УДК 004.054

**Практика применения набора тестов MLPerf Inference**

**для оценки характеристик нейросетевых вычислителей**

*Сергей Владимирович Ландышев*

АО НТЦ «Модуль», Россия, 125190, г. Москва, 4-я ул. 8 марта, д.3. landysh@module.ru

**Аннотация —** В статье приводится краткий обзор системы тестирования MLPerf. Рассматриваются принципы применения тестовых наборов MLPerf Inference в контексте оценки бенчмарков производителем аппаратуры и разработчиком ПО для трансляции и оптимизации нейронных сетей.

# Ключевые слова — MLPerf, Inference, LoadGen, нейронные сети, CNN, Benchmarking

**Practice of using the MLPerf Inference test suite**

**for evaluating the characteristics of neural network computings**

*Sergey Landyshev*

RC Module, Russia, 125190, Moscow, 3 Eighth March 4th Street. landysh@module.ru

**Abstract —** This article provides a brief overview of the MLPerf testing system. The principles of application of MLPerf Inference test suites are considered in the context of benchmark evaluation by a hardware manufacturer and a software developer for translation and optimization of neural networks.

# Keywords: MLPerf, Inference, LoadGen, CNN, Benchmarking

# Введение

Множество комбинаций устройств и программного обеспечения для выполнения нейросетевых вычислений затрудняют оценку производительности систем машинного обучения и инференса. Задача построения системы тестирования, которая была-бы достаточно абстрагирована от архитектурны устройств, обладала репрезентативностью и воспроизводимостью, является сверхсложной и актуальной с востребованным решением.

В настоящее время определился набор специализированного тестового ПО, применяемый, в основном, для оценки производительности отдельных операций (микробенчмарки), например, AI Matrix [1] и DeepBench [2], либо тесты с ограниченным набором моделей и сценариев, например, EEMBC MLMark [3] и TBD [4]. Специализация тестов не является их недостатком, наоборот, именно такие тесты интересны разработчикам устройств и программ. С другой стороны, для потребительской оценки готовых и предлагаемых на рынке решений в области ИНС, эти тесты недостаточно обобщены или недостаточно репрезентативны с точки зрения применения в решении наиболее общих, отраслевых задач.

Одной из последних систем на рынке специализированного тестового ПО являются наборы тестов MLPerf от компании MLCommons. MLPerf предлагает открытый механизм получения бенчмарков с глубоко обоснованным применением множества моделей нейронных сетей и сценариев тестирования. Система имеет относительно недолгую историю, которая началась в 2018 году как стартап для оценки производительности машинного обучения [5]. Насущная потребность в открытой системе тестирования в среде производителей устройств и программного обеспечения определила быстрый рост сообщества разработчиков и потребителей MLPerf по всему миру. В настоящее время MLPerf поддерживают более 70 организаций от стартапов в области программного обеспечения до исследователей из ведущих университетов и гигантов облачных вычислений.

В этой статье будут рассмотрены принципы применения системы тестирования MLPerf в контексте оценки бенчмарков производителем аппаратуры и разработчиком ПО для трансляции и оптимизации нейронных сетей. Рассматривается доступная на момент подготовки статьи версия MLPerf r1.1.

Выбор предлагаемых на рынке устройств от российских производителей, даже учитывая потенциальный выход на рынок, не богат. Все предлагаемые решения на практике пригодны только для проигрывания сетей, для систем обучения эти устройства не могут предоставить требуемые вычислительные мощности. В этой связи, здесь будут рассматриваться только наборы тестов MLPerf Inference.

**Общие сведения**

Система тестирования покрывает четыре типовые задачи, эффективно решаемые с применением нейронных сетей: классификация изображений, обнаружение объектов на изображениях, семантическая сегментация и машинный перевод текста. Для каждой из этих задач, по обоснованиям разработчиков тестов, выбраны соответствующие модели нейронных сетей и исходные тестовые данные, доступные для всех производителей тестируемых устройств (таблица 1).

Таблица 1. Тестовые задачи MLPerf Inference

|  |  |  |  |
| --- | --- | --- | --- |
| *Класс задач* | *Модель* | *Набор* | *Метрика оценки точности* |
| Классификация изображений | ResNet-50 v1.5 [6] | ImageNet [7] | Top-1 |
| Обнаружение объектов на изображениях | SSD-ResNet-50 [8], SSD-MobileNet-v1 | COCO [9] | mAP [10] |
| Семантическая сегментация изображений | 3d U-Net [11], Mask R-CNN [12] | BraTS 2019 [13], COCO | mIoU [14] |
| Перевод | NMT [15], Transformer | WMT Eng.-German | SacreBLEU [16] |

MLPerf Inference определяет четыре сценария тестирования: однопоточный (Single-Stream), многопоточный (Multi-Stream), серверный (Server) и автономный (Offline). Каждый из этих сценариев характеризуется своим правилом распределения нагрузки на тестируемую систему во времени и имитирует обработку в реальных системах обслуживания.

Необходимость выполнения сценариев в процессе тестирования определяется типом набора. В системе предлагаются следующие тестовые наборы:

Таблица 2. Наборы и сценарии

|  |  |
| --- | --- |
| *Набор* | *Сценарии* |
| Datacenter – центр хранения и обработки данных (ЦХОД) | Серверный и автономный |
| Edge – всё, что не относится к ЦХОД | Однопоточный, автономный и многопоточный (опционально) |

Оценка выполняется для двух режимов работы – тест производительности и тест точности.

Основной инструмент MLPerf – библиотека LoadGen, в которой реализуется генератор нагрузки. Библиотека распространяется в исходных кодах и имеет интерфейс С++. В состав MLPerf входят программные оболочки, позволяющие реализовать методы на языке Python. Для применения LoadGen необходимо собрать библиотеку, связать с тестовой программой и по-своему реализовать набор виртуальных методов, в которых организуется взаимодействие программы с тестируемым устройством. Управление тестированием осуществляется через файлы конфигурации.

Итогом тестирования являются текстовые лог-файлы, в которых отображается вся информация о ходе теста и результаты его выполнения, рассчитанные по соответствующим метрикам. В таблице 3 приведены основные критерии в режиме измерения производительности.

Таблица 3. Сценарии и критерии

|  |  |
| --- | --- |
| *Сценарий* | *Основной критерий для производительности* |
| Однопоточный (Single-Stream) | Задержка от начала обработки до выдачи результата (латентность). |
| Многопоточный (Multi-Stream) | Производительность (запросов в секунду). Запросы формируются в несколько потоков. |
| Серверный (Server) | Производительность (запросов в секунду) при Пуассоновском распределении поступления запросов.  |
| Автономный (Offline) | Производительность (запросов в секунду). |

Для оценки точности используются различные метрики, в зависимости от класса задачи (таблица 1). Оценка точности выполняется с применением внешних инструментов, входящих в состав MLPerf, на основе данных, сформированных в генераторе нагрузки.

**Подготовка к тестированию**

Для применения MLPerf Inference необходимо наличие в системе следующих компонентов:

* Утилита CMake, версии не ниже 3.1.
* Python с модулями absl-py и numpy.
* CocoAPI[17] для работы со скриптом оценки точности при обнаружении объектов.

Перед тестированием загружаются и транслируются все необходимые модели. Также загружаются необходимые наборы изображений. Изображения, обрабатываемые в ходе тестирования в MLPerf должны быть предварительно обработаны, то есть приведены к формату, который требуется для загрузки на тестируемое устройство. Такие подготовленные изображения обозначаются как «образцы» (samples). Каждому образцу присваивается идентификатор – его порядковый номер в полном наборе.

Особенности тестирования для определённых классов задач проявляются в режиме оценки точности, так как здесь используются различные метрики. Для оценки теста необходимо иметь эталонные данные, например, для оценки точности по метрике Top-1 необходимо загрузить карту классификации – это текстовый файл, в котором в каждой строке содержится пара значений: наименование файла изображения и номер соответствующего класса. При оценке в классе задач обнаружения объектов используются эталонные данные, входящие в стандартный набор данных COCO [18].

При подготовке образцов можно отметить важное обстоятельство на примере предобработки изображений из набора ImageNet для модели ResNet-50. Изображения из набора представлены в формате JPEG и имеют различные разрешения. При подготовке образцов неизбежны декодирование и масштабирование. Выполнение последней операции вносит неоднозначность, так как существует многообразие методов интерполяции пикселей изображения. Применение «неудачного» метода влечёт за собой существенное ухудшение оценки точности. Например, применение линейной интерполяции недопустимо, так как снижает показатели точности до 5% и приводит к неприемлемой ошибке. Рекомендуется использовать наиболее качественный метод из доступных разработчику. Для достижения нужной оценки качества в [19] рекомендуется выполнить последовательно масштабирование и обрезку относительно центра. Центральная часть результирующего изображения в результате окажется несколько увеличена по сравнению с исходным. На рисунке 1 показаны выполняемые преобразования для изображения из набора ImageNet с id=3.



Рисунок 1 — Подготовка изображений

MLPerf устанавливает разрешённые и запрещённые техники, которые следует учитывать при трансляции моделей и создании образцов.

*Разрешено:*

* Произвольное размещение и форматирование образцов и весов.
* Неограниченное время предобработки.
* Замена операций математически эквивалентными операциями.
* Произвольный порядок обработки внутри сценария.
* Квантование весов.
* Произвольное использование пакетного режима.

*Запрещено:*

* Переобучение моделей.
* Изъятие операций из модели.
* Кэширование запросов (особенно актуально для перевода).
* Оптимизация на основе априорной информации о параметрах генератора нагрузки, например, когда заранее известна последовательность запросов образцов.

**Генератор нагрузки**

LoadGen предоставляется в исходных кодах со сборочными скриптами. Сборка библиотеки выполняется утилитой cmake:

> mkdir MLPerf/loadgen/build

 > cd MLPerf/loadgen/build

 > cmake ..

 > cmake --build.

В результате сборки в каталоге MLPerf/loadgen/build появляется статическая библиотека mlperf\_loadgen.lib.

Сборку можно выполнить как в среде Linux, так и в Windows с инструментом MS VisualStudio (версии с поддержкой С++14).

На рисунке 2 показана схема взаимодействия генератора с тестовыми компонентами.



Рисунок 2 — Схема работы генератора нагрузки

Под тестируемой системой понимается пользовательская программа, в которой реализованы методы взаимодействия с тестируемым устройством.

На схеме взаимодействия операции пронумерованы в хронологическом порядке:

1. После старта программы генератор нагрузки определяет поднабор из нескольких изображений-образцов, которые будут использоваться в текущей обработке и передаёт идентификаторы образцов тестируемой системе.
	1. Тестируемая система считывает файлы выбранных образцов и записывает их в оперативную память. Эта операция исключает время обращения к дисковой памяти из времени тестирования при оценке производительности.
2. Тестируемая система сигнализирует генератору о готовности к тесту.
3. Генератор выдаёт запросы на обработку образцов в соответствии со сценарием теста.
4. Тестируемая система выполняет обработку образцов и передаёт генератору результат обработки. LoadGen формирует текстовые файлы с результатом тестирования.
5. В режиме теста точности запускается скрипт вычисления метрики.

В зависимости от выбранного сценария тестирование выполняется итеративно с повторением операций 1 – 6.



Рисунок 3 — Сценарии LoadGen

На рисунке 3 схематично показано, как генератор нагрузки работает в различных сценариях:

* В однопоточном сценарии образцы обрабатываются последовательно один за другим, с темпом, равным времени обработки одного изображения.
* В многопоточном сценарии предполагается пакетная обработка образцов с заданным темпом. В каждом пакете содержатся N образцов. Тестируемая система должна обработать пакет в заданном временном интервале. Если недостаточно производительности, то интервал пропускается. Тест считается выполненным, если количество пропущенных интервалов не превышает 1%.
* Серверный сценарий – это имитация обработки изображений на облачных серверах, когда запросы на обработку поступают со случайными интервалами с Пуассоновским распределением [20]. Значение интервалов варьируются от 15 до 250 мс.
* В автономном сценарии однократно обрабатывается один набор образцов.

**Тестируемая система**

Тестируемая система (SUT – System Under Test) реализует основную программу – функцию main, которая схематично, без целевой имплементации, может выглядеть так:

#include "loadgen.h"

#include "query\_sample\_library.h"

#include "system\_under\_test.h"

class MySUT : public mlperf::QuerySampleLibrary, public mlperf::SystemUnderTest {

public:

 void Test(int argc, char\* argv[]) {

 mlperf::LogSettings log\_settings;

 mlperf::TestSettings test\_settings;

Initialize(argc, argv, log\_settings, test\_settings);

 mlperf::StartTest(this, this, test\_settings, log\_settings);

 Release();

 }

private:

 const std::string &Name() const override {…}

 std::size\_t TotalSampleCount() override {…}

 std::size\_t PerformanceSampleCount() override {…}

 void LoadSamplesToRam(const std::vector<mlperf::QuerySampleIndex>& samples) override {…}

 void UnloadSamplesFromRam(const std::vector<mlperf::QuerySampleIndex>& samples) override {…}

 void IssueQuery(const std::vector<mlperf::QuerySample> &samples) override {…}

 void FlushQueries() override {…}

 void ReportLatencyResults(const std::vector<mlperf::QuerySampleLatency>& ns) override {…}

void Initialize(int argc, const char\* argv[], mlperf::LogSettings& log\_settings,

mlperf::TestSettings& test\_settings) {…}

void Release() {…}

};

int main(int argc, char\* argv[]) {

 MySUT sut;

 sut.Test(argc, argv);

 return 0;

}

Операции “…” в фигурных скобках заменяются необходимой имплементацией.

Поддержка SUT – это определение своего класса, наследуемого от *mlperf::QuerySampleLibrary* и *mlperf::SystemUnderTest* и реализация виртуальных методов порождающих классов:

* *Name* – метод вызывается для получения названия SUT при формировании журналов тестирования (лог-файлы).
* *TotalSampleCount* – в методе необходимо вернуть общее количество образцов в тестовом наборе, с которыми будет оперировать LoadGen в ходе тестирования.
* *PerformanceSampleCount* – в методе необходимо вернуть количество образцов в тестовом наборе, которые гарантированно могут быть размещены в оперативной памяти компьютера, на котором запускается тест (не оперативная память SUT). В ходе тестирования полный набор образцов, размером *TotalSampleCount* разбивается на порции, размером *PerformanceSampleCount*. Обработка выполняется над порциями образцов для исключения из оценки производительности времени обращения к диску.
* *LoadSamplesToRam* – в методе необходимо считать образцы в оперативную память. Параметром метода является вектор индексов образцов. Количество образцов не может быть больше возвращаемого значения *PerformanceSampleCount.* В MLPerf гарантируется, что каждый образец будет обработан только однократно.
* *UnloadSamplesFromRam* – метод вызывается после обработки порции образцов и здесь необходимо освободить оперативную память, выделенную при вызове *LoadSamplesToRam*. Параметром метода является вектор индексов обработанных образцов.
* *IssueQuery* – в методе необходимо передать набор образцов для обработки в тестируемое устройство. Количество образцов и их индексы передаются в параметре-векторе. В зависимости от тестового сценария, образцы могут быть обработаны в этом же методе или в другом потоке, когда результат обработки формируется позже. Результат обработки образцов передаётся в LoadGen через вызов *mlperf::QuerySamplesComplete*. Формат результата зависит от режима тестирования и класса задачи. Каждый тест состоит из одной или нескольких серий вызовов *IssueQuery* в зависимости от режима и сценария.
* *FlushQueries* – LoadGen вызывает этот метод после последнего вызова *IssueQuery* в серии. Этот вызов не обязательно означает конец теста, поскольку во время теста может быть задействовано несколько серий, например, в режиме оценки точности. Как правило, метод используется в серверном сценарии для немедленного сброса отложенных запросов.
* *ReportLatencyResults* – LoadGen передаёт тестируемому устройству значение задержки между сериями обработки в наносекундах.

В функции *Initialize* выполняются все необходимые действия для инициализации SUT. В функции *Release* освобождаются все выделяемые при инициализации ресурсы для корректного выхода из тестирования.

Последовательность вызовов можно наблюдать с помощью вывода на консоль сообщений из описанных выше виртуальных методов. Например, при работе автономного сценария в режиме оценки точности для общего количества образцов 2000 (*TotalSampleCount* = 2000) и набора, размером 1000 (*PerformanceSampleCount* = 1000), последовательность вызовов выглядит так:

*>> LoadSamplesToRam* – загрузка 1000 образцов в оперативную память.

*>> IssueQuery* – обработка набора из 1000 образцов.

*>> FlushQueries* – обработано 1000 образцов.

>> *UnloadSamplesFromRam* – удаление 1000 образцов из оперативной памяти.

*>> LoadSamplesToRam* – загрузка следующих 1000 образцов в оперативную память.

*>> IssueQuery* – обработка следующего набора из 1000 образцов.

*>> FlushQueries* – обработано 2000 образцов.

>> *UnloadSamplesFromRam* – удаление 1000 образцов из оперативной памяти.

LoadGen предоставляет доступ для настроек параметров тестирования через открытые поля следующих классов:

* *mlperf::TestSettings* – задаются параметры тестирования. Необходимо задать по крайней мере сценарий, режим работы и ожидаемую производительность (образцы в секунду).
* *mlperf::LogSettings* – задаются параметры журналирования. LoadGen выводит свои файлы журналов в заданную здесь директорию. При наименовании файлов журналов могут быть использованы заданные префиксы и суффиксы.

Инициализацию полей можно выполнить как непосредственно в программе, так и с помощью внешнего файла конфигурации. Во втором случае настройки определяются в файле с названием «audit.config», при этом настройки, определённые в файле, будут установлены, даже если параметры инициализируются в программе, то есть приоритетными являются настройки из файла. Конфигурация описывается в текстовом файле в формате «*ключ = значение»*, где «*ключ*» формируется в виде «*модель.сценарий.название»*. Если предполагается публикация результатов тестирования, то следует учитывать, что при верификации тестов разработчики MLCommons используют файл конфигурации.

**Результаты тестирования**

В ходе тестирования LoadGen выполняет журналирование событий и по окончании тестирования формирует набор текстовых файлов, содержащих журнал и результаты тестирования. Название файлов и их каталог размещения могут задаваться пользователем при инициализации полей класса *mlperf::LogSettings.* По умолчанию для всех файлов журнала определён префикс «mlperf\_log». Формируются следующие файлы:

* mlperf\_log\_accuracy.json – результат обработки образцов в режиме оценки точности. Формируется в формате JSON [21]. В каждой записи информация о тестировании одного образца – порядковый номер теста, индекс образца в наборе и результат инференса. Форма записи последнего зависит от используемой метрики оценки точности, например, для метрики Top1 результат содержит идентификатор класса, к которому принадлежит изображение на образце с наибольшей вероятностью.
* mlperf\_log\_detail.txt – содержит информацию о настройках последнего теста.
* mlperf\_log\_summary.txt – результат обработки в режиме оценки производительности.
* mlperf\_log\_trace.json – содержит подробный журнал событий последнего теста.

В режиме оценки производительности генератор нагрузки формирует полный отчёт с измеренными задержками и темпом. Для измерений в режиме оценки точности используются различные метрики в зависимости от класса задачи. В этом случае результат вычисляется в режиме постобработки с применением скриптов, которые обрабатывают файл mlperf\_log\_accuracy.json. Например, для вычисления точности для сетей-классификаторов (метрика Top1) используется скрипт «accuracy-imagenet.py» (размещается в каталоге «mlperf\_r1.1\vision\classification\_and\_detection\tools\»).

**Заключение**

Для устройств, обрабатывающих данные в формате с плавающей точкой float32, обработка любых моделей производится с той же точностью, какая достигалась в обучающих фреймворках. Действительно, и при обучении, и при инференсе нейронных сетей используется одинаковый формат данных. Здесь можно согласиться с репрезентативностью тестов в режиме оценки точности с использованием одной модели, которая является наиболее характерной для того или иного типа задач (классификация, обнаружение, сегментация).

Для устройств, оперирующих с числами с фиксированной точкой и пониженной разрядности, ситуация меняется. Для достижения требуемой точности коэффициенты свёрток квантуются так, чтобы достигалась максимальная корреляция между весами до и после квантования. Наиболее распространёнными методами квантования уже обученных нейронных сетей являются методы, связанные с повышением энтропии. Здесь возможны существенные потери, когда исходные веса имеют изначально высокую энтропию. Существуют различные методики квантования [22], но ни один из них не даёт гарантию достижения требуемой точности. В некоторых случаях требуемая точность может быть достигнута «дообучением» нейронной сети, но эта техника не допускается в регламенте MLPerf. Эффективность трансформации весов зависит от модели сети и от обучающей выборки (например, таблица 1 в [23]), и тестирование с использованием ResNet50 ImageNet в этом случае не репрезентативно. Применение метрики Top-5 даёт более полную оценку точности и, возможно, требуемая в тестах точность уже бы не достигалась. Описанная выше проблема решается введением в MLPerf моделей с целочисленными данными. В последней версии MLPerf уже предлагаются модели с данными в формате int8. До этого предполагалось использование целочисленных моделей только для mobilenet-v1.

Также использование лишь одной модели не характеризует тестируемое устройство с точки зрения универсальности применения. Например, применяемая для оценки классификаторов модель Resnet50 использует 12 операций ONNX из более, чем 150-ти определённых в стандарте [24]. В других моделях может использоваться другой набор операций, которые могут быть не реализованы на тестируемом устройстве, либо могут выполняться не оптимально.

В статье рассмотрены методы применения тестовых наборов MLPerf с использованием средств разработки С++. Приводятся примеры работы с предлагаемыми программными инструментами.

Выявлены следующие преимущества и субъективные недостатки:

Достоинства:

* Выполнена систематизация наборов тестов. В системе предлагаются тесты для решения наиболее актуальных задач, выполняемых на разных вычислительных платформах.
* Определены критерии для оценки производительности и точности, которые сведены в таблицы с открытым доступом.
* Широко представлено сообщество разработчиков и пользователей MLPerf (MLPerf community). Обилие и доступность документации, руководств и статей.
* Определены перспективные задачи для рабочих групп разработчиков – MLPerf находится в активной фазе развития.
* Система поддерживается ведущими мировыми производителями устройств для нейросетевых вычислений.
* Выбранные критерии, модели, тестовые выборки и прочие исходные компоненты тестирования имеют обоснования, которые приводятся в открытой документации.
* Кроссплатформенное решение (работает в ОС Linux и Windows).

Недостатки:

* В случаях с квантованными весами для моделей float 32 тесты не являются репрезентативными.
* Тесты не выявляют полноту реализации нейросетевых операций (набор операций ONNX).
* Имеется неоднозначность в предоставлении исходных данных, а именно в процедуре подготовки образцов. В составе MLPerf нет инструмента однозначно масштабирующего изображения.

Амбициозная задача создания эталонных тестовых наборов для оценки производительности и точности нейросетевых вычислений компанией MLCommons в целом решена. Результаты тестирования доступны всем потенциальным потребителям и могут быть применены в отраслях с высокой степенью достоверности.

**Список литературы**

1. Alibaba, “Ai matrix.” https://aimatrix.ai/en-us/, Alibaba, 2018.
2. Baidu, “DeepBench: Benchmarking Deep Learning Operations on Different Hardware,” https://github.com/baidu-research/DeepBench, 2017.
3. EEMBC, “Introducing the eembc mlmark benchmark,” https://www. eembc.org/mlmark/index.php, Embedded Microprocessor Benchmark Consortium, 2019.
4. H. Zhu, M. Akrout, B. Zheng, A. Pelegris, A. Jayarajan, A. Phanishayee, B. Schroeder, and G. Pekhimenko, “Benchmarking and analyzing deep neural network training,” in IEEE International Symposium on Workload Characterization (IISWC), 2018.
5. “MLCommons history”, https://mlcommons.org/en/history/
6. “ResNet in TensorFlow,” https://github.com/mlperf/training/tree/master/image classification/tensorflow/official, 2019.
7. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in 2009 IEEE conference on computer vision and pattern recognition. IEEE, 2009.
8. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.- Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In European conference on computer vision, pages 21–37. Springer, 2016.
9. T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Doll´ar, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in European conference on computer vision. Springer, 2014.
10. Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, Andrew Zisserman, “The PASCAL Visual Object Classes (VOC) Challenge”, https://homepages.inf.ed.ac.uk/ckiw/postscript/ijcv\_voc09.pdf, стр. 11.
11. Ozgun Cicek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger, “3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation”, arXiv:1606.06650v1 [cs.CV] 21 Jun 2016.
12. Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick, “Mask R-CNN”, Facebook AI Research (FAIR), arXiv:1703.06870v3 [cs.CV] 24 Jan 2018.
13. Feifan Wang,Runzhou Jiang, Liqin Zheng, Chun Meng and Bharat Biswal, "3D U-Net Based Brain Tumor Segmentation and Survival Days Prediction", arXiv:1909.12901v2 [eess.IV] 31 Mar 2020.
14. Azim Ahmadzadeh, Dustin J. Kempton, Yang Chen, and Rafal A. Angryk, "Multiscale iou - a metric for evaluation of salient object detection with fine structures", Department of Computer Science, Georgia State University, USA, arXiv:2105.14572v1 [cs.CV] 30 May 2021.
15. Dzmitry Bahdanau, KyungHyun Cho, Yoshua Bengio, “Neural machine translation by jointly learning to align and translate”, arXiv:1409.0473v7 [cs.CL] 19 May 2016.
16. Matt Post, Amazon Research, "A Call for Clarity in Reporting BLEU Scores", Berlin, Germany, arXiv:1804.08771v2 [cs.CL] 12 Sep 2018.
17. https://github.com/philferriere/cocoapi.
18. “ How to work with object detection datasets in COCO format”, https://towardsdatascience.com/how-to-work-with-object-detection-datasets-in-coco-format-9bf4fb5848a4.
19. https://github.com/mlcommons/training/tree/master/image\_classification/#data-preprocessing.
20. Вентцель Е. С., Овчаров Л. А. “Теория вероятностей и её инженерные приложения”, 2-е изд., М.: Высшая школа, 2000, стр. 135.
21. Pierre Bourhis, Juan L. Reutter, Fernando Suárez, Domagoj Vrgoc, “JSON: data model, query languages and schema specification”, arXiv:1701.02221 [cs.DB] 9 Jan 2017.
22. Sean I. Young, Wang Zhe, David Taubman and Bernd Girod, “Transform Quantization for CNN Compression”, IEEE Transactions on pattern analysis and machine intelligence, vol. 0, no. 0, may 2021.
23. Yunhui Guo, “A Survey on Methods and Theories of Quantized Neural Networks”, arXiv:1808.04752v2 [cs.LG] 16 Dec 2018.
24. https://github.com/onnx/onnx/blob/master/docs/Operators.md.