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. 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**	**MC127.05 и NMCard batch-mode***
alexnet (227x227)	3,45	3,2	12,6	13
inception v3 (299x299)	0,63	0,6	8,12	12,43
inception v3 (512x512)	0,24	0,23	3,93	5,44
resnet 18 (224x224)	2,28	2,2	25	47
squeezenet (224x224)	8,3	8	74,4	100
yolo v2 tiny (416x416)	1,16	1,1	21	30,4
yolo v3 (416x416)	0,1	0,09	3,7	4
yolo v3 tiny (416x416)	1,44	1,38	25,3	33,3

Таблица 1.2 - Latency (ms)

	MC121.01	**NMStick**	**MC127.05 и NMCard**	**MC127.05 и NMCard batch-mode***
alexnet (227x227)	290	302	79	308
inception v3 (299x299)	1587	1653	123	322
inception v3 (512x512)	4166	4340	254	735
resnet 18 (224x224)	439	457	40	85
squeezenet (224x224)	120	125	13	40
yolo v2 tiny (416x416)	862	898	47	132
yolo v3 (416x416)	10000	10416	270	1000
yolo v3 tiny (416x416)	694	725	40	120

* Описание *batch-mode* приводится в разделе "Режимы обработки"

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.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 *.pb*. 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 установкии.

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.zip* - архив для демонстрации работы сети *ALEXNET*.
- *nmdl_ref_data_resnet.zip* - архив для демонстрации работы сети *RESNET18*.
- *nmdl_ref_data_squeezenet.zip* - архив для демонстрации работы сети *SQUEEZENET*.

- *nmdl_ref_yolo2_tiny.zip* - архив для демонстрации работы сети *YOLONET V2 TINY*.

В каталогах содержатся файлы:

- *frame.bmp* - тестовое изображение.
- *model.cfg* - модель в формате *DARKNET*.
- *model.pb* - 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*.

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.

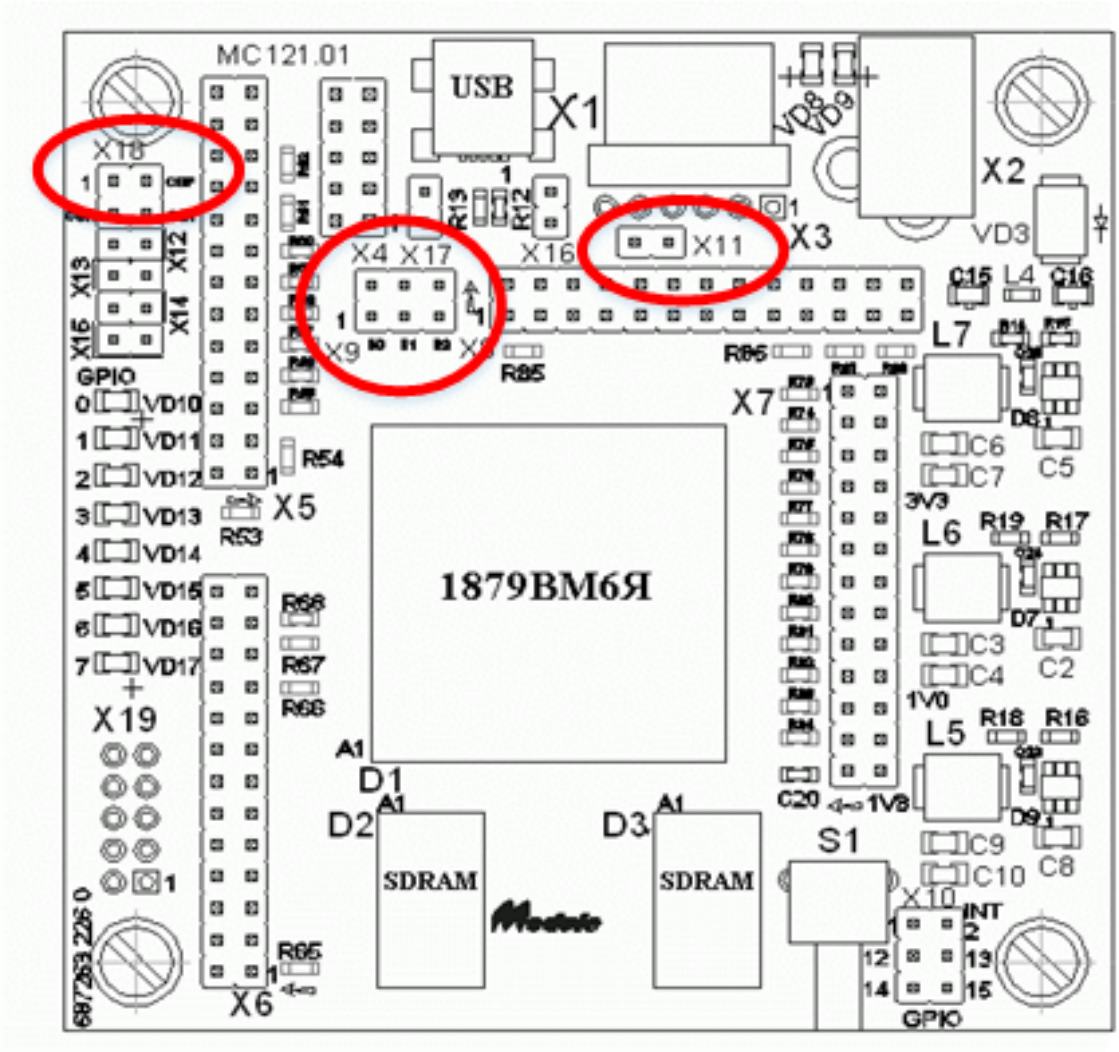

Рисунок 3.1 - Перемычки на модуле MC121.01

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 и .

3.6. Подготовка модулей MC127.05 и NMCard

Для работы с NMDL необходимо установить плату в свободный слот PCIe 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 *.pb или *.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.

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 [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.
- *WEIGHTS_FILENAME* - имя файла с весовыми коэффициентами модели. Нужен только для моделей в формате DarkNet.

Пример:

```
>nmdl_compiler_console MC12705 ONNX
      squeezenet.pb squeezenet.nm8
```

```
>nmdl_compiler_console MC12705 DARKNET
        yolo2t.pb yolo2t.nm8 yolo2t.weights
```

4.1. Поддерживаемые операции

Компилятором поддерживается следующий набор операций:

1. 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
2. MaxPool
    - MaxPool2x2, no pad, stride=1
    - MaxPool2x2, no pad, stride=2
    - MaxPool3x3, no pad, stride=2
    - MaxPool3x3, pad, stride=2
3. AveragePool
    - AveragePool2x2, no pad, stride=1

- AveragePool2x2, no pad, stride=2
- AveragePool3x3, no pad, stride=2
- AveragePool3x3, pad, stride=2

4. GlobalAveragePool
5. Concat (входные тензоры должны иметь чётное количество каналов, количество входных тензоров - не более 7)
6. Add
7. Relu
8. Leaky Relu
9. PRelu
10. BatchNormalization при условии, что операция выполняется сразу после свёртки.
11. Pad
12. GEMM
13. Mat Mul
14. Batch Normalization
15. Sub
16. Mul
17. Div

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 D A B
```

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),
- *D* - делитель для каждого канала пикселя в выражении $dst = src / D + A$ (вещественная величина),
- *A* - слагаемое для каждого канала пикселя в выражении $dst = src / D + A$ (вещественная величина).
- *B* - тип модуля на котором предполагается обработка (допустимые значения: mc12101, mc12705),

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. Режимы обработки

Архитектурные особенности используемых в вычислительных модулях СБИС позволяют реализовать нейросетевые алгоритмы по разному. Разнообразие определяется аппаратными средствами для выполнения параллельных вычислений. В реализации *NMDL* использовалась глубокая оптимизация алгоритмов и выбирались наиболее быстрые способы параллелизма, инкапсулируюя детали разработки. Тем не менее, в реализации *NMDL* для модуля MC127.05 остаётся вариативность решений, которую целесообразно вынести в интерфейс и предоставить пользователю возможность выбрать тот, или иной способ обработки в зависимости от решаемой задачи.

Используемая в модулях *MC127.05* и *NMCard* СБИС К1879ВМ8Я имеет кластерную организацию.

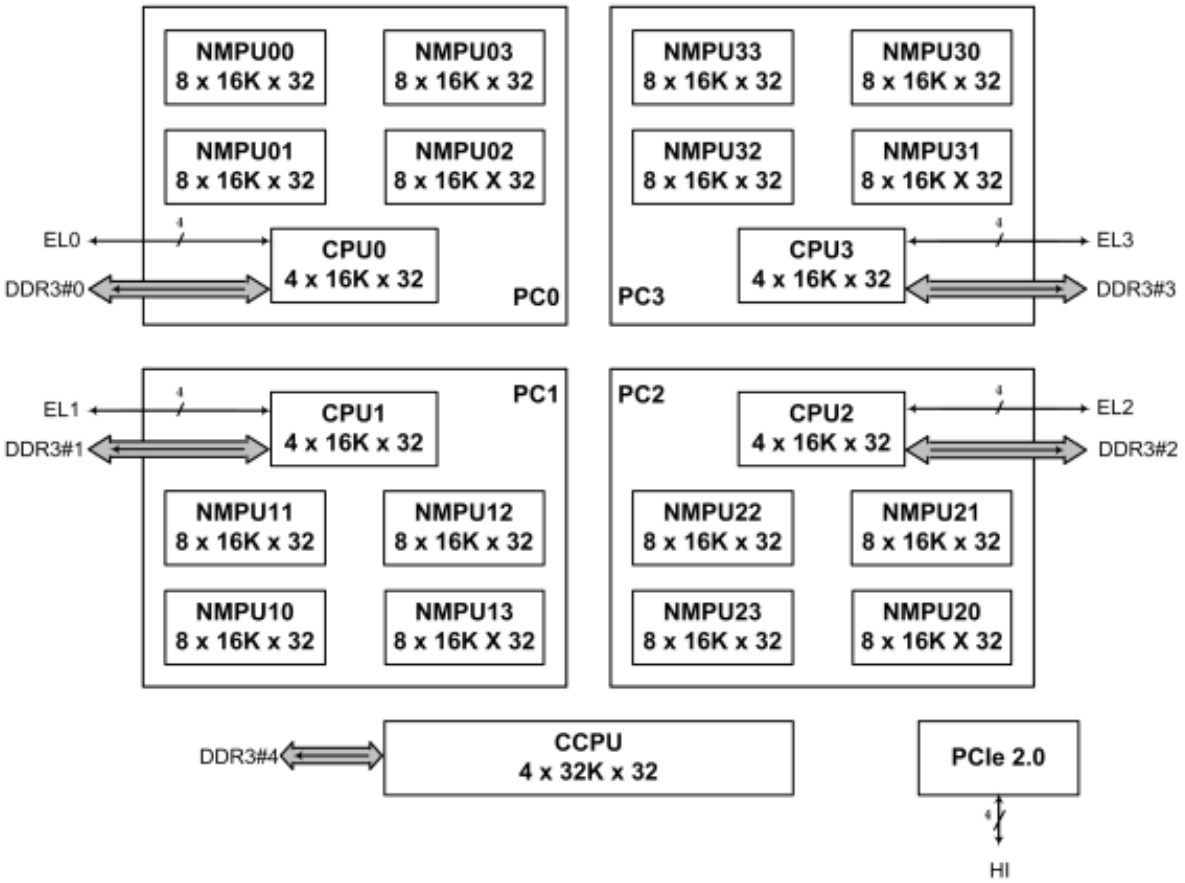

Рисунок 6.1 - Кластеры СБИС К1879ВМ8Я.

На рисунке 6.1:

- *PC* - процессорные кластеры;
- *NMPU* - процессорные узлы NMC4;
- *CPU* - кластерные процессорные ядра ARM Cortex-A5;
- *CCPU* - центральное процессорное ядро ARM Cortex-A5.

Внутри кластеров объём вычислений и данных распределяются на четыре процессорных узла NMC4. Здесь используются тесные аппаратные связи при межпросессорном взаимодействии внутри кластеров и выполнена глубокая оптимизация над функциями-примитивами.

Обработка на кластерах же может быть организована двумя способами: с разделением данных между кластерами и работа в пакетном режиме (*batch-mode*).

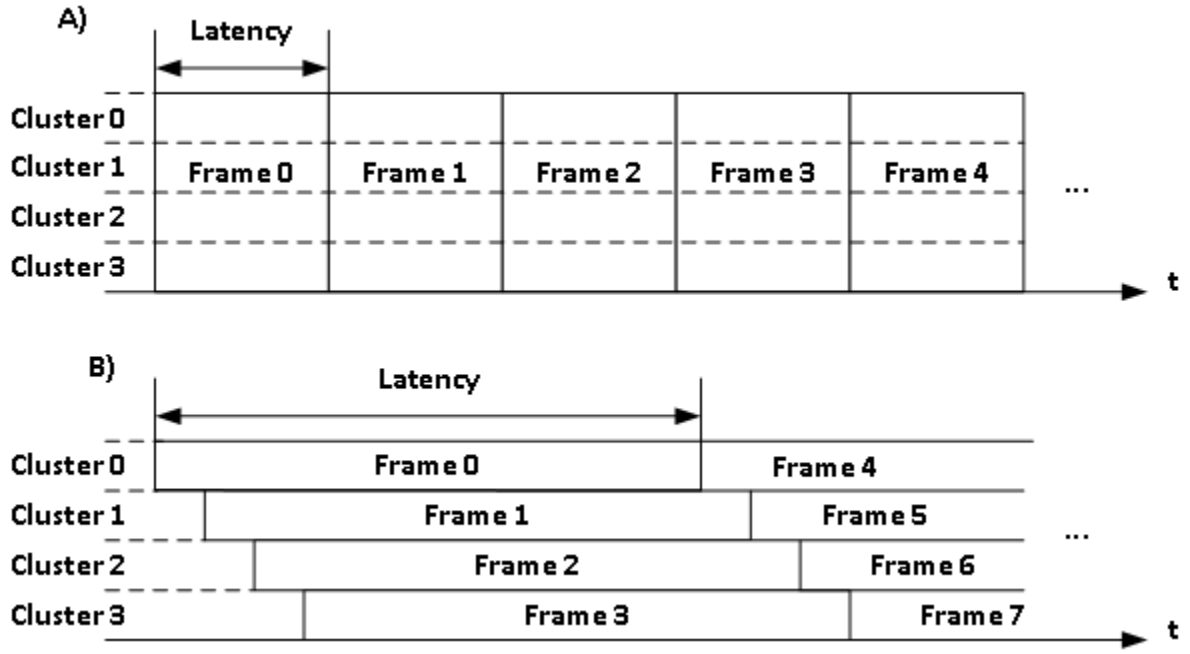

Рисунок 6.2 - Режимы обработки

При обработке с разделением данных (рисунок 6.2 A)) можно добиться минимальной латентности - времени от начала обработки до момента получения результата. Однако в этом случае неизбежны накладные расходы по организации параллелизма. Например, при выполнении свёрток над разделёнными тензорами необходимо компенсировать обработку границ, выполнять переупаковки, транзит данных между кластерами и т. п. Производительность - кадры в секунду - *FPS = 1/Latency*.

При пакетной обработке (рисунок 6.2 В)) на каждом кластере обрабатывается один кадр. Кластеры выполняют одинаковую и независимую обработку. В этом случае накладные расходы по организации распределения данных отсутствует и достигается максимальная производительность, при этом увеличивается латентность. Производительность - кадры в секунду - *FPS = 4/Latency*.

Управление режимами обработки возможно только при работе с модулями *MC127.05*, *NMCard* или с симулятором.

7. 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 7.1.

Рисунок 7.1 - Внешний вид программы в процессе работы

Выбор модуля осуществляется в диалоговом окне *Open Board*, внешний вид которого приведён на рисунке 7.2. Окно содержит список доступных в системе ускорителей, кнопки выбора, а также возможность идентификации устройства посредством светодиодной индикации.

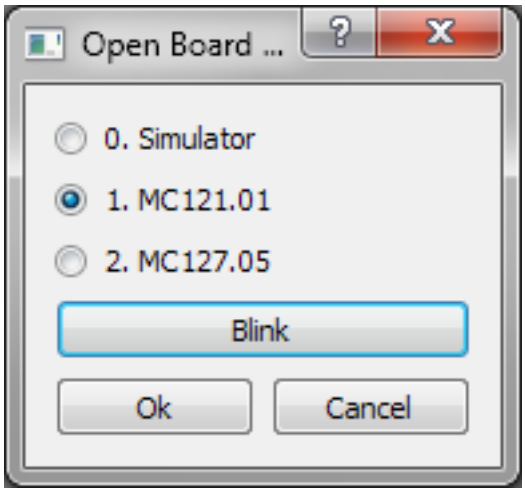

Рисунок 7.2 - Внешний вид окна выбора модуля

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. • **.pb* 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.

Параметр	Описание
	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*. • *yolo2* - нейронная сеть семейства YOLO V2. • *yolo3* - нейронная сеть семейства YOLO V3. 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.

Параметр	**Описание**
yolo_anchors	Параметры начальной инициализации детектируемых прямоугольников (только для сетей *YOLO2* и *YOLO3*).
yolo_confidence_threshold	Порог для отображения результатов детектирования (только для сетей *YOLO2* и *YOLO3*).
yolo_iou_threshold	Порог пересекающихся прямоугольников детектирования (только для сетей *YOLO2* и *YOLO3*)
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 7.3.

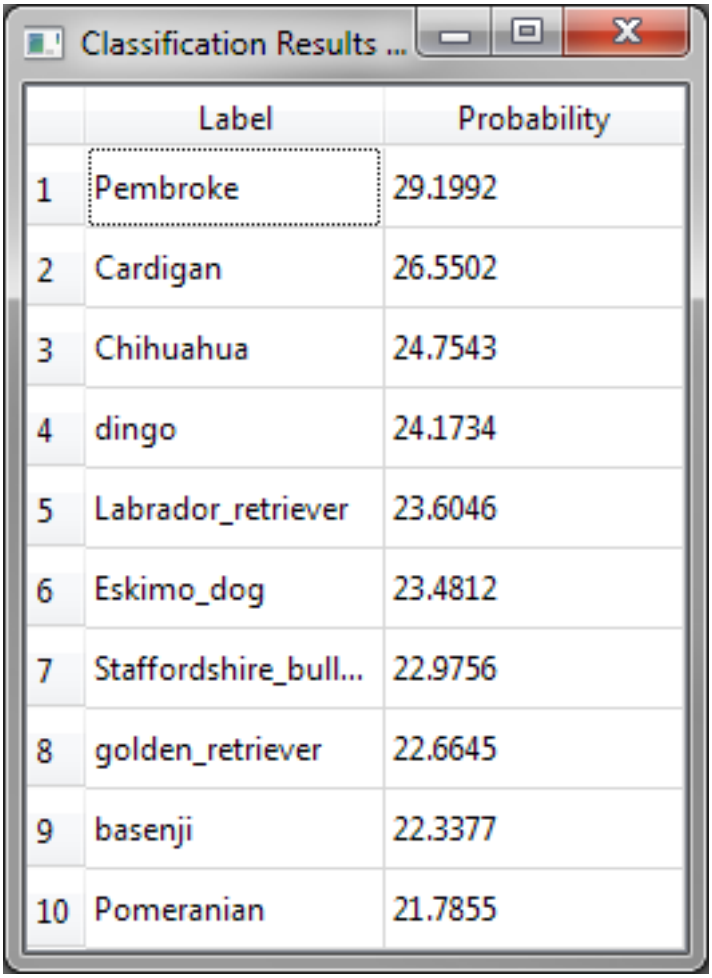

Рисунок 7.3 - Внешний вид окна результатов классификации

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 7.4.

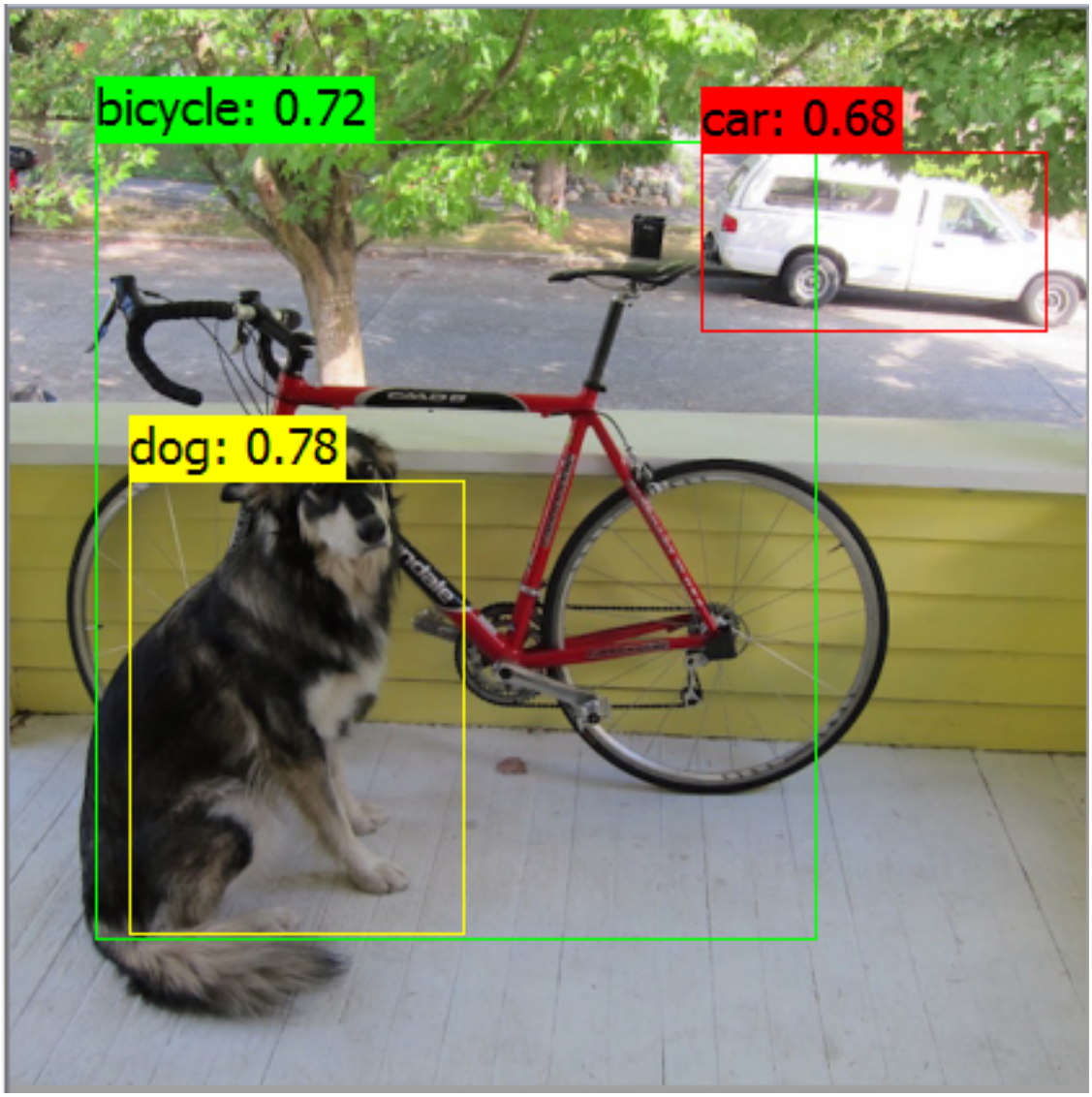

Рисунок 7.4 - Отображение результатов детектирования

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*".

8. Пример использования NMDL

В примере демонстрируется применение библиотеки *NMDL*, программных модулей для компиляции модели и подготовки изображений. Пример состоит из файла исходного кода *example.cpp* и скрипта сборки *CMakeLists.txt* в каталоге *"examples"* 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.pb"*
- 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 <fstream>
002 #include <iostream>
003 #include <string>
004 #include <unordered_map>
005 #include <vector>
006 #include "nmdl.h"
007 #include "nmdl_compiler.h"
008 #include "nmdl_image_converter.h"
009
010 namespace {
011
012 auto Call(NMDL_COMPILER_RESULT result, const std::string &function_na
013  static std::unordered_map<NMDL_COMPILER_RESULT, std::string> map = {
014   {NMDL_COMPILER_RESULT_OK,                          "OK"},
015   {NMDL_COMPILER_RESULT_MEMORY_ALLOCATION_ERROR, "MEMORY_ALLOCATION_E
016   {NMDL_COMPILER_RESULT_MODEL_LOADING_ERROR,     "MODEL_LOADING_ERROR
017   {NMDL_COMPILER_RESULT_INVALID_PARAMETER,       "INVALID_PARAMETER"}
018   {NMDL_COMPILER_RESULT_INVALID_MODEL,           "INVALID_MODEL"},
019   {NMDL_COMPILER_RESULT_UNSUPPORTED_OPERATION,   "UNSUPPORTED_OPERATI
020  };
021  if(result != NMDL_RESULT_OK) {
022   throw std::runtime_error(function_name + ": " + map[result] + ": "
023    NMDL_COMPILER_GetLastError());
024  }
025  return NMDL_RESULT_OK;
026 }
027
028 auto Call(NMDL_RESULT result, const std::string &function_name) {
029  static std::unordered_map<NMDL_RESULT, std::string> map = {
```

```
030    {NMDL_RESULT_OK,                                "OK"},
031    {NMDL_RESULT_INVALID_FUNC_PARAMETER,            "INVALID_FUNC_PARAMETER
032    {NMDL_RESULT_NO_LOAD_LIBRARY,                   "NO_LOAD_LIBRARY"},
033    {NMDL_RESULT_NO_BOARD,                          "NO_BOARD"},
034    {NMDL_RESULT_BOARD_RESET_ERROR,                 "BOARD_RESET_ERROR"},
035    {NMDL_RESULT_INIT_CODE_LOADING_ERROR,           "INIT_CODE_LOADING_ERRO
036    {NMDL_RESULT_CORE_HANDLE_RETRIEVAL_ERROR,       "CORE_HANDLE_RETRIEVAL_
037    {NMDL_RESULT_FILE_LOADING_ERROR,                "FILE_LOADING_ERROR"},
038    {NMDL_RESULT_MEMORY_WRITE_ERROR,                "MEMORY_WRITE_ERROR"},
039    {NMDL_RESULT_MEMORY_READ_ERROR,                 "MEMORY_READ_ERROR"},
040    {NMDL_RESULT_MEMORY_ALLOCATION_ERROR,           "MEMORY_ALLOCATION_ERRO
041    {NMDL_RESULT_MODEL_LOADING_ERROR,               "MODEL_LOADING_ERROR"},
042    {NMDL_RESULT_INVALID_MODEL,                     "INVALID_MODEL"},
043    {NMDL_RESULT_BOARD_SYNC_ERROR,                  "BOARD_SYNC_ERROR"},
044    {NMDL_RESULT_BOARD_MEMORY_ALLOCATION_ERROR,     "BOARD_MEMORY_ALLOCATIO
045    {NMDL_RESULT_NN_CREATION_ERROR,                 "NN_CREATION_ERROR"},
046    {NMDL_RESULT_NN_LOADING_ERROR,                  "NN_LOADING_ERROR"},
047    {NMDL_RESULT_NN_INFO_RETRIEVAL_ERROR,           "NN_INFO_RETRIEVAL_ERRO
048    {NMDL_RESULT_MODEL_IS_TOO_BIG,                  "MODEL_IS_TOO_BIG"},
049    {NMDL_RESULT_NOT_INITIALIZED,                   "NOT_INITIALIZED"},
050    {NMDL_RESULT_BUSY,                              "BUSY"},
051    {NMDL_RESULT_UNKNOWN_ERROR,                     "UNKNOWN_ERROR"}
052   };
053   if(result != NMDL_RESULT_OK) {
054    throw std::runtime_error(function_name + ": " + map[result]);
055   }
056      return NMDL_RESULT_OK;
057 }
058
059 template <typename T>
060 auto ReadFile(const std::string &filename) {
061   std::ifstream ifs(filename, std::ios::binary | std::ios::ate);
062   if(!ifs.is_open()) {
063    throw std::runtime_error("Unable to open input file: " + filename);
064   }
065   auto fsize = static_cast<std::size_t>(ifs.tellg());
066   ifs.seekg(0);
067   std::vector<T> data(fsize / sizeof(T));
068   ifs.read(reinterpret_cast<char*>(data.data()), data.size() * sizeof(
069   return data;
070 }
071
072 void ShowNMDLVersion() {
073   std::uint32_t version_major = 0;
074   std::uint32_t version_minor = 0;
075   Call(NMDL_GetLibVersion(&version_major, &version_minor), "GetLibVers
076   std::cout << "Lib version: " << version_major << "." << version_minc
```

```
077 }
078
079 void CheckBoard(std::uint32_t required_board_type) {
080  std::uint32_t boards;
081  std::uint32_t board_number = -1;
082  Call(NMDL_GetBoardCount(required_board_type, &boards), "GetBoardCoun
083  std::cout << "Detected boards: " << boards << std::endl;
084  if(!boards) {
085   throw std::runtime_error("Board not found");
086  }
087 }
088
089 auto CompileModel(const std::string &filename, std::uint32_t board_ty
090  auto onnx_model = ReadFile<char>(filename);
091  float *nm_model = nullptr;
092  std::uint32_t nm_model_floats = 0u;
093  Call(NMDL_COMPILER_CompileONNX(board_type, onnx_model.data(),
094   onnx_model.size(), &nm_model, &nm_model_floats), "CompileONNX");
095  std::vector<float> result(nm_model, nm_model + nm_model_floats);
096  NMDL_COMPILER_FreeModel(board_type, nm_model);
097  return result;
098 }
099
100 auto GetModelInformation(NMDL_HANDLE nmdl) {
101  NMDL_ModelInfo model_info;
102  Call(NMDL_GetModelInfo(nmdl, &model_info), "GetModelInfo");
103  std::cout << "Output tensor number: " << model_info.output_tensor_nu
104  for(std::size_t i = 0; i < model_info.output_tensor_num; ++i) {
105   std::cout << "Output tensor " << i << ": " <<
106    model_info.output_tensors[0].width << ", " <<
107    model_info.output_tensors[0].height << ", " <<
108    model_info.output_tensors[0].depth <<
109    std::endl;
110  }
111  return model_info;
112 }
113
114 auto PrepareFrame(const std::string &filename, std::uint32_t width,
115   std::uint32_t height, std::uint32_t board_type,
116   std::uint32_t color_format, float divider, float adder) {
117  auto bmp_frame = ReadFile<char>(filename);
118  std::vector<float> nm_frame(NMDL_IMAGE_CONVERTER_RequiredSize(
119    width, height, board_type));
120  if(NMDL_IMAGE_CONVERTER_Convert(bmp_frame.data(), nm_frame.data(), b
121    width, height, color_format, divider, adder, board_type)) {
122   throw std::runtime_error("Imnage conversion error");
123  }
```

```
124  return nm_frame;
125 }
126
127 void WaitForOutput(NMDL_HANDLE nmdl, std::uint32_t batch_num,
128   std::vector<float> &output) {
129  std::uint32_t status = NMDL_PROCESS_FRAME_STATUS_BUSY;
130  while(status == NMDL_PROCESS_FRAME_STATUS_BUSY) {
131   NMDL_GetStatus(nmdl, batch_num, &status);
132  };
133  double fps;
134  Call(NMDL_GetOutput(nmdl, batch_num, output.data(), &fps), "GetOutpu
135  if(batch_num == 0) {
136   std::cout << "FPS: " << fps << std::endl;
137  }
138  std::cout << "Data: " << std::endl;
139  for(auto &cur: output) {
140   std::cout << "\t" << cur << std::endl;
141  }
142 }
143
144 }
145
146 int main() {
147  const std::uint32_t BOARD_TYPE = NMDL_BOARD_TYPE_SIMULATOR;
148  //const uint32_t BOARD_TYPE = NMDL_BOARD_TYPE_MC12705;
149  //const uint32_t BOARD_TYPE = NMDL_BOARD_TYPE_MC12101;
150  const std::uint32_t COMPILER_BOARD_TYPE =
151   BOARD_TYPE == NMDL_BOARD_TYPE_MC12101 ?
152   NMDL_COMPILER_BOARD_TYPE_MC12101 :
153   NMDL_COMPILER_BOARD_TYPE_MC12705;
154  const std::uint32_t IMAGE_CONVERTER_BOARD_TYPE =
155   BOARD_TYPE == NMDL_BOARD_TYPE_MC12101 ?
156   NMDL_IMAGE_CONVERTER_BOARD_TYPE_MC12101 :
157   NMDL_IMAGE_CONVERTER_BOARD_TYPE_MC12705;
158 #ifdef _WIN32
159  const std::string ONNX_MODEL_FILENAME = "..\\nmdl_ref_data\\squeezen
160  const std::string BMP_FRAME_FILENAME = "..\\nmdl_ref_data\\squeezene
161 #else
162  const std::string ONNX_MODEL_FILENAME = "../nmdl_ref_data/squeezenet
163  const std::string BMP_FRAME_FILENAME = "../nmdl_ref_data/squeezenet/
164 #endif
165
166  const NMDL_IMAGE_CONVERTER_COLOR_FORMAT IMAGE_CONVERTER_COLOR_FORMAT
167   NMDL_IMAGE_CONVERTER_COLOR_FORMAT_BGR;
168  const float NM_FRAME_DIVIDER = 1.0f;
169  const float NM_FRAME_ADDER = -114.0f;
170
```

```
171  const std::size_t BATCHES = 4;
172  const std::size_t FRAMES = 5;
173
174  NMDL_HANDLE nmdl = 0;
175
176  try {
177   std::cout << "Query library version..." << std::endl;
178   ShowNMDLVersion();
179
180   std::cout << "Board detection... " << std::endl;
181   CheckBoard(BOARD_TYPE);
182
183   std::cout << "Compile model... " << std::endl;
184   auto nm_model = CompileModel(ONNX_MODEL_FILENAME, COMPILER_BOARD_TY
185
186   std::cout << "NMDL initialization... " << std::endl;
187   Call(NMDL_Create(&nmdl), "Create");
188   Call(NMDL_Initialize(nmdl, BOARD_TYPE, 0, nm_model.data(),
189     nm_model.size(), 0), "Initialize");
190
191   std::cout << "Get model information... " << std::endl;
192   auto model_info = GetModelInformation(nmdl);
193
194   std::cout << "Prepare frame... " << std::endl;
195   auto frame = PrepareFrame(BMP_FRAME_FILENAME, model_info.input_tens
196     model_info.input_tensor.height, IMAGE_CONVERTER_BOARD_TYPE,
197     IMAGE_CONVERTER_COLOR_FORMAT, NM_FRAME_DIVIDER, NM_FRAME_ADDER);
198
199   std::vector<float> output(NMDL_RequiredOutputFloats(&model_info));
200
201   std::cout << "Process single frames... " << std::endl;
202   for(std::size_t i = 0; i < FRAMES; ++i) {
203    Call(NMDL_ProcessFrame(nmdl, 0, frame.data()), "ProcessFrame");
204    WaitForOutput(nmdl, 0, output);
205   }
206   NMDL_Release(nmdl);
207
208   std::cout << "Batch mode process frames... " << std::endl;
209   std::uint32_t cnt_in = 0;
210   std::uint32_t cnt_out = 0;
211   Call(NMDL_Initialize(nmdl, BOARD_TYPE, 0, nm_model.data(),
212     nm_model.size(), 1), "Initialize");
213   for(auto i = 0u; i < BATCHES; ++i) {
214    Call(NMDL_ProcessFrame(nmdl, (cnt_in++) % BATCHES,
215      frame.data()), "ProcessFrame");
216   }
217   for(auto i = BATCHES; i < FRAMES; ++i) {
```

```
218     WaitForOutput(nmdl, (cnt_out++) % BATCHES, output);
219     Call(NMDL_ProcessFrame(nmdl, (cnt_in++) % BATCHES,
220       frame.data()), "ProcessFrame");
221    }
222    for(auto i = 0u; i < BATCHES; ++i) {
223     WaitForOutput(nmdl, (cnt_out++) % BATCHES, output);
224    }
225   }
226   catch (std::exception& e) {
227    std::cerr << e.what() << std::endl;
228   }
229   NMDL_Release(nmdl);
230   NMDL_Destroy(nmdl);
231
232     return 0;
233 }
```

Здесь выполняются следующие действия:

1 .. 8: 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.

12 .. 26: Функция-обёртка для вызовов функций библиотеки компиляции моделей. В случае ошибки формирует вывод об ошибке и инициирует исключение.

28 .. 57: Функция-обёртка для вызовов функций библиотеки нейросетевой обработки. В случае ошибки формирует вывод об ошибке и инициирует исключение.

59 .. 70: Универсальная функция для чтения данных из файла в вектор.

72 .. 77: Функция вывода версии NMDL. Вызов NMDL_GetLibVersion.

79 .. 87: Функция проверки наличия модуля заданного типа. Фактически производится запрос количества обнаруженных модулей заданного типа - NMDL_GetBoardCount.

89 .. 98: 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".

100 .. 112: Функция получения и вывода информации о параметрах выходных тензоров.. Используется вызов NMDL_GetModelInfo.

114 .. 125: Функция подготовки кадра. Здесь выполняется чтение изображения из файла, его декодирование и предобработка (NMDL_IMAGE_CONVERTER_Convert) . Описание предобработки приводится в разделе "Подготовка изображений".

127 .. 142: 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..

147: 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.

158 .. 164: Задаются имена файлов с исходными моделью и изображением.

166 .. 169: 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".

171: Задаётся количество кластеров при пакетной обработке.

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

Далее в главной функции выполняется последовательный вызов описанных вспомогательных функций.

187 .. 189: Инициализация NMDL. Посредством вызова функции NMDL_Create осуществляется инициализация внутренних структур библиотеки NMDL. В функции NMDL_Initialize производится загрузка скомпилированной модели в выбранный ускоритель.

201 .. 205: Пример последовательной обработки кадров в режиме разделения данных по кластерам (см. раздел "Режимы обработки").

208 .. 224: Пример последовательной обработки кадров в режиме пакетной обработки (см. раздел "Режимы обработки").

229 .. 230: При завершении работы освобождаются выделенные ресурсы (NMDL_Release и NMDL_Destroy).

9. Описание идентификаторов, функций и структур NMDL

9.1. Идентификаторы и структуры

9.1.1. 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;
```

- *NMDL_BOARD_TYPE_SIMULATOR* - симулятор модуля *MC127.05*.
- *NMDL_BOARD_TYPE_MC12101* - модуль *MC121.01*.
- *NMDL_BOARD_TYPE_MC12705* - модуль *MC127.05*.
- *NMDL_BOARD_TYPE_NMSTICK* - модуль *NMStick*.
- *NMDL_BOARD_TYPE_NMCARD* - модуль *NMCard*.

9.1.2. NMDL_ModelInfo

Структура с информацией о модели.

```
  typedef struct tagNMDL_ModelInfo {
    NMDL_Tensor input_tensor;
    unsigned int output_tensor_num;
    NMDL_Tensor output_tensors[NMDL_MAX_OUTPUT_TENSORS];
} NMDL_ModelInfo;
```

- *input_tensor* - описание входного тензора (входное изображение),
- *output_tensor_num* - количество выходных тензоров,
- *output_tensors* – выходные тензоры.

NMDL_MAX_OUTPUT_TENSORS - максимальное количество выходных тензоров.

9.1.3. NMDL_PROCESS_FRAME_STATUS

Идентификатор статуса обработки кадра.

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

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

9.1.4. 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_BUSY,
    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 синххронизации 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_BUSY* – устройство находится в состоянии обработки кадра,
- *NMDL_RESULT_UNKNOWN_ERROR* – неизвестная ошибка.

9.1.5. 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.

9.2. Функции

9.2.1. 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.

9.2.2. NMDL_Create

Создание экземпляра NMDL и получение идентификатора экземпляра *NMDL*.

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

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

9.2.3. NMDL_Destroy

Удаление экземпляра NMDL.

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

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

9.2.4. NMDL_GetBoardCount

Запрос количества обнаруженных модулей заданного типа.

```
  NMDL_RESULT NMDL_GetBoardCount(
    unsigned int boart_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.

9.2.5. NMDL_GetLibVersion

Запрос версии библиотеки *NMDL*.

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

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

9.2.6. NMDL_GetModelInfo

Запрос информации о модели.

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

- *nmdl* - [in] дескриптор *NMDL*
- *model_info* - [out] информация о модели.

9.2.7. NMDL_GetOutput

Запрос результата обработки.

```
  NMDL_RESULT NMDL_GetOutput(
    NMDL_HANDLE nmdl,
    unsigned int batch_num,
    float *output,
    double *fps
);
```

- *nmdl* - [in] идентификатор экземпляра *NMDL*.
- *batch_num* - [in] номер кластера 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.
- *output* - [out] указатель на буфер выходных тензоров. Выходные тензоры размещаются друг за другом в порядке их появления в графе обработки, то есть упорядоченные по уровням в графе. Параметры тензоров определяются в структуре NMDL_ModelInfo. Размер требуемого буфера возвращается в функции NMDL_RequiredOutputFloats. Может принимать значение 0.
- *fps* - [out] производительность обработки (кадров в секунду). Значение производительности определено только для кластера 0 (batch_num = 0), для других кластеров записывает 0.0f. Может принимать значение 0.

9.2.8. NMDL_GetStatus

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

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

- *nmdl* - [in] идентификатор экземпляра *NMDL*
- *batch_num* - [in] номер кластера для которого запрашивается статус обработки. Имеет значение только при работе с модулями *MC127.05*, *NMCard* или симулятором. 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.

9.2.9. NMDL_Initialize

Инициализация NMDL.

```
  NMDL_RESULT NMDL_Initialize(
```

```
    NMDL_HANDLE nmdl,
    unsigned int board_type,
    unsigned int board_number,
    const float *model,
    unsigned int model_floats,
    unsigned int use_batch_mode
);
```

- *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.
- *model* - [in] указатель на буфер, содержащий модель в формате *nm7* (для *MC121.01* и *NMStick*) 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.

9.2.10. NMDL_ProcessFrame

Обработка одиночного кадра.

```
  NMDL_RESULT NMDL_ProcessFrame(
    NMDL_HANDLE nmdl,
    unsigned int batch_num,
    const float *frame
);
```

- *nmdl* - [in] идентификатор экземпляра *NMDL*
- *batch_num* - [in] номер кластера на котором запускается обработка. Имеет значение только при работе с модулями *MC127.05*, *NMCard* или симулятором. При работе с *MC121.01* и *NMStick* необходимо установить 0.
- *frame* - [in] указатель на буфер обрабатываемого кадра.

9.2.11. NMDL_Release

Деинициализация NMDL.

```
  void NMDL_Release(
    NMDL_HANDLE nmdl
```

```
);
```

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

9.2.12. NMDL_RequiredOutputFloats

Возвращает размер буфера в элементах *float32*, необходимый для размещения результата (*output*) при вызове NMDL_GetOutput.

```
  unsigned int NMDL_RequiredOutputFloats(
    const NMDL_ModelInfo *model_info
);
```

- *model_info* - [out] информация о модели, ренее полученная с помощью вызова NMDL_GetModelInfo.

10. Описание идентификаторов и функций nmdl_compiler

10.1. Идентификаторы

10.1.1. 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 *MC121.01* и *NMStick*.
- *NMDL_COMPILER_BOARD_TYPE_MC12705* - модули *MC127.05*, *NMCard* и симулятор.

10.1.2. 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.

10.2. Функции

10.2.1. NMDL_COMPILER_CompileDarkNet

Компиляция исходной модели в формате DarkNet.

```
  NMDL_COMPILER_RESULT NMDL_COMPILER_CompileDarkNet(
    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
);
```

- *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*.

10.2.2. NMDL_COMPILER_CompileONNX

Компиляция исходной модели в формате ONNX.

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

- *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 *.pb*.

- *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*.

10.2.3. 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.

10.2.4. NMDL_COMPILER_GetLastError

Возвращает константную строку с описанием последней ошибки.

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

11. Описание идентификаторов и функций nmdl_image_converter

11.1. Идентификаторы и структуры

11.1.1. 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.

11.1.2. 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;
```

11.2. Функции

11.2.1. 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.

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

11.2.2. 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,
    float divider,
    float adder,
    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.
- *divider* - [in] делитель в выражении dst = src / divider+ adder. Приведённая операция выполняется над каждым каналом каждого пикселя исходного изображения.
- *adder* - [in] слагаемое в выражении dst = src / divider+ adder. Приведённая операция выполняется над каждым каналом каждого пикселя исходного изображения.
- *board_type* - [in] идентификатор типа вычислительного модуля на котором предполагается обработка. Одно из значений перечисления NMDL_IMAGE_CONVERTER_BOARD_TYPE.

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

НАУЧНО-ТЕХНИЧЕСКИЙ ЦЕНТР

ЗАО 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: 07/09/2020