# Embedded Person Detection on an STM32F7 with Dynamic Vision Sensor

Pierluigi Gatt
*s3014940*

***Abstract*—Event-based vision sensors offer an alternative to RGB cameras by capturing the changes in a scene. This thesis investigates the optimisation of a convolutional neural network (CNN) for real-time person detection on an STM32F7 microcontroller with a neuromorphic vision sensor, specifically Prophessee's GenX320. Neuromorphic vision sensors offer advantages such as low latency, low power consumption, and the preservation of the recorded's anonymity. However, they pose unique challenges for implementation with a CNN due to their event-based data rather than traditional frame-based outputs. The STM32F7 microcontroller also introduces limitations for CNN deployment, such as its limited internal RAM and Flash and a lack of dedicated hardware acceleration for CNN inference. This thesis reviews and implements various optimisation techniques, such as post-training quantisation, architectural simplification and input size reduction. These efforts aim to minimise resource usage while maintaining detection accuracy. Preexisting CNN models for person detection on RGB datasets like COCO were adapted to fit on the STM32F746G-DISCO's limited 1024 KB and 320 KB of internal RAM and Flash, respectively, alongside processes dedicated to the sensor and display. These optimised models were then trained for inference with a neuromorphic vision sensor using the PEDRo dataset. The applied improvements resulted in a model with an inference time of 422 ms and an AP of 75% (@0.5 IoU on the PEDRo dataset). This study shows that object detection using a neuromorphic vision sensor is feasible on ultra-low-power hardware without significantly compromising accuracy.**

## I. Introduction

Neuromorphic vision sensors like Prophessee's GenX320 feature low power consumption, excellent low-light performance, and an event-driven operation. Rather than capturing continuous frames, these sensors record changes in the scene. Such an approach makes them highly beneficial for applications requiring motion detection and presence monitoring without compromising user anonymity. However, developers typically train off-the-shelf machine learning models for object detection with more feature-rich RGB frame-based datasets such as COCO. COCO (Common Objects in Context) is a widely-used large-scale object detection, segmentation, and captioning dataset comprising over 200,000 labelled entries across 80 object categories [1]. Consequently, these models require significant modification and optimisation for adequate performance with neuromorphic sensors. The STM32F746G-DISCO (F746G) microcontroller board from STMicroelectronics (ST) offers a good pairing with the GenX320 sensor due to its small footprint and low power draw. Despite these advantages, using a microcontroller comes with extra constraints: only 320KB of internal RAM and 1024KB of internal flash, an ARM Cortex-M7 processor running at 216MHz, and, critically, the absence of dedicated hardware acceleration for CNN inference. Although additional external memory (16MB RAM and 16 MB Flash) is available, its use is discouraged due to potential latency and power efficiency loss. This project, therefore, aims to adapt and optimise a CNN-based person detection model to function within these constraints while achieving a reasonable detection speed. Since ST trained their standard models on the COCO dataset, there is no direct comparison to the new model's performance. Therefore, a significant loss in accuracy refers to a substantial reduction in the system's effectiveness for practical, real-time applications.

Fig. 1: STM32F746G-DISCO with GenX320 sensor attached

## II. Background and Related Work

Microcontroller-based object detection models require extensive model optimisations to meet memory and flash constraints while retaining real-time performance. Therefore, deploying a standard object detection model like YOLO (You Only Look Once) [2] without reduction is impossible A prior survey for deep neural networks has found that post-training quantisation (PTQ) is highly effective at compressing CNN detectors with minimal accuracy losses. Quantising from float32 to int8 (or lower) can reduce inference speed significantly [3]. For example, TinissimoYOLO used quantisation-aware training to reduce its 442k parameter YOLO-derived model to less than 0.5MB of space while retaining reasonable accuracy (≈55.9% AP@0.5 IoU on the VOC dataset)[4]. The optimisation enabled deployment on various microcontrollers, such as the STM32H7 and Apollo4b. Notably, their most significant efficiency increase came from using a microcontroller with a dedicated CNN hardware accelerator (Maxim MAX78000), which allowed TinissimoYOLO to achieve

180FPS. However, even on general-purpose hardware, like the STM32H7, the model achieved ≈2.7 FPS by utilising int8 quantisation [4].

Another way to fit complex detectors onto tiny devices is to simplify the machine learning model's architecture. Removing unwanted or redundant heads is a common strategy with a YOLO model. Since YOLO's multiple heads target objects at different scales, even single-scale models can be sufficient for detecting a single type of object. Researchers found that removing one of YOLO's three output heads had little effect on output accuracy while significantly reducing inference speed and memory footprint [5]. Other tiny models like TinissimoYOLO only use one output grid to shrink the model even further [4].

Another optimisation explored is reducing the input size, which can occasionally be the largest layer in a model. Converting the input from a typical RGB image to greyscale reduces the input size by 3x. For person detection, greyscale could perform similarly to RGB since the task relies heavily on shape, contrast, and texture [6]. Typically, the main disadvantage of using greyscale images rather than RGB images is that the model can sometimes use colour to distinguish between objects that are very similar in shape. Since a neuromorphic vision sensor does not output colour information, the colour must be artificially generated to utilise each RGB channel. If trained on this data, the model could pick up on unintentional colour data, resulting in worse real-world performance.

A smaller input also simplifies preprocessing the neuromorphic vision sensor's output. Unlike typical RGB cameras, which are frame-based, neuromorphic vision sensors output asynchronous events whenever a pixel's brightness changes [7]. Typically, these asynchronous events only convey binary information, indicating whether a specific pixel's brightness increases or decreases. One common approach to using an event-based output with a CNN is to combine events over a short time interval into a frame. The prior group working with this camera also took advantage of this approach by using Prophesee's pre-made starter kit for the GenX320 [8]. The starter kit is a repository containing pre-made drivers and example code for using the camera on an STM32F7. It utilises these "frames" to show the camera output on the integrated screen. While the previous group used this output for MNIST detection, studies have shown that machine learning models can use frames created from such event streams in scenarios where the camera is in motion [7].

Finally, the MNIST model the previous group developed had a few issues that made it unusable in real-world scenarios. The main two problems were that the characters shown to the camera had to be in a specific place in the frame and at a particular size and rotation to be detected. By randomly transforming any training data so that the model does not become too dependent on the relevant objects' size and location, these effects can be mitigated [9].

## III. Model Selection

Typically, CNNs made for object detection are large and slow. Researchers have recently tried to shrink these networks to make them more efficient and fit onto small devices. ST provides a repository called ModelZoo, which contains many different machine learning models for various tasks, as well as services to help use and maintain them [10], [11]. These machine-learning models are already relatively compact. Moreover, ST have tested them to ensure they work on many STM microprocessors. While they offer models that can do various things, such as image classification and hand posture recognition, this paper will only focus on the models concerning object detection.

ModelZoo provides six models specifically for object detection: SSD Mobilenet v1/v2, YOLO LC v1, YOLOX Nano, Tiny YOLO V2, and YOLOv8n. Each model comes with two variants. One is trained on and can detect all 80 classes in the COCO dataset. The other is optimised for single-class inference and pre-trained for person detection. Removing the other 79 classes from the model significantly reduces its size and inference latency [12]. There is also a slight boost in Average Precision (AP%), as training can optimise the model from the earliest stages to detect a single class rather than deferring class-specific learning to the final layers. Throughout the remainder of this work, unless stated otherwise, all models mentioned refer specifically to their single-class variant pre-trained for person detection.

Each model's performance needs to be analysed to select between the models provided by ST. Due to the constraints caused by using the F746G, the most critical metrics are RAM size, Flash size, inference latency and AP%. The ST Edge Developer Cloud can be used to determine these values [13]. This online tool allows for testing machine learning models in the ModelZoo (and others) on specific STM microcontrollers and gives detailed performance reports. While it provides most metrics, it cannot provide any data regarding the accuracy of a model. Therefore, the AP% reported in the ModelZoo model description was used for evaluation [12]. Table I compares all models, with an input size of 192×192×3 and an AP@0.5 IoU on the COCO person dataset, quantised to int8.

| Model | Flash (KB) | RAM (KB) | Inference (ms) | AP (%) |
|---|---|---|---|---|
| SSD Mobilenet V1 | 591 | 303 | 274 | 35.8 |
| SSD Mobilenet V2 | 1290 | 662 | 1101 | 40.7 |
| YOLO LC V1 | 336 | 178 | 318 | 39.0 |
| YOLOX Nano | 1087 | 231 | 630 | 45.1 |
| Tiny YOLO V2 | 10240 | 263 | 7670 | 33.7 |
| YOLOv8n | 3144 | 382 | 2229 | 56.9 |

TABLE I: Benchmarks for all models [13]

Upon examining Table I, several models can be ruled out based on resource constraints. Tiny YOLO V2 has a substantial flash footprint and inference latency, making it unsuitable for this use case. Although SSD Mobilenet V2 and YOLOv8n achieve relatively high AP%, their initial flash size, RAM usage, and inference times greatly exceed the acceptable limits for this context. On the other hand, SSD Mobilenet V1 has a flash and RAM size that would immediately fit into the constraints. However, efficiency comes at the cost of accuracy, with an AP% approximately 20% less than the most accurate remaining model, YOLOX Nano.

The remaining two models both show promise. Both fit or are close to fitting into the tight memory and flash constraints of 320KB and 1024KB, respectively, and are close in AP%. While YOLO LC V1 may seem like the obvious choice due to its much lower Flash footprint and RAM size, in real-world testing, it was noted that it had significantly worse performance in identifying people. This reduced performance is likely due to the original YOLO LC model being optimised for detecting small objects such as UAVs, making it less suited for person detection [14]. As a result, YOLOX Nano was selected for this project.

Moreover, the architecture of YOLOX Nano is based more on modern design principles, making it more adaptable and better suited for further optimisation. The model contains three sections: a backbone for feature extraction, a neck for multi-scale feature fusion and three detection heads to output the result.

One of YOLOX Nano's key features is its decoupled detection heads, meaning the classification and bounding box regression predictions are made in separate branches, as shown in Figure 2. YOLOX implements these heads on three scales, where each head outputs at a different scale (1/8, 1/16, 1/32 referred to as P3, P4, and P5, respectively). After receiving the three feature maps from the neck, each head splits into three parallel branches: a regression branch in charge of determining the centre x,y coordinate of the object, as well as its width and height in the frame; an objectness branch used to assess the probability of an object being in this bounding box; and a classification branch, which predicts the class of the object. Decoupled heads dramatically improve convergence speed and allow it to achieve a much better speed-accuracy trade-off compared to prior YOLO models [15].

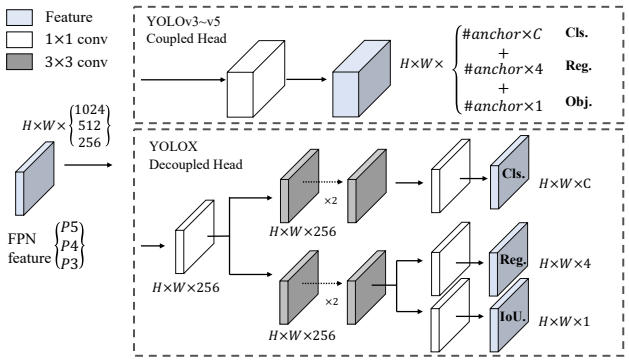



Fig. 2: Comparison between YOLOv3-v5 coupled head and YOLOX decoupled head [15]

YOLOX Nano also uses an anchorless design. Traditional YOLO models (v3/v4/v5) would predict scales for multiple preset anchor boxes per cell [2]. However, YOLOX utilises centre sampling, meaning that only predictions whose centre cell falls into the middle of a ground truth box are considered positive samples. Removing the need for preset anchors allows the output layer to be simpler (since it has fewer boxes to predict) and permits the model to generalise better to different object sizes without manually tuning the anchor boxes' sizes [15].

## IV. Optimizing the CNN

Even when using the most miniature version of the model, with an input size of 192×192×3, its flash footprint exceeds the internal flash of the F746G by ≈30 KB. Moreover, even though the RAM usage of 231 KB appears to be comfortably under the 320 KB internal limit, the actual usable memory is significantly lower (around 230 KB) due to RAM allocation to other system sections, such as processing the neuromorphic sensor's output and the display. With this in mind, the RAM usage must also be reduced by at least 1 KB. To decrease YOLOX Nano's size, further refinement is needed.

### A. Grayscale Input

One of the most memory-intensive components is the input image. Even though it is not part of the model, with an input size of 192×192×3, it occupies ≈110 KB of RAM, making it an essential factor when assessing memory budget.

Typically, traditional RGB cameras require the input to have a depth of 3 to encode proper colour data for the model to use. However, with a neuromorphic vision sensor, colour information is already lost. Therefore, a grayscale implementation is preferred and can reduce the input size by ≈74 KB.

To achieve greyscale input, the input tensor was changed from shape (batch size×192×192×3) to (batch size×192×192×1), and the first convolution was modified accordingly to accept this new input. Initially, the kernel had dimensions (3×3×3×12). While the first two elements represent the width and height of the kernel, the third and fourth elements represent the input and output channels, respectively. Therefore, the shape of the kernel was changed to (3×3×1×12), adjusting the number of input channels but retaining the number of output channels to maintain compatibility with the rest of the model. To retain some data from training, the three input channels were combined using the standard luminance-based grayscale conversion method [16].

### B. Head Removal

Internally, YOLO models have three output heads optimised for objects at different scales. P3 typically detects small objects like animals or fruits, P4 targets medium-sized objects like people or bikes, while P5 focuses on large objects like buildings and occasionally people [17]. This bias to the P4 head can be seen in Figure 3. Even though the pre-trained model is only trained on the "person" class, detections are seen solely on P4, even at small object scales.

Removing the unused P3 and P5 heads could lead to significant performance and memory improvements without sacrificing accuracy. Notably, operations dedicated to the P3 head and P5 head contribute approximately 25k and 225k parameters to the model's 900k parameter total, making them substantial contributors to the model's complexity.

### C. Class Probabilities

Another way to optimise the detection heads is to remove any unnecessary components. YOLO models are

Fig. 3: Detections made by STMicroelectronics' YOLOX Nano labelled with detection head

typically designed for multi-class detection and, therefore, include a dedicated classification branch to predict the most likely class associated with a given bounding box. Classification usually uses a softmax layer, which outputs a probability distribution summing to 1 over every class. With only one class, this probability distribution is always 1, meaning that the classification branch is no longer needed for this use case and was removed. Since the class is always known, the model uses the objectness score (IoU) alone to determine whether an object ("person") is recognised within the designated bounding box.

### D. Profiling-based Optimisation

Resource usage analysis tools like Netron were used to delve deeper into what is causing the model to use up so much memory space. It was observed that the activations of the first three convolutional blocks used up a significant amount of memory space (≈153 KB) and contributed to ≈17% of the total inference time, making them the most significant contributors. Reducing these layers would result in a clear performance improvement. On the other hand, the memory savings will not be one-to-one as optimisation tools often use in-place memory reuse. This technique reuses a minimal activation buffer rather than allocating a separate memory buffer for each intermediate layer.

These first few layers focus the input, reducing its resolution while increasing its channel depth. Then, a "stem" convolution expands the channel count to the desired width. The output is then sent to the CNN's backbone. Since a neuromorphic sensor captures less variation and detail than an RGB camera due to its sparse output, layer fusion can be used to combine these layers.

The original layers use 3×3 convolutional kernels, with two layers configured with a stride of 2 and one with a stride of 1. These were replaced with a kernel of size 7×7 with a stride of 4. This size allows the same input data to represent each output feature.

### E. Quantisation

The float32 version of the machine learning model must be used to implement optimisations and perform training. Therefore, quantisation must be applied after training to reduce the model size. The current, most popular way of performing PTQ is by quantising the weights from float32 to int8. Developers perform PTQ with a representative dataset for more accurate activations and weight mapping. Going a step further, int4 PTQ, a new and experimental technique for ultra-low-bit quantisation, could further halve the size of the weights at the cost of accuracy.

## V. Experimental Setup

Before evaluating the optimised models, a suitable testing environment must be created. It must support all the custom models and train, quantise, and assess them properly. The University of Twente's development environment, JupyterLab, was used for training and evaluation [18]. This service provides multiple GPUs, which can speed up training and inference.

### A. Tool Modification

ST provides various tools for training, evaluating, quantising, and testing different machine learning models for their hardware. These are contained in the ModelZoo services repository [11].

The ModelZoo training service can only run on a single GPU by default. To speed up training times by utilising multiple GPUs, the training function in Python was wrapped in a `MirroredStrategy`. The implementation allows TensorFlow, the main package used by the service to train and evaluate models, to use all available GPUs.

Even though YOLOX Nano models are already supported in the repository, many files need to be modified to support the optimisations made. Notably, both RGB and greyscale model variants are already supported. However, the trainer, evaluator, and quantiser expect the model to have three separate heads and a class confidence output. A fully modified version of the ModelSoo services can be found in Appendix B, which removes these restrictions.

The quantisation tool supported by default in the Modelzoo services is the TensorFlow Lite (TFLite) Converter. However, as of this writing, the TFLite Converter does not support int4 PTQ, a relatively new and experimental technique. Therefore, an alternative optimisation library must be implemented. While Intel's Neural Compressor does not natively support int4 quantisation by default, it can accommodate new data types like int4 with minimal modifications [19], [20]. By installing the `neural_compressor` package and modifying the `tensorflow.yaml` file, int4 quantisation can be enabled. A custom function (`_quantize_neurComp_model`) within `quantise.py` then provides the ModelZoo service with the functionality to perform int4 quantisation.

### B. Dataset Preparation

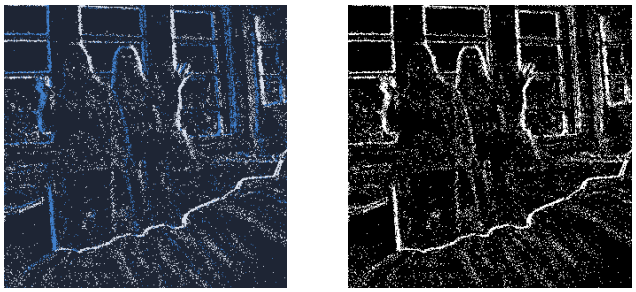
With all modifications, the training, validation and testing datasets can be sourced and adapted for the project's needs. A COCO dataset subset containing only entries with the "person" class will be used to validate STMicroelectronics' accuracy claims. To extract this subset, Modelzoo services include a tool called `dataset_converter`, which converts the COCO dataset into the proper YOLO format and handles subset selection.

A greyscale version of the subset of the COCO dataset needs to be created to fine-tune models when converted to greyscale. The modification was done with a Python script (`convertrgbtogreyscale.ipynb`, included in Appendix B), which goes through all JPGs in a directory

and converts them to greyscale. Since the positions and classes of the objects do not change, the conversion tool does not need to modify the labels.

Finally, the PEDRo dataset is the basis for fine-tuning the model for use with the neuromorphic sensor [21]. PEDRo is a dataset containing over 43000 entries specifically designed for person detection. The DAVIS346, an event camera with a similar output as the GenX320, was used to collect it. One limitation of this dataset is that the camera outputs at a resolution of 346x260. Since the input of the YOLO model has a square aspect ratio, these images must be scaled and stretched to fit. These distortions could result in worse real-world performance with pictures from the GenX320.

Moreover, the pictures in the dataset do not come in the needed JPG format but as many different numpy samples, each representing a frame that contains all of the events within 40 milliseconds. Each event comprises four fields: a timestamp of when the camera took the event, its x coordinate, its y coordinate and the polarity of the event (either 0 for negative or 1 for positive) [21]. A Python script (`convertNumpyToImage.ipynb`, included in Appendix B) was created to convert these frames into JPG greyscale images. The program read each numpy sample and processed all events within it. If no sample were in a specific xy coordinate, that pixel would be pure black (0). However, if there were an event of either polarity, it sets the pixel as pure white (255). No distinction was made between the polarities, as polarity only represents whether the light in that region is getting brighter or dimmer and only depends on how the object is moving relative to the camera and not the object's shape. This effect can be seen in Figure 4. While the different polarities (represented as white and dark blue) have different polarities, both provide the same information for the edge of an object.



(a) Different coloured polarities

(b) Uniformly coloured polarities

Fig. 4: Image from GenX320 comparing two different colour modes

### C. Configuring the Environment

The ModelZoo service has a configuration file that defines how various operations function. ST provides an example configuration that can be used out of the box. However, some changes were made to fit the requirements of the intended training setup.

Since the images in the PEDRo and COCO datasets are not perfectly square, some configuration needs to be done for resizing. The process was set to "fit" the aspect ratio, which would stretch the image rather than crop it. This choice was made to preserve all of the information in the picture, but it does come with the downside of stretching the input, which could lead to inadequate training.

Data augmentation can also be defined to avoid overfitting. Many different parameters can be set to transform and change the brightness of the images used in training. These include cropping, rotating, translating, flipping horizontally and adjusting the brightness and contrast.

For the training step, models were trained with a batch size of 800, selected based on empirical testing to optimise training speed. Each model was trained on the grayscale COCO dataset and then fine-tuned on the PEDRo dataset.

Moreover, an early stopping patience was used to help mitigate overfitting. This patience stops training early if the model's loss on the validation data does not decrease after a certain number of epochs. The check was done on the loss of the validation dataset instead of the training dataset to ensure that performance improvements are generalised outside the training set. Due to its wide array of scenarios and contexts, the patience was set to 60 epochs for the COCO dataset. On the other hand, the PEDRo dataset comprises many images of people in the same context. So that the model does not "forget" the information gained from the COCO dataset, the patience was set much lower (18 epochs).

By default, the TFLite converter was used for PTQ. TFLite quantises the weights to int8 and the activations to int32. If quantisation to int4 was preferred, Intel's Neural Compressor was used. In both cases, the quantisation input type was set to uint8 to accept image luminance values from 0 to 255, and the output type remained float32 since all outputs are between 0 and 1. All evaluations were performed with an IoU evaluation threshold of 0.5.

Benchmarking the model's inference time was done using the ST Edge AI Developer cloud to ensure a consistent testing environment [13]. Since the final deployment will be on the F746G, this specific platform was selected on the Developer Cloud. To use the model on a local F746G, the camera input of 320x320 was downsampled to 192x192 through average pooling. In order to extract the outputs, non-max suppression was used to prevent duplicate detections for the same object.

## VI. Results and Analysis

After all optimisations were implemented and proper training was performed, each optimisation was tested. Table II details specifications about the different models. Optimisations were introduced sequentially, building upon the previous one so that the final performance reflects their cumulative impact. The exception to this is for the final two models labelled "No Head 1&2" and "No Head 1&3," which are both derivatives of the "Layer Comb" model with different heads removed. Int4 quantisation was not performed due to Intel's Neural Compressor expecting a more modern version of TFLite to create and train the models. It was not possible to update the TFLite version due to the dependencies of the ST ModelZoo Services [11]. Therefore, its performance could not be measured.

Figure 5 shows a gradual decrease in flash size, whereas the RAM footprint remains somewhat constant with a

| Model | Details of Optimisation Added |
|---|---|
| int8 | Quantised model to int8 |
| geryscale | Changed input to greyscale |
| No Head 1 | Removed the first head (P3) |
| No Class | Removed the classification branch |
| Layer Comb | Combined first three layers |
| No Head 1&2 | Removed the first and second head (P3, P4) |
| No Head 1&3 | Removed the first and third head (P3, P5) |

TABLE II: Model Details

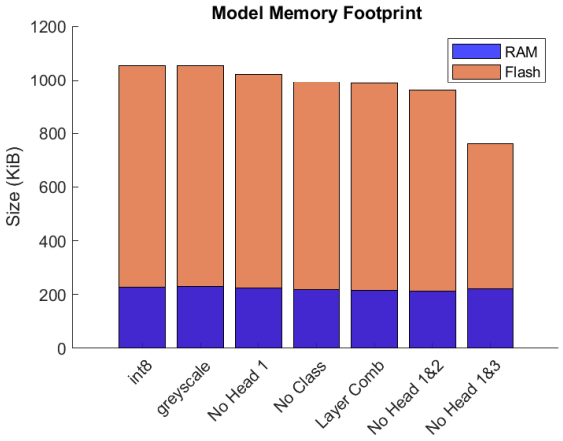



Fig. 5: Flash and RAM footprint of each optimised model

slight downward trend. Significant improvements to flash size come from removing detection heads and the classification branch. Removing P5 alone (model "No Head 1&3") reduces the flash footprint by roughly 200 KB, far more than removing the other two heads. This significant reduction can be associated with the additional layers in the neck needed to perform feature extraction for P5. Moreover, this model is the only exception to the RAM footprint's downward trend, increasing from 212 KiB to 221 KiB. This slight increase in RAM usage is most likely due to the memory optimisation performed by ST's tools before the model is loaded onto the board, as it is trying to find a balance between RAM usage and inference time.

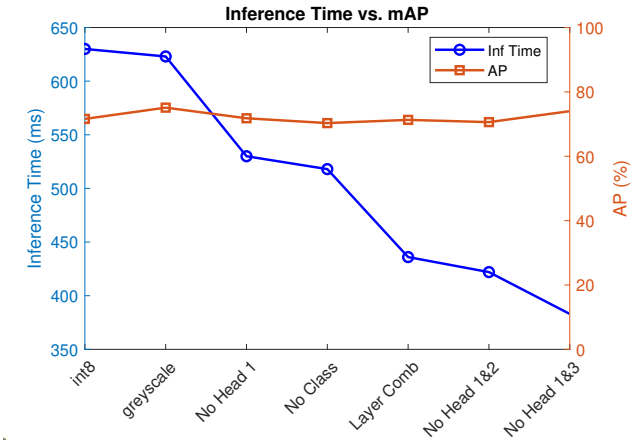


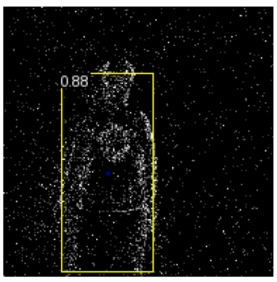

Fig. 6: Inference Time and AP (PEDRo dataset @0.5 IoU) of each optimised model

Figure 6 shows that one of the most significant drops in inference time is the removal of P3. P3 needs to predict 4x as many bounding boxes as P4 and 16x as many bounding boxes as P5. As a result, the omission of the P3 head leads to a much larger reduction in inference time. In contrast, removing the P5 head leads to slightly better performance than removing P4, despite P5 operating at a lower resolution. The observed performance gain is due to the considerable parameter reduction from removing the P5 head.

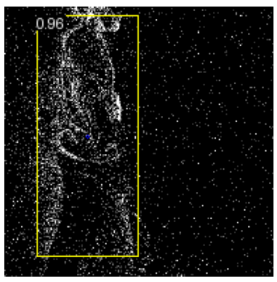
Another significant decrease in inference time is caused by the layer reduction, yielding a drop of roughly 12%. However, due to the aforementioned memory optimisation tool, the improvement does not come with much RAM savings, only dropping the memory usage by around 3 KB.

The AP of each model remains relatively constant. The most prominent improvement in accuracy comes from using the P4 detection head rather than the P5 head as the sole output. This increase stems from the fact that the P4 head is much more optimised to classify objects at the scale of a person.

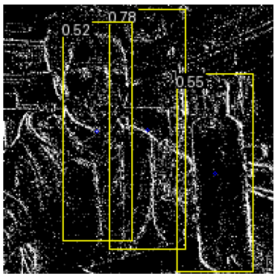
With a significant decrease in Flash footprint and inference time while retaining AP, the model using P4 as the sole detection head best fits the defined use case.



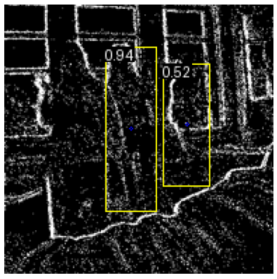

(a) Man standing, empty background, still camera




(b) Man walking, empty background, still camera




(c) Man sitting in an environment with lots of objects, moving camera




(d) Two people waving at the camera, moving camera

Fig. 7: Model inferences on images from the GenX320 sensor

To test accuracy further, inferences were run on a few sample images, as shown in Figure 7. The model handles detecting people well in environments where the camera is not moving due to little background information, such as in Figure 7a and 7b. However, when the camera moves, the sensor picks up more details in the environment since the lighting constantly changes. The model seems to struggle in such complex scenes, especially in environments with multiple objects or people. In Figure 7c, the model gets confused by the bottle on the table, and even though it detects the sitting person, it does so with relatively low

confidence, and the bounding box is not wide enough to contain the whole person. In Figure 7d, the model only detects one person with reasonably high accuracy but also detects their arm as another person. These struggles with detecting people in complex environments can be attributed to insufficient training data in diverse scenarios. Most of the images in the PEDRo dataset are similar since they usually come from a continuous video of a person walking.

Lastly, the finalised model was tested on the F746G. The model was successfully integrated alongside the existing camera and display code, fitting into the RAM and Flash constraints on the microcontroller. The inference time of the model was measured to be 422 ms, a slight increase from the 383 ms measured on the cloud platform. This reduction in performance could be due to multiple factors. First, to support the GenX320 camera, the F746G must run at a slower clock speed of 200 MHz instead of the 216 MHz at which the model was tested using St Edge AI Developer Cloud. Moreover, the Cloud environment tests models alone, without any interference. In actuality, the model shares processing resources with the neuromorphic sensor and display, reducing overall inference speed.

The inference time of 422 ms does not consider input or output processing, both necessary elements to use the model. Testing with these two elements gave a total inference time of 462 ms, allowing the model to achieve 2 FPS.

## VII. Future Directions

While the person detection model managed to reach 2FPS with only 221 KB of RAM and 761 KB of flash, with an inference time of 422 ms, various optimisations can still be made to increase its efficiency. If int4 quantisation is successfully implemented, it could significantly reduce RAM, Flash and inference time. Its drawbacks in accuracy would have to be carefully assessed. However, further quantisation of the model might be viable due to the observed small accuracy drop from float32 to int8.

Another optimisation that can be made is to remove any unnecessary elements from the RGB implementation further. While adjusting the first layer of the model was the only necessary step, the depth of later layers could be reduced to save space and inference time.

Currently, the neck of the model contains the layers which take up the most inference time and storage space. Targeting optimisations to this area could significantly decrease inference latency while reducing the size of the weights.

Furthermore, model training was done with the PEDRo dataset. While this dataset is made from a neuromorphic vision sensor, the camera used to record the dataset did not have a square aspect ratio. This meant the pictures needed to be stretched to train the model with the dataset, which skewed the data. While this resulted in adequate real-world performance, using a dataset created from the GenX320 could give much more accurate results.

To take better advantage of the benefits that come from using a neuromorphic sensor, like the GenX320, an adaptive frame rate could also be implemented. The camera would only generate a frame for inference if it detected several events over a certain threshold. Therefore, this implementation would save energy consumption since it would not process frames unnecessarily.

Finally, an easier way to improve performance is to use a more up-to-date STM board with higher clock speeds and more RAM/Flash storage. Most of the tests made by STMicroelectronics for their machine learning models are done either on an STM32H7 board or an STM32N6. While the STM32H7 is only an improvement on the STM32F7 [22], having a higher clock speed as well as a larger RAM and Flash size, the STM32N6 contains their neural-processing unit (NPU), designed explicitly for neural network inference in computer vision applications [23].

## VIII. Conclusion

Neuromorphic vision sensors provide many benefits compared to traditional RGB cameras, such as improved power efficiency, low latency and preserved anonymity. However, these event-based cameras pose significant complications when used with typical CNNs made for object detection, as they are generally optimised for and trained with RGB datasets, such as COCO. Moreover, to take advantage of the neuromorphic sensor's low-power performance, they must be paired with low-powered hardware, such as a microcontroller. Despite their benefits, using microcontrollers also comes with many limitations in processing power and memory. Therefore, this thesis aimed to optimise and train a CNN for inference on an STM32F746G-DISCO microcontroller using a GenX320 neuromorphic vision sensor as an input. Reductions to the RAM and Flash footprint were achieved by shrinking the model's input size, removing multiple of the model's output heads, removing the classification branch and performing layer fusion. With these optimised layers in place, the model achieved an inference time of 422ms, with an accuracy of 75% (@0.5 IoU) on the PEDRo dataset. In addition, the model ended with a flash and RAM size of 761 KB and 221 KB, respectively, allowing it to be used in the internal flash and RAM of the microcontroller alongside additional processes. While other optimisations could be made to the model for increased improvements, the finalised model achieved all requirements necessary for deployment without compromising real-world accuracy.

## IX. References


[1] "COCO - Common Objects in Context." [Online]. Available: https://cocodataset.org/#home

[2] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," Dec. 2016, arXiv:1612.08242 [cs]. [Online]. Available: http://arxiv.org/abs/1612.08242

[3] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Aug. 2017, arXiv:1703.09039 [cs]. [Online]. Available: http://arxiv.org/abs/1703.09039

[4] J. Moosmann, M. Giordano, C. Vogt, and M. Magno, "TinyissimoYOLO: A Quantized, Low-Memory Footprint, TinyML Object Detection Network for Low Power Microcontrollers," in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Jun. 2023, pp. 1–5, arXiv:2306.00001 [cs]. [Online]. Available: http://arxiv.org/abs/2306.00001

[5] E. Humes, M. Navardi, and T. Mohsenin, "Squeezed Edge YOLO: Onboard Object Detection on Edge Devices," Dec. 2023, arXiv:2312.11716 [cs]. [Online]. Available: http://arxiv.org/abs/2312.11716

[6] G. , "How do you decide whether to utilize grayscale or colour images as input for computer vision tasks?" section: Blogathon. [Online]. Available: https://www.geeksforgeeks.org/how-do-you-decide-whether-to-utilize-grayscale-or-colour-images-as-input-for-computer-vision-tasks/

[7] W. Shariff, M. A. Farooq, J. Lemley, and P. Corcoran, "Event-based YOLO object detection: proof of concept for forward perception system," in *Fifteenth International Conference on Machine Vision (ICMV 2022)*, J. J. Zhou, W. Osten, and D. P. Nikolaev, Eds. Rome, Italy: SPIE, Jun. 2023, p. 12. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/12701/2679341/Event-based-YOLO-object-detection--proof-of-concept-for/10.1117/12.2679341.full

[8] "Starter Kit – STM32F7 (GenX320)." [Online]. Available: https://support.prophesee.ai/portal/en/kb/articles/stm32-discovery-board-for-genx320

[9] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, Dec. 2019, number: 1 Publisher: SpringerOpen. [Online]. Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0

[10] "STMicroelectronics/stm32ai-modelzoo," Apr. 2025, original-date: 2023-01-10T13:58:28Z. [Online]. Available: https://github.com/STMicroelectronics/stm32ai-modelzoo

[11] "STMicroelectronics/stm32ai-modelzoo-services," Apr. 2025, original-date: 2024-11-12T13:43:27Z. [Online]. Available: https://github.com/STMicroelectronics/stm32ai-modelzoo-services

[12] "stm32ai-modelzoo/object_detection at main · STMicroelectronics/stm32ai-modelzoo." [Online]. Available: https://github.com/STMicroelectronics/stm32ai-modelzoo/tree/main/object_detection

[13] S. , "ST Edge AI Developer Cloud." [Online]. Available: https://stedgeai-dc.st.com/home

[14] M. Cui, G. Gong, G. Chen, H. Wang, M. Jin, W. Mao, and H. Lu, "LC-YOLO: A Lightweight Model with Efficient Utilization of Limited Detail Features for Small Object Detection," *Applied Sciences*, vol. 13, no. 5, p. 3174, Jan. 2023, number: 5 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2076-3417/13/5/3174

[15] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO Series in 2021," Aug. 2021, arXiv:2107.08430 [cs]. [Online]. Available: http://arxiv.org/abs/2107.08430

[16] "Grayscale," Feb. 2025, page Version ID: 1276359132. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Grayscale&oldid=1276359132

[17] "COCO Evaluation metrics explained — Picsellia," Aug. 2023. [Online]. Available: https://www.picsellia.com/post/coco-evaluation-metrics-explained

[18] "Research support: Jupyter | JupyterLab | Jupiter | Cloud computing | Service Portal | University of Twente." [Online]. Available: https://www.utwente.nl/en/service-portal/research-support/it-facilities-for-research/jupyterlab

[19] "Perform Model Optimization Using Intel® Neural Compressor." [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/oneapi/neural-compressor.html

[20] l. , "How to Support New Data Type, Like Int4, with a Few Line Changes — Intel® Neural Compressor 3.2 documentation." [Online]. Available: https://intel.github.io/neural-compressor/latest/docs/source/add_new_data_type.html

[21] "SSIGPRO/PEDRo-Event-Based-Dataset," May 2025, original-date: 2023-04-12T20:31:32Z. [Online]. Available: https://github.com/SSIGPRO/PEDRo-Event-Based-Dataset

[22] "STM32H7 - Arm Cortex-M7 and Cortex-M4 MCUs (480 MHz) - STMicroelectronics." [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32h7-series.html

[23] "STM32N6 series - STMicroelectronics." [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32n6-series.html

## APPENDIX A
## AI STATEMENT

During the preparation of this work, I used Grammarly to perform spell and grammar checking as well as improve the work for readability. ChatGPT was used to aid in coding. After using these tools/services, I thoroughly reviewed and edited the content as needed, taking full responsibility for the final outcome.

## APPENDIX B
## CODE REPOSITORIES

The following links lead to various repositories that were important to this project.

Used to optimise the models and prepare the datasets:
https://github.com/piergatt/YoloOptimiseTools

Used to train, quantise and evaluate the models:
https://github.com/piergatt/stm32ai-modelzoo-services-generalised_object_detection

Used to test and deploy model on STM32F746G-DISCO (Request access with a.yousefzadeh@utwente.nl):
https://gitlab.utwente.nl/s3014940/objectdetectiononf746g