions -** The system could maintain stable performance across different environments, ensuring robustness in real-world settings, such as unclear images or bad lighting.
- **Noise Reduction -** The system could process the images of the digits and remove the surrounding noise to improve the accuracy of predictions.

2.5 Summary

This requirements specification identifies the key functional and quality requirements necessary to complete our project. The requirements were extracted from the meetings with our supervisor, reflecting the desired outcomes for this project and serving as guidelines throughout our implementation process. The MoSCoW prioritization method provides a clear framework, highlighting essential features while allowing room for enhancements. By meeting these requirements, the project aims to create a powerful digit recognition solution for neuromorphic vision sensors in an embedded system that balances accuracy and efficiency, achieving both technical and client expectations.

Chapter 3 – Planning and Project Proposal

This chapter outlines our initial planned approach to implementing the project. It details the phased project trajectory, key milestones, and contributions from each team member. Additionally, it covers the organized meetings, the test plan to validate system performance, as well as risk analysis to anticipate and mitigate potential challenges. This chapter provides an overview of how the project was executed, from initial requirements to final testing, ensuring that all objectives are met efficiently within the project timeline.

3.1 Project Approach and Phases

To ensure the systematic and timely development, we identified and structured the project into several phases:

3.1.1 Exploration and Setup of the Hardware Components and Existing Demo

Objective: To familiarize the team with the Prophesee GENX320 neuromorphic vision sensor and the STM32 microcontroller, and have a general understanding of how the existing live demo works and how the components are connected. Also, to set up the necessary tools and environment to be able to modify and execute applications on the microcontroller.

Activities:

- Reviewing the existing general information regarding the Neuromorphic Vision Technology.
- Reading and analyzing the existing documentation and materials from Prophesee.
- Analyzing the documentation and tools for the STM32 microcontroller.
- Setting up and configuring the sensor and microcontroller, experimenting with the camera's event-based output, and testing the initial setup on the STM32.
- Meeting with Mustafa Canıtez, a PhD student who previously worked on a similar project.

Outcome: A general understanding of the camera and microcontroller functionalities, and the technology in general. Additionally, gained a lot of insights from the meeting with the PhD student who worked in the past on a similar project. His insights helped us a lot to understand the

functionality of the live demo from Prophesee, as well as implementation details and ideas from his project, which enabled us to further extend it with our implementation.

3.1.2 Requirements Specification and System Analysis

Objective: To gather detailed and final requirements from the supervisor, analyze similar implementations and come up with our planning of the project.

Activities:

- Organize meetings with the supervisor to gather concrete requirements and pivot the overall planning and implementation details of the project.
- Establish and prioritize the concrete requirements.
- Organize a general plan and structure of the project.
- Establish and divide the responsibilities within the team.

Outcome: After discussions with the supervisor, we clarified and finalized the project's scope. Initially, the objective was to implement a neural network capable of person detection, however, after advising the supervisor, we decided to implement digit detection by further expanding the work of a previous student with a more complex neural network and a better GUI, showing the predicted digit directly on the microcontroller's screen. Also, we established and documented the formal requirements, developed a project plan, and divided the tasks among the team members.

3.1.3 System Design and Model Development

Objective: To design the overall system architecture, including sensor integration, data preprocessing, neural network model design, and the GUI design for representing the predicted digit.

Activities:

- Design the overall structure of the system, including the used libraries and implementations.
- Decide the CNN architecture design and structure for the digit recognition task.
- Create the design for the GUI that represents the predicted digit.

Outcome: Developed a general system architecture, defined the structure of the used CNN, and designed the simple user interface for displaying the predicted digit directly to the screen of the microcontroller.

3.1.4 Implementation and Integration

Objective: To implement the actual CNN and integrate it with the Prophesee sensor and STM32 microcontroller, ensuring it meets real-time performance requirements.

Activities: Code the actual model and integrate it in the live demo of the microcontroller, optimize the neural network for real-time processing, and configure the STM32 to display the predictions.

Outcome: A functional prototype with integrated CNN layers, capturing data from the neuromorphic sensor and displaying results on the STM32 in real-time, capable of presenting the results in a live demo.

3.1.5 Testing and Optimization

Objective: To validate system performance in terms of accuracy and responsiveness.

Activities: Test the model using the MNIST test set, and using manual testing of the MNIST digits on the actual sensor, measure accuracy and latency, and optimize model performance and functionality.

Outcome: A refined and validated system meeting the project's requirements for real-time performance and efficiency.

3.1.6 Documentation and Final Presentation

Objective: To document the full project and prepare a final presentation for review.

Activities: Write the final design report and prepare the final presentation for review with the supervisor.

Outcome: A comprehensive, well-documented documentation of the project and a final presentation ready for reviewing to the client.

3.2 Project Timeline and Milestones

The following table represents a general timeline for the development process of our project. It serves more as a guideline, as several phases of the project might overlap, and start earlier or have a longer duration than expected.

Phase	Timeline	Milestone
Exploration & Setup	Weeks 1–2	Successful sensor and microcontroller setup
Requirements Specification and System Analysis	Weeks 1–2	Defined requirements and general planning of the project
System Design and Model Development	Weeks 3–5	Finalized CNN model and system architecture
Implementation and Integration	Weeks 6–8	Implementation of the model and integration on STM32. A working prototype.
Testing and Optimization	Weeks 9–10	Validated and optimized final system
Documentation and Final Presentation	Week 10	Submission of final report and presentation

3.3 Risk Analysis

To mitigate potential risks, we identified the following challenges and developed contingency strategies:

Hardware Setup and Sensor Configuration:

- **Risk:** Difficulty in configuring and integrating the Prophesee GENX320 neuromorphic sensor with the STM32 microcontroller, which could delay progress in later phases.
- **Mitigation:** Conduct a thorough documentation review, consult the PhD student who previously worked on a similar project in case of encountering problems, and maintain close contact with the supervisor. Additionally, plan extra time in the timeline for setup and troubleshooting to ensure a robust initial configuration.

Model Complexity and STM32 Processing Constraints:

- **Risk:** The CNN model may exceed the STM32's memory and processing capabilities, affecting real-time performance and possibly requiring redesigns.
- **Mitigation**: Keep the model lightweight, implement techniques such as quantization to reduce memory usage, and test early versions on the STM32 to ensure compatibility. Optimize the CNN structure during the design phase to minimize resource demands and maintain a balance between accuracy and performance.

Achieving Real-Time Performance:

- **Risk:** Ensuring the system meets real-time performance requirements on the STM32 microcontroller could be challenging, especially with the latency introduced by CNN processing.
- **Mitigation:** Optimize code during the integration phase, leverage real-time profiling to measure and reduce latency, and adjust CNN parameters as needed to achieve the required speed. Conduct iterative testing on the STM32 during development to monitor real-time performance and make adjustments as necessary.

Accuracy and Model Performance:

- **Risk:** The CNN model may fail to reach the target accuracy due to constraints in training, processing power, or sensor limitations.
- **Mitigation:** Conduct thorough testing on the MNIST dataset and real-time captured inputs, using manual testing with various handwritten digits. Optimize the model iteratively, adjusting parameters and training techniques to maximize accuracy within hardware constraints.

Delays in Documentation and Presentation Preparation:

- **Risk:** Completing the final detailed documentation and preparing a final comprehensive presentation might take more time than anticipated, affecting the quality of the final deliverable.
- **Mitigation:** Begin documenting each phase as soon as it's completed and maintain a shared project journal to track progress. Allocate time in advance for reviewing and refining the documentation, and conduct a preliminary presentation to the team for feedback before the final review.

3.4 Summary

In this chapter, we outlined our structured approach for implementing the project. We developed a phased project trajectory, covering hardware setup, requirements specification, system design, implementation, testing, and final documentation. Each phase includes clearly defined objectives, key activities, and expected outcomes, ensuring a systematic and goal-oriented approach.

The project's trajectory began with an exploration and setup phase to familiarize the team with the Prophesee GENX320 sensor and STM32 microcontroller, followed by detailed requirements specification and system analysis. This phase included meetings with our supervisor and a PhD student with experience on a similar project, which provided valuable insights and led to a pivot from person detection to digit detection. The design and model development phase laid the foundation for the project, including the CNN model architecture, data preprocessing, and GUI design.

Additionally, we prepared a test plan to validate the system performance and developed a risk analysis to identify and address potential challenges throughout the project.

We validated our plan and project approach in alignment with the project's objectives and supervisor's expectations. With this validation, we are ready to proceed to the detailed design and implementation phases of the project, which will be described in the following sections.

Chapter 4 - Design

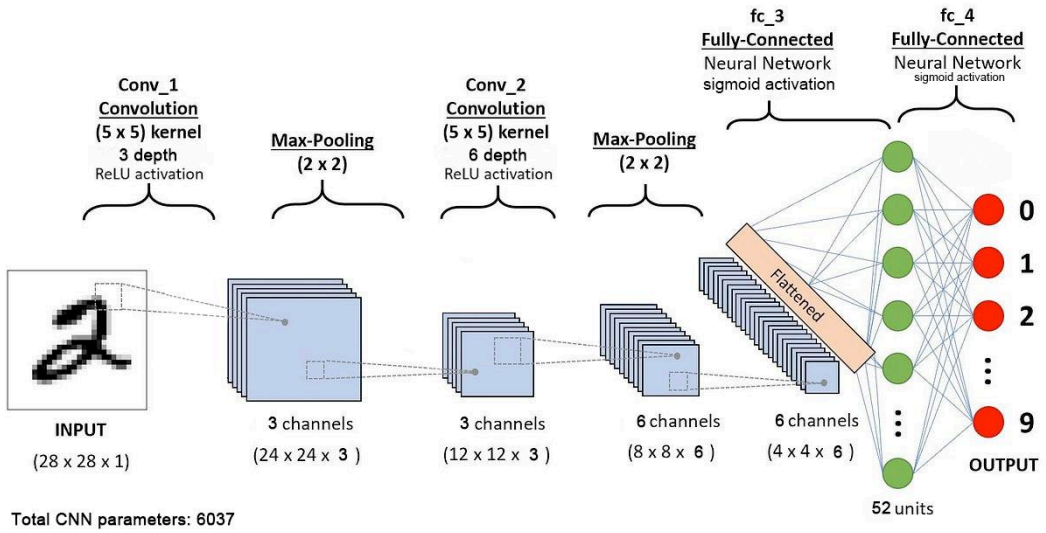
4.1 System Architecture

The system is built around a **neuromorphic vision sensor** that mimics the function of biological sensory neurons. Rather than capturing full-frame images, this sensor detects changes in the visual scene, relaying event-based data to a **microcontroller** for processing. The key components of the architecture include:

- **Neuromorphic Vision Sensor:** This sensor captures event-based visual data by detecting scene changes rather than static images, significantly reducing computational and memory demands. It mimics the human eye's efficiency in only reporting relevant changes, as discussed in the reflection report. It includes the necessary C code from the official Prophesee demo project for the GEN320X NVS to function correctly with the microprocessor.
- **Microcontroller:** The microcontroller used in this project is the STM32F746-DISCO with the above-mentioned sensor mounted on it.
- **Neural Network:** A pre-trained **convolutional neural network (CNN)**, optimized for digit recognition, runs on the microcontroller. Lightweight architecture, Maximum pooling, and quantization were employed to fit the model within the microcontroller's limited memory, enabling real-time event processing similar to biological sensory systems.
- **Graphical Interface:** The UI that shows the predicted digit and the area on the screen where the hand-written digit should be placed to get an accurate prediction.

4.2 Detailed Design

4.2.1 Convolutional Neural Network



The convolutional neural network used in the project consists of 6 different layers which are:

- **Convolutional layer 1**
- **Max pooling layer 1**
- **Convolutional layer 2**
- **Max pooling layer 2**
- **Fully connected layer 1**
- **Fully connected layer 2**

The first convolutional layer is the one that takes the 28x28 picture of a hand-written digit. It uses 3 kernels of 5x5 dimension. Given the specifications, it has 75 weights that are then used in the recognition.

The first max pooling layer shrinks down the size of feature maps that are initially 3 matrixes with 24x24 dimensions after the first convolutional layer. It uses a 2x2 matrix to find the local maximum and save the answer in the resulting matrixes. The result of this layer is 3 12x12 matrixes.

The second convolutional layer takes the 3 matrixes that are 12x12 after the previous layer and passes 6 kernels with the dimensions of 5x5 through them. As a result, it gets 6 8x8 matrixes and saves a total of 450 weights.

The second max pooling layer is essentially identical to the first one and uses the same dimensions, so after the procedure, the result is 6 matrixes of 4x4.

Before the image gets to the fully connected layers it is flattened into a single dimension array with the length of 4*4*6.

The first fully connected layer has 84 neurons and thus 96*52 weights that the data has to pass through. In the end, the result is an array of 52 elements.

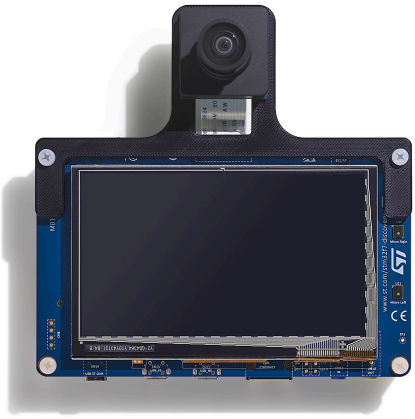
The second fully connected layer has only 10 neurons and a total of 52*10 weights. The final array contains the predictions of how similar every digit of the 10 is to the hand-written initial digit.

The final step is the function that finds the maximum from the final array and prints the index of the maximum coefficient.

4.2.2 Neuromorphic Vision Camera

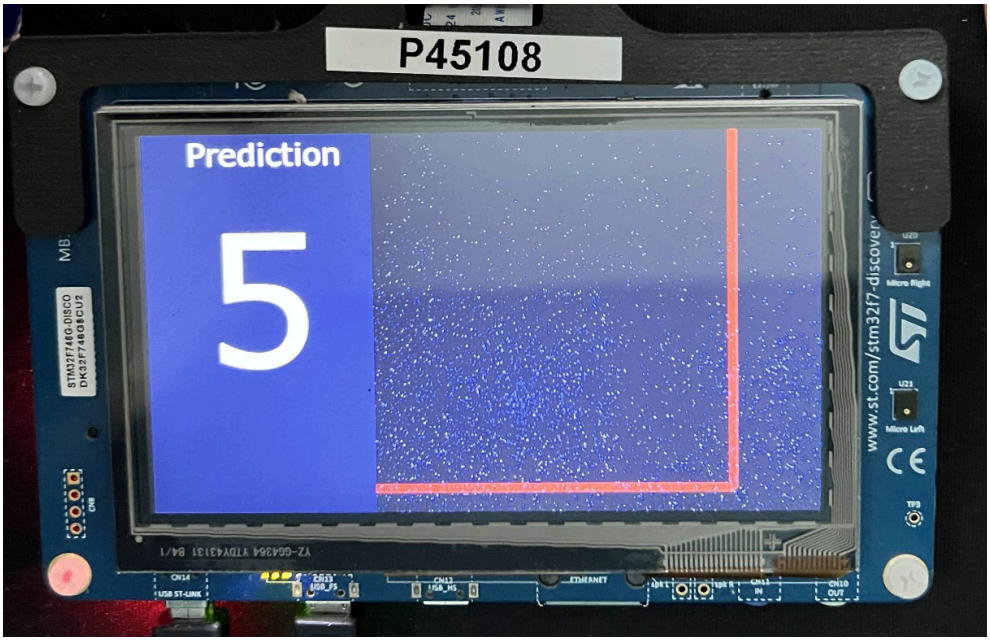
The Prophesee GEN320X camera is mounted on the microcontroller. The Prophesee official demo implementation has the drivers, pins, and scripts that are necessary for the correct functioning already configured. So, the frames captured by the camera were correctly saved in a convenient buffer by default.

4.2.3 Microcontroller



This is the full final hardware setup of the project. The above-mentioned camera is mounted in the STM32F46 DISCOVERY model. It is considered an energy-efficient microcontroller and is the model that is used in the official demo implementation from Prophesee, so it is highly compatible with the GEN320X and already has the necessary drivers and pins set up.

4.2.4 Graphical Interface:



The graphical interface is displayed on the microcontroller screen. On the left side of the screen is the label that contains the prediction from the neural network. On the right is the configured output from the sensor that is set by default and also the region marked by a red square where the digit should be placed to get a correct result.

Chapter 5 - Implementation

5.1 Neural Network Implementation

5.1.1 Training

The training for this specific neural Network is done by using the https://github.com/arnogranier/cppCNN/ framework in C++. Initially, the plan was to use the same framework for training and also prediction on the microcontroller. However, the STM32 microcontroller project programming is mainly done in C, and since the framework uses C++ abstraction and attributes that are not compatible with C there are compatibility issues. The framework was not created with embedded applications in mind, it uses double precision type for storing weights and performing calculations which would greatly exceed the memory and processing capabilities of our board. Moreover, it contains a lot of methods that are not necessary for the architecture of our project and only add complexity to the program. Given these arguments, the C++ framework was used only for training, testing, and saving weights of various CNN architectures. TensorFlow was also a choice during the analysis of possible training libraries, but the decision was not in its favor since the C++ library was less complex and more transparent with clear documentation. The training was done on the MNIST dataset that contains 60000 pictures and labels for training and 10000 pictures and labels for testing. The pictures are converted from grayscale to only black and white(0 or 1) for compatibility reasons with the output taken from the sensor. The learning rates are different for the feature detection and the classifier layers:

- Feature detection learning rate: 0.0003
- Classifier learning rate: 0.3

For optimal training and getting good accuracy without overfitting, we chose the number of 20 epochs for training. Our model has 6 layers with specifications explained in the design section. The total number of weights is calculated as

- First convolutional layer: 75 (3*5*5)
- First max pooling layer: 0
- Second convolutional layer: 450 (6*3*5*5)

- Second max pooling layer: 0
- First fully connected layer:4992 (96*52)
- Second fully connected layer: 520 (52*10)

The total number of weights is 6037. The weights are then quantized using a scaling factor of 10000 to later fit into an array of int16_t elements. Initially, the weights were used and saved only in their original type which is double without quantization. The accuracy without quantization is higher, but the memory usage is 4 times bigger which is why the 6037 weights are quantized and saved in a .txt file for later to be dequantized on the microcontroller only when the float value is necessary. After the training process, the file with the quantized weights is transformed into 4 strings (for each layer that requires weights) that look like a declaration of an array in C. This is done to copy and paste the weights in the form of an array in a separate C script that can be used straight away without reading the weights from a file first. Reading the weights from a file and then storing them in data structures interfered with the memory allocated for the loading of the UI and different elements.

5.1.2 Predicting

For a prediction mechanism, we have implemented the forward propagation functionality of the CNN from scratch, based on the order of the weights in the weights file generated by the C++ library for the prediction to go as intended after training. In comparison to the C++ library, our implementation of the layers is a lot simpler and easier to debug, and it does not contain unnecessary functionality and is more memory efficient. The methods that are used in our implementation:

- Conv - the method used for the convolutional layers
- Sigmoid function
- Max_pooling used for the MP layers
- Predict is used to find the maximum from the 10-element array and return the digit
- Calculate_the_weight_of_a_9by9_block used to compress the image
- fullyConnectedLayer is used for both fully connected layers
- Predict_main is the main function where all the layers are used and the prediction is returned
- Dequantize_weight is used to dequantize a single weight when it is necessary to use it for multiplication or other operations

The weights are saved in the weights.c script in the form of arrays and used by the methods described above. The prediction is done by feeding the 28x28 picture to the predict_main function

where all the layers are used and the picture is passed through them. At the end, the function returns the predicted digit.

5.2 CNN with NVS output

The main action of reading the input from the camera and passing it to the CNN is performed inside task_frame_buffer.c. The camera input is stored inside a variable called frame_buffer_l1_one. From this buffer, a specific region of 252x252 is taken starting from the top left corner. This is a requirement since this region can then be compressed to 28x28 if every region of 9x9 is shrunk to a single pixel. The function Calculate_the_weight_of_a_9by9_block takes each 9x9 block of this buffer and adds up all the values(0 or 1) into one variable(minimum-0, maximum-81). After that, each summed value is compared to a threshold. If a block contains fewer black pixels than the threshold then its corresponding pixel in the 28x28 image will be white, otherwise black. This threshold is at 30 since it proved to be more balanced in keeping the features of the digit visible, but not shaving off too much. Later, the 28x28 image is passed to the CNN via the predict_main function. The predicted digit is stored in a variable that will subsequently be passed to the label that shows the prediction inside the screen1_view.c script. The buffer is updated every cycle of the for statement thus ensuring that the image is always read as quickly as possible. However, since the usage of the CNN and pre-processing algorithms uses a lot of computational power if used on every cycle it could crash the runtime. That is why the prediction is limited to once every 30 cycles which is equivalent to once around 500 ms.

5.3 Graphic Interface Design

To create the GUI of the project the MX Designer software from STM32 was used. It offers a handful of useful and easy-to-use tools to create designs for the microcontroller application. It is used together

with the STM32IDE from which the code can be edited and also the runs are performed. After all the elements of the UI are placed on the chosen screen the code is generated inside the IDE. For this project, the UI consists of the label that shows the predicted digit and also the red square that highlights the region where the hand-written digit should be placed.

Chapter 6 – Testing and Evaluation

6.1 Test Plan

To assess the system's performance, we conducted tests focused on three key areas:

- **Accuracy:** The accuracy of the system was tested in 2 different ways. Firstly, the accuracy was tested inside the CPP library using the predictions from 10000 MNIST pictures. The second tests were run when the CNN was already mounted on the microcontroller by directly showing 200 pictures of MNIST pictures to the camera on 3 different tests and monitoring the outcome manually.
- **Latency:** Using the HAL_GetTick() method to track the start of the frame processing and the end of it. The delay was tracked when the CNN was commented out and when it was active and working together with the pre-processing algorithm.
- **System Efficiency:** Although power consumption wasn't a primary focus, we briefly observed the system's performance under normal conditions to check its operational stability.

Given limited resources, the testing prioritized basic functionality over comprehensive evaluation, providing initial insights into the system's behavior.

6.2 Test Results

- **Accuracy:**
  On the first test performed using the CPP library and MNIST dataset pictures the accuracy was 96,5%. On the second test performed on the microcontroller itself using the method described in the test plan, the accuracy dropped to 93%. One of the reasons for the change is that the NVS captures some noise and also during pre-processing some of the features of the digit and its thickness sometimes are shaved off. Moreover, we are losing some weight precision in the weights quantization process, which can also affect accuracy.
- **Latency:**
  The latency of the frame processing was tested 3 times. The first test was performed on the frame processing algorithm that does not contain the CNN or any pre-processing and the

time that it took was 22 ms. The second test was performed on the frame processing algorithm with the CNN which showed to be 45 ms from which 22 ms are for the general frame processing, 7 ms are pre-processing and the rest of 16 ms are for CNN prediction.

6.3 User Feedback

We conducted informal testing with a small group of users, who provided the following feedback:

- **Positive:** Users found the system responsive and easy to use, appreciating the real-time feedback and quick digit recognition.
- **Areas for Improvement:** Sometimes the system can make a wrong prediction if the digit is not shown in the center of the red square, is too tilted, or is very zoomed out to the camera, and if the digit is very heavily scorched or deformed to the point where even a human can barely recognize it.

6.4 Conclusion of Testing

Overall, the tests performed on the system worked well and showed that this implementation of the CNN can accurately predict digits if some specific conditions are met. Moreover, the processing time from the CNN showed to be even less than the predefined processing of the frame written by Prophesee, which can be considered a success. However, it could be possible to improve it to such an extent that it will be able to run on every frame productively without any freezes.

Chapter 7 - Conclusion

This project successfully demonstrated the potential of using neuromorphic vision sensors for efficient, real-time digit recognition on microcontrollers. By utilizing the event-driven nature of these sensors, the system was able to process changes in visual data with minimal computational demand, highlighting the advantages of this technology in environments with limited resources.

The system met its core objectives, achieving satisfactory accuracy and low latency in digit recognition tasks. By integrating neuromorphic vision with machine learning models, we showed that even on constrained hardware, it is possible to achieve high-performance real-time vision processing. These results indicate promising applications in fields such as autonomous systems, robotics, and wearable devices, where quick response times and low power consumption are crucial.

Despite the system's success under standard conditions, the project also revealed several areas for enhancement. Future efforts will aim to expand the system's capabilities to recognize more complex objects beyond handwritten digits. This will involve scaling the neural network and refining event-processing algorithms to handle more diverse inputs while maintaining efficiency.

Improving the system's resilience under varying conditions, such as changes in lighting or motion, will also be key for broader real-world applications. Furthermore, optimizing the interaction between software and hardware could enhance scalability, allowing the system to tackle more complex tasks without sacrificing performance.

In summary, this project represents a significant advancement in low-power, practical vision processing systems using neuromorphic sensors. It provides a strong foundation for future work focused on broadening functionality, improving adaptability, and exploring new applications in dynamic, real-world environments.

Chapter 8 – Future Work

1. **Expanding Object Detection Beyond Digits**
   Future work could extend the system's capabilities to recognize more than just digits. By training the model on more diverse datasets, it could detect letters, shapes, and even real-world objects. This would open the door to applications in areas such as smart surveillance, where it could monitor environments in real-time, industrial automation for more precise machinery control, and assistive technologies that help users interact with their surroundings more effectively.
2. **Quantization improvement**
   In this project, we used quantization from double weights to int16_t weights and then again to float when the weights were necessary for calculation. However, there are a couple of things that can be changed to make the system work even faster. First of all, quantization can be done to int8_t rather than int16_t. This would improve the speed of prediction since the amount of data to be processed is divided by 2. Moreover, instead of dequantizing the weights every time they are needed for a calculation, the calculation can be done on quantized weights which would remove the necessity for dequantization entirely.
3. **Exploring Alternative Microcontroller Platforms**
   Exploring microcontrollers with advanced AI capabilities, such as those featuring dedicated neural processing units (NPUs), could greatly enhance the system's speed and efficiency. These specialized platforms would make the system more capable of handling real-time processing in more complex applications, expanding its usability across sectors such as autonomous vehicles and advanced robotics.
4. **Improving Robustness Under Varying Conditions**
   To ensure the system works reliably across different environments, future iterations should focus on improving performance in challenging conditions like poor lighting or unusual input patterns. This could be achieved by incorporating techniques such as data augmentation, which simulates various conditions during training, or enhanced pre-processing methods. These improvements would make the system more resilient and consistent in diverse settings.
5. **CNN Evolution**
   Since our project has only 2 convolution layers, 2 max pool layers, and 2 fully connected layers including sigmoid activation and ReLU activation, our model can be called simple. That is why some techniques could be used to make it more advanced such as biases, more layers, etc.

6. **Compatibility issue**

   At the moment the CNN implemented for this project is compatible only with the weights format provided by the CPP library mentioned above. This means it would not work with the weights created by TensorFlow or any other popular libraries. It could become a problem once the scope is to mount a more advanced NN that cannot be trained on the CPP library.

7. **Testing improvement**

   Testing when the CNN is already mounted on the microcontroller becomes a long, manual, and time-consuming process. To test the accuracy of the system all the pictures should be shown to the camera directly which means the results have to be monitored manually by someone. For us, every accuracy test on the device with 200 MNIST images takes around 20-30 minutes, which is quite long for one test. Moreover, a 200-image test set is too small of a sample to be able to assess the accuracy correctly and as close to the truth as possible.

References

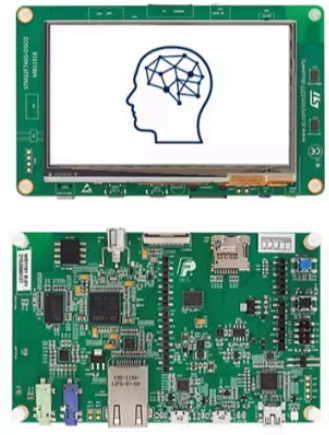
1. https://www.prophesee.ai/wp-content/uploads/2024/07/Metavision-Starter-Kit-STM32F7-GenX320-Product-Brief-2024-COB.pdf
2. 32F746GDISCOVERY - Discovery kit with STM32F746NG MCU - STMicroelectronics
3. https://github.com/arnogranier/cppCNN/

Appendix A

GenX320 module Specification

PARAMETER	UNIT	SPECIFICATION
Sensor Model		GenX320
GexX320 COB dimension	mm	8 x 8 x 5
Lens Model		SND2636A1SD-8M • 1.8 mm • f/2.8
#Connector pins		30
Control Interface		I²C
Data Interface		Parallel – STM32 Digital Camera Interface (DCMI)
Lens Mount		M12
Minimum object aperture		f/2.8
Focal Length		1.8 mm
HFoV/VFoV	deg	58
DFoV	deg	76
IR cut filter		No
Integrated EEPROM		Yes (256Kbit)

STM32F746-DISCO Specification



Features

- STM32F746NGH6 Arm® Cortex® core-based microcontroller with 1 Mbyte of Flash memory and 340 Kbytes of RAM, in BGA216 package
- 4.3" RGB 480×272 color LCD-TFT with capacitive touch screen
- Ethernet compliant with IEEE-802.3-2002
- USB OTG HS
- USB OTG FS
- SAI audio codec
- Two ST-MEMS digital microphones
- 128-Mbit Quad-SPI Flash memory
- 128-Mbit SDRAM (64 Mbits accessible)
- Two user and reset push-buttons