Final Project Report:

Food Detection System

**UTDesign Fall 2021**

**UTDesign Team #1306**

Omar Resendiz (CE)

Chujie Guan (EE)

Mariela Ramirez (CE)

Karthik Gopan (CE)

Jordan Rider (CE)

Corporate Sponsor:

Maxim Integrated

Corporate Advisor:

Mr. Brian Rush

Instructor:

Dr. Marco Tacca

Date: December 19th, 2021

Abstract:

The main goal for our project is to use our sponsor’s board, the MAX78000 featherboard, as the hardware for a neural network we developed to identify 10 different food types using the onboard camera. To create our machine learning neural network, we develop and train it on our own PCs then use Maxim’s synthesizer to create C code to upload it onto the board. We train and adjust this model until we are satisfied with the results.

Table of Contents:

1. Introduction
2. Review of Conceptual and Preliminary Design
3. Basic solution Description
4. Performance optimization and design of system components
5. Testing Procedure
6. Project implementation/Operation and assessment
7. Conclusion, overview of work statement
8. Future work recommendations
9. Project Management Summary
10. Introduction

As technology continues to advance, the various applications of image recognition and processing continue to prove their value. Some of the important applications of image processing in the field of science and technology include computer vision, face detection, augmented reality, X-rays, and Ultrasonic scanning. This project uses a MAX78000FTHR board to detect various types of food with the use of a convolutional neural network. The final objective for this project is to create an AI neural network that can identify foods from images taken by the camera on the MAX78000 feather board. The program must run on the board itself, with the output being shown on a computer terminal. This project means to showcase the capabilities of a low power solution for Convolutional Neural Networks.

1. Review of Conceptual and Preliminary Design

For this project we need to create and train a neural network from a data set of food images, synthesize the neural network and verify functionality, run a trained model on MAX78000FTHR board, and create documentation for the project. The neural network must be able to identify at least 10 different food types from various angles. Must be able to identify foods using the integrated camera on the MAX78000. Output results to the terminal on the host computer/or on the onboard screen that was soldered on. We have prepared our MAX78000FTHR board to use an LCD display. A neural network has been created to train off of a data set that is scaled to 64x64 and artificially expanded through image augmentation.

1. Basic Solution Description

The results we expect to achieve are to create a machine learning AI using pytorch and modified MAXIM code to identify at least 10 different foods from various angles, and to have the LCD show the camera feed and label the food item shown (either what food it thinks it sees or “unknown” for things that aren't in the data set.

1. Performance optimization and design of system components

Development of the Neural Network:

To start out development of the neural network, we used examples provided by Maxim as a base. We used their cats\_vs\_dogs example as it was the one recommended by our mentor as it has a simple kind of use case: using the camera to take a picture and then feed that picture through the convolutional neural network to try and identify what it is. The example program however only has to identify two things, so we expanded it to identify 10 different items and as such had to adjust the amount of layers we could feed the image to. We had to cut the amount of layers down to 9, and limit the size of the image as otherwise we ran into memory limitations. After our adjustments, we managed to get something working at the end where it could identify a food type at least 95% of the time with 90% or higher accuracy.

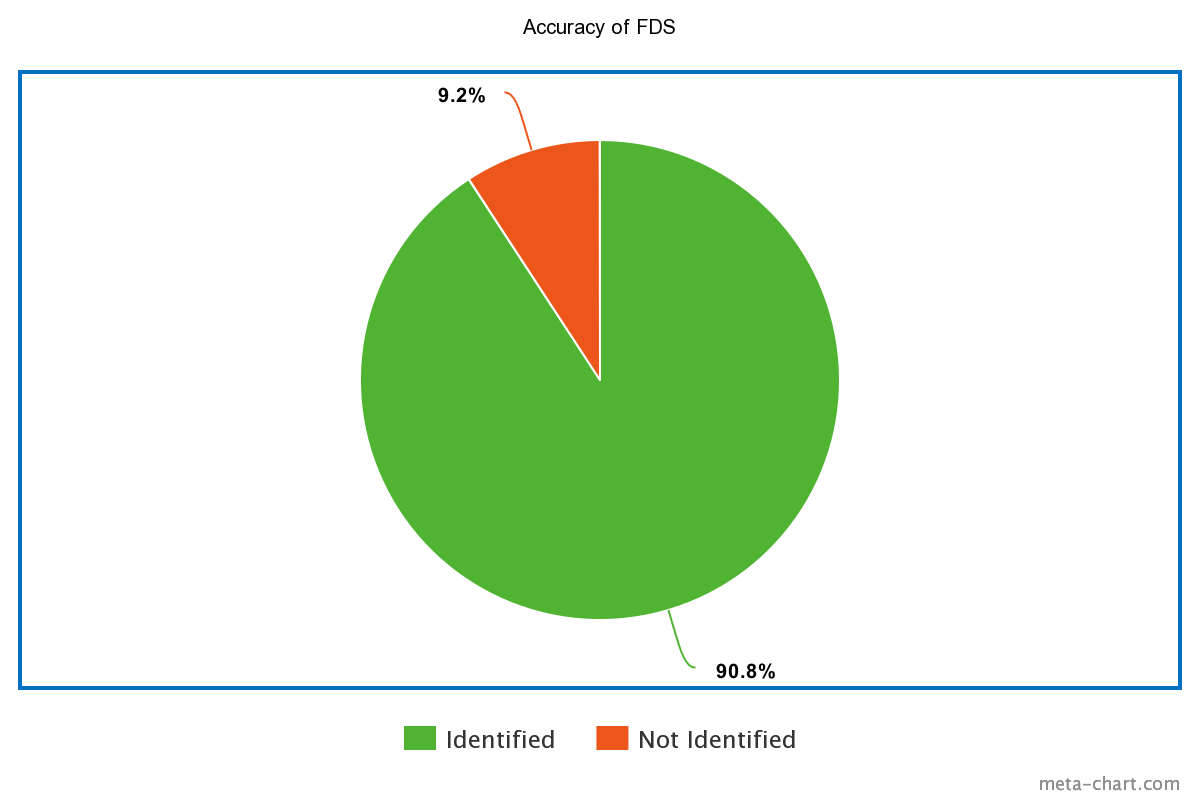
Dataset Image Augmentation:

The dataset we used consisted of 10 food types with roughly 1000 photos within each food type. Because we were dealing with a fewer number of food types, we wanted to ensure the model was as confident with those food types as possible when making its decisions, so we chose to expand our dataset. Finding an additional 1000 images of any one food type can be tedious when thinking of the various angles and lighting to be taken into consideration. Our solution was to use PyTorch to create code that would duplicate and augment our current dataset. By translating images along either x or y axis, rotating them various degrees, and adjusting brightness and contrast, we were able to successfully artificially expand our dataset by 3-4 times.

1. Testing Procedure

Accuracy Testing:

For our testing, we focused on prioritizing accuracy. We tested the accuracy of our model by first training it on our own computers, adjusting the model after compiling and training it, and by adding more images to our dataset. Out of our entire dataset, we organized it 80/20 where 80% of the images would be used for training the model and the remaining 20% would act as the test. Once we felt like the model was decently accurate enough on our computers, we then transitioned the code into the synthesizer and then uploaded the code to the MAX78000 board and tested it out by taking pictures and seeing what the output was on the terminal.



Plugging our data (of all of the tests) into excel, we came out of final testing with an average of 90.4% across all 10 food types, which we believe to be acceptable for the board.

Another metric, that wasn’t a priority, was the speed at which the board identified foods. From our testing this turned out to be very fast, under 500 milliseconds every time for the data to show up on the terminal screen.

1. Project implementation/Operation and assessment

This project can be implemented into food/photo identification operations, as the the main scope of this project is to realize the potential of Maxim’s 78000 FTHR microcontroller. This device also features voice to speech identification, which allows for seamless control over the microcontroller if it was hooked to a microwave, for example. In one situation, the microcontroller detects the food and identifies it. It then realizes how long the food should be microwaved for and cooks it for that period of time.

For training, we used a dataset that was artificially expanded using Imageio and Opencv to create a training set and test set. The model was then trained using augmented training set on an Ubuntu machine. Necessary scripts are run that dictate model training parameters.

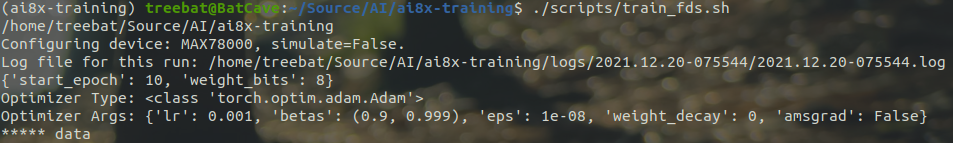


Fig 2: Model training on Ubuntu

Quantization of the training model was necessary as the model was too large and we wanted to produce a smaller model. We were able to determine and evaluate the model accuracy by using the test set. Based on the evaluation results, we decide if any model or dataset alternation are needed. When conducting the project synthesis we require a random sample image from the training dataset to evaluate the trained model, quantized model, and a network description of the model.

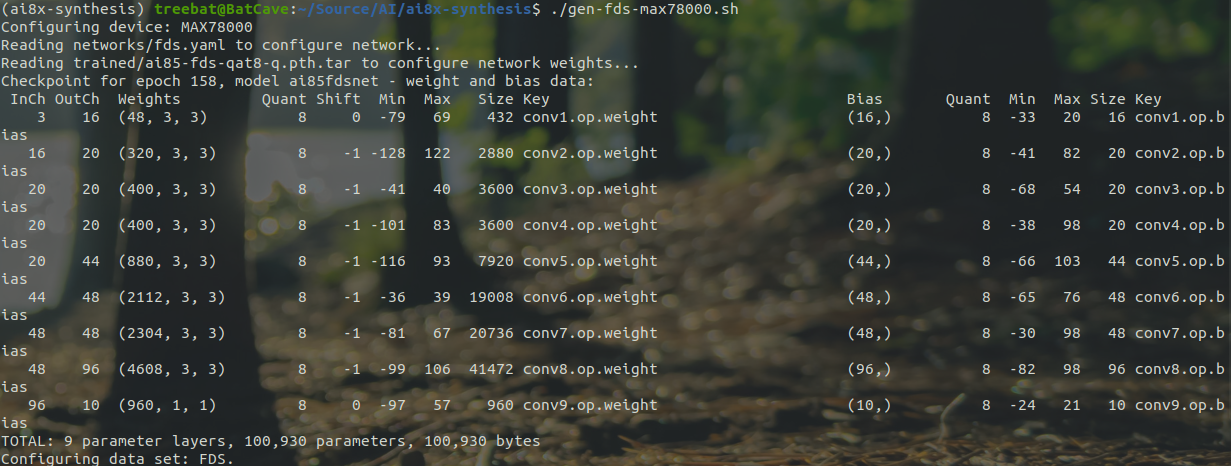


Fig 3: Model synthesis to generate C code

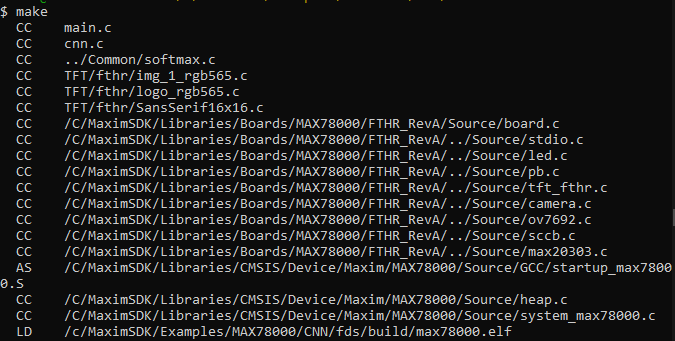


Fig 2: Run Makefile to generate programmable code

Once the project is synthesized using, the C files generated are used to make the programmable code for the MAX78000FTHR board. Through MinGW, we are able to use OpenOCD to connect to the board. Once connected we upload code onto the board first by halting all processes then loading programmable files onto the board. Once programmed the board feeds random sample images to the model and outputs results through the serial monitor on Putty. Using the board’s built-in camera, an image is created and fed into the model. The food item is identified and named if known.

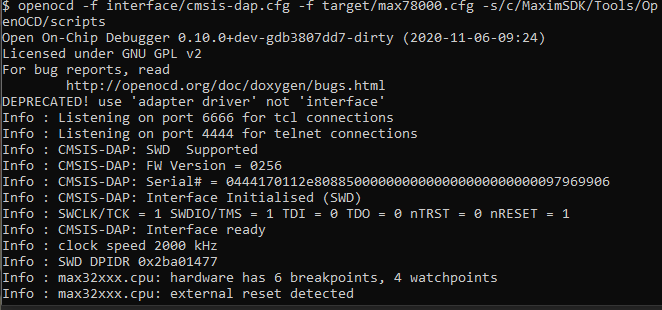


Fig 3: Interfacing into board using OpenOCD on MinGW

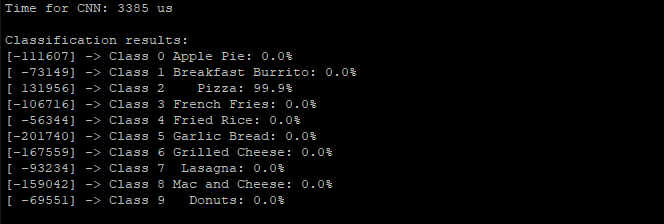


Fig 4: Results from sample image fed into trained model

1. Conclusion, overview of work statement

In conclusion, the team managed to implement the main functions as were required by MAXIM when they sent out the project specifications. We set out to be able to identify food items using the MAXIM board and the onboard CNN accelerator and narrowed our scope to identify 10 food items that could be considered microwavable foods, in order to fit into the constraints of the hardware. Every member on the team gained knowledge on working with machine learning software and hardware and we believe we have become better engineers from our time working between ourselves and with MAXIM.

1. Future work recommendations

For the future, we would recommend fine-tuning the model and code to see how many more layers you could fit onto the board without it running out of memory. We’re sure there is room for improvement, as we are not as familiar with the board as MAXIM is.

1. Project Management Summary

Milestones:

* Created basic ai8x training neural network using augmented dataset
* Created custom class to be used with pytorch.datasets
* Performed convolutional neural network programs on Eclipse IDE
* Achieved an accuracy of 90% when identifying food items
* Artificially expanded dataset by 3~4 times with our custom image augmentor

1. Ethics:

This project is for educational purposes and improving our understanding of its appropriate applications. We acknowledge that we use open source software and modify some existing code. Our system is designed to function within the scope of our project and is inspired from existing MAXIM projects.

1. Lifelong Learning

Through the use of deep learning software and the training of models, every single member of the team has gained experience in the process of AI learning. The use of convolutional neural networks is very promising for the future in robotics and artificial intelligence as it proves itself to be a valuable skill. The team also learned to work with each other despite different schedules and availability through frequent communication. This team-work skill is important in all aspects of work that allow for increased productivity. In addition to valuable skills learned, we were also given the opportunity to have real world experience as a development team working closely with a company. This experience has been invaluable for all members of the team as we were able to use previously learned knowledge to a real world application, and it's even given some of us insight into possible career paths

1. Multidisciplinary teams

Omar Resendiz- Main neural network primary developer (Developed neural networks/Trained datasets)

Karthik Gopan- Main neural network secondary developer (Developed neural networks/Trained datasets)

Chujie Guan- Board testing and adjusting (Assembled display board onto microcontroller, test outputs)

Mariela Ramirez- Board testing and adjusting (Assembled display board onto microcontroller, test outputs)

Jordan Rider- Data augmentation code (Increased dataset by duplicating and changing parameters of images to produce more variety in images)

1. Contemporary issues

Memory Limitations: When we ran our initial dataset of 101 food types with a total of 101,000 images, we ran into memory limitations as well as a very long training time. We decided to cut the amount of layers down to only 9 in order to decomplexify our CNN and effectively shortened our training time. We also restricted the size of the images as well as shortened our data set to only 10 food types to resolve the issue regarding memory limitations.

1. Standards

In this project we strived for accuracy on the identification system. We were determined to have an accuracy of at least 90% when identifying food items.

1. Appendices

Synthesized C Code:

#include <stdlib.h>

#include <stdint.h>

#include <string.h>

#include <stdio.h>

#include "mxc.h"

#include "gcfr\_regs.h"

#include "cnn.h"

#include "weights.h"

void CNN\_ISR(void)

{

// Acknowledge interrupt to all groups

\*((volatile uint32\_t \*) 0x50100000) &= ~((1<<12) | 1);

\*((volatile uint32\_t \*) 0x50500000) &= ~((1<<12) | 1);

\*((volatile uint32\_t \*) 0x50900000) &= ~((1<<12) | 1);

\*((volatile uint32\_t \*) 0x50d00000) &= ~((1<<12) | 1);

CNN\_COMPLETE; // Signal that processing is complete

#ifdef CNN\_INFERENCE\_TIMER

cnn\_time = MXC\_TMR\_SW\_Stop(CNN\_INFERENCE\_TIMER);

#else

cnn\_time = 1;

#endif

}

int cnn\_continue(void)

{

cnn\_time = 0;

\*((volatile uint32\_t \*) 0x50100000) |= 1; // Re-enable group 0

return CNN\_OK;

}

int cnn\_stop(void)

{

\*((volatile uint32\_t \*) 0x50100000) &= ~1; // Disable group 0

return CNN\_OK;

}

void memcpy32(uint32\_t \*dst, const uint32\_t \*src, int n)

{

while (n-- > 0) {

\*dst++ = \*src++;

}

}

// Kernels:

static const uint32\_t kernels\_0[] = KERNELS\_0;

static const uint32\_t kernels\_1[] = KERNELS\_1;

static const uint32\_t kernels\_2[] = KERNELS\_2;

static const uint32\_t kernels\_3[] = KERNELS\_3;

static const uint32\_t kernels\_4[] = KERNELS\_4;

static const uint32\_t kernels\_5[] = KERNELS\_5;

static const uint32\_t kernels\_6[] = KERNELS\_6;

static const uint32\_t kernels\_7[] = KERNELS\_7;

static const uint32\_t kernels\_8[] = KERNELS\_8;

static const uint32\_t kernels\_9[] = KERNELS\_9;

static const uint32\_t kernels\_10[] = KERNELS\_10;

static const uint32\_t kernels\_11[] = KERNELS\_11;

static const uint32\_t kernels\_12[] = KERNELS\_12;

static const uint32\_t kernels\_13[] = KERNELS\_13;

static const uint32\_t kernels\_14[] = KERNELS\_14;

static const uint32\_t kernels\_15[] = KERNELS\_15;

static const uint32\_t kernels\_16[] = KERNELS\_16;

static const uint32\_t kernels\_17[] = KERNELS\_17;

static const uint32\_t kernels\_18[] = KERNELS\_18;

static const uint32\_t kernels\_19[] = KERNELS\_19;

static const uint32\_t kernels\_20[] = KERNELS\_20;

static const uint32\_t kernels\_21[] = KERNELS\_21;

static const uint32\_t kernels\_22[] = KERNELS\_22;

static const uint32\_t kernels\_23[] = KERNELS\_23;

static const uint32\_t kernels\_24[] = KERNELS\_24;

static const uint32\_t kernels\_25[] = KERNELS\_25;

static const uint32\_t kernels\_26[] = KERNELS\_26;

static const uint32\_t kernels\_27[] = KERNELS\_27;

static const uint32\_t kernels\_28[] = KERNELS\_28;

static const uint32\_t kernels\_29[] = KERNELS\_29;

static const uint32\_t kernels\_30[] = KERNELS\_30;

static const uint32\_t kernels\_31[] = KERNELS\_31;

static const uint32\_t kernels\_32[] = KERNELS\_32;

static const uint32\_t kernels\_33[] = KERNELS\_33;

static const uint32\_t kernels\_34[] = KERNELS\_34;

static const uint32\_t kernels\_35[] = KERNELS\_35;

static const uint32\_t kernels\_36[] = KERNELS\_36;

static const uint32\_t kernels\_37[] = KERNELS\_37;

static const uint32\_t kernels\_38[] = KERNELS\_38;

static const uint32\_t kernels\_39[] = KERNELS\_39;

static const uint32\_t kernels\_40[] = KERNELS\_40;

static const uint32\_t kernels\_41[] = KERNELS\_41;

static const uint32\_t kernels\_42[] = KERNELS\_42;

static const uint32\_t kernels\_43[] = KERNELS\_43;

static const uint32\_t kernels\_44[] = KERNELS\_44;

static const uint32\_t kernels\_45[] = KERNELS\_45;

static const uint32\_t kernels\_46[] = KERNELS\_46;

static const uint32\_t kernels\_47[] = KERNELS\_47;

static const uint32\_t kernels\_48[] = KERNELS\_48;

static const uint32\_t kernels\_49[] = KERNELS\_49;

static const uint32\_t kernels\_50[] = KERNELS\_50;

static const uint32\_t kernels\_51[] = KERNELS\_51;

static const uint32\_t kernels\_52[] = KERNELS\_52;

static const uint32\_t kernels\_53[] = KERNELS\_53;

static const uint32\_t kernels\_54[] = KERNELS\_54;

static const uint32\_t kernels\_55[] = KERNELS\_55;

static const uint32\_t kernels\_56[] = KERNELS\_56;

static const uint32\_t kernels\_57[] = KERNELS\_57;

static const uint32\_t kernels\_58[] = KERNELS\_58;

static const uint32\_t kernels\_59[] = KERNELS\_59;

static const uint32\_t kernels\_60[] = KERNELS\_60;

static const uint32\_t kernels\_61[] = KERNELS\_61;

static const uint32\_t kernels\_62[] = KERNELS\_62;

static const uint32\_t kernels\_63[] = KERNELS\_63;

int cnn\_load\_weights(void)

{

\*((volatile uint8\_t \*) 0x50180001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50180000, kernels\_0, 279);

\*((volatile uint8\_t \*) 0x50184041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50184000, kernels\_1, 243);

\*((volatile uint8\_t \*) 0x50188041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50188000, kernels\_2, 243);

\*((volatile uint8\_t \*) 0x5018c041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x5018c000, kernels\_3, 243);

\*((volatile uint8\_t \*) 0x50190041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50190000, kernels\_4, 243);

\*((volatile uint8\_t \*) 0x50194041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50194000, kernels\_5, 243);

\*((volatile uint8\_t \*) 0x50198041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50198000, kernels\_6, 243);

\*((volatile uint8\_t \*) 0x5019c041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x5019c000, kernels\_7, 243);

\*((volatile uint8\_t \*) 0x501a0041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x501a0000, kernels\_8, 243);

\*((volatile uint8\_t \*) 0x501a4041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x501a4000, kernels\_9, 243);

\*((volatile uint8\_t \*) 0x501a8041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x501a8000, kernels\_10, 243);

\*((volatile uint8\_t \*) 0x501ac041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x501ac000, kernels\_11, 243);

\*((volatile uint8\_t \*) 0x501b0041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x501b0000, kernels\_12, 243);

\*((volatile uint8\_t \*) 0x501b4041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x501b4000, kernels\_13, 243);

\*((volatile uint8\_t \*) 0x501b8041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x501b8000, kernels\_14, 243);

\*((volatile uint8\_t \*) 0x501bc041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x501bc000, kernels\_15, 243);

\*((volatile uint8\_t \*) 0x50580001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50580000, kernels\_16, 279);

\*((volatile uint8\_t \*) 0x50584041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50584000, kernels\_17, 243);

\*((volatile uint8\_t \*) 0x50588041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50588000, kernels\_18, 243);

\*((volatile uint8\_t \*) 0x5058c041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x5058c000, kernels\_19, 243);

\*((volatile uint8\_t \*) 0x50590041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50590000, kernels\_20, 243);

\*((volatile uint8\_t \*) 0x50594041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50594000, kernels\_21, 243);

\*((volatile uint8\_t \*) 0x50598041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50598000, kernels\_22, 243);

\*((volatile uint8\_t \*) 0x5059c041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x5059c000, kernels\_23, 243);

\*((volatile uint8\_t \*) 0x505a0041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x505a0000, kernels\_24, 243);

\*((volatile uint8\_t \*) 0x505a4041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x505a4000, kernels\_25, 243);

\*((volatile uint8\_t \*) 0x505a8041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x505a8000, kernels\_26, 243);

\*((volatile uint8\_t \*) 0x505ac041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x505ac000, kernels\_27, 243);

\*((volatile uint8\_t \*) 0x505b0041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x505b0000, kernels\_28, 243);

\*((volatile uint8\_t \*) 0x505b4041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x505b4000, kernels\_29, 243);

\*((volatile uint8\_t \*) 0x505b8041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x505b8000, kernels\_30, 243);

\*((volatile uint8\_t \*) 0x505bc041) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x505bc000, kernels\_31, 243);

\*((volatile uint8\_t \*) 0x50980001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50980000, kernels\_32, 279);

\*((volatile uint8\_t \*) 0x50984081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50984000, kernels\_33, 207);

\*((volatile uint8\_t \*) 0x50988081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50988000, kernels\_34, 320);

\*((volatile uint8\_t \*) 0x5098c081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x5098c000, kernels\_35, 320);

\*((volatile uint8\_t \*) 0x50990001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50990000, kernels\_36, 392);

\*((volatile uint8\_t \*) 0x50994001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50994000, kernels\_37, 392);

\*((volatile uint8\_t \*) 0x50998001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50998000, kernels\_38, 392);

\*((volatile uint8\_t \*) 0x5099c001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x5099c000, kernels\_39, 392);

\*((volatile uint8\_t \*) 0x509a0001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x509a0000, kernels\_40, 392);

\*((volatile uint8\_t \*) 0x509a4001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x509a4000, kernels\_41, 392);

\*((volatile uint8\_t \*) 0x509a8001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x509a8000, kernels\_42, 392);

\*((volatile uint8\_t \*) 0x509ac001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x509ac000, kernels\_43, 392);

\*((volatile uint8\_t \*) 0x509b0001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x509b0000, kernels\_44, 392);

\*((volatile uint8\_t \*) 0x509b4001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x509b4000, kernels\_45, 392);

\*((volatile uint8\_t \*) 0x509b8001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x509b8000, kernels\_46, 392);

\*((volatile uint8\_t \*) 0x509bc001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x509bc000, kernels\_47, 392);

\*((volatile uint8\_t \*) 0x50d80001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50d80000, kernels\_48, 392);

\*((volatile uint8\_t \*) 0x50d84001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50d84000, kernels\_49, 392);

\*((volatile uint8\_t \*) 0x50d88001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50d88000, kernels\_50, 392);

\*((volatile uint8\_t \*) 0x50d8c001) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50d8c000, kernels\_51, 392);

\*((volatile uint8\_t \*) 0x50d90081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50d90000, kernels\_52, 320);

\*((volatile uint8\_t \*) 0x50d94081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50d94000, kernels\_53, 320);

\*((volatile uint8\_t \*) 0x50d98081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50d98000, kernels\_54, 320);

\*((volatile uint8\_t \*) 0x50d9c081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50d9c000, kernels\_55, 320);

\*((volatile uint8\_t \*) 0x50da0081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50da0000, kernels\_56, 320);

\*((volatile uint8\_t \*) 0x50da4081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50da4000, kernels\_57, 320);

\*((volatile uint8\_t \*) 0x50da8081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50da8000, kernels\_58, 320);

\*((volatile uint8\_t \*) 0x50dac081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50dac000, kernels\_59, 320);

\*((volatile uint8\_t \*) 0x50db0081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50db0000, kernels\_60, 320);

\*((volatile uint8\_t \*) 0x50db4081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50db4000, kernels\_61, 320);

\*((volatile uint8\_t \*) 0x50db8081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50db8000, kernels\_62, 320);

\*((volatile uint8\_t \*) 0x50dbc081) = 0x01; // Set address

memcpy32((uint32\_t \*) 0x50dbc000, kernels\_63, 320);

return CNN\_OK;

}

static const uint8\_t bias\_0[] = BIAS\_0;

static void memcpy\_8to32(uint32\_t \*dst, const uint8\_t \*src, int n)

{

while (n-- > 0) {

\*dst++ = \*src++;

}

}

int cnn\_load\_bias(void)

{

memcpy\_8to32((uint32\_t \*) 0x50108000, bias\_0, sizeof(uint8\_t) \* 10);

return CNN\_OK;

}

int cnn\_init(void)

{

\*((volatile uint32\_t \*) 0x50001000) = 0x00000000; // AON control

\*((volatile uint32\_t \*) 0x50100000) = 0x00100008; // Stop SM

\*((volatile uint32\_t \*) 0x50100004) = 0x0000040e; // SRAM control

\*((volatile uint32\_t \*) 0x50100008) = 0x00000007; // Layer count

\*((volatile uint32\_t \*) 0x50500000) = 0x00100008; // Stop SM

\*((volatile uint32\_t \*) 0x50500004) = 0x0000040e; // SRAM control

\*((volatile uint32\_t \*) 0x50500008) = 0x00000007; // Layer count

\*((volatile uint32\_t \*) 0x50900000) = 0x00100008; // Stop SM

\*((volatile uint32\_t \*) 0x50900004) = 0x0000040e; // SRAM control

\*((volatile uint32\_t \*) 0x50900008) = 0x00000007; // Layer count

\*((volatile uint32\_t \*) 0x50d00000) = 0x00100008; // Stop SM

\*((volatile uint32\_t \*) 0x50d00004) = 0x0000040e; // SRAM control

\*((volatile uint32\_t \*) 0x50d00008) = 0x00000007; // Layer count

return CNN\_OK;

}

int cnn\_configure(void)

{

// Layer 0 group 0

\*((volatile uint32\_t \*) 0x50100010) = 0x00010041; // Rows

\*((volatile uint32\_t \*) 0x50100090) = 0x00010041; // Columns

\*((volatile uint32\_t \*) 0x50100310) = 0x00012000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50100410) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50100590) = 0x00006b60; // Layer control

\*((volatile uint32\_t \*) 0x50100a10) = 0x00007800; // Layer control 2

\*((volatile uint32\_t \*) 0x50100610) = 0x00000078; // Mask offset and count

\*((volatile uint32\_t \*) 0x50100690) = 0x0000003f; // TRAM ptr max

\*((volatile uint32\_t \*) 0x50100790) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50100710) = 0x00010001; // Mask and processor enables

// Layer 0 group 1

\*((volatile uint32\_t \*) 0x50500010) = 0x00010041; // Rows

\*((volatile uint32\_t \*) 0x50500090) = 0x00010041; // Columns

\*((volatile uint32\_t \*) 0x50500310) = 0x00012000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50500410) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50500590) = 0x00000b60; // Layer control

\*((volatile uint32\_t \*) 0x50500a10) = 0x00007800; // Layer control 2

\*((volatile uint32\_t \*) 0x50500610) = 0x00000078; // Mask offset and count

\*((volatile uint32\_t \*) 0x50500690) = 0x0000003f; // TRAM ptr max

\*((volatile uint32\_t \*) 0x50500790) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50500710) = 0x00010001; // Mask and processor enables

// Layer 0 group 2

\*((volatile uint32\_t \*) 0x50900010) = 0x00010041; // Rows

\*((volatile uint32\_t \*) 0x50900090) = 0x00010041; // Columns

\*((volatile uint32\_t \*) 0x50900310) = 0x00012000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50900410) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50900590) = 0x00000b60; // Layer control

\*((volatile uint32\_t \*) 0x50900a10) = 0x00007800; // Layer control 2

\*((volatile uint32\_t \*) 0x50900610) = 0x00000078; // Mask offset and count

\*((volatile uint32\_t \*) 0x50900690) = 0x0000003f; // TRAM ptr max

\*((volatile uint32\_t \*) 0x50900790) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50900710) = 0x00010001; // Mask and processor enables

// Layer 0 group 3

\*((volatile uint32\_t \*) 0x50d00010) = 0x00010041; // Rows

\*((volatile uint32\_t \*) 0x50d00090) = 0x00010041; // Columns

\*((volatile uint32\_t \*) 0x50d00310) = 0x00012000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50d00410) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50d00590) = 0x00000b60; // Layer control

\*((volatile uint32\_t \*) 0x50d00a10) = 0x00007800; // Layer control 2

\*((volatile uint32\_t \*) 0x50d00610) = 0x00000078; // Mask offset and count

\*((volatile uint32\_t \*) 0x50d00690) = 0x0000003f; // TRAM ptr max

\*((volatile uint32\_t \*) 0x50d00790) = 0x00022000; // Post processing register

// Layer 1 group 0

\*((volatile uint32\_t \*) 0x50100014) = 0x00010041; // Rows

\*((volatile uint32\_t \*) 0x50100094) = 0x00010041; // Columns

\*((volatile uint32\_t \*) 0x50100194) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x50100214) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x50100294) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x50100314) = 0x00000800; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50100414) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50100594) = 0x0000cba0; // Layer control

\*((volatile uint32\_t \*) 0x50100a14) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50100614) = 0x000000f8; // Mask offset and count

\*((volatile uint32\_t \*) 0x50100694) = 0x0000001f; // TRAM ptr max

// Layer 1 group 1

\*((volatile uint32\_t \*) 0x50500014) = 0x00010041; // Rows

\*((volatile uint32\_t \*) 0x50500094) = 0x00010041; // Columns

\*((volatile uint32\_t \*) 0x50500194) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x50500214) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x50500294) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x50500314) = 0x00000800; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50500414) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50500594) = 0x00000ba0; // Layer control

\*((volatile uint32\_t \*) 0x50500a14) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50500614) = 0x000000f8; // Mask offset and count

\*((volatile uint32\_t \*) 0x50500694) = 0x0000001f; // TRAM ptr max

// Layer 1 group 2

\*((volatile uint32\_t \*) 0x50900014) = 0x00010041; // Rows

\*((volatile uint32\_t \*) 0x50900094) = 0x00010041; // Columns

\*((volatile uint32\_t \*) 0x50900194) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x50900214) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x50900294) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x50900314) = 0x00000800; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50900414) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50900594) = 0x00000ba0; // Layer control

\*((volatile uint32\_t \*) 0x50900a14) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50900614) = 0x000000f8; // Mask offset and count

\*((volatile uint32\_t \*) 0x50900694) = 0x0000001f; // TRAM ptr max

\*((volatile uint32\_t \*) 0x50900714) = 0xfff0fff0; // Mask and processor enables

// Layer 1 group 3

\*((volatile uint32\_t \*) 0x50d00014) = 0x00010041; // Rows

\*((volatile uint32\_t \*) 0x50d00094) = 0x00010041; // Columns

\*((volatile uint32\_t \*) 0x50d00194) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x50d00214) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x50d00294) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x50d00314) = 0x00000800; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50d00414) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50d00594) = 0x00000ba0; // Layer control

\*((volatile uint32\_t \*) 0x50d00a14) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50d00614) = 0x000000f8; // Mask offset and count

\*((volatile uint32\_t \*) 0x50d00694) = 0x0000001f; // TRAM ptr max

\*((volatile uint32\_t \*) 0x50d00714) = 0x000f000f; // Mask and processor enables

// Layer 2 group 0

\*((volatile uint32\_t \*) 0x50100018) = 0x00010021; // Rows

\*((volatile uint32\_t \*) 0x50100098) = 0x00010021; // Columns

\*((volatile uint32\_t \*) 0x50100318) = 0x00010000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50100418) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50100518) = 0x00000800; // SRAM read ptr

\*((volatile uint32\_t \*) 0x50100598) = 0x00002b20; // Layer control

\*((volatile uint32\_t \*) 0x50100a18) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50100618) = 0x00800178; // Mask offset and count

\*((volatile uint32\_t \*) 0x50100698) = 0x0000001f; // TRAM ptr max

\*((volatile uint32\_t \*) 0x50100798) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50100718) = 0xffffffff; // Mask and processor enables

// Layer 2 group 1

\*((volatile uint32\_t \*) 0x50500018) = 0x00010021; // Rows

\*((volatile uint32\_t \*) 0x50500098) = 0x00010021; // Columns

\*((volatile uint32\_t \*) 0x50500318) = 0x00010000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50500418) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50500518) = 0x00000800; // SRAM read ptr

\*((volatile uint32\_t \*) 0x50500598) = 0x00000b20; // Layer control

\*((volatile uint32\_t \*) 0x50500a18) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50500618) = 0x00800178; // Mask offset and count

\*((volatile uint32\_t \*) 0x50500698) = 0x0000001f; // TRAM ptr max

\*((volatile uint32\_t \*) 0x50500798) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50500718) = 0xffffffff; // Mask and processor enables

// Layer 2 group 2

\*((volatile uint32\_t \*) 0x50900018) = 0x00010021; // Rows

\*((volatile uint32\_t \*) 0x50900098) = 0x00010021; // Columns

\*((volatile uint32\_t \*) 0x50900318) = 0x00010000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50900418) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50900518) = 0x00000800; // SRAM read ptr

\*((volatile uint32\_t \*) 0x50900598) = 0x00000b20; // Layer control

\*((volatile uint32\_t \*) 0x50900a18) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50900618) = 0x00800178; // Mask offset and count

\*((volatile uint32\_t \*) 0x50900698) = 0x0000001f; // TRAM ptr max

\*((volatile uint32\_t \*) 0x50900798) = 0x00022000; // Post processing register

// Layer 2 group 3

\*((volatile uint32\_t \*) 0x50d00018) = 0x00010021; // Rows

\*((volatile uint32\_t \*) 0x50d00098) = 0x00010021; // Columns

\*((volatile uint32\_t \*) 0x50d00318) = 0x00010000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50d00418) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50d00518) = 0x00000800; // SRAM read ptr

\*((volatile uint32\_t \*) 0x50d00598) = 0x00000b20; // Layer control

\*((volatile uint32\_t \*) 0x50d00a18) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50d00618) = 0x00800178; // Mask offset and count

\*((volatile uint32\_t \*) 0x50d00698) = 0x0000001f; // TRAM ptr max

\*((volatile uint32\_t \*) 0x50d00798) = 0x00022000; // Post processing register

// Layer 3 group 0

\*((volatile uint32\_t \*) 0x5010001c) = 0x00010021; // Rows

\*((volatile uint32\_t \*) 0x5010009c) = 0x00010021; // Columns

\*((volatile uint32\_t \*) 0x5010019c) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x5010021c) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x5010029c) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x5010031c) = 0x00000800; // SRAM write ptr

\*((volatile uint32\_t \*) 0x5010041c) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x5010059c) = 0x0000cba0; // Layer control

\*((volatile uint32\_t \*) 0x50100a1c) = 0x0001d800; // Layer control 2

\*((volatile uint32\_t \*) 0x5010061c) = 0x010002d8; // Mask offset and count

\*((volatile uint32\_t \*) 0x5010069c) = 0x0000000f; // TRAM ptr max

\*((volatile uint32\_t \*) 0x5010079c) = 0x00022000; // Post processing register

// Layer 3 group 1

\*((volatile uint32\_t \*) 0x5050001c) = 0x00010021; // Rows

\*((volatile uint32\_t \*) 0x5050009c) = 0x00010021; // Columns

\*((volatile uint32\_t \*) 0x5050019c) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x5050021c) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x5050029c) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x5050031c) = 0x00000800; // SRAM write ptr

\*((volatile uint32\_t \*) 0x5050041c) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x5050059c) = 0x00000ba0; // Layer control

\*((volatile uint32\_t \*) 0x50500a1c) = 0x0001d800; // Layer control 2

\*((volatile uint32\_t \*) 0x5050061c) = 0x010002d8; // Mask offset and count

\*((volatile uint32\_t \*) 0x5050069c) = 0x0000000f; // TRAM ptr max

\*((volatile uint32\_t \*) 0x5050079c) = 0x00022000; // Post processing register

// Layer 3 group 2

\*((volatile uint32\_t \*) 0x5090001c) = 0x00010021; // Rows

\*((volatile uint32\_t \*) 0x5090009c) = 0x00010021; // Columns

\*((volatile uint32\_t \*) 0x5090019c) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x5090021c) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x5090029c) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x5090031c) = 0x00000800; // SRAM write ptr

\*((volatile uint32\_t \*) 0x5090041c) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x5090059c) = 0x00000ba0; // Layer control

\*((volatile uint32\_t \*) 0x50900a1c) = 0x0001d800; // Layer control 2

\*((volatile uint32\_t \*) 0x5090061c) = 0x010002d8; // Mask offset and count

\*((volatile uint32\_t \*) 0x5090069c) = 0x0000000f; // TRAM ptr max

\*((volatile uint32\_t \*) 0x5090079c) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x5090071c) = 0xffffffff; // Mask and processor enables

// Layer 3 group 3

\*((volatile uint32\_t \*) 0x50d0001c) = 0x00010021; // Rows

\*((volatile uint32\_t \*) 0x50d0009c) = 0x00010021; // Columns

\*((volatile uint32\_t \*) 0x50d0019c) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x50d0021c) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x50d0029c) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x50d0031c) = 0x00000800; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50d0041c) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50d0059c) = 0x00000ba0; // Layer control

\*((volatile uint32\_t \*) 0x50d00a1c) = 0x0001d800; // Layer control 2

\*((volatile uint32\_t \*) 0x50d0061c) = 0x010002d8; // Mask offset and count

\*((volatile uint32\_t \*) 0x50d0069c) = 0x0000000f; // TRAM ptr max

\*((volatile uint32\_t \*) 0x50d0079c) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50d0071c) = 0xffffffff; // Mask and processor enables

// Layer 4 group 0

\*((volatile uint32\_t \*) 0x50100020) = 0x00010011; // Rows

\*((volatile uint32\_t \*) 0x501000a0) = 0x00010011; // Columns

\*((volatile uint32\_t \*) 0x501001a0) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x50100220) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x501002a0) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x50100320) = 0x00010000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50100420) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50100520) = 0x00000800; // SRAM read ptr

\*((volatile uint32\_t \*) 0x501005a0) = 0x0000eba0; // Layer control

\*((volatile uint32\_t \*) 0x50100a20) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50100620) = 0x02e003d8; // Mask offset and count

\*((volatile uint32\_t \*) 0x501006a0) = 0x00000007; // TRAM ptr max

\*((volatile uint32\_t \*) 0x501007a0) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50100720) = 0xffffffff; // Mask and processor enables

// Layer 4 group 1

\*((volatile uint32\_t \*) 0x50500020) = 0x00010011; // Rows

\*((volatile uint32\_t \*) 0x505000a0) = 0x00010011; // Columns

\*((volatile uint32\_t \*) 0x505001a0) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x50500220) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x505002a0) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x50500320) = 0x00010000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50500420) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50500520) = 0x00000800; // SRAM read ptr

\*((volatile uint32\_t \*) 0x505005a0) = 0x00000ba0; // Layer control

\*((volatile uint32\_t \*) 0x50500a20) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50500620) = 0x02e003d8; // Mask offset and count

\*((volatile uint32\_t \*) 0x505006a0) = 0x00000007; // TRAM ptr max

\*((volatile uint32\_t \*) 0x505007a0) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50500720) = 0xffffffff; // Mask and processor enables

// Layer 4 group 2

\*((volatile uint32\_t \*) 0x50900020) = 0x00010011; // Rows

\*((volatile uint32\_t \*) 0x509000a0) = 0x00010011; // Columns

\*((volatile uint32\_t \*) 0x509001a0) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x50900220) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x509002a0) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x50900320) = 0x00010000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50900420) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50900520) = 0x00000800; // SRAM read ptr

\*((volatile uint32\_t \*) 0x509005a0) = 0x00000ba0; // Layer control

\*((volatile uint32\_t \*) 0x50900a20) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50900620) = 0x02e003d8; // Mask offset and count

\*((volatile uint32\_t \*) 0x509006a0) = 0x00000007; // TRAM ptr max

\*((volatile uint32\_t \*) 0x509007a0) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50900720) = 0xffffffff; // Mask and processor enables

// Layer 4 group 3

\*((volatile uint32\_t \*) 0x50d00020) = 0x00010011; // Rows

\*((volatile uint32\_t \*) 0x50d000a0) = 0x00010011; // Columns

\*((volatile uint32\_t \*) 0x50d001a0) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x50d00220) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x50d002a0) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x50d00320) = 0x00010000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50d00420) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50d00520) = 0x00000800; // SRAM read ptr

\*((volatile uint32\_t \*) 0x50d005a0) = 0x00000ba0; // Layer control

\*((volatile uint32\_t \*) 0x50d00a20) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50d00620) = 0x02e003d8; // Mask offset and count

\*((volatile uint32\_t \*) 0x50d006a0) = 0x00000007; // TRAM ptr max

\*((volatile uint32\_t \*) 0x50d007a0) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50d00720) = 0x0fff0fff; // Mask and processor enables

// Layer 5 group 0

\*((volatile uint32\_t \*) 0x50100024) = 0x00010009; // Rows

\*((volatile uint32\_t \*) 0x501000a4) = 0x00010009; // Columns

\*((volatile uint32\_t \*) 0x501001a4) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x50100224) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x501002a4) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x50100324) = 0x00000800; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50100424) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x501005a4) = 0x0000cba0; // Layer control

\*((volatile uint32\_t \*) 0x50100a24) = 0x0000e800; // Layer control 2

\*((volatile uint32\_t \*) 0x50100624) = 0x03e004c8; // Mask offset and count

\*((volatile uint32\_t \*) 0x501006a4) = 0x00000003; // TRAM ptr max

\*((volatile uint32\_t \*) 0x501007a4) = 0x00022000; // Post processing register

// Layer 5 group 1

\*((volatile uint32\_t \*) 0x50500024) = 0x00010009; // Rows

\*((volatile uint32\_t \*) 0x505000a4) = 0x00010009; // Columns

\*((volatile uint32\_t \*) 0x505001a4) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x50500224) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x505002a4) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x50500324) = 0x00000800; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50500424) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x505005a4) = 0x00000ba0; // Layer control

\*((volatile uint32\_t \*) 0x50500a24) = 0x0000e800; // Layer control 2

\*((volatile uint32\_t \*) 0x50500624) = 0x03e004c8; // Mask offset and count

\*((volatile uint32\_t \*) 0x505006a4) = 0x00000003; // TRAM ptr max

\*((volatile uint32\_t \*) 0x505007a4) = 0x00022000; // Post processing register

// Layer 5 group 2

\*((volatile uint32\_t \*) 0x50900024) = 0x00010009; // Rows

\*((volatile uint32\_t \*) 0x509000a4) = 0x00010009; // Columns

\*((volatile uint32\_t \*) 0x509001a4) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x50900224) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x509002a4) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x50900324) = 0x00000800; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50900424) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x509005a4) = 0x00000ba0; // Layer control

\*((volatile uint32\_t \*) 0x50900a24) = 0x0000e800; // Layer control 2

\*((volatile uint32\_t \*) 0x50900624) = 0x03e004c8; // Mask offset and count

\*((volatile uint32\_t \*) 0x509006a4) = 0x00000003; // TRAM ptr max

\*((volatile uint32\_t \*) 0x509007a4) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50900724) = 0xfffcfffc; // Mask and processor enables

// Layer 5 group 3

\*((volatile uint32\_t \*) 0x50d00024) = 0x00010009; // Rows

\*((volatile uint32\_t \*) 0x50d000a4) = 0x00010009; // Columns

\*((volatile uint32\_t \*) 0x50d001a4) = 0x00000001; // Pooling rows

\*((volatile uint32\_t \*) 0x50d00224) = 0x00000001; // Pooling columns

\*((volatile uint32\_t \*) 0x50d002a4) = 0x00000001; // Stride

\*((volatile uint32\_t \*) 0x50d00324) = 0x00000800; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50d00424) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50d005a4) = 0x00000ba0; // Layer control

\*((volatile uint32\_t \*) 0x50d00a24) = 0x0000e800; // Layer control 2

\*((volatile uint32\_t \*) 0x50d00624) = 0x03e004c8; // Mask offset and count

\*((volatile uint32\_t \*) 0x50d006a4) = 0x00000003; // TRAM ptr max

\*((volatile uint32\_t \*) 0x50d007a4) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50d00724) = 0xffffffff; // Mask and processor enables

// Layer 6 group 0

\*((volatile uint32\_t \*) 0x50100028) = 0x00010005; // Rows

\*((volatile uint32\_t \*) 0x501000a8) = 0x00010005; // Columns

\*((volatile uint32\_t \*) 0x50100328) = 0x00010000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50100428) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50100528) = 0x00000800; // SRAM read ptr

\*((volatile uint32\_t \*) 0x501005a8) = 0x00002b20; // Layer control

\*((volatile uint32\_t \*) 0x50100a28) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50100628) = 0x01800278; // Mask offset and count

\*((volatile uint32\_t \*) 0x501006a8) = 0x00000003; // TRAM ptr max

\*((volatile uint32\_t \*) 0x501007a8) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50100728) = 0xffffffff; // Mask and processor enables

// Layer 6 group 1

\*((volatile uint32\_t \*) 0x50500028) = 0x00010005; // Rows

\*((volatile uint32\_t \*) 0x505000a8) = 0x00010005; // Columns

\*((volatile uint32\_t \*) 0x50500328) = 0x00010000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50500428) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50500528) = 0x00000800; // SRAM read ptr

\*((volatile uint32\_t \*) 0x505005a8) = 0x00000b20; // Layer control

\*((volatile uint32\_t \*) 0x50500a28) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50500628) = 0x01800278; // Mask offset and count

\*((volatile uint32\_t \*) 0x505006a8) = 0x00000003; // TRAM ptr max

\*((volatile uint32\_t \*) 0x505007a8) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50500728) = 0x3fff3fff; // Mask and processor enables

// Layer 6 group 2

\*((volatile uint32\_t \*) 0x50900028) = 0x00010005; // Rows

\*((volatile uint32\_t \*) 0x509000a8) = 0x00010005; // Columns

\*((volatile uint32\_t \*) 0x50900328) = 0x00010000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50900428) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50900528) = 0x00000800; // SRAM read ptr

\*((volatile uint32\_t \*) 0x509005a8) = 0x00000b20; // Layer control

\*((volatile uint32\_t \*) 0x50900a28) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50900628) = 0x01800278; // Mask offset and count

\*((volatile uint32\_t \*) 0x509006a8) = 0x00000003; // TRAM ptr max

\*((volatile uint32\_t \*) 0x509007a8) = 0x00022000; // Post processing register

// Layer 6 group 3

\*((volatile uint32\_t \*) 0x50d00028) = 0x00010005; // Rows

\*((volatile uint32\_t \*) 0x50d000a8) = 0x00010005; // Columns

\*((volatile uint32\_t \*) 0x50d00328) = 0x00010000; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50d00428) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50d00528) = 0x00000800; // SRAM read ptr

\*((volatile uint32\_t \*) 0x50d005a8) = 0x00000b20; // Layer control

\*((volatile uint32\_t \*) 0x50d00a28) = 0x0000f800; // Layer control 2

\*((volatile uint32\_t \*) 0x50d00628) = 0x01800278; // Mask offset and count

\*((volatile uint32\_t \*) 0x50d006a8) = 0x00000003; // TRAM ptr max

\*((volatile uint32\_t \*) 0x50d007a8) = 0x00022000; // Post processing register

// Layer 7 group 0

\*((volatile uint32\_t \*) 0x5010032c) = 0x00000400; // SRAM write ptr

\*((volatile uint32\_t \*) 0x501003ac) = 0x00000001; // Write ptr time slot offs

\*((volatile uint32\_t \*) 0x5010042c) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x501005ac) = 0x0001c920; // Layer control

\*((volatile uint32\_t \*) 0x50100a2c) = 0x0000480f; // Layer control 2

\*((volatile uint32\_t \*) 0x5010062c) = 0x2be030d8; // Mask offset and count

\*((volatile uint32\_t \*) 0x5010012c) = 0x00000100; // 1D

\*((volatile uint32\_t \*) 0x501007ac) = 0x00023000; // Post processing register

// Layer 7 group 1

\*((volatile uint32\_t \*) 0x5050032c) = 0x00000400; // SRAM write ptr

\*((volatile uint32\_t \*) 0x505003ac) = 0x00000001; // Write ptr time slot offs

\*((volatile uint32\_t \*) 0x5050042c) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x505005ac) = 0x00010920; // Layer control

\*((volatile uint32\_t \*) 0x50500a2c) = 0x0000480f; // Layer control 2

\*((volatile uint32\_t \*) 0x5050062c) = 0x2be030d8; // Mask offset and count

\*((volatile uint32\_t \*) 0x5050012c) = 0x00000100; // 1D

\*((volatile uint32\_t \*) 0x505007ac) = 0x00022000; // Post processing register

// Layer 7 group 2

\*((volatile uint32\_t \*) 0x5090032c) = 0x00000400; // SRAM write ptr

\*((volatile uint32\_t \*) 0x509003ac) = 0x00000001; // Write ptr time slot offs

\*((volatile uint32\_t \*) 0x5090042c) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x509005ac) = 0x00010920; // Layer control

\*((volatile uint32\_t \*) 0x50900a2c) = 0x0000480f; // Layer control 2

\*((volatile uint32\_t \*) 0x5090062c) = 0x2be030d8; // Mask offset and count

\*((volatile uint32\_t \*) 0x5090012c) = 0x00000100; // 1D

\*((volatile uint32\_t \*) 0x509007ac) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x5090072c) = 0xfffcfffc; // Mask and processor enables

// Layer 7 group 3

\*((volatile uint32\_t \*) 0x50d0032c) = 0x00000400; // SRAM write ptr

\*((volatile uint32\_t \*) 0x50d003ac) = 0x00000001; // Write ptr time slot offs

\*((volatile uint32\_t \*) 0x50d0042c) = 0x00002000; // Write ptr mask offs

\*((volatile uint32\_t \*) 0x50d005ac) = 0x00010920; // Layer control

\*((volatile uint32\_t \*) 0x50d00a2c) = 0x0000480f; // Layer control 2

\*((volatile uint32\_t \*) 0x50d0062c) = 0x2be030d8; // Mask offset and count

\*((volatile uint32\_t \*) 0x50d0012c) = 0x00000100; // 1D

\*((volatile uint32\_t \*) 0x50d007ac) = 0x00022000; // Post processing register

\*((volatile uint32\_t \*) 0x50d0072c) = 0xffffffff; // Mask and processor enables

return CNN\_OK;

}

int cnn\_start(void)

{

cnn\_time = 0;

\*((volatile uint32\_t \*) 0x50100000) = 0x00100808; // Enable group 0

\*((volatile uint32\_t \*) 0x50500000) = 0x00100809; // Enable group 1

\*((volatile uint32\_t \*) 0x50900000) = 0x00100809; // Enable group 2

\*((volatile uint32\_t \*) 0x50d00000) = 0x00100809; // Enable group 3

#ifdef CNN\_INFERENCE\_TIMER

MXC\_TMR\_SW\_Start(CNN\_INFERENCE\_TIMER);

#endif

CNN\_START; // Allow capture of processing time

\*((volatile uint32\_t \*) 0x50100000) = 0x00100009; // Master enable group 0

return CNN\_OK;

}

// Custom unload for this network: 32-bit data, shape: [10, 1, 1]

int cnn\_unload(uint32\_t \*out\_buf)

{

volatile uint32\_t \*addr;

addr = (volatile uint32\_t \*) 0x50401000;

\*out\_buf++ = \*addr++;

\*out\_buf++ = \*addr++;

\*out\_buf++ = \*addr++;

\*out\_buf++ = \*addr++;

addr = (volatile uint32\_t \*) 0x50409000;

\*out\_buf++ = \*addr++;

\*out\_buf++ = \*addr++;

\*out\_buf++ = \*addr++;

\*out\_buf++ = \*addr++;

addr = (volatile uint32\_t \*) 0x50411000;

\*out\_buf++ = \*addr++;

\*out\_buf++ = \*addr++;

return CNN\_OK;

}

int cnn\_enable(uint32\_t clock\_source, uint32\_t clock\_divider)

{

// Reset all domains, restore power to CNN

MXC\_GCFR->reg3 = 0xf; // Reset

MXC\_GCFR->reg1 = 0xf; // Mask memory

MXC\_GCFR->reg0 = 0xf; // Power

MXC\_GCFR->reg2 = 0x0; // Iso

MXC\_GCFR->reg3 = 0x0; // Reset

MXC\_GCR->pclkdiv = (MXC\_GCR->pclkdiv & ~(MXC\_F\_GCR\_PCLKDIV\_CNNCLKDIV | MXC\_F\_GCR\_PCLKDIV\_CNNCLKSEL))

| clock\_divider | clock\_source;

MXC\_SYS\_ClockEnable(MXC\_SYS\_PERIPH\_CLOCK\_CNN); // Enable CNN clock

NVIC\_SetVector(CNN\_IRQn, CNN\_ISR); // Set CNN complete vector

return CNN\_OK;

}

int cnn\_boost\_enable(mxc\_gpio\_regs\_t \*port, uint32\_t pin)

{

mxc\_gpio\_cfg\_t gpio\_out;

gpio\_out.port = port;

gpio\_out.mask = pin;

gpio\_out.pad = MXC\_GPIO\_PAD\_NONE;

gpio\_out.func = MXC\_GPIO\_FUNC\_OUT;

MXC\_GPIO\_Config(&gpio\_out);

MXC\_GPIO\_OutSet(gpio\_out.port, gpio\_out.mask);

return CNN\_OK;

}

int cnn\_boost\_disable(mxc\_gpio\_regs\_t \*port, uint32\_t pin)

{

mxc\_gpio\_cfg\_t gpio\_out;

gpio\_out.port = port;

gpio\_out.mask = pin;

gpio\_out.pad = MXC\_GPIO\_PAD\_NONE;

gpio\_out.func = MXC\_GPIO\_FUNC\_OUT;

MXC\_GPIO\_Config(&gpio\_out);

MXC\_GPIO\_OutSet(gpio\_out.port, gpio\_out.mask);

return CNN\_OK;

}

int cnn\_disable(void)

{

// Disable CNN clock

MXC\_SYS\_ClockDisable(MXC\_SYS\_PERIPH\_CLOCK\_CNN);

// Disable power to CNN

MXC\_GCFR->reg3 = 0xf; // Reset

MXC\_GCFR->reg2 = 0xf; // Iso

MXC\_GCFR->reg0 = 0x0; // Power

MXC\_GCFR->reg1 = 0x0; // Mask memory

MXC\_GCFR->reg3 = 0x0; // Reset

return CNN\_OK;