uTensor

Características principais:

Implementa o framework Tensorflow de Deep Learning da Google de forma otimizada para sistemas ARM. Ferramenta offline. O sistema contém o core para rodar um modelo de rede neural, implementação de operadores, escalonadores e gerenciador de memória. O runtime core pesa 2KB.

O uTensor não treina redes, apenas aplica um modelo treinado para classificação em sistemas embarcados.

Onde é utilizado:

Qualquer sistema embarcado que necessita de classificação de dados durante a execução. Isso inclui dados de leitura de sensores ou imagens de entrada com câmeras ou desenhos in touch (vídeo de implementação com touchpad: <https://www.youtube.com/watch?v=FhbCAd0sO1c> )

Partes importante do código:

Exemplo de código de treinamento com exportação para uTensor:

#import de bibliotecas

from \_\_future\_\_ import absolute\_import, division, print\_function, unicode\_literals

from utensor\_cgen.api.export import tflm\_keras\_export

import tensorflow as tf

import numpy as np

from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D

from tensorflow.keras import Model

# UTILIZANDO DATASET MNIST (Números de 0 a 9, em imagens de 28x28 pixels)

mnist = tf.keras.datasets.mnist

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# FUNÇÃO QUE CRIA O MODELO

class MyModel(Model):

def \_\_init\_\_(self):

super(MyModel, self).\_\_init\_\_()

self.conv1 = Conv2D(8, 3, activation='relu')

self.pool = MaxPooling2D()

self.flatten = Flatten()

self.d1 = Dense(10)

def call(self, x):

x0 = self.conv1(x)

x1 = self.pool(x0)

x2 = self.flatten(x1)

return self.d1(x2)

# O modelo contém uma camada de convolução de 8 matrizes de 3x3, seguido de um MaxPooling (compressão de imagem com valores mais ativos), achatamento para vetor e finalmente uma saída com 10 classificações (0 a 9)

model = MyModel()

train\_your\_model(model)

#Calibrando as imagens de entrada

num\_calibration\_steps = 128

calibration\_dtype = tf.float32

def representative\_dataset\_gen():

for \_ in range(num\_calibration\_steps):

rand\_idx = np.random.randint(0, x\_test.shape[0]-1)

sample = x\_test[rand\_idx]

sample = sample[tf.newaxis, ...]

sample = tf.cast(sample, dtype=calibration\_dtype)

yield [sample]

# Exportando o modelo treinado para o uTensor

tflm\_keras\_export(

model,

representive\_dataset=representative\_dataset\_gen,

model\_name='my\_model',

target='utensor',

)

# PARTE ESSENCIAL PARA O uTENSOR ^

Já no uTensor, a leitura do modelo se dá, por exemplo, com:

#include "models/my\_model.hpp"//Arquivo do modelo gerado anteriormente

#include "tensor.hpp"

#include "mbed.h"

#include <stdio.h>

#include "input\_data.h"//Dados de entrada

Serial pc(USBTX, USBRX, 115200);

int main(void) {

Context ctx; // Variável de Contexto que permite inferência (classificar)

Tensor\* input\_x = new WrappedRamTensor<float>({1, 784}, (float\*) input\_data); // Passa a imagem de entrada para um Tensor

get\_deep\_mlp\_ctx(ctx, input\_x); // Passa o tensor para o Contexto

S\_TENSOR pred\_tensor = ctx.get("y\_pred:0");// Tensor de saída da predição

ctx.eval(); //Função que roda o modelo para inferir uma classe

int pred\_label = \*(pred\_tensor->read<int>(0,0)); //Passando resultado para uma variável

printf("Predicted label: %d\r\n", pred\_label); // Classe da imagem printada

return 0;

}

Phoenix-RTOS

Phoenix-RTOS é o sucessor do Phoenix sistema operacional, desenvolvido a partir de 1999 por Pawel Pisarczyk no Departamento de Eletrônica e Tecnologia da Informação da Universidade de Tecnologia de Varsóvia.

## Principais características

* Arquitetura de microkernel, ou seja, implementa apenas o mínimo de funções primitivas no sistema operacional.

O kernel é responsável por:

* Gerenciamento de memória
* Gerenciamento de threads e processos
* Comunicação e sincronização entre threads e processos

O kernel do Phoenix-RTOS é dividido em cinco subsistemas

* hal - camada de abstração de hardware
* lib - rotinas comuns
* vm - gerenciamento de memória virtual
* proc - gerenciamento de processos e threads
* teste - teste para outros subsistemas
* Biblioteca padrão própria (libphoenix) compativel com ANSI C89 estendida com funções de mapeamento de memória e gerenciamento de processes e threads.
* libphoenix fornece suporte a POSIX.
* suporta servidores, são implementações de funcionalidades que o kernel não possui como por exemplo gerenciador de arquivos e gerenciamento de dispositivos são feitos através de servidores.

### arquiteturas suportadas:

* armv7m3-stm32l152xd
* armv7m3-stm32l152xe
* armv7m4-stm32l4x6
* armv7m7-imxrt105x
* armv7m7-imxrt106x
* armv7m7-imxrt117x
* armv7a7-imx6ull
* ia32-generic
* riscv64-spike

## Onde é usado

Usado em grande escala em dispositivos Smart Grid (medidores de energia inteligentes, concentradores de dados, medidores de gás inteligentes)

## Partes importantes do código

### printf.c

Implementação da função printf, estão função faz a impressão no arquivo de saída padrão.

/\*

\* Phoenix-RTOS

\*

\* libphoenix

\*

\* printf.c

\*

\* Copyright 2017 Phoenix Systems

\* Author: Adrian Kepka

\*

\* This file is part of Phoenix-RTOS.

\*

\* %LICENSE%

\*/

#include "stdio.h"

#include "format.h"

#include "sys/debug.h"

#include "unistd.h"

typedef struct \_printf\_ctx\_t {

size\_t n;

} printf\_ctx\_t;

static void printf\_feed(void \*context, char c)

{

size\_t \*n = context;

\*n = \*n + 1;

putchar(c);

}

int printf(const char \*format, ...)

{

int retVal;

va\_list arg;

va\_start(arg, format);

retVal = vprintf(format, arg);

va\_end(arg);

return retVal;

}

int vprintf(const char \*format, va\_list arg)

{

size\_t n = 0;

format\_parse(&n, printf\_feed, format, arg);

return n;

}

### pthread.c

Implementações das funções ptheads da API POSIX que são utilizadas para programação paralela e concorrente.

/\*

\* Phoenix-RTOS

\*

\* libphoenix

\*

\* pthread

\*

\* Copyright 2017, 2019 Phoenix Systems

\* Author: Pawel Pisarczyk, Marcin Baran

\*

\* This file is part of Phoenix-RTOS.

\*

\* %LICENSE%

\*/

#include <stdlib.h>

#include <errno.h>

#include <limits.h>

#include <sys/list.h>

#include <sys/mman.h>

#include <sys/minmax.h>

#include <pthread.h>

#define CEIL(value, size) ((((value) + (size) - 1) / (size)) \* (size))

typedef struct pthread\_ctx {

handle\_t id;

void \*(\*start\_routine)(void \*);

void \*arg;

void \*retval;

struct pthread\_ctx \*next;

struct pthread\_ctx \*prev;

int detached;

} pthread\_ctx;

static pthread\_ctx \*pthread\_list = NULL;

static const pthread\_attr\_t pthread\_attr\_default = {

.stackaddr = NULL,

.policy = SCHED\_RR,

.priority = 4,

.detached = PTHREAD\_CREATE\_JOINABLE,

.stacksize = CEIL(PTHREAD\_STACK\_MIN, PAGE\_SIZE)

};

static void start\_point(void \*args)

{

pthread\_ctx \*ctx = (pthread\_ctx \*)args;

ctx->retval = (void \*)(ctx->start\_routine(ctx->arg));

endthread();

}

static pthread\_ctx \*find\_pthread(handle\_t id)

{

pthread\_ctx \*ctx = pthread\_list;

if (ctx != NULL) {

do {

if (ctx->id == id)

return ctx;

ctx = ctx->next;

} while (ctx != pthread\_list);

}

return NULL;

}

int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr,

void \*(\*start\_routine)(void \*), void \*arg)

{

const pthread\_attr\_t \*attrs = &pthread\_attr\_default;

if (attr != NULL)

attrs = attr;

void \*stack = mmap(attrs->stackaddr, attrs->stacksize,

PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANONYMOUS, NULL, 0);

if (stack == MAP\_FAILED || stack == NULL)

return -EAGAIN;

pthread\_ctx \*ctx = (pthread\_ctx \*)malloc(sizeof(pthread\_ctx));

if (ctx == NULL) {

munmap(stack, attrs->stacksize);

return -EAGAIN;

}

ctx->retval = NULL;

ctx->detached = attrs->detached;

ctx->start\_routine = start\_routine;

ctx->arg = arg;

\*thread = (pthread\_t)ctx;

int err = beginthreadex(start\_point, attrs->priority, stack,

attrs->stacksize, (void \*)ctx, &ctx->id);

if (err != EOK) {

munmap(stack, attrs->stacksize);

free(ctx);

thread = NULL;

}

else if (ctx->detached == PTHREAD\_CREATE\_JOINABLE) {

if (pthread\_list != NULL && pthread\_list->id == 0)

pthread\_list = NULL;

LIST\_ADD(&pthread\_list, ctx);

}

return err;

}

int pthread\_join(pthread\_t thread, void \*\*value\_ptr)

{

if (thread == pthread\_self())

return -EDEADLK;

pthread\_ctx \*ctx = (pthread\_ctx \*)thread;

if (ctx == NULL)

return -EINVAL;

if (ctx->detached != PTHREAD\_CREATE\_DETACHED) {

int err;

do {

err = threadJoin(0);

if (err < 0) {

return err;

}

else if (ctx->id != err) {

pthread\_ctx \*ghost = find\_pthread(err);

if (ghost != NULL)

ghost->detached = PTHREAD\_CREATE\_DETACHED;

}

} while (ctx->id != err);

}

if (value\_ptr != NULL)

\*value\_ptr = ctx->retval;

LIST\_REMOVE(&pthread\_list, ctx);

free(ctx);

return EOK;

}

int pthread\_detach(pthread\_t thread)

{

pthread\_ctx \*ctx = (pthread\_ctx \*)thread;

if (ctx == NULL)

return -ESRCH;

if (ctx->detached != PTHREAD\_CREATE\_JOINABLE)

return -EINVAL;

LIST\_REMOVE(&pthread\_list, ctx);

free(ctx);

return EOK;

}

pthread\_t pthread\_self(void)

{

return (pthread\_t)find\_pthread(gettid());

}

int pthread\_equal(pthread\_t t1, pthread\_t t2)

{

return t1 == t2;

}

void pthread\_exit(void \*value\_ptr)

{

pthread\_ctx \*ctx = (pthread\_ctx \*)pthread\_self();

if (ctx != NULL)

ctx->retval = value\_ptr;

endthread();

}

int pthread\_attr\_init(pthread\_attr\_t \*attr)

{

\*attr = pthread\_attr\_default;

return EOK;

}

int pthread\_attr\_destroy(pthread\_attr\_t \*attr)

{

return EOK;

}

int pthread\_attr\_setstackaddr(pthread\_attr\_t \*attr, void \*stackaddr)

{

if (attr == NULL)

return -EINVAL;

attr->stackaddr = stackaddr;

return EOK;

}

int pthread\_attr\_getstackaddr(const pthread\_attr\_t \*attr,void \*\*stackaddr)

{

if (attr == NULL)

return -EINVAL;

\*stackaddr = attr->stackaddr;

return EOK;

}

int pthread\_attr\_setstacksize(pthread\_attr\_t \*attr, size\_t stacksize)

{

if (attr == NULL || stacksize < PTHREAD\_STACK\_MIN)

return -EINVAL;

attr->stacksize = CEIL(stacksize, PAGE\_SIZE);

return EOK;

}

int pthread\_attr\_getstacksize(const pthread\_attr\_t \*attr, size\_t \*stacksize)

{

if (attr == NULL)

return -EINVAL;

\*stacksize = attr->stacksize;

return EOK;

}

int pthread\_attr\_setstack(pthread\_attr\_t \*attr, void \*stackaddr,

size\_t stacksize)

{

return pthread\_attr\_setstackaddr(attr, stackaddr) |

pthread\_attr\_setstacksize(attr, stacksize);

}

int pthread\_attr\_getstack(const pthread\_attr\_t \*attr, void \*\*stackaddr,

size\_t \*stacksize)

{

return pthread\_attr\_getstackaddr(attr, stackaddr) |

pthread\_attr\_getstacksize(attr, stacksize);

}

int pthread\_attr\_setschedparam(pthread\_attr\_t \*attr,

const struct sched\_param \*param)

{

if (attr == NULL)

return -EINVAL;

if (param->sched\_priority > sched\_get\_priority\_max(SCHED\_RR) ||

param->sched\_priority < sched\_get\_priority\_min(SCHED\_RR))

return -ENOTSUP;

attr->priority = param->sched\_priority;

return EOK;

}

int pthread\_attr\_getschedparam(const pthread\_attr\_t \*attr,

struct sched\_param \*param)

{

if (attr == NULL)

return -EINVAL;

param->sched\_priority = attr->priority;

return EOK;

}

int pthread\_attr\_setschedpolicy(pthread\_attr\_t \*attr, int policy)

{

if (policy < SCHED\_FIFO || policy > SCHED\_OTHER)

return -EINVAL;

else if (policy == SCHED\_FIFO || policy == SCHED\_OTHER)

return -ENOTSUP;

attr->policy = policy;

return EOK;

}

int pthread\_attr\_getschedpolicy(const pthread\_attr\_t \*attr, int \*policy)

{

if (attr == NULL)

return -EINVAL;

\*policy = attr->policy;

return EOK;

}

int pthread\_attr\_setscope(pthread\_attr\_t \*attr, int contentionscope)

{

return EOK;

}

int pthread\_attr\_getscope(const pthread\_attr\_t \*attr,

int \*contentionscope)

{

if (attr == NULL)

return -EINVAL;

return EOK;

}

int pthread\_attr\_setdetachstate(pthread\_attr\_t \*attr, int detachstate)

{

if (detachstate != PTHREAD\_CREATE\_DETACHED ||

detachstate != PTHREAD\_CREATE\_JOINABLE)

return -EINVAL;

attr->detached = detachstate;

return EOK;

}

int pthread\_attr\_getdetachstate(const pthread\_attr\_t \*attr,

int \*detachstate)

{

if (attr == NULL)

return -EINVAL;

\*detachstate = attr->detached;

return EOK;

}

int pthread\_attr\_getinheritsched(const pthread\_attr\_t \*restrict attr,

int \*restrict inheritsched);

int pthread\_attr\_setinheritsched(pthread\_attr\_t \*attr,

int inheritsched);

int pthread\_setschedprio(pthread\_t thread, int prio);

int pthread\_getschedparam(pthread\_t thread, int \*policy,

struct sched\_param \*restrict param);

int pthread\_setschedparam(pthread\_t thread, int policy,

const struct sched\_param \*param);

int pthread\_mutex\_init(pthread\_mutex\_t \*mutex, const pthread\_mutexattr\_t \*attr)

{

(void)attr;

return mutexCreate(mutex);

}

int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex)

{

if (mutex == NULL)

return -EINVAL;

return mutexLock(\*mutex);

}

int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex)

{

if (mutex == NULL)

return -EINVAL;

return mutexTry(\*mutex);

}

int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex)

{

if (mutex == NULL)

return -EINVAL;

return mutexUnlock(\*mutex);

}

int pthread\_mutex\_destroy(pthread\_mutex\_t \*mutex)

{

if (mutex == NULL)

return -EINVAL;

return resourceDestroy(\*mutex);

}

int sched\_get\_priority\_max(int policy)

{

if (policy == SCHED\_RR)

return 7;

return -EINVAL;

}

int sched\_get\_priority\_min(int policy)

{

if (policy == SCHED\_RR)

return 0;

return -EINVAL;

}