NMDL - Программное обеспечение для реализации глубоких нейронных сетей на платформе NeuroMatrix®

NMDL
Руководство пользователя

НАУЧНО-ТЕХНИЧЕСКИЙ ЦЕНТР

Содержание

Список иллюстраций

Список таблиц

1. Общие сведения

1.1. Назначение

Программный модуль *NMDL* позволяет запускать предварительно обученную глубокую сверточную нейронную сеть на вычислительных модулях *MC121.01*, *MC127.05*, *NMStick*, *NMCard* и на симуляторе модуля *MC127.05*. Программный модуль состоит из 2 частей. Одна часть работает на персональном компьютере (хост) под управлением 64 разрядных ОС Microsoft® Windows 7/10 или Linux, другая часть запускается и работает на процессоре вычислительного модуля. Связь устройств *MC121.01* и *NMStick* с хостом осуществляется по каналу USB2.0, для связи модулей *MC127.05* и *NMCard* с хостом используется интерфейс PCIe.

Для работы с *NMDL* необходимо предварительно установить в системе ПО поддержки используемых вычислительных модулей. Для работы с симулятором установка ПО поддержки не требуется.

NMDL выполняет обработку пользовательских исходных изображений в стоответствии с заданой моделью нейросети. Перед обработкой необходимо подготовить данные модели и изображений.

Модель предварительно подготавливается специальным компилятором из состава *NMDL*. Исходные модели могут быть представлены в формате *ONNX* или *DarkNet*. Компилятором *NMDL* поддерживаются не все операции, определённые в *ONNX*. Список поддерживаемых операций и другие ограничения приведены в разделе "Поддерживаемые операции".

Изображения также должны быть предварительно обработаны специальным конвертером изображений. Только подготовленные модели и изображения могут быть заружены и обработаны на вычислительных модулях.

Библиотека предоставляет программный интерфейс C/C++.

Далее по тексту *NMDL* (в верхнем регистре) - обозначение комплекта ПО, *nmdl* (в нижнем регистре) - файлы программного модуля.

1.2. Режимы обработки

Обработка входных данных (инференс) производится в соответствии с графом обработки, определённым в нейросетевой модели. Каждая из моделей обрабатывается на вычислительном устройстве - юните. Устройства и модули, выполненные на базе *СБИС 1879ВМ6Я* - *MC121.01* и *NMStick* - имеют один юнит. Модули *MC127.05*, *NMCard* и прочие устройства, выполненные на базе *СБИС К1879ВМ8Я* имеют четыре юнита.

На устройствах с процессором *К1879ВМ8Я* возможна одновременная и независимая обработка различных моделей. Такая обработка иллюстрируется на рисунке 1.1 - на

каждом юните независимо обрабатывается "своя" модель нейронной сети. Такой режим обработки обозначается "single unit" - инференс выполняется на одном юните.

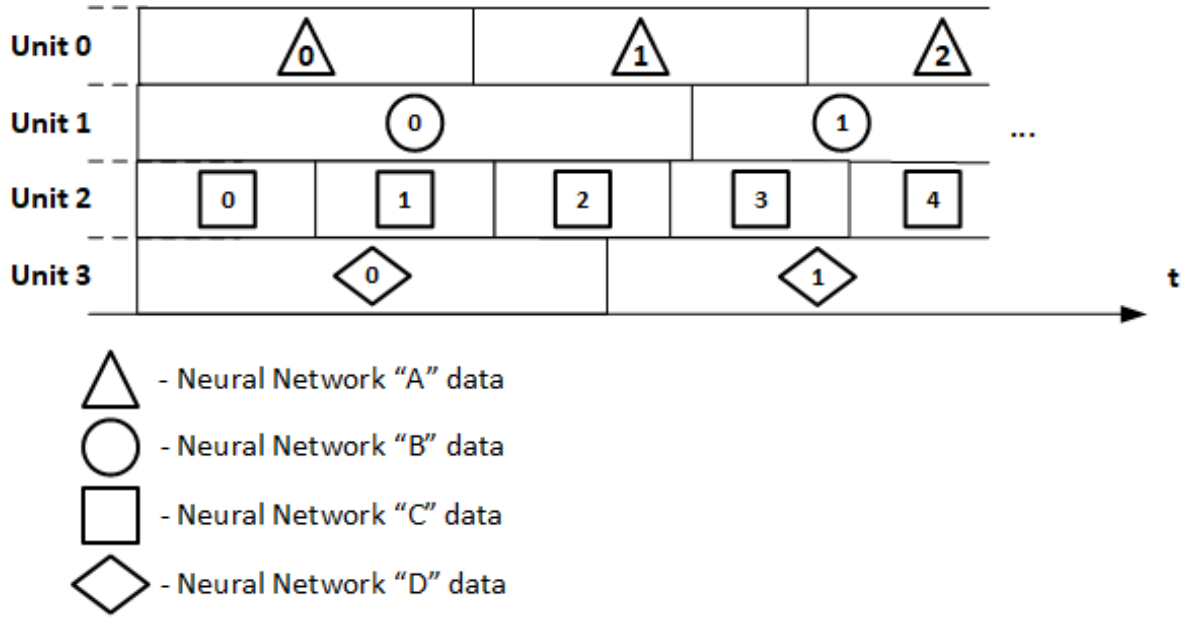

Рисунок 1.1 - Одновременная обработка нескольких нейронных сетей в режиме "single unit"

Четыре юнита можно настроить на обработку одинаковых моделей, когда каждый юнит выполняет один и тот же граф обработки, при этом можно организовать пакетную обработку, достигая максимальной производительности обработки потока данных, например, потока кадров от видеокамеры (см. рисунок 1.2).

Пекетная обработка характеризуется высоким значением задержки (Latency) - времени от запуска обработки до получения результата.

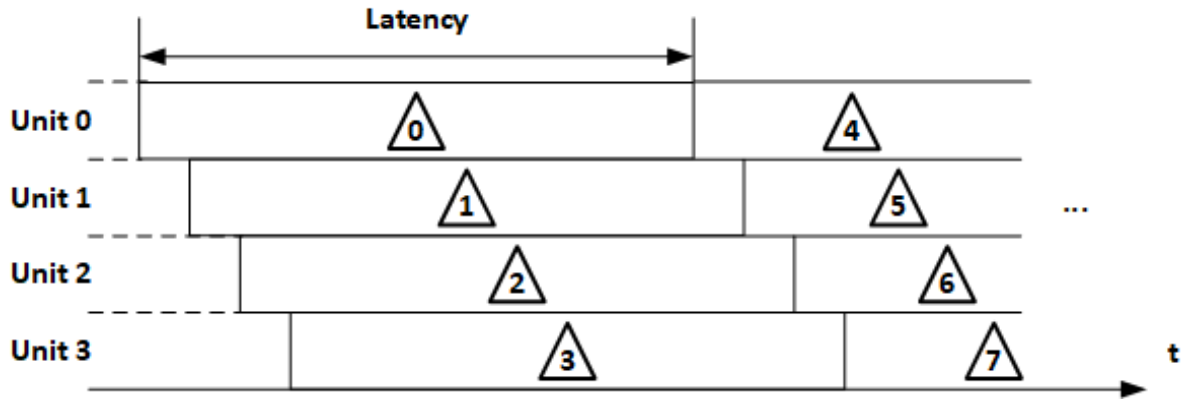

Рисунок 1.2 - Пакетная обработка в режиме "single unit"

Для устройств на базе *СБИС К1879ВМ8Я* в *NMDL* реализован режим обработки одной модели на четырёх юнитах с равномерным разделением данных (*"multi unit"*) (см. рисунок 1.2). При этом достигается минимальная задержка. Производительность здесь, как правило, несколько ниже из-за накладных расходов по организации параллелизма. Например, при выполнении свёрток над разделёнными тензорами необходимо компенсировать обработку границ, выполнять переупаковки, транзит данных между кластерами и т. п.

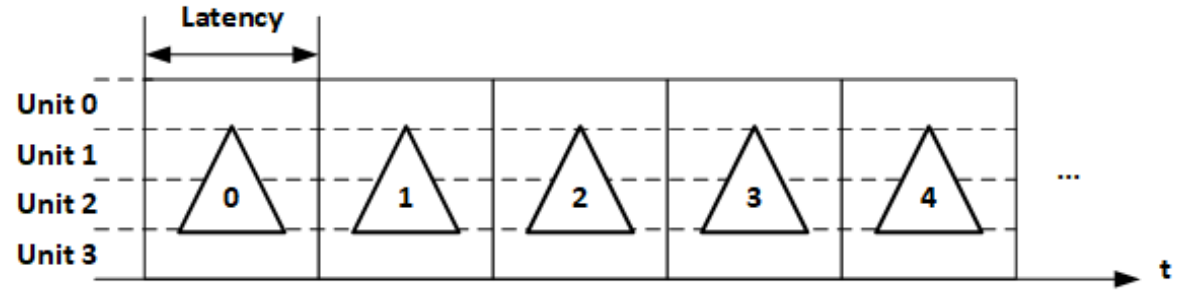

Рисунок 1.3 - Режим обработки "multi unit"

1.3. Быстрый старт

В разделе приводится пошаговая инструкция для быстрого начала работы с демонстрационной программой. Более полную информацию о *NMDL* можно получить в соответсвующих разделах руководства. Здесь описывается демонстрация обработки изображения с применением нейронной сети для классификации *Squeezenet* на симуляторе устройства на базе *СБИС К1879ВМ8Я*.

- Установите дистрибутив *NMDL* так, как описано в разделе Установка.

- Перейдите в каталог *bin* в директории установки *NMDL* (по-умолчанию в Windows "c:\Program Files\Module\NMDL", по-умолчанию в Linux "/opt/nmdl") и запустите демонстрационную программу *nmdl_gui*.

- Выберите устройство с помощью меню *File – Open Board...*. Убедитесь, что в диалоговом окне выбран симулятор.

- Выберите описание модели с помощью меню *File – Open Description...*. В диалоговом окне выбора файла выберите файл *PATH_TO_NMDL/nmdl_ref_data/squeezenet_imagenet/description08.xml*.

- Выберите обрабатываемое изображение с помощью меню *File – Open Picture...*. В диалоговом окне выбора файла выберите файл *PATH_TO_NMDL/nmdl_ref_data/squeezenet_imagenet/frame.bmp*.

- Запустите обработку с помощью меню *File – Run*. В результате обработки всплывёт окно классификации с вычисленными вероятностями в порядке уменьшения. Верный класс для выбранного изображения - "lakeside".

1.4. Производительность NMDL

В таблицах приводятся значения производительности (кадры в секунду FPS) и значения задержки от начала обработки кадра до получения результата (Latency). Рядом с названием сети в скобках указан размер обрабатываемого изображения.

Таблица 1.1 - FPS

	MC121.01	**NMStick**	**MC127.05 и NMCard (multi unit mode)**	**MC127.05 и NMCard (single unit mode)**
alexnet (227x227)	3,45	3,2	12,6	13
inception v3 (299x299)	0,63	0,6	12,8	20,3
inception v3 (512x512)	0,24	0,23	3,93	5,44
resnet 18 (224x224)	2,28	2,2	25	47
resnet 50 (224x224)	0,8	0,75	12,2	20,6
squeezenet (224x224)	8,3	8	74,4	100
u-net (512x512)*	-	-	2	2
yolo v2 tiny (416x416)	1,16	1,1	21	30,4
yolo v3 (416x416)	0,1	0,09	3,7	4,5
yolo v3 tiny (416x416)	1,44	1,38	27,3	35,3
yolo v5s (640x640)	-	-	4,7	5,3
yolo v5l (640x640)	-	-	1,39	1,43

Таблица 1.2 - Latency (ms)

	MC121.01	**NMStick**	**MC127.05 и NMCard (multi unit mode)**	**MC127.05 и NMCard (single unit mode)**
alexnet (227x227)	290	302	79	308
inception v3 (299x299)	1587	1653	78	197
inception v3 (512x512)	4166	4340	254	735
resnet 18 (224x224)	439	457	40	85
resnet 50 (224x224)	1250	1300	82	194
squeezenet (224x224)	120	125	13	40
u-net (512x512)*	-	-	500	2000
yolo v2 tiny (416x416)	862	898	47	132
yolo v3 (416x416)	10000	10416	270	889
yolo v3 tiny (416x416)	694	725	36	113
yolo v5s (640x640)	-	-	212	754
yolo v5l (640x640)	-	-	720	2797

* В модели u-net произведена замена слоёв *transposed_convolution* на *upsampling*.

2. Состав ПО

ПО реализации нейронных сетей состоит из программных модулей (API), утилит и руководства.

Файлы API для разработки программ с использованием *NMDL*:

- *nmdl.dll/nmdl.so* - программный модуль для применения обученной нейронной сети. См. раздел "Описание идентификаторов, функций и структур nmdl".
- *nmdl.lib* - библиотека для раннего связывания программ с *NMDL* в среде MSVC++.
- *nmdl.h* - заголовочный файл с описанием структур и функций API.
- *nmdl_compiler.dll/nmdl_compiler.so* - программный модуль - компилятор моделей *ONNX*/*DarkNet* во внутреннее представление. См. "Описание идентификаторов и функций nmdl_compiler"
- *nmdl_compiler.lib* - библиотека для раннего связывания модуля компилятора моделей в среде MSVC++.
- *nmdl_compiler.h* - заголовочный файл с описанием структур и функций компилятора моделей.
- *nmdl_image_converter.dll/nmdl__image_converter.so* - программный модуль для подготовки обрабатываемых изображений. См. "Описание идентификаторов и функций nmdl_image_converter"
- *nmdl_image_converter.dll / nmdl__image_converter.so* - a program module for preparing processed images. See "Description of nmdl_image_converter identifiers and functions"
- *nmdl_image_converter.lib* - модуль для раннего связывания модуля подготовки изображений в среде MSVC++.
- *nmdl_image_converter.h* - заголовочный файл с описанием структур и функций для подготовки изображений.

Заголовочные файлы и библиотеки раннего связывания размещаются в каталогах *include* и *lib* директории *NMDL*.

Утилиты:

- *nmdl_compiler_console* - утилита командной строки для компиляции моделей из форматов *ONNX* и *DarkNet* во внутренний формат для загрузки на вычислительные модули. Файл модели *ONNX* обычно имеет расширение *.onnx*. Модель в формате *DarkNet* сохраняется в двух файлах - с расширением *.cfg* и расширением *.weights*. Подготовленная модель для устройств *MC121.01* и *NMStick* имеет расширение *.nm7*. Модель для *MC127.05* и *NMCard* имеет расширение *.nm8*. См. раздел "Компиляция модели".

- *nmdl_nmdl_image_converter_console* - утилита командной строки для подготовки обрабатываемых изображений. См. раздел "Подготовка изображений".

- *nmdl_gui* - оконная утилита для демонстрации функциональных возможностей *NMDL*. См. раздел "Демонстрационная программа".

3. Установка

3.1. Установка в Windows

NMDL распространяется в виде установочного дистрибутива. Поддерживаются только 64-х битовые системы.

Для установки дистрибутива запустите исполняемый файл инсталлятора с правами администратора. Следуйте инструкциям мастера установки.

Для работы с программными модулями необходимо включить их в область "видимости" операционной системы. Одно из решений - создание переменной среды окружения, например, с именем *NMDL*, в которой записывается путь к каталогу с заголовочными и бинарными файлами, и добавление созданной переменной к переменной окружения *PATH*.

Для использования модуля nmdl в C/C++ программах включите в исходный файл директиву #include "nmdl.h" и свяжите программу с библиотекой *NMDL*. Если используется среда разработки MSVC++, то для раннего связывания подключите к проекту файл *nmdl.lib*. Можно создать и использовать переменную окружения *NMDL* для задания пути к файлам *nmdl.h* и *nmdl.lib*. Таким же образом можно подключить модули *nmdl_compiler* и *nmdl_image_converter*.

3.2. Установка в Linux

Распространяется в виде .deb пакета. Поддерживаются только 64-х битовые системы семейства Debian.

Для установки используйте менеджер пакетов dpkg.

Например:

```
dpkg -i NMDL.deb
```

3.3. Дополнительные компоненты

Вместе с программным модулем можно получить дополнительные компоненты для проверки и демонстрации работы *NMDL*.

Компоненты распространяются в архивах:

- *nmdl_ref_data_alexnet_imagenet.tar* - архив для демонстрации работы сети *ALEXNET*.
- *nmdl_ref_data_inception_v3_imagenet.tar* - архив для демонстрации работы сети *INCEPTION V3*.
- *nmdl_ref_data_resnet_18_imagenet.tar* - архив для демонстрации работы сети *RESNET18*.

- *nmdl_ref_data_resnet_50_imagenet.tar* - архив для демонстрации работы сети *RESNET50*.
- *nmdl_ref_data_squeezenet_imagenet.tar* - архив для демонстрации работы сети *SQUEEZENET*.
- *nmdl_ref_data_unet_no_transp_conv_covid.tar* - архив для демонстрации работы сети *U-NET*. В модели произведена замена слоёв *transposed_convolution* на *upsampling*.
- *nmdl_ref_yolo2_tiny_pascal_voc.tar* - архив для демонстрации работы сети *YOLONET V2 TINY*.
- *nmdl_ref_yolo3_coco.tar* - архив для демонстрации работы сети *YOLONET V3*.
- *nmdl_ref_yolo3_tiny_coco.tar* - архив для демонстрации работы сети *YOLONET V3 TINY*.
- *nmdl_ref_yolo5s_coco.tar* - архив для демонстрации работы сети *YOLONET V5S*. Работает только на модулях со СБИС К1879ВМ8Я - МС127.05, NMCard, NMMezzo, NMQuad, а также на симуляторе.

В каталогах содержатся файлы:

- *frame.bmp* - тестовое изображение.
- *model.cfg* - модель в формате *DARKNET*.
- *model.onnx* - модель в формате *ONNX*.
- *model.weights* - весовые коэффициенты модели в формате *DARKNET*.
- *description07.xml* - файл описания модели для устройств *MC121.01* и *NMStick* для запуска в программе nmdl_gui.
- *description08.xml* - файл описания модели для модулей *MC127.05*, *NMCard* и симулятора для запуска в программе nmdl_gui.

Для подготовки демонстрационных данных, которые будут обрабатываться на вычислительном модуле необходимо распаковать их в каталог *nmdl_ref_data*, выполнить компиляцию моделей и подготовку изображений так, как это описывается в разделах "Компиляция модели" и "Подготовка изображений" данного руководства. Для удобства компиляции в архив включены скрипты *prepare.cmd* и *prepare.sh*. В результате работы скрипта в каждом каталоге появятся файлы:

- *frame07* - подготовленное для обработки на *MC121.01* и *NMStick* изображение.
- *frame08* - подготовленное для обработки на *MC127.05* и *NMCard* изображение.
- *model.nm7* - компилированная модель для загрузки на *MC121.01* и *NMStick*.
- *model.nm8* - компилированная модель для загрузки на *MC127.05* и *NMCard* (режим "single unit").

- *model_mu.nm8* - компилированная модель для загрузки на *MC127.05* и *NMCard* (режим "multi unit").

3.4. Подготовка модуля MC121.01

При использовании NMDL с устройствами *MC121.01* необходимо до подключения модуля к USB-разъёму ПК установить следующие перемычки (см. рисунок 3.1):

- Контакт 1-2 разъема X9.
- Контакт 3-4 разъема X9.
- Контакт 5-6 разъема X9.
- При использовании питания от USB контакт 1-2 разъема X11 (при использовании блока питания контакт разомкнуть).
- Контакт 1-2 разъема X18.

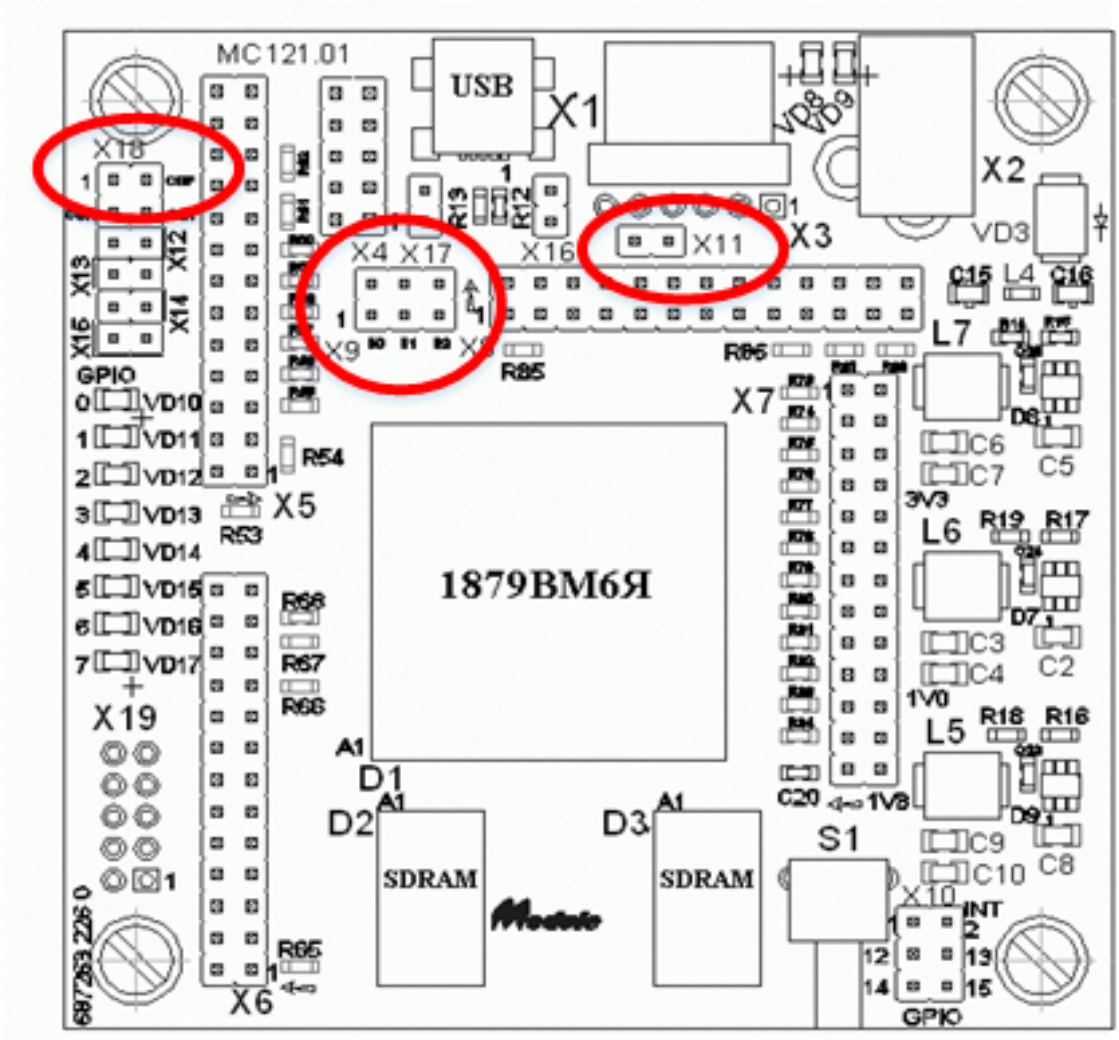

Рисунок 3.1 - Перемычки на модуле MC121.01

3.5. Подготовка устройства NMStick

При использовании *NMDL* с *NMStick* необходимо выполнить инсталляцию ПО поддержки устройства (входит в комплект поставки изделия) и подключить устройство к USB-разъёму ПК.

3.6. Подготовка модулей MC127.05, NMCard, NMMez

Для работы с *NMDL* необходимо установить плату в свободный слот PCIe (для *NMMezzo* в разъём PCIe несущей платы) и выполнить инсталляцию ПО поддержки модуля, входящее в комплект поставки изделия.

4. Компиляция модели

Нейронная сеть должна быть предварительно обучена с помощью нейросетевого пакета (например, Microsoft® Cognitive Toolkit - CNTK), и преобразована к форматам *ONNX* или *DarkNet*.

Исходная модель в формате *ONNX*, как правило, хранится в файле со структурой Google Protocol Buffer и имеет расширение **.onnx*.

Модели для сетей типа *YOLO* могут храниться в формате *DarkNet.* В этом случае модель описывается двумя файлами - файл с расширением **.cfg* содержит описание графа обработки, файл **.weights* содержит веса для свёрток.

Файлы моделей *ONNX* и *DarkNet* являются результатом обучения нейронной сети и одновременно исходными данными для компилятора, который в результате обработки создаёт файлы с расширением *nm7* для устройств *MC121.01* и *NMStick*, или *nm8* для модулей *MC127.05*, *NMCard* и симулятора.

Если в модели ONNX входные и выходные тензоры имеют динамические размерности, то для компиляции модели необходимо сделать их статическими. Фиксацию можно сделать при помощи следующего примера на python. Здесь предполагается, что размерности входного тензора динамические. Мы фиксируем их к значениям [1, 3, 600, 640], где 1 - размер пакета (batch), 3 - каналы, 600 - высота, 640 - ширина. Имя входного слоя - "input1", имена выходных слоёв - "output1", "output2" и "output3". Выходные размеры будут вычисляться автоматически:

```
   #Model fixation script
import onnx
from onnx.tools import update_model_dims
from onnx import helper, shape_inference

model = onnx.load('path/to/the/dynamic_dim_model.onnx')

# update model dimensions
variable_length_model = update_model_dims.update_inputs_outputs_dims(model, \
{'input1': [1, 3, 600, 640]}, {'output1': ['1', 'C','H','W'], \
'output2': ['1', 'C','H','W'], 'output3': ['1', 'C','H','W']})

# out dim will be calculated automatically
inferred_model = shape_inference.infer_shapes(variable_length_model)

# Check the model
onnx.checker.check_model(inferred_model)

# Save the ONNX model
onnx.save(inferred_model, 'static_dim_model.onnx')
```

Модели *nm7* и *nm8* являются бинарными образами скомпилированных пользовательских моделей. Их структура не документируется и зависит от целевого устройства и версии компилятора.

Компилятор выполнен в виде динамически загружаемого модуля и может быть встроен в программу пользователя. Можно также воспользоваться консольной утилитой nmdl_compiler_console для выполнения преобразования моделей из командной строки.

```
nmdl_compiler_console BOARD_TYPE NN_TYPE
      SRC_FILENAME DST_FILENAME IS_MULTI_UNIT [WEIGHTS_FILENAME]
```

Аргументы командной строки:

- *BOARD_TYPE* - тип модуля. Допустимые значения: *"MC12101"* - преобразование модели в формат nm7 для устройств *MC121.01* и *NMStick*, *"MC12705"* - преобразование модели в формат nm8 для модулей *MC127.05*, *NMCard* и симулятора.
- *NN_TYPE* - формат файла входной модели. Допустимые значения: *"ONNX"* или *"DARKNET"*.
- *SRC_FILENAME* - имя файла исходной модели.
- *DST_FILENAME* - имя файла выходной модели.
- *IS_MULTI_UNIT* - 0: модель будет обрабатываться в режиме *"single unit"*, 1: модель будет обрабатываться в режиме *"multi unit"* (см. раздел "Режимы обработки").
- *WEIGHTS_FILENAME* - имя файла с весовыми коэффициентами модели. Нужен только для моделей в формате *DarkNet*.

Пример компиляции модели squeezenet для прогона на одном юните:

```
>nmdl_compiler_console MC12705 ONNX
      squeezenet.onnx squeezenet.nm8 0
```

Пример компиляции модели squeezenet для прогона в режиме "multy unit":

```
>nmdl_compiler_console MC12705 DARKNET
      yolo2t.onnx yolo2t.nm8 1 yolo2t.weights
```

4.1. Поддерживаемые операции

Компилятором поддерживается следующий набор операций:

1. Abs
2. Add
3. AveragePool
   - AveragePool2x2, no pad, stride=1
   - AveragePool2x2, no pad, stride=2
   - AveragePool3x3, no pad, stride=2
   - AveragePool3x3, pad, stride=2
4. BatchNormalization (при условии, что операция выполняется сразу после свёртки).

5. Clip*.
6. Concat (по осям каналов и ширины, количество входных тензоров - не более 7).
7. Convolution
    - Conv1x1, stride=1
    - Conv1x1, stride=2
    - Conv3x3, no pad, all strides
    - Conv3x3, pad, stride=1
    - Conv3x3, pad, stride=2
    - Conv5x5, no pad, all strides
    - Conv5x5, pad, stride=1
    - Conv7x7, no pad, all strides
    - Conv7x7, pad, stride=2
    - Conv11x11, no pad, all strides
    - Conv7x1, pad_w, stride=1
    - Conv1x7, pad_h, stride=1
    - Conv3x1, pad_w, stride=1
    - Conv1x3, pad_h, stride=1
8. ConvTranspose (kernel 2x2, stride 2x2)
9. Div
10. GEMM
11. GlobalAveragePool
12. Leaky Relu
13. Mat Mul
14. MaxPool
    - MaxPool2x2, no pad, stride=1
    - MaxPool2x2, no pad, stride=2
    - MaxPool3x3, no pad, stride=2

- MaxPool3x3, pad, stride=2
- MaxPoolNxN (N - odd value), stride=1*

15. Mul
16. Pad
17. PRelu
18. Relu
19. Reshape
20. Resize* (resize nearest + resize linear, half fixel, scale 2x2)
21. Sigmoid
22. Slice
23. Sub*
24. Transpose
25. Upsample nearest*

* - только для модулей MC127.05, NMCard, NMMezzo и NMQuad.

NMDL может работать с моделями с одним обрабатываемым изображением, то есть обрабатывается один входной тензор.

Количество терминальных узлов (выходные тензоры) - не более 5.

5. Подготовка изображений

Обрабатываемые изображения необходимо предварительно сконвертировать в вещественный формат (тензор). Сделать это можно с помощью утилиты *nmdl_image_converter_console*

```
nmdl_image_converter_console SRC_FILE DST_FILE W H F DR DG DB AR AG AB BT
```

Аргументы командной строки:

- *SRC_FILE* - имя входного файла (*bmp*, *gif*, *jpg*, *emf*, *png*, *tiff*),
- *DST_FILE* - имя выходного файла,
- *W* - ширина тензора изображения,
- *H* - высота тензора изображения,
- *F* - порядок следования цветовых каналов в тензоре изображения (допустимые значения: *rgb*, *rbg*, *grb*, *gbr*, *brg*, *bgr*, intensity - один канал градации серого),
- *DR* - делитель для красного канала пикселя в выражении $dst = src / D + A$ (вещественная величина),
- *DG* - делитель для зелёного канала пикселя в выражении $dst = src / D + A$ (вещественная величина),
- *DB* - делитель для голубого канала пикселя в выражении $dst = src / D + A$ (вещественная величина),
- *AR* - слагаемое для красного канала пикселя в выражении $dst = src / D + A$ (вещественная величина).
- *AG* - слагаемое для зелёного канала пикселя в выражении $dst = src / D + A$ (вещественная величина).
- *AB* - слагаемое для голубого канала пикселя в выражении $dst = src / D + A$ (вещественная величина).
- *BT* - тип модуля на котором предполагается обработка (допустимые значения: mc12101, mc12705).

Для изображений с градацией серого, когда аргумент F задан intensity, используются только делитель DR и слагаемое AR. Остальные делители (DG и DB) и слагаемые (AG и AB) игнорируются и могут быть установлены в любые значения.

Программа масштабирует входное изображение относительно центра в соответствии с заданными аргументами.

Подготовленные изображения имеют либо планарный (для *MC121.01* и *NMStick*), либо пиксельный (для *MC127.05* и *NMCard*) формат расположения элементов типа *float32*.

В планарном формате в файл записываются поочерёдно плоскости каждого из каналов. В этом случае буфер можно представить как массив в стиле C/C++:

```
float image[CHANNELS][HEIGHT][WIDTH];
```

То есть самый быстро меняющийся индекс - это индекс по ширине изображения. Количество каналов - три (RGB каналы), или один для одноканальных изображений (градации серого). В подготовленном буфере размер выравнивается по ширине изображения. Для изображений с чётной шириной размер буфера составит: *size = width * height * channels* (float32), для изображений с нечётной шириной: *size = (width + 1) * height * channels* (float32).

В пиксельном формате в файл записываются последовательно каналы пикселей изображения. Буфер можно представить как массив в стиле C/C++:

```
float image[HEIGHT][WIDTH][CHANNELS];
```

То есть самый быстро меняющийся индекс - это индекс по каналам. Количество каналов - три (RGB каналы), или один для одноканальных изображений (градации серого). В подготовленном буфере размер выравнивается по каналам изображения до ближайшего чётного значения. Размер буфера с подготовленным изображением для модулей *MC127.05* и *NMCard* составит: *size = width * height * (channels + 1)* (float32).

6. Демонстрационная программа

Для демонстрации функциональных возможностей библиотеки NMDL была разработана утилита nmdl_gui. В задачи утилиты входят:

- Инициализация библиотеки *NMDL*;
- Обнаружение и идентификация доступных ускорителей (*симулятор*, *MC121.01*, *MC127.05*, *NMStick*, *NMCard*);
- Загрузка файла модели нейронной сети (*.nm7* или *.nm8*);
- Загрузка файла описания нейронной сети (*xml*);
- Предварительная обработка и загрузка изображений на вычислитей модуль;
- Запуск обработки на вычислительном модуле;
- Обработка и вывод результатов.

При запуске программы появляется главное окно программы со строками меню, состояния и клиентской областью.

В меню расположены органы управления программой. В клиентской области располагается изображение и результаты обработки. В строке состояния выводится следующая информация:

- Выбранный ускоритель (*симулятор*, *MC121.01*, *MC127.05*, *NMStick*, *NMCard*);
- Выбранная модель нейронной сети;
- Выбранное описание нейронной сети;
- Выбранное изображение;
- Скорость обработки (кадров/с).

Внешний вид программы в процессе работы приведён на рисунке 6.1.

Рисунок 6.1 - Внешний вид программы в процессе работы

Выбор модуля осуществляется в диалоговом окне *Open Board*, внешний вид которого приведён на рисунке 6.2. Окно содержит список доступных в системе ускорителей, кнопки выбора, а также возможность идентификации устройства посредством светодиодной индикации.

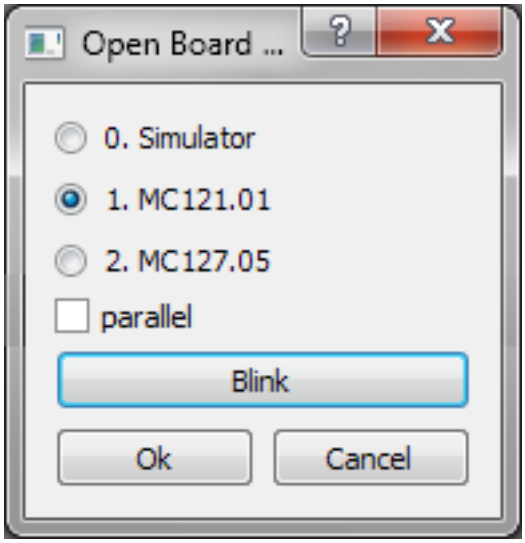

Рисунок 6.2 - Внешний вид окна выбора модуля

Включение флага *"parallel"* обеспечивает обработку потока изображений в параллельном режиме. В этом режиме производится, по возможности, одновременная обработка нескольких кадров с использованием всех обнаруженных выбранных ускорителей в системе и всех юнитов выбранного ускорителя. При достаточной пропускной способности канала, когда время передачи данных занимает существенно меньше времени инференса, производительность обработки одного кадра (один инференс) увеличивается кратно в соответствии с количеством ускорителей. Например, при использовании четырёх ускорителей *MC127.05* в режиме *"single unit"* производительность увеличится примерно в 16 раз (четыре юнита в четырёх ускорителях), в режиме "multi unit" производительность увеличится приблизительно в четыре раза. Параллельная обработка производится только при запуске в автоматическом режиме, когда обрабатывается поток изображений - меню *"File->Run Auto"*.

Выбор описания нейронной сети осуществляется в диалоговом окне *Open Description*. Описание нейронной сети осуществляется в формате *XML* и содержит информацию о необходимом формате изображения, а также параметры интерпретации выходных параметров.

Параметр	**Описание**
model	Строка с названием файла модели нейронной сети в одном из следующих форматах: • **.nm7* - описание модели для загрузки на модули *MC121.01* и *NMStick*. • **.nm8* - описание модели для загрузки на модули *MC127.05*, *NMCard* или на симулятор.

Параметр	**Описание**
	• **.onnx* или **.onnx* - описание модели в формате ONNX Protobuf. Модель может быть загружена на любой модуль. Перед загрузкой на модуль модель предварительно конвертируется в **.nm7* или в **.nm8* в зависимости от типа модуля. • **.cfg* - описание модели в формате DARKNET. Файл с весовыми коэффициентами **.weights* должен размещаться в том же каталоге и иметь то же название, что и файл описания модели **.cfg*. Модель может быть загружена на любой модуль. Перед загрузкой на модуль модель предварительно конвертируется в **.nm7* или в **.nm8* в зависимости от типа модуля. Можно указать как абсолютный путь к файлу, так и путь, относительно размещения файла описания XML.
format	Формат пикселя, к которому будет преобразованы пиксели входного изображения перед обработкой. Может принимать значения: • rgb • rbg • grb • gbr • brg • bgr Это значение соответствует порядку следования каналов входного тензора.
divider	Масштабирующий коэффициент для каждой цветовой компоненты пикселя входного изображения. Используется при преобразовании изображения во входной тензор (см. раздел "Подготовка изображений").
adder	Коэффициент смещения для каждой цветовой компоненты пикселя входного изображения. Используется при преобразовании изображения во входной тензор (см. раздел "Подготовка изображений").
neural_network	Выбранная нейронная сеть Может принимать значения: • *classifier* - нейронная сеть - классификатор, например *ALEXNET*, *RESNET*, *SQUEEZENET*.

Параметр	Описание
	• *unet* - нейронная сеть семейства U-Net. • *yolo2* - нейронная сеть семейства YOLO V2. • *yolo3* - нейронная сеть семейства YOLO V3. • *yolo5* - нейронная сеть семейства YOLO V5. Результатом работы классификатора является вектор вероятностей принадлежности изображения определённым классам. В демонстрационной программе классы и вероятности обнаружения выводятся во всплывающем окне. Сети YOLO в результате обработки выдают информацию о нескольких обнаруженных объектах и их расположении на исходном изображении. В демонстрационной программе обнаруженые объекты обозначаются непосредственно на исходном.изображении в описывающих прямоугольниках.
yolo_anchors	Параметры начальной инициализации детектируемых прямоугольников (только для сетей *YOLO*).
yolo_confidence_threshold	Порог для отображения результатов детектирования (только для сетей *YOLO*).
yolo_iou_threshold	Порог пересекающихся прямоугольников детектирования (только для сетей *YOLO*)
labels	Список строк с названиями классов, которые будут отображаться в окне результата обработки.

Диалоговое окно *Open Picture* позволяет открыть одно или несколько изображений для обработки.

При наличии всех данных становится доступной кнопка *Run*, запускающая обработку. Запуск можно инициировать с помощью комбинации *"Ctrl+R"*. Для каждого переданного изображения осуществляется предварительная обработка в соответствии с заданным описанием нейронной сети для последующей обработки на ускорителе. В результате обработки заполняется выходная структура с N тензорами. В зависимости от нейронной сети, выходная структура интерпретируется различным образом. Для сетей классификации (таких как *SqueezeNet*) появляется дополнительное окно с классами и вероятностью. Внешний вид этого окна приведён на рисунке 6.3.

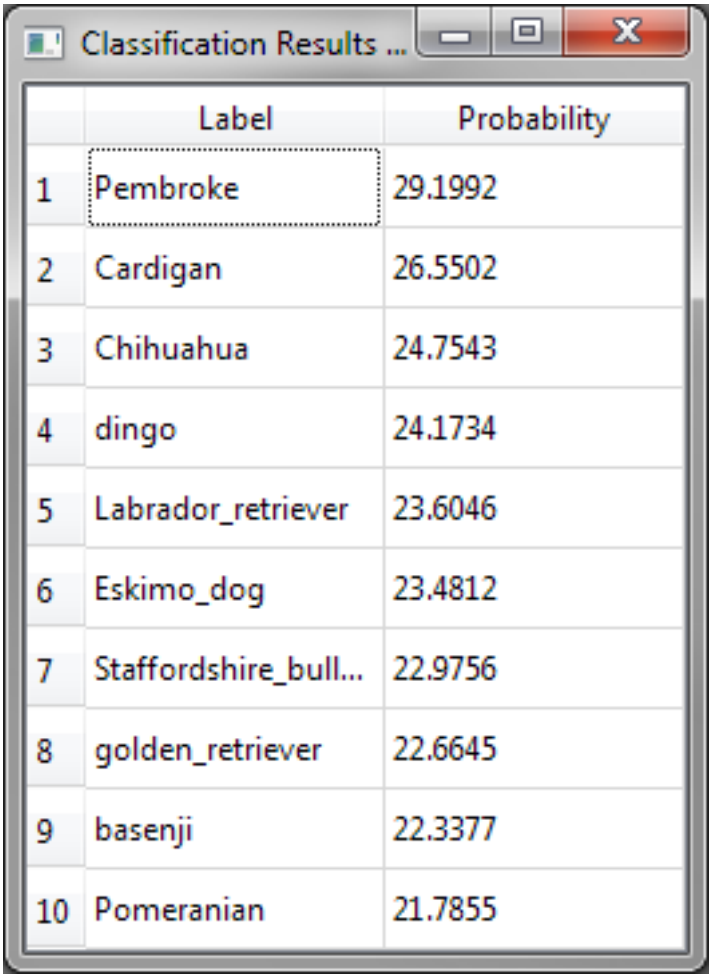

Рисунок 6.3 - Внешний вид окна результатов классификации

Для сетей, которые выполняют местоопределение объекта (таких как *YOLO*), результат показывается непосредственно на исходном изображении. После окончания работы ускорителя осуществляется наложение прямоугольников на обнаруженные объекты, приводится название класса и вероятность обнаружения. Пример наложения результата на изображение приведён на рисунке 6.4.

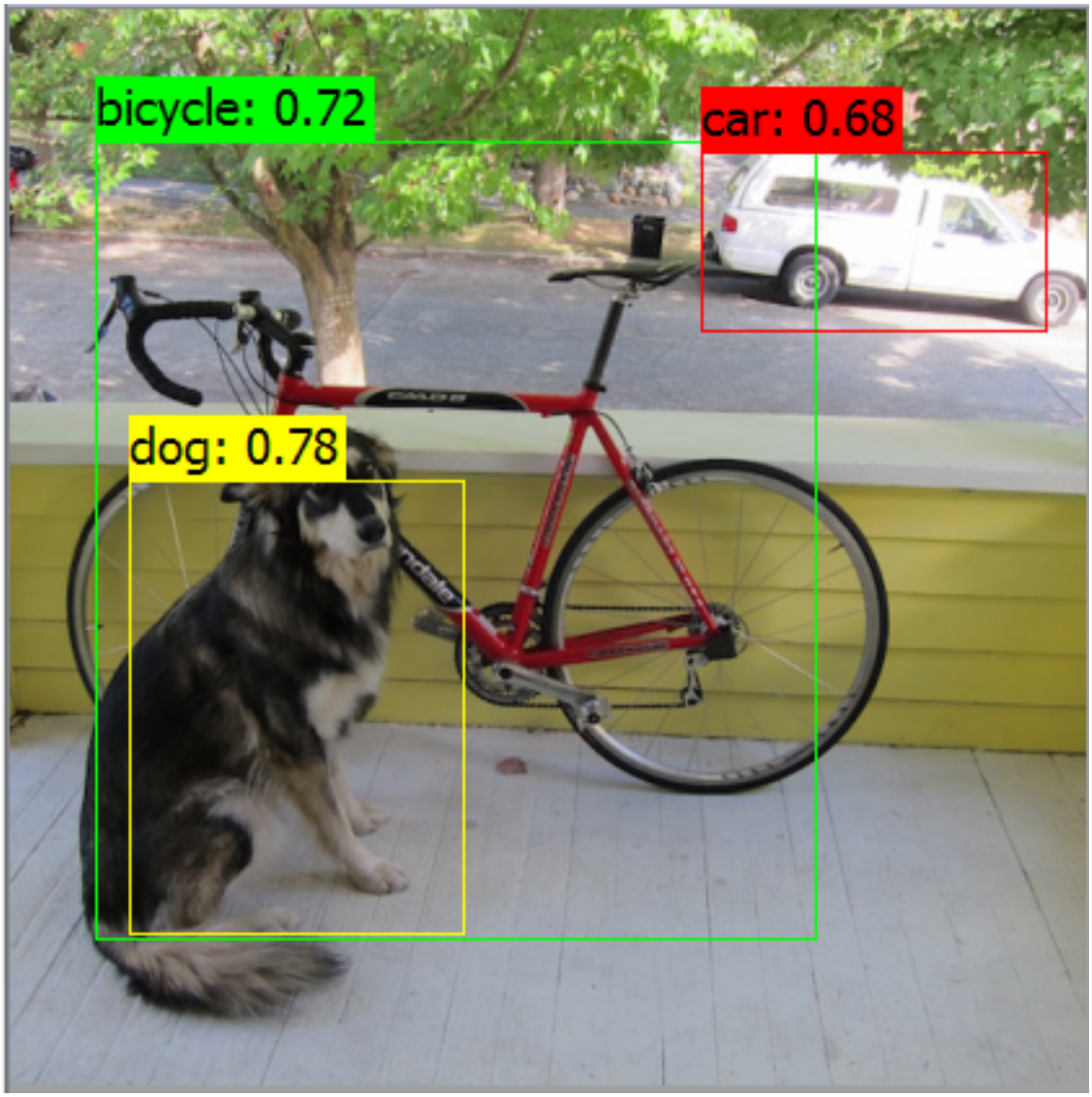

Рисунок 6.4 - Отображение результатов детектирования

Программа позволяет обработать последовательность изображений. Для этого необходимо при выборе изображений выделить несколько файлов. После этого при запуске обработки (*Run*) будет обработано одно изображение. При повторном запуске - второе и т. д. Можно также запустить обработку в автоматическом режиме (*Run Auto*). Для останова автоматической обработки нажмите клавишу "*пробел*".

Для отображения дополнительной информации используется строка состояния. В строку выводится название выбранного устройства, файла описания нейронной сети, файла изображения, а также измеренное программой значение fps с учётом пересылки кадра и полученрия результата. Скорость обработки без учёта времени пересылок выводится в скобках.

7. Пример использования NMDL

В примере демонстрируется применение библиотеки *NMDL*, программных модулей для компиляции модели и подготовки изображений. Пример состоит из файла исходного кода *example1.cpp* и скрипта сборки *CMakeLists.txt* в каталоге *"examples/example1"* установочной директории.

Предполагается, что исходные данные для обработки находятся в каталоге *"nmdl_ref_data/squeezenet"* в каталоге установки, это означает, что в примере демонстрируется обработка нейронной сети *squeezenet*. Исходными данными являются:

- Модель нейронной сети в формате ONNX - файл *"nmdl_ref_data/squeezenet/model.onnx"*
- Обрабатываемое изображение в формате BMP - файл *"nmdl_ref_data/squeezenet/frame.bmp"*

В примере показаны вызовы функций библиотек в порядке, необходимом для осуществления корректной работы.

```
  001 #include <array>
002 #include <fstream>
003 #include <iostream>
004 #include <string>
005 #include <unordered_map>
006 #include <vector>
007 #include "nmdl.h"
008 #include "nmdl_compiler.h"
009 #include "nmdl_image_converter.h"
010
011 //#define _USE_DARKNET_
012
013 namespace {
014
015 auto Call(NMDL_COMPILER_RESULT result, const std::string &function_name) {
016     static std::unordered_map<NMDL_COMPILER_RESULT, std::string> map = {
017         {NMDL_COMPILER_RESULT_OK,                       "OK"},
018         {NMDL_COMPILER_RESULT_MEMORY_ALLOCATION_ERROR, "MEMORY_ALLOCATION_ERROR"},
019         {NMDL_COMPILER_RESULT_MODEL_LOADING_ERROR,     "MODEL_LOADING_ERROR"},
020         {NMDL_COMPILER_RESULT_INVALID_PARAMETER,       "INVALID_PARAMETER"},
021         {NMDL_COMPILER_RESULT_INVALID_MODEL,           "INVALID_MODEL"},
022         {NMDL_COMPILER_RESULT_UNSUPPORTED_OPERATION,   "UNSUPPORTED_OPERATION"}
023     };
024     if(result != NMDL_COMPILER_RESULT_OK) {
025         throw std::runtime_error(function_name + ": " + map[result] + ": " +
026             NMDL_COMPILER_GetLastError());
027     }
028     return NMDL_RESULT_OK;
029 }
030
031 auto Call(NMDL_RESULT result, const std::string &function_name) {
032     static std::unordered_map<NMDL_RESULT, std::string> map = {
033         {NMDL_RESULT_OK,                            "OK"},
034         {NMDL_RESULT_INVALID_FUNC_PARAMETER,        "INVALID_FUNC_PARAMETER"},
035         {NMDL_RESULT_NO_LOAD_LIBRARY,               "NO_LOAD_LIBRARY"},
036         {NMDL_RESULT_NO_BOARD,                      "NO_BOARD"},
037         {NMDL_RESULT_BOARD_RESET_ERROR,             "BOARD_RESET_ERROR"},
038         {NMDL_RESULT_INIT_CODE_LOADING_ERROR,       "INIT_CODE_LOADING_ERROR"},
039         {NMDL_RESULT_CORE_HANDLE_RETRIEVAL_ERROR,   "CORE_HANDLE_RETRIEVAL_ERROR"},
040         {NMDL_RESULT_FILE_LOADING_ERROR,            "FILE_LOADING_ERROR"},
041         {NMDL_RESULT_MEMORY_WRITE_ERROR,            "MEMORY_WRITE_ERROR"},
042         {NMDL_RESULT_MEMORY_READ_ERROR,             "MEMORY_READ_ERROR"},
043         {NMDL_RESULT_MEMORY_ALLOCATION_ERROR,       "MEMORY_ALLOCATION_ERROR"},
```

```
044         {NMDL_RESULT_MODEL_LOADING_ERROR,           "MODEL_LOADING_ERROR"},
045         {NMDL_RESULT_INVALID_MODEL,                 "INVALID_MODEL"},
046         {NMDL_RESULT_BOARD_SYNC_ERROR,              "BOARD_SYNC_ERROR"},
047         {NMDL_RESULT_BOARD_MEMORY_ALLOCATION_ERROR, "BOARD_MEMORY_ALLOCATION_ERROR"},
048         {NMDL_RESULT_NN_CREATION_ERROR,             "NN_CREATION_ERROR"},
049         {NMDL_RESULT_NN_LOADING_ERROR,              "NN_LOADING_ERROR"},
050         {NMDL_RESULT_NN_INFO_RETRIEVAL_ERROR,       "NN_INFO_RETRIEVAL_ERROR"},
051         {NMDL_RESULT_MODEL_IS_TOO_BIG,              "MODEL_IS_TOO_BIG"},
052         {NMDL_RESULT_NOT_INITIALIZED,               "NOT_INITIALIZED"},
053         {NMDL_RESULT_BUSY,                          "BUSY"},
054         {NMDL_RESULT_UNKNOWN_ERROR,                 "UNKNOWN_ERROR"}
055     };
056     if(result != NMDL_RESULT_OK) {
057         throw std::runtime_error(function_name + ": " + map[result]);
058     }
059     return NMDL_RESULT_OK;
060 }
061
062 template <typename T>
063 auto ReadFile(const std::string &filename) {
064     std::ifstream ifs(filename, std::ios::binary | std::ios::ate);
065     if(!ifs.is_open()) {
066         throw std::runtime_error("Unable to open input file: " + filename);
067     }
068     auto fsize = static_cast<std::size_t>(ifs.tellg());
069     ifs.seekg(0);
070     std::vector<T> data(fsize / sizeof(T));
071     ifs.read(reinterpret_cast<char*>(data.data()), data.size() * sizeof(T));
072     return data;
073 }
074
075 void ShowNMDLVersion() {
076     std::uint32_t major = 0;
077     std::uint32_t minor = 0;
078     std::uint32_t patch = 0;
079     Call(NMDL_GetLibVersion(&major, &minor, &patch), "GetLibVersion");
080     std::cout << "Lib version: " << major << "." << minor
081             << "." << patch << std::endl;
082 }
083
084 void CheckBoard(std::uint32_t required_board_type) {
085     std::uint32_t boards;
086     std::uint32_t board_number = -1;
087     Call(NMDL_GetBoardCount(required_board_type, &boards), "GetBoardCount");
088     std::cout << "Detected boards: " << boards << std::endl;
089     if(!boards) {
090         throw std::runtime_error("Board not found");
091     }
092 }
093
094 #ifdef _USE_DARKNET_
095 auto CompileModel(const std::string &config_filename,
096         const std::string &weights_filename,
097         std::uint32_t board_type,
098         bool is_multi_unit) {
099     float *nm_model = nullptr;
100     std::uint32_t nm_model_floats = 0u;
101     auto config = ReadFile<char>(config_filename);
102     auto weights = ReadFile<char>(weights_filename);
103     Call(NMDL_COMPILER_CompileDarkNet(is_multi_unit, board_type,
104             config.data(), config.size(), weights.data(), weights.size(),
105             &nm_model, &nm_model_floats), "CompileONNX");
106     std::vector<float> result(nm_model, nm_model + nm_model_floats);
107     NMDL_COMPILER_FreeModel(board_type, nm_model);
108     return result;
109 }
110 #else
111 auto CompileModel(const std::string &model_filename, std::uint32_t board_type,
112         bool is_multi_unit) {
113     float *nm_model = nullptr;
114     std::uint32_t nm_model_floats = 0u;
```

```
115     auto model = ReadFile<char>(model_filename);
116     Call(NMDL_COMPILER_CompileONNX(is_multi_unit, board_type, model.data(),
117             model.size(), &nm_model, &nm_model_floats), "CompileONNX");
118     std::vector<float> result(nm_model, nm_model + nm_model_floats);
119     NMDL_COMPILER_FreeModel(board_type, nm_model);
120     return result;
121 }
122 #endif
123
124 auto GetModelInformation(NMDL_HANDLE nmdl, std::uint32_t unit_num) {
125     NMDL_ModelInfo model_info;
126     Call(NMDL_GetModelInfo(nmdl, unit_num, &model_info), "GetModelInfo");
127     std::cout << "Input tensor number: " << model_info.input_tensor_num << std::endl;
128     for(std::size_t i = 0; i < model_info.input_tensor_num; ++i) {
129         std::cout << "Input tensor " << i << ": " <<
130             model_info.input_tensors[i].width << ", " <<
131             model_info.input_tensors[i].height << ", " <<
132             model_info.input_tensors[i].depth <<
133             std::endl;
134     }
135     std::cout << "Output tensor number: " << model_info.output_tensor_num << std::endl;
136     for(std::size_t i = 0; i < model_info.output_tensor_num; ++i) {
137         std::cout << "Output tensor " << i << ": " <<
138             model_info.output_tensors[i].width << ", " <<
139             model_info.output_tensors[i].height << ", " <<
140             model_info.output_tensors[i].depth <<
141             std::endl;
142     }
143     return model_info;
144 }
145
146 auto PrepareInput(const std::string &filename, std::uint32_t width,
147         std::uint32_t height, std::uint32_t board_type,
148         std::uint32_t color_format, const float rgb_divider[3],
149         const float rgb_adder[3]) {
150     auto bmp_frame = ReadFile<char>(filename);
151     std::vector<float> input(NMDL_IMAGE_CONVERTER_RequiredSize(
152             width, height, color_format, board_type));
153     if(NMDL_IMAGE_CONVERTER_Convert(bmp_frame.data(), input.data(), bmp_frame.size(),
154             width, height, color_format, rgb_divider, rgb_adder, board_type)) {
155         throw std::runtime_error("Image conversion error");
156     }
157     return input;
158 }
159
160 void WaitForOutput(NMDL_HANDLE nmdl, std::uint32_t unit_num, float *outputs[]) {
161     std::uint32_t status = NMDL_PROCESS_FRAME_STATUS_INCOMPLETE;
162     while(status == NMDL_PROCESS_FRAME_STATUS_INCOMPLETE) {
163         NMDL_GetStatus(nmdl, unit_num, &status);
164     };
165     double fps;
166
167     Call(NMDL_GetOutput(nmdl, unit_num, outputs, &fps), "GetOutput");
168     std::cout << "First four result values:" << std::endl;
169     for(std::size_t i = 0; i < 4; ++i) {
170         std::cout << outputs[0][i] << std::endl;
171     }
172     std::cout << "FPS:" << fps << std::endl;
173 }
174
175 }
176
177 int main() {
178     const std::uint32_t BOARD_TYPE = NMDL_BOARD_TYPE_SIMULATOR;
179     //const uint32_t BOARD_TYPE = NMDL_BOARD_TYPE_MC12705;
180     //const uint32_t BOARD_TYPE = NMDL_BOARD_TYPE_MC12101;
181     const std::uint32_t COMPILER_BOARD_TYPE =
182         BOARD_TYPE == NMDL_BOARD_TYPE_MC12101 ?
183         NMDL_COMPILER_BOARD_TYPE_MC12101 :
184         NMDL_COMPILER_BOARD_TYPE_MC12705;
185     const std::uint32_t IMAGE_CONVERTER_BOARD_TYPE =
```

```
186         BOARD_TYPE == NMDL_BOARD_TYPE_MC12101 ?
187         NMDL_IMAGE_CONVERTER_BOARD_TYPE_MC12101 :
188         NMDL_IMAGE_CONVERTER_BOARD_TYPE_MC12705;
189 #ifdef _USE_DARKNET_
190     const std::string DARKNET_CONFIG_FILENAME =
191             "../nmdl_ref_data/yolo_v3_tiny_coco/model.cfg";
192     const std::string DARKNET_WEIGHTS_FILENAME =
193             "../nmdl_ref_data/yolo_v3_tiny_coco/model.weights";
194     const std::string BMP_FRAME_FILENAME =
195             "../nmdl_ref_data/yolo_v3_tiny_coco/frame.bmp";
196     const NMDL_IMAGE_CONVERTER_COLOR_FORMAT IMAGE_CONVERTER_COLOR_FORMAT =
197         NMDL_IMAGE_CONVERTER_COLOR_FORMAT_RGB;
198     const float NM_FRAME_RGB_DIVIDER[3] = {255.0f, 255.0f, 255.0f};
199     const float NM_FRAME_RGB_ADDER[3] = {0.0f, 0.0f, 0.0f};
200 #else
201     const std::string ONNX_MODEL_FILENAME =
202             "../nmdl_ref_data/squeezenet_imagenet/model.onnx";
203     const std::string BMP_FRAME_FILENAME =
204             "../nmdl_ref_data/squeezenet_imagenet/frame.bmp";
205     const NMDL_IMAGE_CONVERTER_COLOR_FORMAT IMAGE_CONVERTER_COLOR_FORMAT =
206         NMDL_IMAGE_CONVERTER_COLOR_FORMAT_BGR;
207     const float NM_FRAME_RGB_DIVIDER[3] = {1.0f, 1.0f, 1.0f};
208     const float NM_FRAME_RGB_ADDER[3] = {0.0f, 0.0f, 0.0f};
209 #endif
210     const std::size_t BATCHES = 4;
211     const std::size_t FRAMES = 5;
212
213     NMDL_HANDLE nmdl = 0;
214
215     try {
216         std::cout << "Query library version..." << std::endl;
217         ShowNMDLVersion();
218
219         std::cout << "Board detection... " << std::endl;
220         CheckBoard(BOARD_TYPE);
221
222         std::cout << "NMDL initialization... " << std::endl;
223         Call(NMDL_Create(&nmdl), "Create");
224
225         std::cout << "Use multi unit... " << std::endl;
226
227         std::cout << "Compile model... " << std::endl;
228 #ifdef _USE_DARKNET_
229         auto model = CompileModel(DARKNET_CONFIG_FILENAME, DARKNET_WEIGHTS_FILENAME,
230                 COMPILER_BOARD_TYPE, true);
231 #else
232         auto model = CompileModel(ONNX_MODEL_FILENAME, COMPILER_BOARD_TYPE, true);
233 #endif
234
235         std::array<const float*, NMDL_MAX_UNITS> models = {model.data()};
236         std::array<std::uint32_t, NMDL_MAX_UNITS> model_floats =
237                 {static_cast<std::uint32_t>(model.size())};
238         Call(NMDL_Initialize(nmdl, BOARD_TYPE, 0, 0, models.data(),
239                 model_floats.data()), "Initialize");
240
241         std::cout << "Get model information... " << std::endl;
242         auto model_info = GetModelInformation(nmdl, 0);
243
244         std::cout << "Prepare inputs... " << std::endl;
245         auto input = PrepareInput(BMP_FRAME_FILENAME, model_info.input_tensors[0].width,
246                 model_info.input_tensors[0].height, IMAGE_CONVERTER_BOARD_TYPE,
247                 IMAGE_CONVERTER_COLOR_FORMAT, NM_FRAME_RGB_DIVIDER, NM_FRAME_RGB_ADDER);
248         std::array<const float*, 1> inputs = {input.data()};
249
250         std::cout << "Reserve outputs... " << std::endl;
251         std::vector<std::vector<float>> output_tensors(model_info.output_tensor_num);
252         std::vector<float*> outputs(model_info.output_tensor_num);
253         for(std::size_t i = 0; i < model_info.output_tensor_num; ++i) {
254             output_tensors[i].resize(static_cast<std::size_t>(
255                     model_info.output_tensors[i].width) *
256                     model_info.output_tensors[i].height *
```

```
257                     model_info.output_tensors[i].depth);
258             outputs[i] = output_tensors[i].data();
259         }
260
261         std::cout << "Process inputs... " << std::endl;
262         for(std::size_t i = 0; i < FRAMES; ++i) {
263             Call(NMDL_Process(nmdl, 0, inputs.data()), "Process");
264             WaitForOutput(nmdl, 0, outputs.data());
265         }
266         NMDL_Release(nmdl);
267
268         std::cout << "Process batch... " << std::endl;
269
270         std::cout << "Compile model... " << std::endl;
271 #ifdef _USE_DARKNET_
272         model = CompileModel(DARKNET_CONFIG_FILENAME, DARKNET_WEIGHTS_FILENAME,
273                 COMPILER_BOARD_TYPE, false);
274 #else
275         model = CompileModel(ONNX_MODEL_FILENAME, COMPILER_BOARD_TYPE, false);
276 #endif
277         models = {model.data(), model.data(), model.data(), model.data()};
278         model_floats = {
279                 static_cast<std::uint32_t>(model.size()),
280                 static_cast<std::uint32_t>(model.size()),
281                 static_cast<std::uint32_t>(model.size()),
282                 static_cast<std::uint32_t>(model.size())};
283         Call(NMDL_Initialize(nmdl, BOARD_TYPE, 0, 0, models.data(),
284                 model_floats.data()), "Initialize");
285
286         std::uint32_t cnt_in = 0;
287         std::uint32_t cnt_out = 0;
288         for(auto i = 0u; i < BATCHES; ++i) {
289             Call(NMDL_Process(nmdl, (cnt_in++) % BATCHES, inputs.data()),
290                     "ProcessFrame");
291         }
292         for(auto i = BATCHES; i < FRAMES; ++i) {
293             WaitForOutput(nmdl, (cnt_out++) % BATCHES, outputs.data());
294             Call(NMDL_Process(nmdl, (cnt_in++) % BATCHES, inputs.data()),
295                     "ProcessFrame");
296         }
297         for(auto i = 0u; i < BATCHES; ++i) {
298             WaitForOutput(nmdl, (cnt_out++) % BATCHES, outputs.data());
299         }
300     }
301     catch (std::exception& e) {
302         std::cerr << e.what() << std::endl;
303     }
304     NMDL_Release(nmdl);
305     NMDL_Destroy(nmdl);
306
307     return 0;
308 }
```

Здесь выполняются следующие действия:

1 .. 9: Подключение заголовочных файлов. *"nmdl.h"* - описание библиотеки нейросетевой обработки, *"nmdl_compiler.h"* - описание библиотеки для компиляции моделей, *"nmdl_image_converter.h"* - описание библиотеки для подготовки изображений.

15 .. 29: Функция-обёртка для вызовов функций библиотеки компиляции моделей. В случае ошибки формирует вывод об ошибке и инициирует исключение.

31 .. 60: Функция-обёртка для вызовов функций библиотеки нейросетевой обработки. В случае ошибки формирует вывод об ошибке и инициирует исключение.

62 .. 73: Универсальная функция для чтения данных из файла в вектор.

75 .. 82: Функция вывода версии NMDL NMDL_GetLibVersion.

84 .. 92: Функция проверки наличия модуля заданного типа. Фактически производится запрос количества обнаруженных модулей заданного типа - NMDL_GetBoardCount.

94 .. 122: Функция компиляции исходной модели. Вызывается функция NMDL_COMPILER_CompileONNX. Здесь происходит выделение памяти внутри функции компиляции, поэтому после работы выделенная память освобождается вызовом NMDL_COMPILER_FreeModel, а результат компиляции копируется в возвращаемый вектор float. В целевой программе можно выполнить предварительную компиляцию модели с помощью утилиты *nmdl_compiler_console* так, как описано в разделе "Компиляция модели".

124 .. 144: Функция получения и вывода информации о параметрах входных и выходных тензоров. Используется вызов NMDL_GetModelInfo.

146 .. 158: Функция подготовки кадра. Здесь выполняется чтение изображения из файла, его декодирование и предобработка (NMDL_IMAGE_CONVERTER_Convert). Описание предобработки приводится в разделе "Подготовка изображений".

160 .. 173: Функция ожидания обработки кадра. Принимает номер кластера на котором производится обработка (*batch_num*) и буфер-вектор для хранения результата обработки. Функция блокируется в цикле запроса статуса обработки (NMDL_GetStatus). После получения статуса *NMDL_PROCESS_FRAME_STATUS_FREE* производится копирование результата вызовом NMDL_GetOutput..

178: Задание типа модуля ускорителя. В зависимости от типа ускорителя определяются и типы для работы с библиотекой компиляции моделей и библиотекой подготовки изображений. В примере используется симулятор модуля MC127.05. Для работы с другими типами раскомментируйте соответствующую строку.

189 .. 209: Задаются имена файлов с исходными моделью и изображением. Задаются параметры для подготовки изображения. Для исходной модели требуется подготовить изображение с пикселями в формате blue-green-red. Каждый пиксель модифицируется по формуле Y = X / 1.0 - 114. Это преобразование необходимо для обработки модели *squeezenet*. Для других моделей необходимо использовать соответствующие параметры, которые определяются при создании модели и являются исходными данными, привязанными к конкретной модели. Подробнее о подготовке изображений см. в разделе "Подготовка изображений".

210: Задаётся количество кластеров при пакетной обработке.

211: Задаётся количество обрабатываемых кадров. В примере производится многократная обработка одного кадра.

Далее в главной функции выполняется последовательный вызов описанных вспомогательных функций.

223: Инициализация NMDL. Посредством вызова функции NMDL_Create осуществляется инициализация внутренних структур библиотеки NMDL.

225 .. 266: Пример последовательной обработки кадров в режиме разделения данных по кластерам *"multi unit"* (см. раздел "Режимы обработки").

229 .. 232: компиляция модели.

235 .. 239: инициализация. В функции NMDL_Initialize производится загрузка скомпилированной модели в выбранный ускоритель.

245: формирование входного тензора.

250 .. 259: резервирование буферов для выходных тензоров.

262 .. 265: обработка и получение результата.

268 .. 299: Пример последовательной обработки кадров в режиме пакетной обработки *"batch mode"* (см. раздел "Режимы обработки").

304, 305: При завершении работы освобождаются выделенные ресурсы (NMDL_Release и NMDL_Destroy).

8. Описание идентификаторов, функций и структур NMDL

8.1. Идентификаторы и структуры

8.1.1. NMDL_BOARD_TYPE

Типы модулей.

```
   typedef enum tagNMDL_BOARD_TYPE {
     NMDL_BOARD_TYPE_SIMULATOR,
     NMDL_BOARD_TYPE_MC12101,
     NMDL_BOARD_TYPE_MC12705,
     NMDL_BOARD_TYPE_NMSTICK,
     NMDL_BOARD_TYPE_NMCARD,
     NMDL_BOARD_TYPE_NMMEZZO,
     NMDL_BOARD_TYPE_NMQUAD
} NMDL_BOARD_TYPE;
```

- *NMDL_BOARD_TYPE_SIMULATOR* - симулятор модуля *MC127.05*.
- *NMDL_BOARD_TYPE_MC12101* - модуль *MC121.01*. Модуль на базе СБИС К1879ВМ6Я.
- *NMDL_BOARD_TYPE_MC12705* - модуль *MC127.05*. Представляет собой серверный вычислитель со СБИС К1879ВМ8Я, подключаемый к стандартным портам PCI-E на материнской плате.
- *NMDL_BOARD_TYPE_NMSTICK* - модуль *NMStick*. Представляет собой спецвычислитель со СБИС К1879ВМ6Я в форм-факторе USB Flash drive.
- *NMDL_BOARD_TYPE_NMCARD* - модуль *NMCard*. Представляет собой спецвычислитель со СБИС К1879ВМ8Я, подключаемый в слот расширения PCIe на материнской плате компьютера.
- *NMDL_BOARD_TYPE_NMMEZZO* - модуль *NMMezzo*. Представляет собой спецвычислитель со СБИС К1879ВМ8Я, подключаемый по шине PCIe к несущей плате пользователя.
- *NMDL_BOARD_TYPE_NMQUAD* - модуль *NMQuad*. Представляет собой спецвычислитель со СБИС К1879ВМ8Я, подключаемый в слот расширения PCIe на материнской плате компьютера.

8.1.2. NMDL_ModelInfo

Структура с информацией о модели.

```
   typedef struct tagNMDL_ModelInfo {
     unsigned int input_tensor_num;
     NMDL_Tensor input_tensors[NMDL_MAX_INPUT_TENSORS];
```

```
    unsigned int output_tensor_num;
    NMDL_Tensor output_tensors[NMDL_MAX_OUTPUT_TENSORS];
} NMDL_ModelInfo;
```

- *input_tensor_num* - количество входных тензоров,
- *input_tensors* – входные тензоры.
- *output_tensor_num* - количество выходных тензоров,
- *output_tensors* – выходные тензоры.

NMDL_MAX_INPUT_TENSORS - максимальное количество входных тензоров.

NMDL_MAX_OUTPUT_TENSORS - максимальное количество выходных тензоров.

8.1.3. NMDL_PROCESS_FRAME_STATUS

Идентификатор статуса обработки кадра.

```
  typedef enum tagNMDL_PROCESS_FRAME_STATUS {
    NMDL_PROCESS_FRAME_STATUS_FREE,
    NMDL_PROCESS_FRAME_STATUS_INCOMPLETE
} NMDL_PROCESS_FRAME_STATUS;
```

- *NMDL_PROCESS_FRAME_STATUS_FREE* - обработка кадра не производится,
- *NMDL_PROCESS_FRAME_STATUS_INCOMPLETE* - выполняется обработка кадра.

8.1.4. NMDL_RESULT

Возвращаемый результат.

```
  typedef enum tagNMDL_RESULT {
    NMDL_RESULT_OK,
    NMDL_RESULT_INVALID_FUNC_PARAMETER,
    NMDL_RESULT_NO_BOARD,
    NMDL_RESULT_BOARD_RESET_ERROR,
    NMDL_RESULT_INIT_CODE_LOADING_ERROR,
    NMDL_RESULT_CORE_HANDLE_RETRIEVAL_ERROR,
    NMDL_RESULT_FILE_LOADING_ERROR,
    NMDL_RESULT_MEMORY_WRITE_ERROR,
    NMDL_RESULT_MEMORY_READ_ERROR,
    NMDL_RESULT_MEMORY_ALLOCATION_ERROR,
    NMDL_RESULT_MODEL_LOADING_ERROR,
    NMDL_RESULT_INVALID_MODEL,
    NMDL_RESULT_BOARD_SYNC_ERROR,
    NMDL_RESULT_BOARD_MEMORY_ALLOCATION_ERROR,
    NMDL_RESULT_NN_CREATION_ERROR,
    NMDL_RESULT_NN_LOADING_ERROR,
    NMDL_RESULT_NN_INFO_RETRIEVAL_ERROR,
    NMDL_RESULT_MODEL_IS_TOO_BIG,
    NMDL_RESULT_NOT_INITIALIZED,
    NMDL_RESULT_INCOMPLETE,
    NMDL_RESULT_UNKNOWN_ERROR
} NMDL_RESULT;
```

- *NMDL_RESULT_OK* - нет ошибок,

- *NMDL_RESULT_INVALID_FUNC_PARAMETER* - неверный параметр,
- *NMDL_RESULT_NO_BOARD* – нет модуля,
- *NMDL_RESULT_BOARD_RESET_ERROR* – ошибка сброса модуля,
- *NMDL_RESULT_INIT_CODE_LOADING_ERROR* – ошибка загрузки кода инициализации библиотеки загрузки и обмена,
- *NMDL_RESULT_CORE_HANDLE_RETRIEVAL_ERROR* – ошибка получения идентификатора вычислителя,
- *NMDL_RESULT_FILE_LOADING_ERROR* – ошибка загрузки программного образа,
- *NMDL_RESULT_MEMORY_WRITE_ERROR* – ошибка записи в память модуля,
- *NMDL_RESULT_MEMORY_READ_ERROR* – ошибка чтения из памяти модуля,
- *NMDL_RESULT_MEMORY_ALLOCATION_ERROR* – ошибка выделения памяти,
- *NMDL_RESULT_MODEL_LOADING_ERROR* – ошибка загрузки модели нейронной сети,
- *NMDL_RESULT_INVALID_MODEL* – ошибка в модели,
- *NMDL_RESULT_BOARD_SYNC_ERROR* – ошибка синхронизации с модулем,
- *NMDL_RESULT_BOARD_MEMORY_ALLOCATION_ERROR* – ошибка выделения памяти на модуле,
- *NMDL_RESULT_NN_CREATION_ERROR* – ошибка создания модели на модуле,
- *NMDL_RESULT_NN_LOADING_ERROR* – ошибка загрузки модели нейронной сети,
- *NMDL_RESULT_NN_INFO_RETRIEVAL_ERROR* – ошибка запроса информации о модели,
- *NMDL_RESULT_MODEL_IS_TOO_BIG* – модель не может быть размещена в памяти модуля,
- *NMDL_RESULT_NOT_INITIALIZED* – библиотека функций NMDL не инициализирована,
- *NMDL_RESULT_INCOMPLETE* – устройство находится в состоянии обработки кадра,
- *NMDL_RESULT_UNKNOWN_ERROR* – неизвестная ошибка.

8.1.5. NMDL_Tensor

Структура, описывающая тензор.

```
   typedef struct tagNMDL_Tensor {
     unsigned int width;
     unsigned int height;
     unsigned int depth;
} NMDL_Tensor;
```

- *width* - ширина тензора,
- *height* - высота тензора,
- *depth* – глубина тензора.

8.2. Функции

8.2.1. NMDL_Blink

Светодиодная индикация для идентификации модуля.

```
   NMDL_RESULT NMDL_Blink(
     unsigned int board_type,
     unsigned int board_number
);
```

- *board_type* - [in] тип модуля, на котором вызвается процедура светодиодной индикации. Одно из значений перечисления NMDL_BOARD_TYPE.
- *board_number* - [in] порядковый номер модуля, на котором вызвается процедура светодиодной индикации.

8.2.2. NMDL_Create

Создание экземпляра NMDL и получение идентификатора экземпляра *NMDL*.

```
   NMDL_RESULT NMDL_Create(
     NMDL_HANDLE *nmdl
);
```

- *nmdl* - [out] идентификатор экземпляра *NMDL*.

После работы с экземпляром NMDL необходимо освободить выделенные ресурсы вызовом NMDL_Destroy.

8.2.3. NMDL_Destroy

Удаление экземпляра NMDL.

```
   void NMDL_Destroy(
     NMDL_HANDLE nmdl
);
```

- *nmdl* - [in] идентификатор экземпляра *NMDL*.

Функция вызывается для освобождения ресурсов, выделенных при вызовах NMDL_Create иNMDL_Initialize .

8.2.4. NMDL_GetBoardCount

Запрос количества обнаруженных модулей заданного типа.

```
   NMDL_RESULT NMDL_GetBoardCount(
     unsigned int board_type,
     unsigned int *boards
);
```

- *board_type* - [in] тип опрашиваемых модулей. Одно из значений перечисления NMDL_BOARD_TYPE.
- *boards* - [out] количество обнаруженных модулей.

Для симулятора MC127.05 (тип *NMDL_BOARD_TYPE_SIMULATOR*) всегда обнаруживается один модуль.

8.2.5. NMDL_GetLibVersion

Запрос версии библиотеки *NMDL*.

```
   NMDL_RESULT NMDL_GetLibVersion(
     unsigned int *major,
     unsigned int *minor,
     unsigned int *patch
);
```

- *major* - [out] старший номер версии.
- *minor* - [out] младший номер версии.
- *patch* - [out] номер патча.

8.2.6. NMDL_GetModelInfo

Запрос информации о модели.

```
   NMDL_RESULT NMDL_GetModelInfo(
     NMDL_HANDLE nmdl,
     unsigned int unit_num,
     NMDL_ModelInfo *model_info
);
```

- *nmdl* - [in] дескриптор *NMDL*.
- *unit_num* - [in] номер юнита для которого запрашивается информация. Имеет значение только при работе с модулями со *СБИС К1879ВМ8Я* (*MC127.05*, *NMCard*, *NMMezzo*, *NMQuad* или симулятор). При работе с *MC121.01* и *NMStick* необходимо установить 0.
- *model_info* - [out] информация о модели.

8.2.7. NMDL_GetOutput

Запрос результата обработки.

```
   NMDL_RESULT NMDL_GetOutput(
     NMDL_HANDLE nmdl,
     unsigned int unit_num,
     float *outputs[],
     double *fps
);
```

- *nmdl* - [in] идентификатор экземпляра *NMDL*.
- *unit_num* - [in] номер юнита для которого запрашивается результат обработки. Имеет значение только при работе с модулями *MC127.05*, *NMCard* или симулятором. При работе с *MC121.01* и *NMStick* необходимо установить 0.
- *outputs* - [out] массив указателей на буферы для выходных тензоров. Выходные тензоры нумеруются друг за другом в порядке их появления в графе обработки, то есть упорядоченные по уровням в графе. Память под буферы выделяется пользователем. Размер каждого тензора вычисляется как произведение его ширины, высоты и глубины (каналы). Количество и параметры тензоров определяются в структуре NMDL_ModelInfo, которую можно получить вызовом NMDL_GetModelInfo.
- *fps* - [out] производительность обработки (кадров в секунду). Может принимать значение 0.

Пример вызова *NMDL_GetOutput* с выделением памяти под результат на C++.

```
   ...
NMDL_ModelInfo model_info;
NMDL_GetModelInfo(nmdl_handle, unit_num, &model_info);
std::vector<std::vector<float>> output_tensors(model_info.output_tensor_num);
std::vector<float*> outputs(model_info.output_tensor_num);
for(std::size_t i = 0; i < model_info.output_tensor_num; ++i) {
    output_tensors[i].resize(static_cast<std::size_t>(
            model_info.output_tensors[i].width) *
            model_info.output_tensors[i].height *
            model_info.output_tensors[i].depth);
    outputs[i] = output_tensors[i].data();
  }
double fps;
NMDL_GetOutput(nmdl_handle, unit_num, outputs.data(), &fps);
...
```

8.2.8. NMDL_GetStatus

Запрос статуса обработки. Функция вызывается для проверки окончания обработки кадра.

```
   NMDL_RESULT NMDL_GetStatus(
     NMDL_HANDLE nmdl,
     unsigned int unit_num,
     unsigned int *status
);
```

- *nmdl* - [in] идентификатор экземпляра *NMDL*
- *unit_num* - [in] номер кластера для которого запрашивается статус обработки. Имеет значение только при работе с модулями *MC127.05*, *NMCard* или симулятором, которые имеют в своём составе несколько юнитов. При работе с *MC121.01* и *NMStick* необходимо установить 0.
- *status* - [out] состояное кластера в момент вызова функции. Одно из значений перечисления NMDL_PROCESS_FRAME_STATUS.

Пример использования *NMDL_GetStatus* для организации поллинга. Блокирующая функция для ожидания окончания обработки:

```
   auto WaitForOutput(NMDL_HANDLE nmdl, std::uint32_t unit_num, float *output[]) {
     std::uint32_t status = NMDL_PROCESS_FRAME_STATUS_INCOMPLETE;
     while(status == NMDL_PROCESS_FRAME_STATUS_INCOMPLETE) {
         NMDL_GetStatus(nmdl, unit_num, &status);
     };
     double fps;
     NMDL_GetOutput(nmdl, unit_num, output, &fps);
     return fps;
}
```

8.2.9. NMDL_Initialize

Инициализация NMDL.

```
   NMDL_RESULT NMDL_Initialize(
     NMDL_HANDLE nmdl,
     unsigned int board_type,
     unsigned int board_number,
     unsigned int proc_number,
     const float *model[NMDL_MAX_UNITS],
     const unsigned int model_floats[NMDL_MAX_UNITS]
);
```

- *nmdl* - [in] идентификатор экземпляра *NMDL*.
- *board_type* - [in] тип инициализируемого модуля. Одно из значений перечисления NMDL_BOARD_TYPE.
- *board_number* - [in] порядковый номер инициализируемого модуля.
- *proc_number* - [in] порядковый номер инициализируемого процессора. Для всех однопроцессорных модулей и симулятора необходимо установить 0.
- *model* - [in] массив указателей на буферы, содержащие образы скомпилированных моделей. или *nm8* (для *MC127.05*, *NMCard* и симулятора).
- *model_floats* - [in] размер модели в вещественных числах с одинарной точностью.
- *use_batch_mode* - [in] флаг использования режима пакетной обработки. Имеет значение только для модулей *MC127.05*, *NMCard* или симулятора. При работе с *MC121.01* и *NMStick* необходимо установить 0.

Для устройств на базе процессора К1879ВМ6Я таких, как *MC121.01* и *NMStick*, имеется один юнит для прогона нейронной сети, поэтому здесь необходимо задать только одну модель. Например:

```
   // NMDL_HANDLE nmdl - library descriptor created in NMDL_Create.
// std::vector<float> model - compiled model data.
std::array<const float*, NMDL_MAX_UNITS> models = {
 model.data()
};
std::array<std::uint32_t, NMDL_MAX_UNITS> model_floats = {
 static_cast<std::uint32_t>(model.size())
};
NMDL_Initialize(nmdl, NMDL_BOARD_TYPE_MC12101, 0, models.data(), model_floats.data());
```

Для устройств на базе процессора *К1879ВМ8Я - MC127.05*, *NMCard*, *NMMezzo*, *NMQuad* и симуляторе - можно задать несколько моделей, так как здесь можно использовать до четырёх юнитов. Подробнее о режимах обработки см. раздел Режимы обработки. Например:

```
   // NMDL_HANDLE nmdl - library descriptor created in NMDL_Create.
// std::vector<float> model_0 - unit 0 compiled model data.
// std::vector<float> model_1 - unit 1 compiled model data.
// std::vector<float> model_2 - unit 2 compiled model data.
// std::vector<float> model_3 - unit 3 compiled model data.
std::array<const float*, NMDL_MAX_UNITS> models = {
 model_0.data(), model_1.data(), model_2.data(), model_3.data()
};
std::array<std::uint32_t, NMDL_MAX_UNITS> model_floats = {
 static_cast<std::uint32_t>(model_0.size()),
 static_cast<std::uint32_t>(model_1.size()),
 static_cast<std::uint32_t>(model_2.size()),
 static_cast<std::uint32_t>(model_3.size())
};
NMDL_Initialize(nmdl, NMDL_BOARD_TYPE_NMCARD, 0, models.data(), model_floats.data());
```

Если модель скомпилирована для устройств на базе процессора *К1879ВМ8Я* для прогона в режиме *"multi unit"*, то нужно задать только одну модель - эта модель содержит в себе данные для инициализации всех четырёх юнитов. Подробнее о режимах обработки см. раздел Режимы обработки. Например:

```
   // NMDL_HANDLE nmdl - library descriptor created in NMDL_Create.
// std::vector<float> model - compiled model data  to run in "multi unit" mode.
std::array<const float*, NMDL_MAX_UNITS> models = {
 model.data()
};
std::array<std::uint32_t, NMDL_MAX_UNITS> model_floats = {
 static_cast<std::uint32_t>(model.size())
};
NMDL_Initialize(nmdl, NMDL_BOARD_TYPE_NMCARD, 0, models.data(), model_floats.data());
```

После работы с функциями NMDL необходимо освободить выделенные ресурсы вызовом функции деинициализации NMDL_Release.

8.2.10. NMDL_Process

Обработка входных тензоров.

```
   NMDL_RESULT NMDL_Process(
     NMDL_HANDLE nmdl,
     unsigned int unit_num,
```

```
    const float *frame[]
);
```

- *nmdl* - [in] идентификатор экземпляра *NMDL*
- *unit_num* - [in] номер юнита на котором запускается обработка. Имеет значение только при работе с модулями *MC127.05*, *NMCard*, *NMMezzo*, *NMQuad* или симулятором в режиме независимой обаработки на юнитах. При работе с *MC121.01* и *NMStick*, или в режиме обработки *"multi unit"* необходимо установить 0.
- *frame* - [in] массив, содержащий указатели на буферы со входными тензорами.

Количество и геометрия входных тензоров должно соответствовать загруженной модели нейронной сети. Пример вызова:

```
  // NMDL_HANDLE nmdl - library descriptor created in NMDL_Create.
// std::vector<float> input_tensor_0 - first input tensor data.
// std::vector<float> input_tensor_1 - second input tensor data.
std::array<const float*, 2> input_tensors = {
 input_tensor_0.data(), input_tensor_1.data()
};
NMDL_Process(nmdl, 0, input_tensors.data());
```

8.2.11. NMDL_Release

Деинициализация NMDL.

```
  void NMDL_Release(
    NMDL_HANDLE nmdl
);
```

- *nmdl* - [in] идентификатор экземпляра *NMDL*

9. Описание идентификаторов и функций nmdl_compiler

9.1. Идентификаторы

9.1.1. NMDL_COMPILER_BOARD_TYPE

Типы модулей.

```
   typedef enum tagNMDL_COMPILER_BOARD_TYPE {
     NMDL_COMPILER_BOARD_TYPE_MC12101,
     NMDL_COMPILER_BOARD_TYPE_MC12705
} NMDL_COMPILER_BOARD_TYPE;
```

- *NMDL_COMPILER_BOARD_TYPE_MC12101* - модули на базе *СБИС К1879ВМ6Я* - *MC121.01* и *NMStick*.
- *NMDL_COMPILER_BOARD_TYPE_MC12705* - модули на базе *СБИС К1879ВМ8Я* - *MC127.05*, *NMCard*, *NMMezzo*, *NMQuad* и симулятор.

9.1.2. NMDL_COMPILER_RESULT

Возвращаемый результат.

```
   typedef enum tagNMDL_COMPILER_RESULT {
     NMDL_COMPILER_RESULT_OK,
     NMDL_COMPILER_RESULT_MEMORY_ALLOCATION_ERROR,
     NMDL_COMPILER_RESULT_MODEL_LOADING_ERROR,
     NMDL_COMPILER_RESULT_INVALID_PARAMETER,
     NMDL_COMPILER_RESULT_INVALID_MODEL,
     NMDL_COMPILER_RESULT_UNSUPPORTED_OPERATION
} NMDL_COMPILER_RESULT;
```

- *NMDL_COMPILER_RESULT_OK* - нет ошибок,
- *NMDL_COMPILER_RESULT_MEMORY_ALLOCATION_ERROR* – ошибка выделения памяти,
- *NMDL_COMPILER_RESULT_MODEL_LOADING_ERROR* – ошибка загрузки модели нейронной сети,
- *NMDL_COMPILER_RESULT_INVALID_PARAMETER* - неверный параметр,
- *NMDL_COMPILER_RESULT_INVALID_MODEL* – ошибка в модели,
- *NMDL_COMPILER_RESULT_UNSUPPORTED_OPERATION* – модель содержит неподдерживаемую операцию.

9.2. Функции

9.2.1. NMDL_COMPILER_CompileDarkNet

Компиляция исходной модели в формате DarkNet.

```
   NMDL_COMPILER_RESULT NMDL_COMPILER_CompileDarkNet(
     unsigned int is_multi_unit,
     unsigned int board,
     const char* src_model,
     unsigned int src_model_size,
     const char* src_weights,
     unsigned int src_weights_size,
     float** dst_model,
     unsigned int* dst_model_floats
);
```

- *is_multi_unit* - [in] флаг использования режима обработки *"multi unit"* (см. раздел Режимы обработки). 0 - не используется, 1 - используется. Параметр актуален только для модулей типа board = NMDL_COMPILER_BOARD_TYPE_MC12705. Для board = NMDL_COMPILER_BOARD_TYPE_MC12101 параметр игнорируется и может принимать любое значение.

- *board* - [in] идентификатор типа модуля для которого производится компиляция модели. Одно из значений перечисления NMDL_COMPILER_BOARD_TYPE.

- *src_model* - [in] буфер исходной модели в формате *DarkNet*, предварительно считанной из файла *.cfg*.

- *src_model_size* - [in] размер буфера исходной модели в байтах.

- *src_weights* - [in] буфер коэффициентов, предварительно считанный из файла *.weights*.

- *src_weights_size* - [in] размер буфера коэффициентов в байтах.

- *dst_model* - [out] буфер компилированной модели. Выделение памяти происходит в функции.

- *dst_model_floats* - [out] размер буфера компилированной модели в *float32*.

9.2.2. NMDL_COMPILER_CompileONNX

Компиляция исходной модели в формате ONNX.

```
   NMDL_COMPILER_RESULT NMDL_COMPILER_CompileONNX(
     unsigned int is_multi_unit,
     unsigned int board,
     const char* src_model,
     unsigned int src_model_size,
     float** dst_model,
     unsigned int* dst_model_floats
);
```

- *is_multi_unit* - [in] флаг использования режима обработки *"multi unit"* (см. раздел Режимы обработки). 0 - не используется, 1 - используется. Параметр актуален только для модулей типа board = NMDL_COMPILER_BOARD_TYPE_MC12705. Для board = NMDL_COMPILER_BOARD_TYPE_MC12101 параметр игнорируется и может принимать любое значение.

- *board* - [in] идентификатор типа модуля для которого производится компиляция модели. Одно из значений перечисления NMDL_COMPILER_BOARD_TYPE.

- *src_model* - [in] буфер исходной модели в формате *ONNX*, предварительно считанной из файла *.onnx*.

- *src_model_size* - [in] размер буфера исходной модели в байтах.

- *dst_model* - [out] буфер компилированной модели. Выделение памяти происходит в функции.

- *dst_model_floats* - [out] размер буфера компилированной модели в *float32*.

9.2.3. NMDL_COMPILER_FreeModel

Освобождение выделенной при компиляции памяти.

```
   NMDL_COMPILER_RESULT NMDL_COMPILER_FreeModel(
     unsinged int board,
     char* dst_model
);
```

- *board* - [in] идентификатор типа модуля для которого производедена компиляция модели. Одно из значений перечисления NMDL_COMPILER_BOARD_TYPE.

- *dst_model* - [in] буфер освобождаемой памяти. Память для буфера была выделена при вызове NMDL_COMPILER_CompileDarkNet или NMDL_COMPILER_CompileONNX.

9.2.4. NMDL_COMPILER_GetLastError

Возвращает константную строку с описанием последней ошибки.

```
   const char *NMDL_COMPILER_GetLastError();
```

10. Описание идентификаторов и функций nmdl_image_converter

10.1. Идентификаторы и структуры

10.1.1. NMDL_IMAGE_CONVERTER_BOARD_TYPE

Типы модулей.

```
   typedef enum tagNMDL_IMAGE_CONVERTER_BOARD_TYPE {
     NMDL_IMAGE_CONVERTER_BOARD_TYPE_MC12101,
     NMDL_IMAGE_CONVERTER_BOARD_TYPE_MC12705
} NMDL_IMAGE_CONVERTER_BOARD_TYPE;
```

- *NMDL_IMAGE_CONVERTER_BOARD_TYPE_MC12101* - модули *MC121.01* и *NMStick*.
- *NMDL_IMAGE_CONVERTER_BOARD_TYPE_MC12705* - модули *MC127.05*, *NMCard* и симулятор.

10.1.2. NMDL_IMAGE_CONVERTER_COLOR_FORMAT

Идентификаторы формата пикселя. Описывает порядок следования цветовых компонент RGB в пикселе.

```
   typedef enum tagNMDL_IMAGE_CONVERTER_COLOR_FORMAT {
     NMDL_IMAGE_CONVERTER_COLOR_FORMAT_RGB,
     NMDL_IMAGE_CONVERTER_COLOR_FORMAT_RBG,
     NMDL_IMAGE_CONVERTER_COLOR_FORMAT_GRB,
     NMDL_IMAGE_CONVERTER_COLOR_FORMAT_GBR,
     NMDL_IMAGE_CONVERTER_COLOR_FORMAT_BRG,
     NMDL_IMAGE_CONVERTER_COLOR_FORMAT_BGR,
     NMDL_IMAGE_CONVERTER_COLOR_FORMAT_INTENSITY,
} NMDL_IMAGE_CONVERTER_COLOR_FORMAT;
```

10.2. Функции

10.2.1. NMDL_IMAGE_CONVERTER_Convert

Подготовка изображения.

```
   int NMDL_IMAGE_CONVERTER_Convert(
     const char* src,
     float* dst,
     unsigned int src_size,
     unsigned int dst_width,
     unsigned int dst_height,
     unsigned int dst_color_format,
     const float rgb_divider[3],
     const float rgb_adder[3],
     unsigned int board_type
```

```
);
```

- *src* - [in] буфер с образом исходного изображения. Содержимое буфера соответствует содержимому файла изображения. Изображения могут быть представлены в форматах .bmp, .gif, .jpg и .png.
- *dst* - [out] буфер для подготовленного изображения. Память для буфера должна быть предварительно выделена. Размер буфера должен быть не менше значения, возвращаемого функцией NMDL_IMAGE_CONVERTER_RequiredSize.
- *src_size* - [in] размер буфера исходного изображения в байтах.
- *dst_width* - [in] ширина подготовленногоизображения.
- *dst_height* - [in] высота подготовленногоизображения.
- *dst_color_format* - [in] идентификатор формата пикселя подготовленного изображения. Одно из значений перечисления NMDL_IMAGE_CONVERTER_COLOR_FORMAT.
- *rgb_divider* - [in] делитель в выражении dst = src / divider+ adder. Приведённая операция выполняется над каналами каждого пикселя исходного изображения. rgb_divider[0] - для красного канала, rgb_divider[1] - для зелёного канала, rgb_divider[2] - для голубого канала.
- *rgb_adder* - [in] слагаемое в выражении dst = src / divider+ adder. Приведённая операция выполняется над каналами каждого пикселя исходного изображения. rgb_adder[0] - для красного канала, rgb_adder[1] - для зелёного канала, rgb_adder[2] - для голубого канала.
- *board_type* - [in] идентификатор типа вычислительного модуля на котором предполагается обработка. Одно из значений перечисления NMDL_IMAGE_CONVERTER_BOARD_TYPE.

Для изображений с градацией серого, когда поле dst_color_format = NMDL_IMAGE_CONVERTER_COLOR_FORMAT_INTENSITY, используются только делитель rgb_divider[0] и слагаемое rgb_adder[0]. Остальные делители (rgb_divider[1] и rgb_divider[2]) и слагаемые (rgb_adder[1] и rgb_adder[2]) игнорируются и могут быть установлены в любые значения.

Возвращаемое значение: 0 - нормальное завершение, -1 - ошибка.

10.2.2. NMDL_IMAGE_CONVERTER_RequiredSize

Возвращает размер буфера в элементах *float32* для хранения подготовленного изображения.

```
int NMDL_IMAGE_CONVERTER_RequiredSize(
  unsigned int dst_width,
  unsigned int dst_height,
  unsigned int dst_color_format,
  unsigned int board_type
```

```
);
```

- *dst_width* - [in] ширина подготовленногоизображения.
- *dst_height* - [in] высота подготовленногоизображения.
- *dst_color_format* - [in] идентификатор формата пикселя подготовленного изображения. Одно из значений перечисления NMDL_IMAGE_CONVERTER_COLOR_FORMAT.
- *board_type* - [in] идентификатор типа вычислительного модуля на котором предполагается обработка. Одно из значений перечисления NMDL_IMAGE_CONVERTER_BOARD_TYPE.

НАУЧНО-ТЕХНИЧЕСКИЙ ЦЕНТР

АО НТЦ "Модуль"
А/Я 166, Москва, 125190, Россия
Тел: +7 (499) 152-9698
Факс: +7 (499) 152-4661
E-Mail: rusales@module.ru
WWW: http://www.module.ru

Напечатано в России. Дата публикации: 28/03/2023