АННОТАЦИЯ

Подивилов О.М. Система распознавания образов с использованием аппаратной нейросети на основе ПЛИС. – Челябинск: ЮУрГУ, ЭУз-571, ПЗ 2023, 100 с., 28 ил., 1 листинг программы, 4 табл., библиогр., список – 61 наим., 10 л. Раздаточного материала ф. А4

Выпускная квалификационная работа выполнена с целью изучения и разработки методов распознавания образов на базе фреймворка Vitis-AI фирмы Xilinx.

В обзорной части выпускной квалификационной работы рассмотрены существующие средства разработки систем машинного зрения, присутствующие на рынке. Во второй главе приведен обзор существующих моделей нейросетей и их особенностей. Обосновано использование конкретной модели нейросети для проекта.

Практическая часть выпускной квалификационной работы содержит описание маршрута проектирования системы распознавания образов на основе фреймворка Vitis-AI и отладочной платы Alinx AXU2CG-A. Разработан прототип системы машинного зрения для нахождения дефектов изготовления печатных плат и дефектов при печати полиграфической продукции. Проведены предварительные испытания работы элементов системы распознавания образов на основе Alinx AXU2CG-A.

Обоснована целесообразность применения аппаратных ускорителей математических вычислений для использования в области технологий распознавания образов.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы. Тема машинного зрения и применения в машинном зрении нейросетей имеет огромную актуальность в современном мире. Рассмотрим некоторые возможности применения нейросетей в различных областях человеческой деятельности и в области машинного зрения и распознавания образов, в частности.

Современные электронные технологии предоставляют нам уникальные возможности для быстрой разработки и тестирования идей, возникающих у математиков, программистов, экономистов. В областях, где можно применить определенные формальные идеи. В частности, современные технологии машинного интеллекта позволяют с огромными скоростями проводить факторизацию больших сложных множеств (задачи распознавания), проводить анализ огромных объемов статистической информации для выделения определенных закономерностей, которые до недавнего времени невозможно было обнаружить.

Огромный потенциал имеется для использования нейросетей в физике, астрономии и других науках связанных с обработкой больших объемов данных. Их классификации и выявлению новых закономерностей. Вот некоторые конкретные примеры:

1. Анализ данных и обработка изображений. Нейросети могут использоваться для обработки данных, полученных в физических экспериментах или при астрономических наблюдениях. Они способны автоматически находить закономерности в рядах данных, выделять особенности и извлекать новую информацию. Например, нейросети могут быть использованы для классификации звезд и галактик по их спектральным характеристикам или для обнаружения и классификации космических объектов на фотографиях.

2. Аппроксимация и моделирование сложных функций. Нейросети могут использоваться для аппроксимации сложных функций или моделирования законов физики, особенно в случаях, когда аналитическое решение сложно или невозможно получить. Это может помочь в создании новых моделей и понимании физических явлений.

Варианты использования нейросетей в экономике и бизнесе также огромны:

1. Прогнозирование спроса. Нейросети могут использоваться для прогнозирования спроса на товары и услуги. Они анализируют исторические данные о рыночных трендах, клиентах и продажах, других факторах, для предсказания будущего спроса. Это позволяет компаниям более точно планировать маркетинговые кампании, производство и запасы.
2. Анализ и предсказания рыночных трендов. Нейросети способны анализировать огромные объемы данных о рынке, включая финансовые данные, новости, социальные медиа и другие источники информации. Они могут выявлять скрытые закономерности и тренды, помогая предсказывать направление рынка и принимать более обоснованные инвестиционные решения.
3. Персонализация и рекомендации. Нейросеть может использоваться для создания персонализированных рекомендаций продуктов и услуг на основе данных о предпочтениях и поведении клиентов. Это может помочь улучшить опыт покупателей, увеличить лояльность и повысить конверсию.
4. Обработка естественного языка. Нейросети можно использовать для обработки и анализа естественного языка, что позволяет автоматически классифицировать и извлекать информацию из текстовых данных. Это будет полезно, например, при автоматической обработке отзывов

клиентов, анализа текстовых данных из социальных сетей или создании виртуальных помощников.

5. Оптимизация бизнес-процессов. Нейросеть можно использовать для оптимизации различных бизнес-процессов, таких как управление запасами, логистика, планирование производства и прогнозирование спроса. Они смогут помочь автоматизировать принятие решений, улучшить эффективность и снизить издержки.
6. Генерация контента. Нейросети можно использовать для генерации различных видов контента, таких как тексты, изображения, музыка и видео. Это может быть полезно при создании персонализированных рекламных материалов, создании контента для веб сайтов, социальных сетей и т.п.

Цель работы – продемонстрировать возможность создания программно-аппаратного комплекса на основе отладочной платы Alinx AXUCGA для обнаружения дефектов в процессе травления печатных плат и дефектов при производстве полиграфической продукции в типографиях. А также исследованию его возможностей и ограничений.

Задачи работы – машинного зрения для распознавания дефектов печатных плат и дефектов при печати полиграфической продукции. Исследовать возможности разработанной системы. Рассмотреть возможности её дальнейшего совершенствования.

1 ОБЗОР СУЩЕСТВУЮЩИХ СИСТЕМ РАЗРАБОТКИ

1.1 Компоненты систем машинного зрения

Машинное зрение – это направление современной науки в области робототехники и искусственного интеллекта, а также связанные с ним технологии получения изображений объектов реального мира, их использования и обработки полученных данных для решения разного рода прикладных задач без участия, частичного или полного, человека.

Компоненты системы машинного зрения:

– Несколько или одна аналоговая, или цифровая камера (цветная или черно-белая) с оптикой, подходящей для получения изображений.

– Программное обеспечение для работы и редактирования изображений. Для камер аналогового типа нужен оцифровщик изображений.

– CPU (современный PC с процессором, состоящим из нескольких ядер или встроенный процессор, например – ЦСП).

– Программные пакеты машинного зрения, которые предоставляют инструментарий для разработки отдельных приложений программного обеспечения.

– Сетевое оборудование и каналы связи для передачи информации о полученных результатах.

– Один вариант устройства, которое включает в себя все вышеперечисленные пункты – умная камера.

– Специальные источники света (галогенные лампы, светодиоды, люминесцентные лампы и т. д.)

– Специальное программное обеспечение для обработки изображений и обнаружения соответствующих свойств.

– Датчик для синхронизации частей обнаружения (часто оптический или магнитный датчик) для захвата и обработки изображений.

– Приводы определенной формы, используемые для сортировки или отбрасывания бракованных деталей.

Классы задач машинного зрения:

– распознавание;

– идентификация;

– обнаружение;

– распознавание текста;

– восстановление 3D формы по 2D изображениям;

– оценка движения;

– восстановление сцены;

– восстановление изображений;

– выделение на изображениях структур определенного вида, сегментация изображений;

– анализ оптического потока

Мы сконцентрируемся на первом классе задач, а именно распознавании. Данный класс задач можно разделить на несколько типов [1]:

Идентификация:

Распознается индивидуальный экземпляр объекта, принадлежащего к какому-либо классу.

Примеры: идентификация определённого человеческого лица или отпечатка пальца, или автомобиля.

Обнаружение:

Обнаружение, основанное на относительно простых и быстрых вычислениях иногда используется для нахождения небольших участков в анализируемом изображении, которые затем анализируются с помощью приемов, более требовательных к ресурсам, для получения правильной интерпретации.

Распознавание текста:

Поиск изображений по содержанию: нахождение всех изображений в большом наборе изображений, которые имеют определенное различными путями

содержание. Оценка положения: определение положения или ориентации определенного объекта относительно камеры.

Оптическое распознавание знаков:

распознавание символов на изображениях печатного или рукописного текста (обычно для перевода в текстовый формат, наиболее удобный для редактирования или индексации. Например, ASCII).

Рассмотрим необходимые вычислительные мощности для решения задач идентификации паттернов и объектов. Классически, для решения этой задачи, необходимо иметь достаточно мощный и, желательно, многоядерный процессор. Во всех задачах распознавания, основным элементом программного обеспечения, традиционно является библиотека OpenCV [2]. В этом смысле OpenCV фактически стал отраслевым стандартом в области машинного зрения. Минимальные системные требования к процессору и памяти для OpenCV, это процессор x86 или ARM с частотой ~1Ггц и объем оперативной памяти порядка 1 Гбайт. В промышленных системах используется практически всегда Linux, как базовая операционная система.

Использование SoC (System on Crystal), т.е. гетерогенных систем, в которых объединены модули процессора и плис в одном кристалле, дает дополнительные преимущества для ускорения работы системы распознавания.

Вторым основным элементом системы машинного зрения является блок распознавания подготовленных библиотекой OpenCV изображений или видеопотока. Он может быть создан различными способами. Это может быть некий (реализованный на C++) алгоритм (в случае, если он нам известен и дает хорошие результаты при тестировании). Это может быть предварительно натренированная нейросеть, часть которой, или она вся прошита в блок FPGA(ПЛИС) SoC. Или просто моделируется программой в компьютере. В случае аппаратной прошивки данная сеть становится, фактически, hardware блоком, с которым CPU общается с использованием аппаратных запросов [3]. Рассмотрим простой пример такой реализации [4].

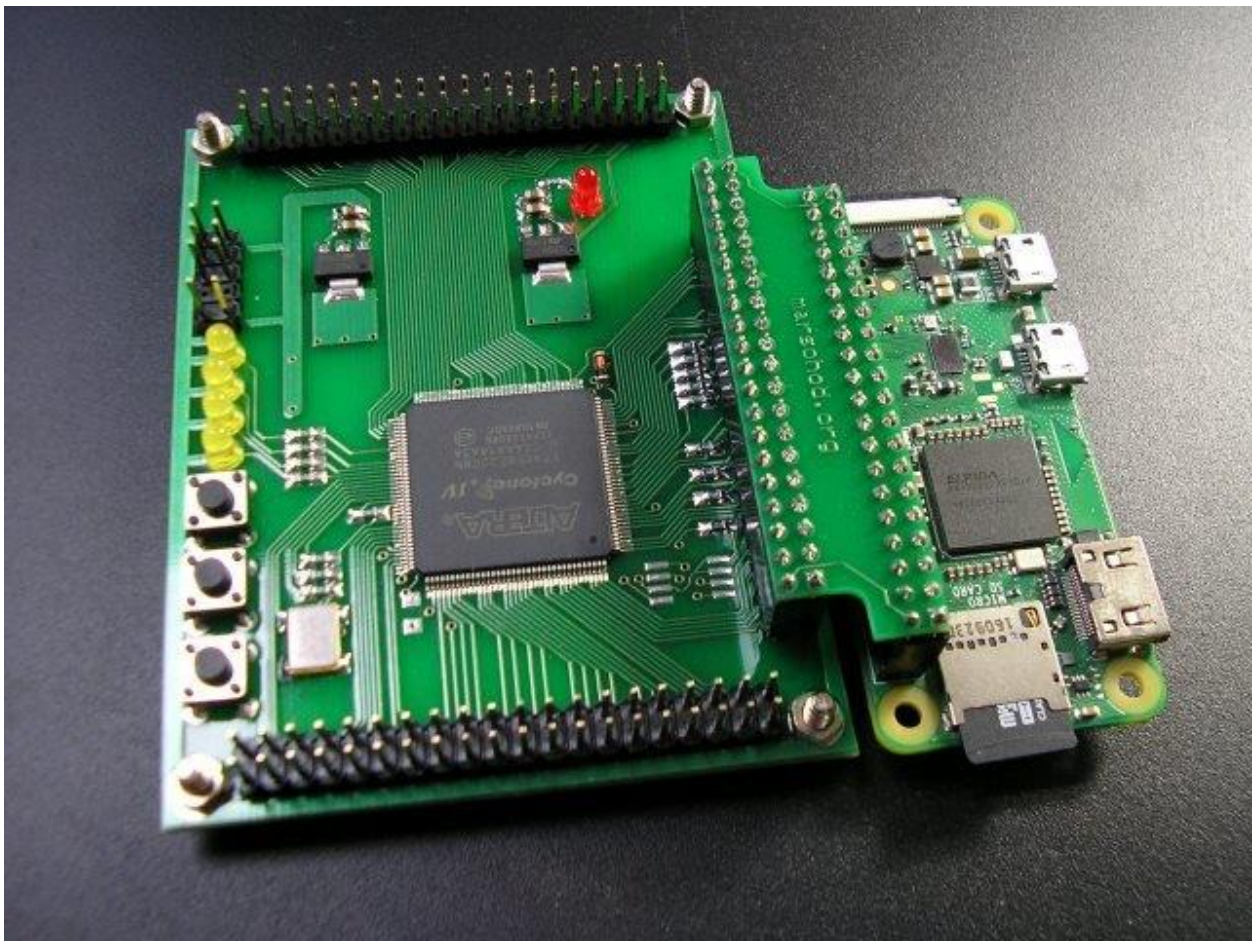

Рисунок 1.1 – Внешний вид модулей Marsohod + Raspberry Zero

На рисунке 1.1 мы можем увидеть два электронных устройства, объединенных вместе через переходник. Справа видим широко известный микрокомпьютер Raspberry. В данной конструкции он выполняет функции центрального процессорного устройства. Слева видим плату Марсоход, которая производится в Таганроге фирмой «Инпро плюс» [5],[6].

Такая комбинация интересна возможностью размещения на чипе FPGA некоторой специализированной цифровой схемы, которая способна выполнять какие-то специализированные операции, со скоростью намного превышающей скорость работы центрального процессора скажем той же Raspberry (Малинки). Если посмотреть на скорость обработки векторных и тензорных операций (что как раз необходимо для эффективной работы нейросети), то она может превышать скорость обработки центральным процессором в сотни и тысячи раз. Все зависит от возможностей распараллеливания алгоритма и объема FPGA. Ровно такая же

аппаратная комбинация имеется на кристаллах Soc типа Xilinx Zynq (Ultrascale+) или Cyclone V.

Гетерогенной системой в микроэлектронике называется сочетание на одном кристалле или плате нескольких разнородных компонентов, выполняющих разные вычислительные задачи. Это может быть CPU + GPU или CPU + FPGA или сочетания всех трех компонентов. Сейчас уже стало нормой, что современные процессоры имеют видеоядро. Это позволяет экономить на дискретной видеокарте. Рассмотрим эволюцию гетерогенных систем во времени. В дальнейшем нас будет интересовать связка CPU + FPGA.

На английском, данная связка называется SoC (System on Chip).

В 1984 году была основана знаменитая американская компания Xilinx.

В 1985 году инженер этой компании Росс Фриман разработал совершенно новый класс микросхем. Микросхемы FPGA. Первенцем стал чип Xilinx XC2064, имевший 1000 вентилей [7].

Эта же компания создала в 2011 году первый SoC – Zynq-7000.

Zynq имеет на борту процессор ARM A9, программируемую логику и некоторую периферию.

В 2013 году свою SoC Cyclone V SE выпустила компания Altera. Чипы были похожи, но у Altera чип работал на более высокой частоте и имел более развитую периферию.

В 2015 году компанию Altera приобрела компания Intel, заплатив за нее 16,7 миллиардов долларов. Данное слияние привело к созданию в 2016 году гетерогенного процессора Intel – комбинации на одной подложке серверного процессора Xeon E5-2600 v4 и FPGA Altera Arria 10 [8].

Общую конструкцию системы на кристалле можно увидеть на рисунке 1.2.

(Сделано в пакете BPWIN [9],[10]).

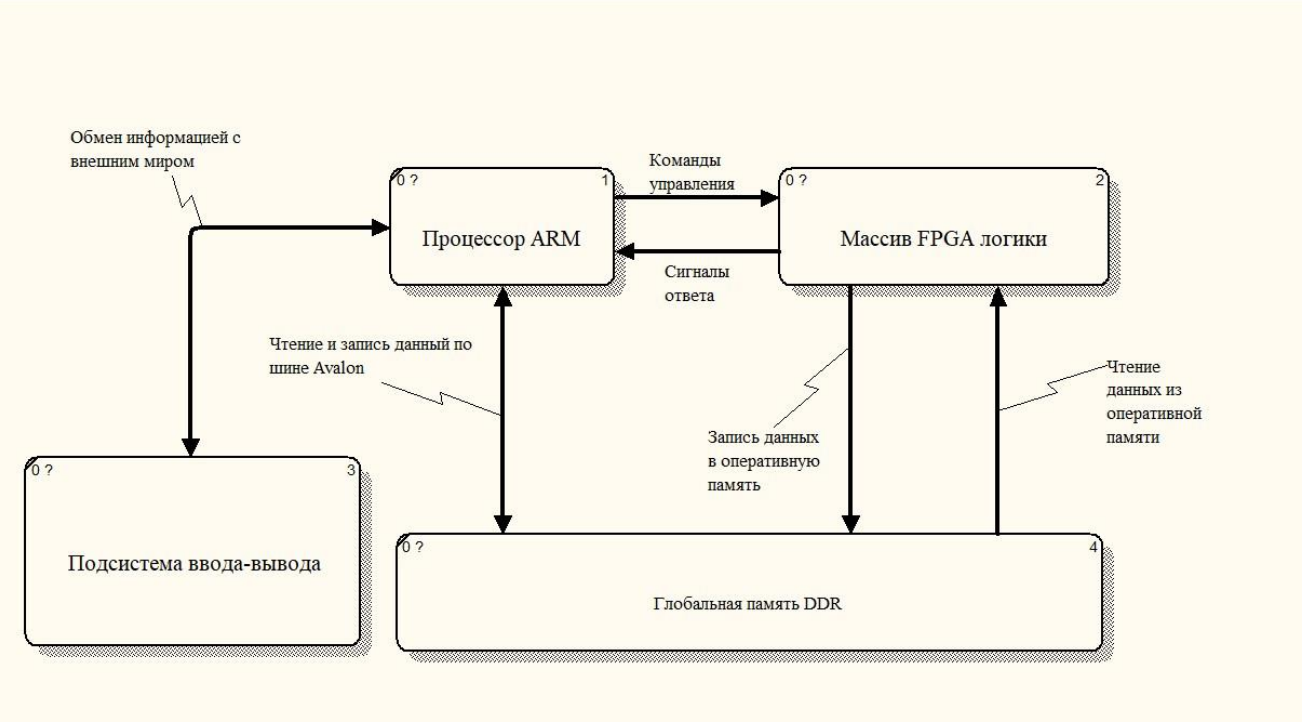

Рисунок 1.2 – Общая структура SoC (Система на Кристалле)

Рассмотрим составляющие SoC и их взаимодействие в процессе работы.

Компоненты:

1) Процессор ARM. Связан с подсистемой ввода-вывода, с внешней памятью может общаться напрямую или через массив FPGA логики. Внешняя память в кристалл SoC не входит и на фото приведена для прояснения принципов работы системы.
2) Массив FPGA логики. Напрямую связан с процессором. Может адаптивно перепрограммироваться командами процессора. Встроенная логика, при соответствующей прошивке, может выполнять любую роль, заданную при программировании. Например, роль математического сопроцессора, на который можно возложить выполнение тяжелых или сложных для процессора операций. Матричное или тензорное умножение. Нейросетевые вычисления.
3) Подсистема ввода-вывода. Служит для общения процессора с внешним миром по различным каналам связи. UART, CAN, параллельная шина, и т.п. [11],[12],[13]

Нейросеть может быть описана обобщенной идеализированной математической моделью [14], [15]:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j, \quad (1)$$

где u_k - выходной вектор;

$w_{kj} = \{w_{k1}, w_{k2}, w_{k3}, \ldots . w_{km}\}$ – веса синаптических связей нейрона k;

x_j – вектор входных сигналов, подаваемых на нейрон k.

Полученная функция u_k входит в функцию:

$$y_k = F(u_k + A_k), \quad (2)$$

где A_k – порог возбуждения нейрона;

y_k – выходной вектор нейросети.

В результате имеем систему линейных уравнений, коэффициенты которых w_{kj} настраиваются определенным образом в процессе обучения сети для прохождения тестового вектора с высоким процентом совпадения по результату.

Таким образом, в самом простом варианте, в FPGA размещается схема типа DSP вычислителя, которая на входной вектор x_j будет давать выходной вектор y_k.

Сверточная нейронная сеть (CNN) – это сеть глубокого обучения, который может принимать входное изображение, присваивать важность (усваиваемые веса и смещения) различным областям и\или объектам в изображении и может отличать одно от другого. Предварительной обработки в CNN требуется значительно меньше, по сравнению с другими алгоритмами классификации.

Тогда как в примитивных методах фильтры сконструированы вручную, CNN, при достаточном обучении, способны сами изучать эти фильтры/характеристики.

Архитектура CNN подобна архитектуре связности нейронов человеческом мозге и была вдохновлена организацией зрительной коры. Отдельные нейроны реагируют на раздражители только в ограниченной зоне поля зрения, называемой рецептивным полем. Совокупность таких полей накладывается, чтобы покрыть всю зону поля зрения [16].

CNN умеет захватывать пространственно-временные закономерности на изображении с помощью соответствующих фильтров. И соответственно постепенно начинает понимать сложность изображения [17].

Алгоритм работы сверточной нейросети (CNN)

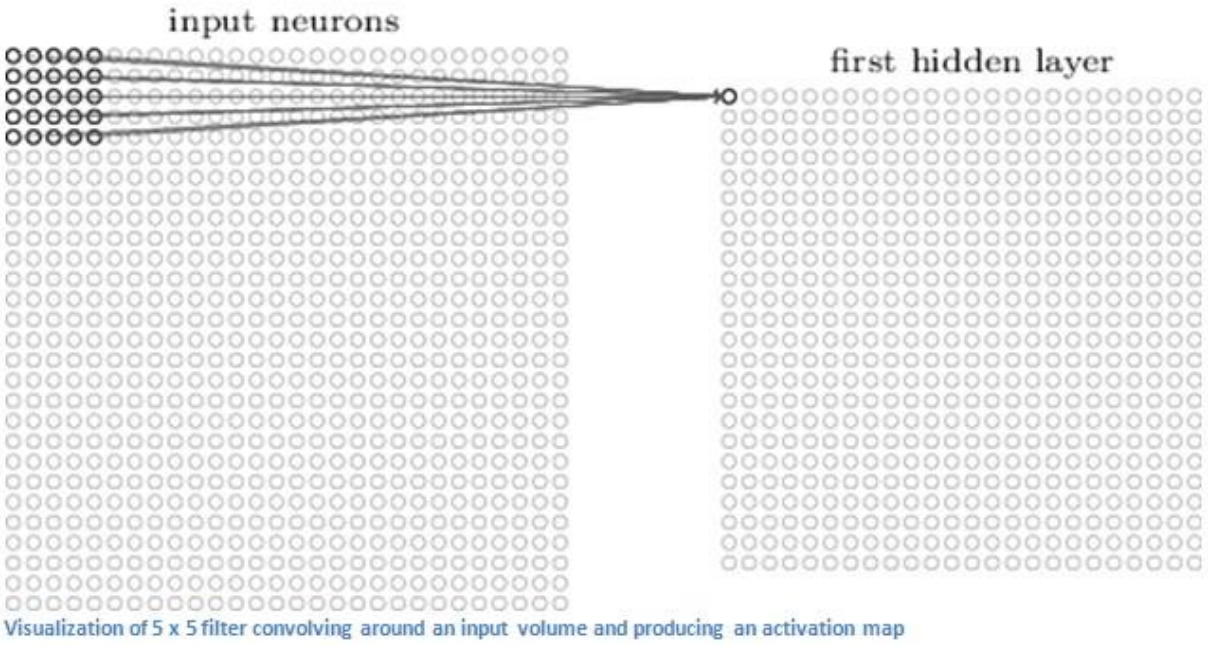

Рисунок 1.3 – Иллюстрация сверточного преобразования

1. Ядро свертки

Понятие ядра свертки [18]. Ядро свертки, это некий фильтр, который мы накладываем на исходную матрицу изображения, с целью получить изображение меньшего размера. Фактически, примененная к изображению один или несколько раз свертка, позволяет усилить и выделить на изображении определенные признаки. На рисунке 1.3 представлена схема сверточного преобразования.

2. Ядро пулинга (объединяющее ядро)

Пулинговое ядро выполняет роль уменьшения размера изображения для его дальнейшего анализа [19]. Это важно для извлечения доминирующих признаков, инвариантных относительно вращения и позиционирования, что поддерживает процесс эффективного обучения модели (рисунок 1.4).

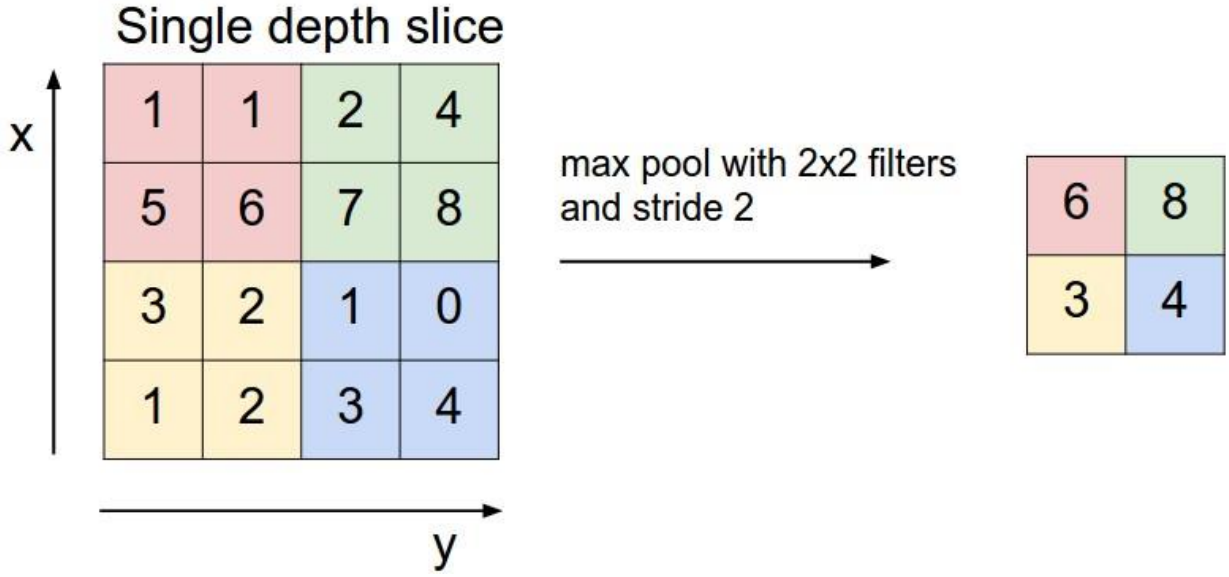

Рисунок 1.4 – Иллюстрация механизма пулинга (pooling)

3. Нормализующее ядро (LRN)

Корректирует получившуюся матрицу изображения, для того чтобы значения величины каждого пиксела не выходили за пределы определенных значений. Математический смысл нормализации:

$$X_i' = \frac{PX_i}{\sum X_i}, \tag{3}$$

где X_i – значение пиксела i до нормализации;

P – разрядность в битах изображения, которое нужно получить;

X_i' – получаемое после нормализации значение i-го пиксела.

Есть более сложные типы нормализации, но здесь мы не будем на них акцентироваться.

4. Аппаратный блок, реализующий алгоритм

Блок-схема, реализующая вышеописанный алгоритм, показана на рисунке 1.5. Когда, в выделенную для буфера хранения изображения область памяти процессор кладет матрицу изображения, он подает команду на начало преобразования ускорителя DCNN [20]. Следующим шагом Блок чтения из памяти считывает матрицу из памяти и передает Ядру свертки. Ядро свертки представляет из себя тензорный преобразователь, настроенный как натренированная нейросеть на выделение определенных признаков на изображении. Он может быть перепрограммирован на другие признаки распознавания, при перезагрузке системы. После выполнения необходимых действий, Ядро свертки передает полученную матрицу Объединяющему ядру (Pooling). Это ядро выполняет пост обработку получившейся матрицы.

Результат работы Объединяющего ядра записывается в глобальную память Блоком записи в память. Последнюю стадию обработки выполняет Ядро нормализации.

Рабочий цикл создания системы можно разделить на несколько этапов.

1 этап – проектирование нейросети. На этом этапе мы должны определить архитектуру нейросети с которой будем работать. На основании этой архитектуры мы спроектируем блок схему преобразователя, которая будет залита в виде прошивки в FPGA.

2 этап – тренировка нейросети. На этом этапе мы первоначально инициализируем матрицу весов w_{kj} случайными числами по определенному алгоритму и начинаем циклы тренировки нейросети

3 этап – эксплуатация уже настроенной и обученной нейросети для практических задач.

Этапы разработки и эксплуатации делаются на разных аппаратных конфигурациях.

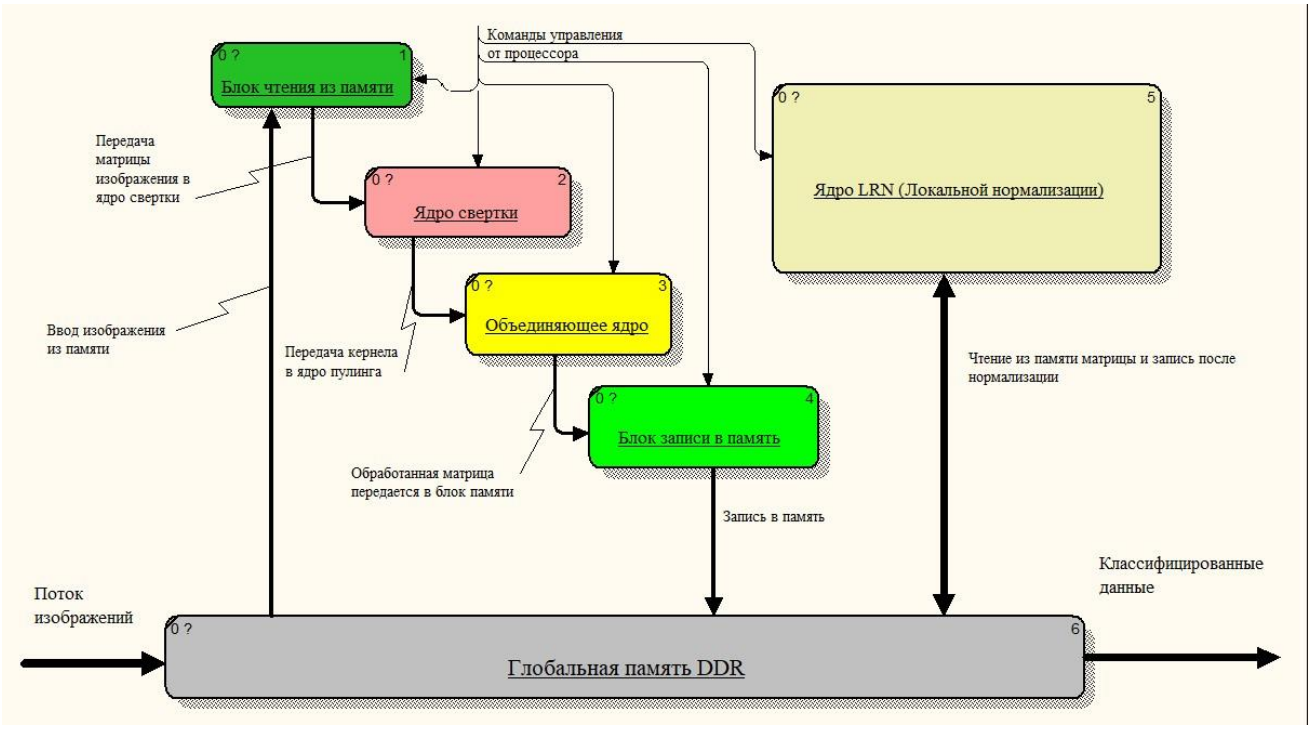

Рисунок 1.5 – Ускоритель DCNN

Создание и тренировка нейросети осуществляется обычном компьютере с ускорителем GPU типа Tesla и с использованием фреймворка Tensorflow или Keras. Это позволяет сделать пред-обученный вариант нейросети и загружать её в FPGA непосредственно только для эксплуатации.

В графическом виде путь реализации нейросети в FPGA показан на рисунке 1.6 [21].

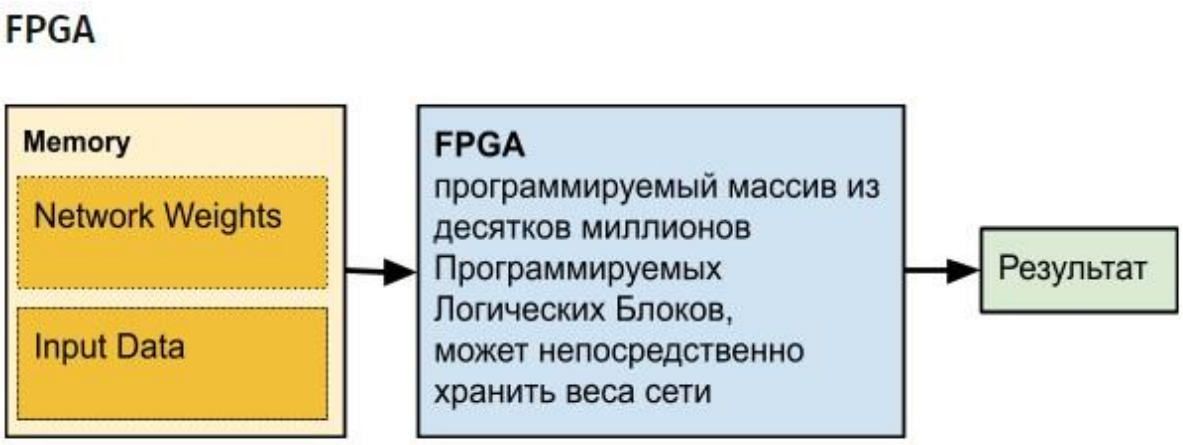

Рисунок 1.6 – Маршрут реализации нейросети

Схема реализации нейросети на Soc показан на рисунке 1.7.

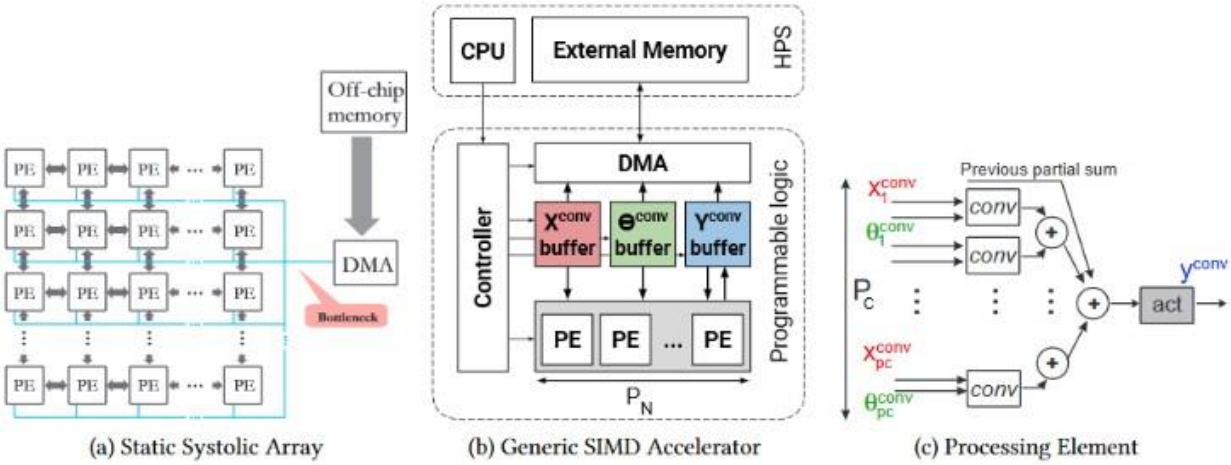

Figure 4: Generic Data-paths of FPGA-based CNN accelerators

Рисунок 1.7 – Иллюстрация блок-схемы реализации ускорителя на Soc

На сегодняшний день существует большое количество фреймворков (IDE, систем разработки) для проектирования и создания систем машинного зрения, с использованием нейросетевых технологий. Проведем сравнительный анализ таких фреймворков. Основными игроками этого рынка являются как давно известные компании типа nVidia, Xilinx, Intel, так и новички, которые появились совсем недавно, но уже хорошо себя зарекомендовали в области распознавания образов с той или иной стороны. Например, в 2019 году фирма Canaan Inc. выпустила на рынок новый процессор Kendryte K210 [22], который имеет в своем составе аппаратные команды работы с нейросетями. Рассматривать будем именно комплексные аппаратно-программные блоки, предназначенные для сквозного проектирования готового прототипа для испытаний и дальнейшего использования его как основы для серийного производства.

1.2 Отладочные платы и фреймворки для разработки nVidia

nVidia выпускает широкий спектр аппаратных решений для разработки и промышленного использования систем машинного зрения [23].

Примеры:

а) GPU nVidia известна своими графическими процессорами (GPU), которые являются ключевым компонентом для обработки больший объемов информации в системах машинного зрения. GPU nVidia типа Kepler, Pascal. Turing, Volta, Amper (перечислены по мере увеличения производительности и годов выпуска), обладают высокой вычислительной мощностью и параллельной архитектурой. Это делает их отличным вариантом для построения высокомощных вычислительных кластеров, способных выполнять очень трудоемкие вычислительные задачи. В контексте нашего исследования они отлично приспособлены для проектирования и тренировки нейросетей, содержащих десятки и сотни тысяч нейронов.

б) CUDA (Compute Unified Device Architecture) – архитектура и набор инструментов разработки, предоставляемых nVidia, которые позволяют разработчикам эффективно использовать GPU для выполнения общего назначения вычислений. CUDA обеспечивает прямой доступ к вычислительным мощностям GPU и позволяет разрабатывать и оптимизировать параллельные алгоритмы машинного зрения [24].

в) NVIDIA Deep Learning SDK – набор инструментов и библиотек, предназначенных для разработки и развертывания глубоких нейронных сетей на аппаратных платформах nVidia. Он включает в себя такие компоненты, как библиотеки cuDNN(CUDA Deep Neural Network), которые предоставляют оптимизированные алгоритмы для выполнения операций глубокого обучения, а также TensorRT, который обеспечивает оптимизацию и ускорение развертывания обученных моделей глубокого обучения.

г) NVIDIA TensorRT – оптимизационный и разверточный фреймворк, разработанный nVidia для ускорения инференса (выполнения) моделей глубокого

обучения. Он использует оптимизации на уровне алгоритма и аппаратной архитектуры, чтобы обеспечить максимальную производительность при выполнении моделей на GPU. TensorRT поддерживает различные форматы моделей и позволяет оптимизировать их для развертывания на различных платформах [25].

д) NVIDIA Jetson – платформа для разработки и развертывания систем и приложений компьютерного зрения на периферийные устройства, такие как автономные роботы, дроны и встраиваемые системы. Jetson основан на энергоэффективных GPU nVidia и предлагает различные модели, такие как Jetson Nano, Jetson Xavier, Jetson ASX, Jetson TX2, с разной вычислительной мощностью и функциональностью. Эти платформы обладают достаточной производительностью для выполнения задач машинного зрения в реальном времени на устройствах с ограниченными ресурсами.

е) NVIDIA Clara – платформа и инструменты для разработки и развертывания медицинских приложений машинного обучения и компьютерного зрения. Clara предлагает набор библиотек и инструментов, специально разработанных для медицинских изображений и анализа данных, таких как сегментация, классификация и обнаружение объектов. Она позволяет медицинским исследователям и разработчикам создавать точные автоматизированные системы диагностики и обработки медицинских изображений.

Для наших целей перспективными представляются отладочные платы серии Jetson. Рассмотрим их более подробно.

Приведем сравнительный анализ пяти основных параметров отладочных плат nVidia Jetson: Jetson Nano [26] (рисунок 1.8), Jetson TX2 [27] (рисунок 1.9), Jetson Xavier NX [28], Jetson AGX Xavier [29], Jetson Xavier NX Developer Kit[30].

а) Производительность и вычислительная мощность:

– Jetson Nano (рисунок 1.8): Оснащен 64-битным четырехъядерным процессором ARM Cortex-A57 и графическим процессором NVIDIA Maxwell со 128 ядрами CUDA. Обеспечивает производительность до 472 GFLOPS.

– Jetson TX2 (рисунок 1.9): имеет 64-битный восьмиядерный процессор ARM Cortex-A57 и графический процессор NVIDIA Pascal со 256 ядрами CUDA. Обеспечивает производительность до 1,3 TFLOPS.

– Jetson Xavier NX: включает 64-битный шестиядерный процессор ARMv8.2 Carmel и графический процессор NVIDIA Volta со 384 ядрами CUDA. Обеспечивает производительность до 21 TFLOPS.

– Jetson AGX Xavier: Оснащен 64-битным восьмиядерным процессором ARMv8.2 Carmel и графическим процессором NVIDIA Volta со 512 ядрами CUDA. Обеспечивает производительность до 32 TFLOPS.

б) Память:

– Jetson Nano (рисунок 1.8): имеет 4 ГБ LPDDR4 RAM и 16 ГБ eMMC хранилище.

– Jetson TX2 (рисунок 1.9): предлагает 8 ГБ LPDDR4 RAM и 32 ГБ eMMC хранилище.

– Jetson Xavier NX: включает 8 ГБ LPDDR4 RAM и 16 ГБ eMMC хранилище.

– Jetson AGX Xavier: обладает 16 ГБ LPDDR4 RAM и 32 ГБ eMMC хранилищем.

в) Нейросетевые возможности:

– Jetson Nano (рисунок 1.8): поддерживает различные фреймворки глубокого обучения, такие как TensorFlow, PyTorch и Caffe.

– Jetson TX2 (рисунок 1.9): обладает той же нейросетевой функциональностью, что и Jetson Nano, но с более высокой производительностью и памятью.

– Jetson Xavier NX: предоставляет более мощные вычислительные возможности для глубокого обучения, поддерживая также TensorRT и NVIDIA DeepStream.

– Jetson AGX Xavier: обеспечивает самые высокие возможности глубокого обучения с поддержкой больших моделей нейронных сетей и многочисленных видеопотоков.

г) Видео и графика:

– Jetson Nano (рисунок 1.8): поддерживает декодирование и кодирование видео разрешением до 4K.

Рисунок 1.8 – Внешний вид Jetson Nano

– Jetson TX2 (рисунок 1.9): обладает аналогичными возможностями по декодированию и кодированию видео.

– Jetson Xavier NX: обеспечивает поддержку декодирования и кодирования видео в разрешении до 8K.

– Jetson AGX Xavier: обладает аналогичными возможностями в видео, что и Jetson Xavier NX, но с более высокой производительностью и способностью обрабатывать несколько видеопотоков одновременно.

д) Интерфейсы и порты:

– Jetson Nano (рисунок 1.8): имеет HDMI, USB 2.0, USB 3.0, Gigabit Ethernet, MIPI CSI для подключения камеры и GPIO для взаимодействия с внешними устройствами.

– Jetson TX2(рисунок 1.9): предлагает аналогичные порты и интерфейсы, что и Jetson Nano, но с дополнительными возможностями, такими как PCI Express и SATA.

– Jetson Xavier NX: поддерживает HDMI, USB 3.1, USB 2.0, Gigabit Ethernet, MIPI CSI, PCI Express и GPIO.

– Jetson AGX Xavier: обладает широким набором портов и интерфейсов, включая HDMI, USB 3.1, USB 2.0, Gigabit Ethernet, MIPI CSI, PCI Express, SATA, CAN и GPIO.

е) Цены устройств (мин, макс)

– Jetson Nano (рисунок 1.8):

Минимальная цена: около 99 евро. Максимальная цена: около 149 евро.

– Jetson TX2(рисунок 1.9):

Минимальная цена: около 399 евро. Максимальная цена: около 599 евро.

– Jetson Xavier NX:

Минимальная цена: около 499 евро. Максимальная цена: около 699 евро.

– Jetson AGX Xavier:

Минимальная цена: около 799 евро. Максимальная цена: около 1,099 евро.

Сводная таблица параметров отладочных плат серии Jetson приведена ниже (таблица 1.1):

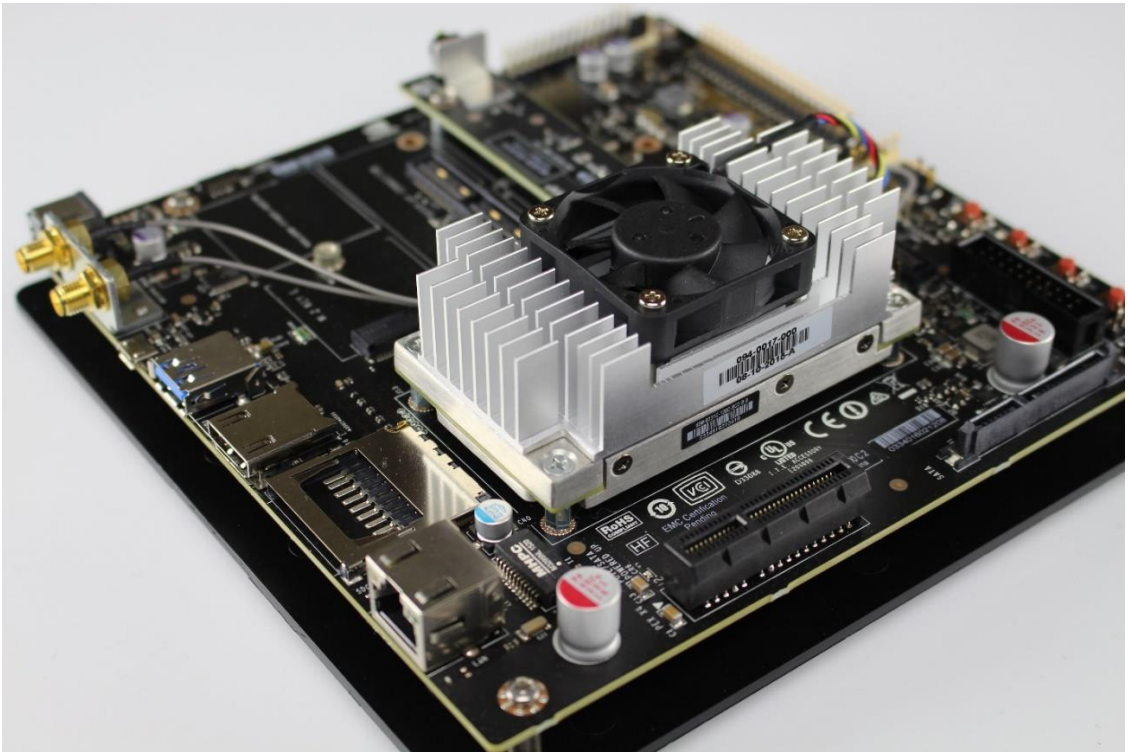

Рисунок 1.9 – Внешний вид Jetson TX2

Таблица 1.1 – Параметры отладочных плат серии Jetson

Название платы	Jetson nano	Jetson TX2	Jetson Xavier NX	Jetson AGX Xavier
Производительность (TFLOPS)	0.472	1.3	21	32
Память (RAM/eMMC)(Gbyte)	4/16	8/32	8/16	16/32
Нейросетевые возможности	TensorFlow, PyTorch и Caffe	TensorFlow, PyTorch и Caffe	То же + TensorRT и NVIDIA DeepStream	То же + TensorRT и NVIDIA DeepStream
Видео и графика	4K.	4K.	8K.	То же. Более производительна
Порты и интерфейсы	HDMI, USB 3.0, USB 2.0, Gigabit Ethernet, MIPI CSI и GPIO.	То же + PCI-e, SATA	HDMI,USB 3.1,USB 2.0,Gigabit Ethernet, MIPI CSI, PCI Express и GPIO.	HDMI, USB 3.1, USB 2.0, Gigabit Ethernet, MIPI CSI, PCI Express, SATA, CAN и GPIO
Цена мин. (евро)	99	399	499	799
Цена макс.(евро)	149	599	699	1099

1.3 Отладочные платы и фреймворки для разработки Xilinx

Варианты плат для разработок систем машинного зрения на чипах Xilinx:

а) Xilinx Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit (рисунок 1.10).

Состав платы [31]:

Микросхема: Xilinx Zynq UltraScale+ XCZU7EV-2FFVC1156 MPSoC

Логических ячеек - 504K

Память на чипе - 38Мб

DSP блоков - 1728

Аппаратный видео кодек - 1 (H.264/H.265)

ЦПУ: Quad-core ARM Cortex-A53 MPCore и Dual-core ARM Cortex-R5 MPCore

Графический процессор: Mali-400 MP2

Память: 4 ГБ DDR4 SDRAM

Хранилище данных: 8 ГБ eMMC Flash, 16 МБ QSPI Flash

Видеоинтерфейсы: HDMI вход и выход, DisplayPort

Сетевые интерфейсы: 10/100/1000 Ethernet

Разъемы: USB 3.0, USB 2.0, JTAG, SD

Слоты расширения: 1 слот PCI Express Gen2 x4, 1 слот M.2

Инструментарий разработки: Vivado и SDK, Vitis, Vitis-AI.

Это отладочная плата, основанная на микросхеме Zynq UltraScale+ MPSoC от Xilinx. Она предназначена для разработки и прототипирования систем машинного зрения. Плата обладает мощными вычислительными возможностями благодаря сочетанию программной и аппаратной обработки сигналов.

Включает в себя различные интерфейсы для подключения камер, дисплеев и других периферийных устройств.

Имеет поддержку инструментария Vivado и SDK для разработки программного обеспечения.

Цена: 1678 долларов США.

Рисунок 1.10 – Внешний вид платы ZCU104 Evaluation Kit

б) Xilinx Zynq-7000 All Programmable SoC ZC702 Evaluation Kit (рисунок 1.11) [32]:

Микросхема: Xilinx Zynq-7000 SoC XC7Z020-CLG484-1

ЦПУ: Dual-core ARM Cortex-A9 MPCore

Графический процессор: не применимо

Память: 1 ГБ DDR3 SDRAM

Хранилище данных: 4 ГБ eMMC Flash, 128 МБ Quad-SPI Flash

Видеоинтерфейсы: HDMI вход и выход, VGA

Сетевые интерфейсы: 10/100/1000 Ethernet

Разъемы: USB 2.0, JTAG, SD

Слоты расширения: 1 слот FMC, 1 слот PMOD

Инструментарий разработки: Vivado

Рисунок 1.11 – Внешний вид платы ZC702 Evaluation Kit

Это отладочная плата, основанная на микросхеме Zynq-7000 SoC от Xilinx. Она предназначена для разработки и тестирования систем машинного зрения, включая распознавание образов.

Плата обладает мощными вычислительными возможностями и программируемой логикой FPGA для ускорения обработки данных.

Включает в себя интерфейсы для подключения камер, дисплеев, Ethernet и других периферийных устройств. Поставляется с интегрированной средой разработки Vivado для создания программного обеспечения.

Цена: 1160 долларов США.

в) Xilinx Alveo Data Center Accelerator Cards [33]:

Это серия акселераторов для обработки данных в центрах обработки данных и облачных средах. Карты Alveo обладают высокой производительностью и энергоэффективностью благодаря использованию FPGA. Они могут быть использованы для различных задач машинного зрения и обработки образов,

включая нейронные сети. Карты Alveo поставляются с программным обеспечением Vitis для разработки и оптимизации приложений на FPGA.

В следствии высокой цены и малой доступности в России, мы исключили эти платы из обзора, как малоперспективные для наших целей. Тем не менее приведем общие характеристики серии:

Архитектура FPGA:

Поддержка FPGA-матрицы от Xilinx с высокой производительностью и гибкостью. Различные размеры FPGA-матрицы, такие как UltraScale, UltraScale+, Versal.

Память:

Встроенная память DDR для хранения данных и инструкций.

Высокоскоростная память HBM (High Bandwidth Memory) для ускорения работы с данными.

Интерфейсы:

Высокоскоростные интерфейсы PCIe (Peripheral Component Interconnect Express) для подключения к хост-системе. Ethernet-интерфейсы для сетевого взаимодействия. Возможность подключения камер и других периферийных устройств через различные интерфейсы.

Вычислительная мощность:

Высокая производительность и параллельная обработка благодаря использованию FPGA-технологии. Возможность выполнения вычислений с низкой задержкой и высокой энергоэффективностью.

Программное обеспечение и разработка:

Инструментарий разработки Vitis для разработки и оптимизации приложений на FPGA.

Поддержка популярных фреймворков машинного обучения и глубокого обучения, таких как TensorFlow, PyTorch, Caffe и других.

Различные библиотеки и инструменты для упрощения разработки и оптимизации алгоритмов и моделей.

На сегодняшний день, кроме отладочных плат, продаваемых под маркой Xilinx, существует много отладочных плат третьих производителей, которые делают их с использованием чипов Xilinx. Они стоят существенно дешевле официальных плат Xilinx, а по показателям часто не уступают и даже существенно обходят официальные платы этой фирмы. Для примера можно привести:

г) Alinx AXU9EG. Xilinx Zynq UltraScale+ MPSoc XCZU9EG FPGA Development board High-end Integrated Deve [34] (рисунок 1.12).

Состав:

– FPGA: Zynq UltraScale+ MPSOC, XCZU9EG-2FFVB1156I

– ARM Cortex™-A53 x4, Cortex-R5 x2, Mali™-400MP2 GPU

– PS 4GB DDR4, 64bit, PL 2GB DDR4, 32bit

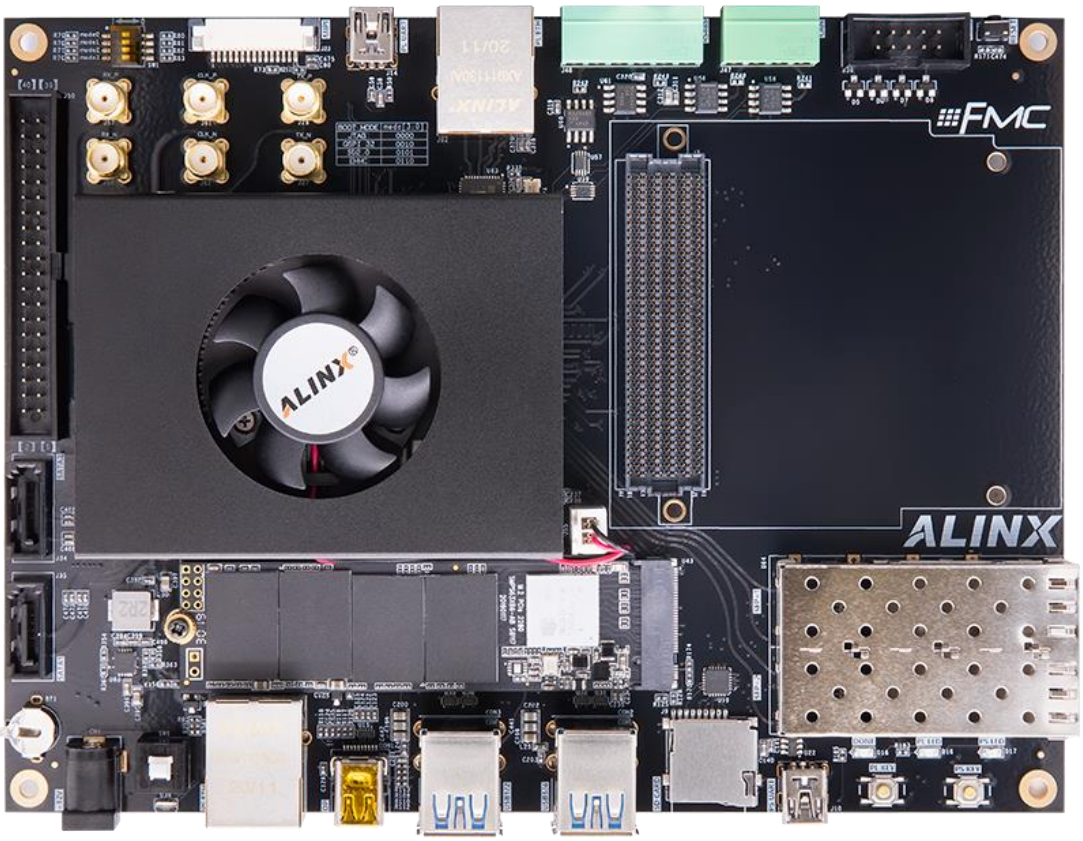

Рисунок 1.12 - Внешний вид платы Alinx AXU9EG

– 8GB eMMC FLASH, 64MB FLASH

– Standard FMC HPC Interface, Connect various FMC boards

Цена: 2280 долларов США.

д) Alinx AXU2CGB. Xilinx Zynq UltraScale+ MPSOC XCZU2CG FPGA Development board AI Artificial Intelligen [35] (рисунок 1.13).

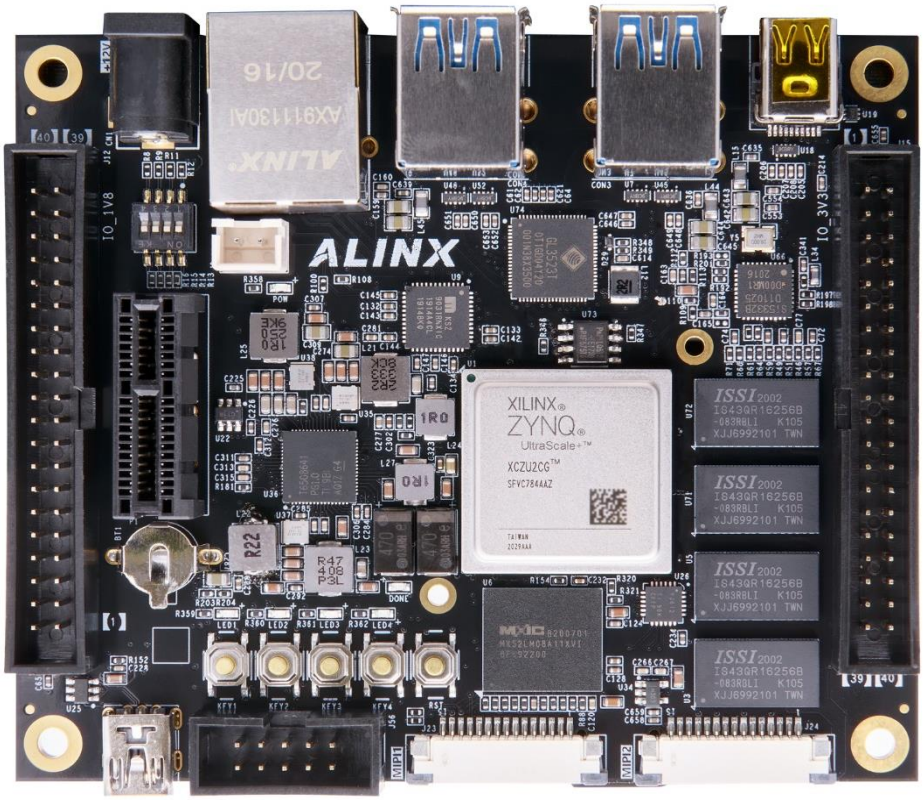

Рисунок 1.13 - Внешний вид платы Alinx AXU2CGB

Состав:

– FPGA: Xilinx Zynq UltraScale+ MPSoCs, XCZU2CG-1SFVC784E

– ARM Cortex™-A53 x2 1.2Ghz, Cortex-R5 x2 500Mhz

– 2GByte DDR4, 32bit, Data Speed 2400Mbps for PS Side

– 1GB eMMC, 256Mb QSPI FLASH

– Mini DP interface, Supports 4K x 2K@30Fps Output

– 4*USB 3.0 Host, Speed up to 5.0 Gb/s

– 1*Gigabit Ethernet Interface

Цена: 380 долларов США.

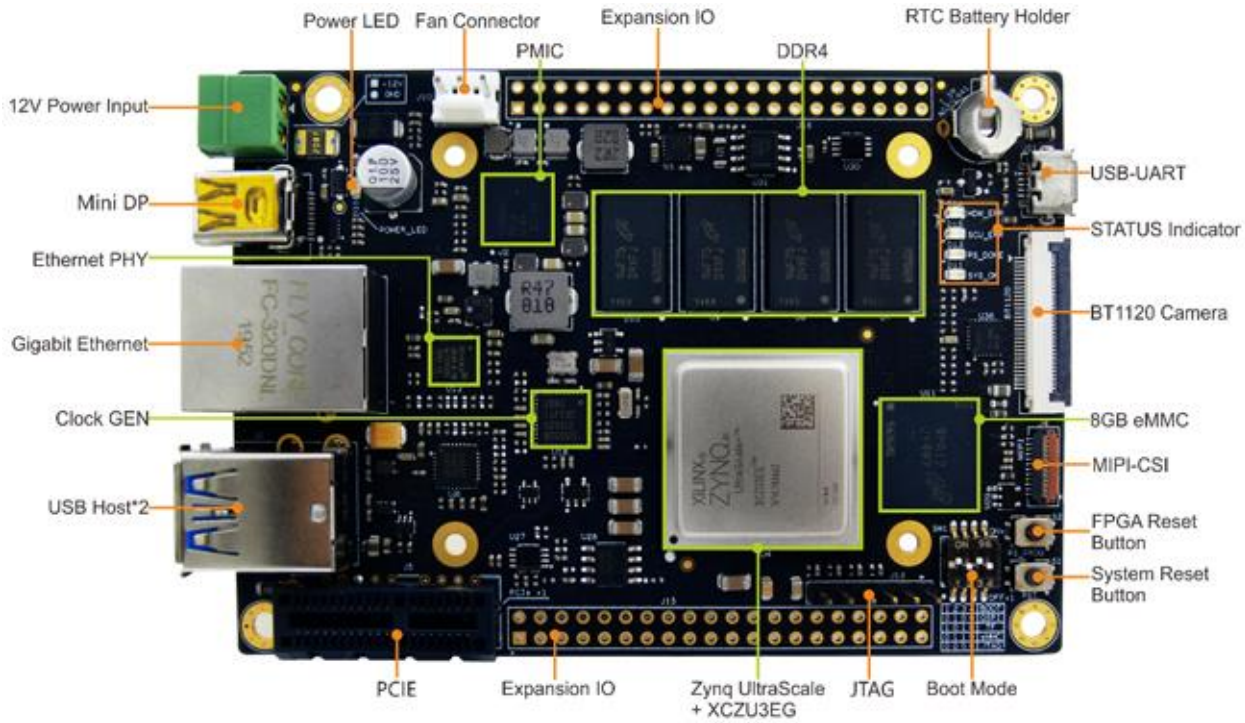

Рисунок 1.14 - Внешний вид платы MYIR MYS-ZU3EG-8E4D-EDGE-K2

e) MYIR MYS-ZU3EG-8E4D-EDGE-K2 [36] (рисунок 1.14).

Состав:

– Xilinx Zynq UltraScale+ ZU3EG MPSoC based on 1.2 GHz Quad Arm Cortex-A53 and 600MHz Dual Cortex-M4 Cores

– 4GB DDR4 SDRAM (64-bit, 2400MHz)

– 8GB eMMC Flash, 32MB QSPI Flash, 32KB EEPROM

– USB2.0, USB3.0, Gigabit Ethernet, TF, DP, PCIe, MIPI-CSI, BT1120, USB-UART, JTAG…

– Computing Power up to 1.2TOPS, MobileNet up to 100FPS

– Ready-to-Run PetaLinux and Supports Xilinx Vitis

– Supports Baidu's PaddlePaddle Deep Learning AI Framework

Цена: 449 долларов США.

Средства разработки систем машинного зрения и нейросетей:

Xilinx Vitis-AI [37]:

Это программная среда разработки для создания и оптимизации моделей и приложений машинного обучения на устройствах Xilinx.

Vitis-AI предоставляет инструменты и библиотеки для разработки и оптимизации нейросетей.

Она поддерживает различные фреймворки глубокого обучения, такие как TensorFlow и PyTorch.

Vitis-AI позволяет разработчикам создавать эффективные решения для различных сценариев машинного зрения и распознавания образов.

Сводная таблица параметров отладочных плат с чипами Xilinx приведена ниже (таблица 1.2).

Таблица 1.2 – Параметры отладочных плат Xilinx

Название платы	ZCU104	ZC702	AXU9EG	AXU2CGB	ZU3EG-8E4D
Кол-во ядер процессора	4/2R	2	4/2R	2/2R	4/2R
Графический процессор	Mali-400 MP2	нет	Mali-400 MP2	нет	нет
Память оперативная	4 Гб DDR4	1Гб DDR3	6 Гб DDR4	2Гб DDR4	4 Гб DDR4
Память eMMC	8 Гб	4 Гб	8 Гб	8 Гб	8 Гб
Нейросетевые возможности	Vitis-AI, TensorFlow	PyTorch	Vitis-AI, TensorFlow	Vitis-AI, TensorFlow	Vitis-AI, TensorFlow
Порты и интерфейсы	USB2.0, USB3.0, Gigabit Ethernet, TF, HDMI ,DP, PCIe, USB-UART, JTAG	USB2.0, Gigabit Ethernet, TF, USB-UART, JTAG, FMC, PMOD	USB2.0, USB3.0, Gigabit Ethernet, TF, DP, PCIe, MIPI-CSI, BT1120, USB-UART, JTAG	USB2.0, USB3.0, Gigabit Ethernet, TF, DP, PCIe, MIPI-CSI, BT1120, USB-UART, JTAG	USB2.0, USB3.0, Gigabit Ethernet, TF, DP, PCIe, MIPI-CSI, BT1120, USB-UART, JTAG
Цена (долл. США)	1678	1160	2280	380	499

1.4 Отладочные платы и фреймворки для разработки Altera-Intel

Далее рассмотрим системы для разработки машинного зрения и нейросетей от объединения Altera-Intel. Интел достаточно давно занимается разработкой нескольких фреймворков для создания и тренировки нейросетей. На сегодняшний день можно упомянуть, к примеру, о таких SDK как Intel Distribution of OpenVINO Toolkit [38],[39]. OpenVINO (Open Visual Inference and Neural

network Optimization) или Intel DevCloud for the Edge. Это облачный сервис, предоставляемый Intel, который позволяет разработчикам прототипировать и тестировать свои модели машинного обучения на различных аппаратных платформах Intel. DevCloud for the Edge предоставляет доступ к вычислительным ресурсам и инструментам Intel для разработки и оптимизации моделей машинного обучения для развертывания на реальных устройствах. Представляет собой оптимизированный набор инструментов и библиотек, разработанных Intel для ускорения и оптимизации инференса моделей глубокого обучения. Он поддерживает широкий спектр архитектур процессоров Intel и устройств, включая FPGA и VPU, позволяя разрабатывать эффективные приложения машинного зрения на платформах Intel.

Кроме того, Intel также предоставляет ряд фреймворков и библиотек, которые облегчают разработку и оптимизацию приложений машинного зрения:

а) Intel OpenVINO Toolkit: Упомянутый ранее инструментарий OpenVINO также включает в себя библиотеки и API для работы с моделями глубокого обучения, оптимизации инференса и развертывания на устройствах Intel.

б) Intel Math Kernel Library (MKL): MKL представляет собой библиотеку оптимизированных функций для выполнения математических операций. Она включает в себя функции для работы с векторами, матрицами, свертками и другими операциями, которые часто используются в алгоритмах машинного зрения.

в) Intel Deep Learning Deployment Toolkit (DLDT): DLDT предоставляет инструменты и библиотеки для конвертации и оптимизации моделей глубокого обучения, а также для их развертывания на устройствах Intel.

г) Intel Media SDK: это набор инструментов для аппаратного ускорения кодирования, декодирования и обработки видео и аудио. Он может быть полезен при разработке приложений машинного зрения, связанных с обработкой видео.

Компания Altera, до слияния с Intel, разрабатывала и производила достаточно большой ассортимент отладочных плат, которые позиционировались в

том числе для нейросетевых разработок и разработок машинного зрения, в частности. Рассмотрим современные платы, пригодные для этого, а также их совместимость с соответствующими фреймворками для машинного зрения и разработки нейросетей. Состав OpenVINO Toolkit представлен на рисунке 1.15.

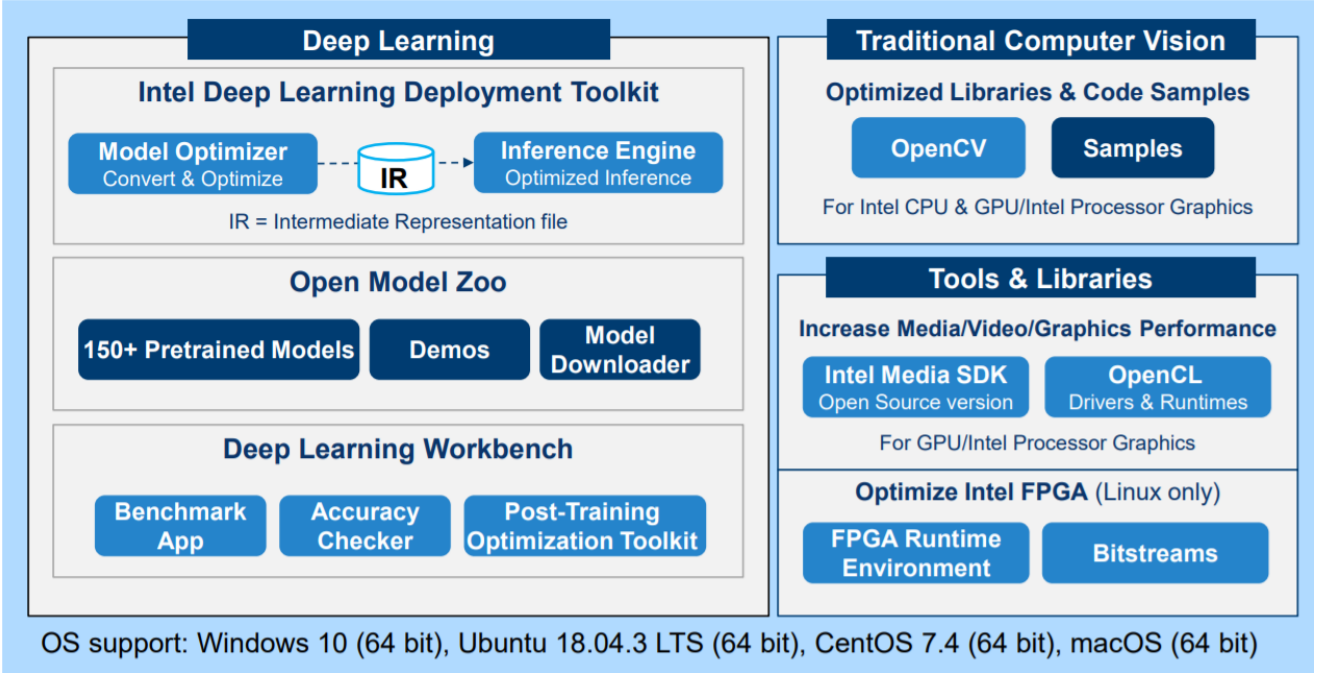

Рисунок 1.15 – Состав Intel Distribution of OpenVINO Toolkit

OpenVINO (Open Visual Inference and Neural network Optimization) Toolkit – это набор инструментов, разработанных компанией Intel, для оптимизации и ускорения работы нейронных сетей на различных устройствах, включая центральные процессоры (CPU), графические процессоры (GPU), программируемые матричные устройства (FPGA) и ускорители нейронных сетей (NCS, VPU).

Ключевые особенности и возможности OpenVINO Toolkit:

а) Компиляция и оптимизация: OpenVINO Toolkit предоставляет средства для компиляции и оптимизации нейронных сетей, созданных в популярных фреймворках, таких как TensorFlow, Caffe, MXNet и других. Он использует набор оптимизаций, включая квантизацию, сжатие моделей и автоматическое выделение функций, чтобы обеспечить максимальную производительность на целевых устройствах.

б) Многоплатформенность: OpenVINO Toolkit поддерживает широкий спектр целевых устройств, включая CPU, GPU, FPGA и VPU. Это позволяет разработчикам эффективно использовать аппаратные ресурсы каждого устройства для выполнения нейронных сетей с наилучшей производительностью.

в) Распределенные вычисления: OpenVINO Toolkit позволяет распределенное выполнение нейронных сетей на нескольких устройствах или в нескольких контейнерах Docker, что обеспечивает масштабируемость и ускорение вычислений.

г) Инференс в реальном времени: OpenVINO Toolkit обеспечивает возможность выполнения нейронных сетей в режиме реального времени, что особенно полезно для приложений, требующих низкой задержки, например, в области компьютерного зрения и обработки видео.

д) Удобство использования: OpenVINO Toolkit предоставляет высокоуровневые API на различных языках программирования, включая C++, Python, Java и C#. Это делает его доступным и удобным для разработчиков с разным уровнем опыта.

е) Интеграция с другими инструментами: OpenVINO Toolkit интегрируется с другими инструментами и библиотеками Intel, такими как Intel Distribution of OpenVINO, Intel Deep Learning Deployment Toolkit

К большому недостатку OpenVINO Toolkit можно отнести очень небольшое количество официально поддерживаемых отладочных плат. Поддерживаются различные устройства GPU и CPU с набором команд AVX2, AVX512. Поддержана Raspberry Pi 4 Model B (рисунок 1.16) [40], nVIDIA Jetson Nano, Android девайсы. Поддержана также Intel Arria 10 (рисунок 1.17) [41] и очень оригинальное устройство - Intel Neural Compute Stick 2 [42].

Рассмотрим технические характеристики плат, которые поддерживаются фремйворком OpenVINO:

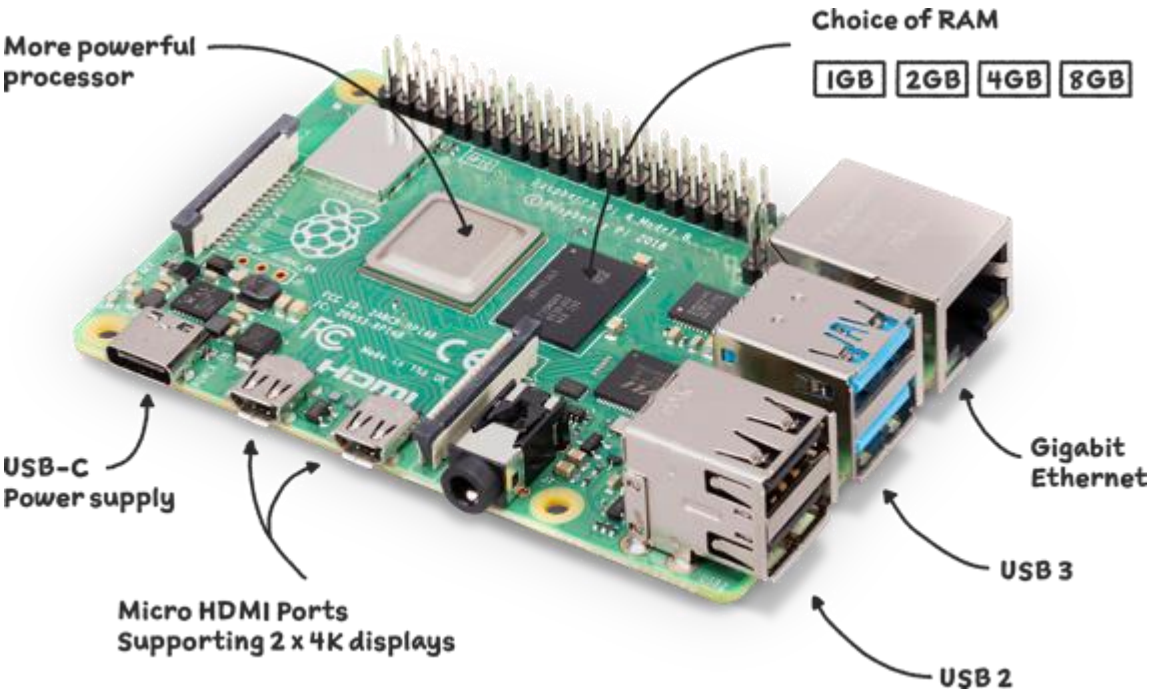

Рисунок 1.16 – Микрокомпьютер Raspberry Pi 4 Model B

а) Состав и технические характеристики платы Raspberry Pi 4 Model B (рисунок 1.16):

– Процессор: Broadcom BCM2711, 4-ядерный ARM Cortex-A72 (64-бит), тактовая частота 1,5 ГГц.

– Оперативная память (RAM): Варианты с 2 ГБ, 4 ГБ и 8 ГБ LPDDR4-3200 SDRAM.

– Графический процессор: VideoCore VI с поддержкой OpenGL ES 3.x.

– Видеовыходы: 2x микро HDMI порта с поддержкой разрешений до 4K при 60 Гц.

– Мультимедиа: Декодирование видео до 4Kp60, поддержка H.265 (HEVC) видео до 4Kp60, H.264 (AVC) видео до 1080p60.

– Хранение данных: Карты памяти MicroSD до 32 ГБ (возможна поддержка больших емкостей с использованием карты памяти SDXC).

– USB: 2x USB 2.0 порта, 2x USB 3.0 порта.

– Сеть: Гигабитный Ethernet порт (RJ-45), поддержка беспроводной связи Wi-Fi 802.11ac и Bluetooth 5.0.

– GPIO: 40 контактов GPIO (General-Purpose Input/Output) для подключения различных периферийных устройств и расширений.

Рисунок 1.17 – Terasic TR10a-HL Arria 10 FPGA

– Операционная система: поддерживает различные операционные системы, включая Raspbian (официальная ОС Raspberry Pi), Ubuntu, Windows 10 IoT Core и другие.

б) Состав и технические данные Terasic TR10a-HL Arria 10 FPGA Development Kit (рисунок 1.17):

– FPGA: Altera Arria 10 GX 10AX115N3F45E2SG FPGA, содержащая 115000 логических элементов (LE) и 6480 DSP-блоков.

– Память: DDR4 SDRAM: 4 ГБ (2x 2 ГБ) DDR4 SDRAM с тактовой частотой 2400 МГц. QDR IV SRAM: 64 МБ QDR IV SRAM.

– Хранение данных: Поддержка MicroSD-карт. Поддержка SATA интерфейса для подключения жесткого диска.

– Видео: Входы и выходы HDMI: 2 входа и 2 выхода HDMI с поддержкой разрешения до 1080p.

– Входы и выходы VGA: 1 вход и 1 выход VGA.

– Интерфейсы: Ethernet: 2 порта 10/100/1000 Ethernet. USB: 2 порта USB 3.0 и 2 порта USB 2.0. UART: 2 UART порта. GPIO: Разъем GPIO для подключения внешних устройств. PCIe: x8 Gen3 PCIe слот.

– Операционная система: Linux предустановлена на MicroSD-карте и поддерживается платой. Цена: 6624 доллара США

в) Состав и технические данные Intel Neural Compute Stick 2 (NCS2)

– Нейронный процессор: Intel Movidius Myriad X Vision Processing Unit (VPU).

– Архитектура: 16 SHAVE (Streaming Hybrid Architecture Vector Engine) ядер для выполнения вычислений с низкой мощностью.

– Интерфейс: USB 3.0 Type-A.

– Оперативная память (RAM): 4 ГБ LPDDR4.

– Скорость работы: до 4 тераОпс (TOPS) в инференсе и до 1 тераОпс в обучении.

– Разрешение видео: Поддержка видео с разрешением до 4K.

– Тепловое ограничение (TDP): 2 Вт.

Поддерживаемые фреймворки и библиотеки: TensorFlow, Caffe, MXNet, OpenVINO Toolkit и другие.

Поддерживаемые операционные системы: Linux и Windows.

Поддержка разработки: Intel Distribution of OpenVINO Toolkit обеспечивает инструменты для разработки и оптимизации моделей глубокого обучения.

Использование энергии: Низкое энергопотребление и мобильность делают NCS2 удобным для различных устройств и сценариев использования.

Intel Neural Compute Stick 2 (NCS2) предоставляет возможность выполнения высокоэффективных нейронных сетей на устройствах с ограниченными ресурсами. Он позволяет ускорить обработку данных и выполнение инференса нейронных сетей, делая его полезным для широкого спектра приложений, включая компьютерное зрение, робототехнику, автономные транспортные средства и другие области. Ниже приведена сводная таблица 1.3 отладочных плат для проектирования нейросетей и машинного зрения, работающих с фреймворком OpenVINO Toolkit.

Таблица 1.3 – Параметры отладочных плат Altera-Intel

Название платы	Raspberry Pi 4 Model B	Terasic TR10a-HL Arria 10 FPGA	Intel Neural Compute Stick 2 (NCS2)	Jetson nano
Количество ядер процессора	4 ядра ARM Cortex-A72	неприменимо	16 VPU SHAVE	4 ядра ARM Cortex-A57
Графический процессор	VideoCore VI	неприменимо	неприменимо	NVIDIA Maxwell, 4K
Оперативная память	2-32Gb	72 Mbit SRAM + QDR2/256MB Flash	4 Gb	4Gb
Нейросетевые возможности	OpenVINO Toolkit	OpenVINO Toolkit	OpenVINO Toolkit, TensorFlow, Caffe, ONNX, MXNet и другие	OpenVINO Toolkit, TensorFlow, PyTorch и Caffe
Порты и интерфейсы	2 USB 3.0 ports; 2 USB 2.0 micro-HDMI MIPI DSI MIPI CSI	PCI-E x8, 4 QSFP+, RS422, GPIO	USB 3.0	HDMI, USB 3.0, USB 2.0, Gigabit Ethernet, MIPI CSI и GPIO
Цена долл. США	От 35	6624	120	149

Сравнение и выводы.

При сравнении различных систем разработки машинного зрения, возникает сложная задача сопоставления иногда совершенно различных технических исполнений. Несмотря на это, у всех рассмотренных фреймворков и отладочных плат можно выделить общие конструктивные и программно-аппаратные модули. Отладочные платы Jetson имеют, например, комбинацию мощного многоядерного

процессора ARM + GPU (графического и математического акселератора). Платы же Xilinx имеют в своем составе чип SoC, который содержит в себе также мощный многоядерный процессор ARM + блок FPGA, на котором средствами фреймворка Vitis-AI [43] размещается DPU (Deep Learning Processor Unit) Процессорный блок глубокого обучения. Подобная же конструкция используется в (не рассмотренном в данной работе) процессоре Kendryte K210. Представляет также интерес фреймворк fpgaConvNet [44]. Вариант реализации нейросети на ПЛИС описан в работе [45]. Далее, процесс конструирования и обучения нейросети вынесен во вне этих программно-аппаратных комплексов. Этот процесс можно сравнить с компиляцией и компоновкой обычной компьютерной программы на мощном компьютере или суперкомпьютере и сервере. Делается это потому, что тренировка большой нейросети может потребовать огромных вычислительных мощностей, которыми микрокомпьютер не может обладать в принципе. С другой стороны, подготовленная таким образом и натренированная нейросеть может быть размножена и настроена на микрокомпьютерах в неограниченном количестве экземпляров. С точки зрения практического использования, выбор типа фреймворка и отладочной платы может определяться личными предпочтениями и навыками общения разработчика с тем или иным набором инструментов. С точки зрения интегральных характеристик отладочные платы Jetson и Xilinx Zynq Ultrascale+ близки по параметрам и, в принципе, находятся в одной ценовой нише. С точки зрения доступности в наше время можно выбрать платы с чипами Xilinx от третьих производителей типа Alinx или MYIR. Они более доступны и обладают более демократичной ценой. С точки зрения коммерческого производства и распространения, данная тема требует дополнительных исследований.

2 ОБЗОР СУЩЕСТВУЮЩИХ МОДЕЛЕЙ НЕЙРОСЕТЕЙ

2.1 История развития моделей и архитектур нейросетей

Рассмотрим эволюцию моделей и архитектур нейронных сетей за последние 10–12 лет. Примерно с 2011 года можно наблюдать активный рост научных исследований и публикации в этой области. Появилось множество значимых достижений области искусственного интеллекта, и впечатляющих результатов. Расскажем о нескольких ключевых моделях, которые повлияли на прогресс нейросетей в последние годы [46],[47].

– AlexNet (2012):

В 2012 году модель AlexNet, разработанная Алексеем Криссхаржевским и Ильёй Сосниным, привлекла широкое внимание на конкурсе ImageNet Large Scale Visual Recognition Challenge (ILSVRC). AlexNet стал первой глубокой сверточной нейронной сетью, которая существенно превзошла предыдущие подходы. Она имела пять сверточных слоев и три полносвязных слоя, а также использовала ReLU вместо более традиционной функции активации.

– GoogLeNet (2014):

GoogLeNet, представленная командой исследователей Google под руководством Сергея Иоффе и Кристиана Зампецери, предложила новую архитектуру нейронной сети, известную как Inception. Она использовала модуль с параллельными сверточными слоями разных размеров ядра и помогла снизить количество параметров сети. GoogLeNet стал первой моделью, превысившей человеческую точность на задаче классификации изображений ImageNet.

– ResNet (2015):

ResNet, или Residual Neural Network, разработанная командой исследователей Microsoft, представила новый подход к глубокому обучению, используя остаточные блоки. Эта модель помогла преодолеть проблему затухания градиента, возникающую при обучении глубоких нейронных сетей. ResNet с 152

слоями стал первой моделью, способной обучаться на такой глубине и достигать высоких результатов в различных задачах компьютерного зрения.

– Generative Adversarial Networks (GANs) (2014):

GAN, или генеративно-состязательные сети, представленные Ианом Гудфеллоу и его коллегами, стали мощным инструментом для генерации новых данных. GAN состоит из двух моделей: генератора и дискриминатора, которые соревнуются друг с другом. Генератор старается создать поддельные данные, неотличимые от реальных, в то время как дискриминатор старается их различить. GANs применяются в различных задачах, включая генерацию изображений, текста и звука.

– Transformer (2017):

Модель Transformer, представленная Васильем Форгати и его коллегами, стала революцией в обработке естественного языка. Transformer использует механизм внимания для обработки последовательностей без использования рекуррентных нейронных сетей. Эта модель достигла значительных результатов в задачах машинного перевода, генерации текста и вопросно-ответных системах.

– BERT (2018):

BERT (Bidirectional Encoder Representations from Transformers) - это модель, представленная исследователями Google. Она использует большие предобученные языковые модели на основе Transformer для эффективного понимания контекста в естественном языке. BERT существенно улучшил результаты в задачах обработки естественного языка, включая вопросно-ответные системы, классификацию текста и анализ тональности.

– GPT (Generative Pre-trained Transformer) (2018):

GPT, разработанная OpenAI, представила новый подход к генерации текста с использованием предобученных моделей на основе Transformer. Эта модель обучается на огромных объемах текстовых данных и способна генерировать качественные тексты, а также выполнять задачи, связанные с обработкой естественного языка, такие как перевод, анализ тональности и ответы на вопросы.

– AlphaZero (2018):

AlphaZero, разработанный компанией DeepMind, стал значительным прорывом в области обучения с подкреплением. Эта модель обучается играть в шахматы, го и шашки только на основе самоигры и без каких-либо предварительных данных или правил. AlphaZero продемонстрировал сверхчеловеческие навыки и преодолел лучшие шахматные и го программы.

– StyleGAN (2019):

StyleGAN, разработанная исследователями NVIDIA, представила инновационный подход к генерации и модификации изображений. Она позволяет контролировать различные стили и атрибуты генерируемых изображений, такие как возраст, пол, эмоции и стиль рисования. StyleGAN нашла применение в генерации реалистичных портретов, создании фотореалистичных сцен и дизайне персонажей.

– BERT-based модели и их дальнейшее развитие:

С 2018 года по настоящее время было разработано множество BERT-based моделей, основанных на архитектуре BERT. Некоторые из них включают RoBERTa, DistilBERT, ALBERT и ELECTRA. Эти модели превзошли BERT в задачах обработки естественного языка, улучшив качество и эффективность представления текста и понимания контекста.

– GPT-3 (2020):

GPT-3, разработанная OpenAI, является одной из самых масштабных и мощных моделей обработки естественного языка на текущий момент. Она имеет огромное количество параметров и способна выполнять различные задачи, такие как автозаполнение текста, перевод, генерация стихов и ответы на вопросы. GPT-3 вызвала большой интерес и обсуждения в сообществе искусственного интеллекта.

Прогресс в области компьютерного зрения:

С 2018 года и до настоящего времени произошел значительный прогресс в области компьютерного зрения. Модели, такие как EfficientNet, YOLOv4, и

ResNeSt, показали значительное улучшение в обнаружении объектов, семантической сегментации и анализе изображений. Также появились методы, основанные на генеративных моделях, такие как CycleGAN и Pix2Pix, для решения задач, связанных с генерацией и модификацией изображений [48],[49].

В дальнейшем мы рассмотрим те из моделей нейросетей, которые показали хорошие результаты именно в задачах машинного зрения, а именно распознавания и классификации объектов в поле зрения камеры.

Рисунок 2.1 иллюстрирует развитие нейросетей за последние 10 лет.

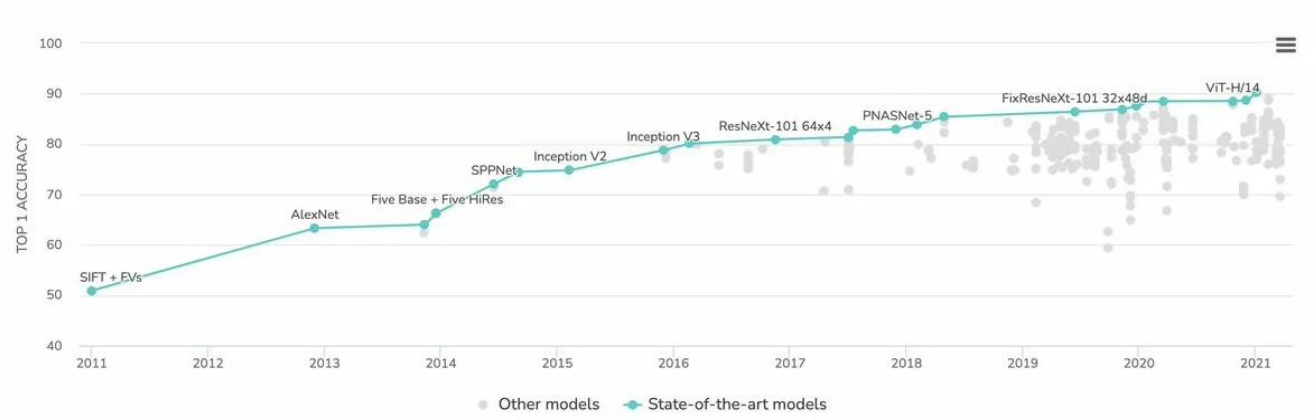

Рисунок 2.1 – Развитие моделей нейросетей в период 2011-2021 гг.

Из современных моделей нейросетей, которые широко используются для распознавания образов, можно рассмотреть следующие:

– ResNet (Residual Neural Network):

ResNet – это архитектура сверточной нейронной сети, которая внесла значительный вклад в область распознавания образов. Она представляет собой глубокую нейронную сеть с пропусками (skip connections), которые позволяют эффективно обучать глубокие модели, избегая проблему затухания градиентов. ResNet имеет различные варианты, включая ResNet-50, ResNet-101, ResNet-152 и другие.

– VGG (Visual Geometry Group):

VGG – это архитектура сверточной нейронной сети, которая известна своей простотой и эффективностью. Она состоит из нескольких сверточных слоев

и полносвязных слоев с небольшим размером фильтра (3x3) и фиксированным числом фильтров. VGG имеет различные варианты, например, VGG-16 и VGG-19, которые различаются по числу слоев.

– Inception (GoogLeNet):

Inception или GoogLeNet – это архитектура сверточной нейронной сети, разработанная компанией Google. Она известна своим модульным подходом, в котором используются различные размеры фильтров (1x1, 3x3, 5x5) параллельно для извлечения различных уровней признаков. Это позволяет сети иметь более широкий охват признаков и эффективно использовать ресурсы.

– DenseNet (Densely Connected Convolutional Networks):

DenseNet – архитектура сверточной нейронной сети, в которой каждый слой имеет прямые связи со всеми предыдущими слоями. Это создает плотные связи между слоями и позволяет модели эффективно использовать признаки из предыдущих слоев. DenseNet известен своей способностью более эффективно использовать параметры и обеспечивать хорошую производительность с меньшим количеством параметров.

– EfficientNet:

EfficientNet – это семейство моделей нейросетей, которые основаны на методе автоматического масштабирования модели с использованием коэффициента масштабирования. Этот подход позволяет создавать модели с различными размерами и глубиной, чтобы достичь оптимального баланса между производительностью и вычислительной сложностью. EfficientNet показывает высокую эффективность в распознавании образов при меньшем количестве параметров.

Каждая из этих моделей имеет свои особенности и преимущества в задаче распознавания образов. Выбор конкретной модели будет зависеть от требований задачи, доступных ресурсов и ограничений, таких как скорость обработки и объем памяти.

2.2 Модель ResNet (Residual Neural Network)

Модель ResNet (Residual Neural Network) – глубокая сверточная нейронная сеть, которая была представлена в 2015 году командой исследователей Microsoft.

Она представляет собой важный прорыв в области глубокого обучения и позволяет эффективно обучать модели с большим количеством слоев. Конструкция ResNet основана на концепции остаточных блоков (residual blocks).

Остаточные блоки позволяют пропускать один или несколько сверточных слоев, добавляя пути (residual connections), которые обходятся вокруг этих слоев. Остаточная связь позволяет сохранять и передавать информацию и градиенты через слои, облегчая обучение глубоких сетей. Структура ResNet обычно состоит из нескольких блоков, которые повторяются. Каждый блок содержит несколько сверточных слоев, Batch Normalization и функцию активации, такую как ReLU. Отличительной особенностью ResNet является наличие остаточных связей, которые добавляются после каждого блока. Это позволяет градиентам свободно проходить через блоки и избежать затухания градиентов.

– Сильные стороны модели ResNet:

1) Способность обучать глубокие модели: Остаточные блоки позволяют эффективно обучать модели с большим количеством слоев. Это позволяет модели ResNet достигать высокой точности в различных задачах компьютерного зрения.

2) Решение проблемы затухания градиентов: Остаточные связи позволяют градиентам легко протекать через слои, облегчая обучение глубоких сетей и устраняя проблему затухания градиентов.

3) Хорошая обобщающая способность: Модель ResNet имеет хорошую способность обобщения, что означает, что она может эффективно распознавать и классифицировать объекты, которые она ранее не видела, благодаря своей глубине и структуре.

4) Применимость к различным задачам: ResNet была успешно применена в различных задачах компьютерного зрения, включая классификацию изображений, семантическую сегментацию и обнаружение объектов.

– Слабые стороны модели ResNet:

1) Высокое количество параметров: Использование остаточных блоков приводит к увеличению количества параметров в сравнении с другими моделями. Это может потребовать больших вычислительных ресурсов и времени для обучения.

2) Потребление памяти: Большое количество слоев и параметров в модели ResNet требует значительное количество памяти для хранения и вычислений.

3) Возможность переобучения: при использовании очень глубоких моделей ResNet есть риск переобучения, особенно при недостаточном количестве обучающих данных или при неправильной регуляризации.

4) Сложность интерпретации: из-за большого количества слоев и связей, модель ResNet может быть сложной для интерпретации и понимания, особенно для анализа, почему она принимает определенные решения.

Не смотря на некоторые слабости, модель ResNet остается одной из наиболее влиятельных и широко используемых архитектур в области компьютерного зрения, и ее идеи остаточных блоков привнесли значительный вклад в развитие глубокого обучения.

2.3 Модель VGG (Visual Geometry Group)

Модель VGG (Visual Geometry Group) – глубокая сверточная нейронная сеть, представленная в 2014 году исследовательской группой Visual Geometry Group из Университета Оксфорд. Она известна своей простой и легко интерпретируемой архитектурой. Конструкция модели VGG основана на использовании сверточных слоев с малыми фильтрами размером 3x3, которые последовательно объединяются. Архитектура VGG состоит из нескольких последовательных блоков, каждый из которых содержит несколько сверточных слоев, а затем слой субдискретизации (пулинга) для уменьшения размерности.

– Сильные стороны модели VGG:

1) Простота и понятность: Архитектура VGG проста и легко интерпретируема. Она состоит из последовательного стека сверточных слоев, что делает ее понятной для понимания и анализа.

2) Показывает хорошие результаты в классификации: VGG демонстрирует высокую точность в задачах классификации изображений.

3) Модель VGG стала одной из первых успешных моделей, добившейся высокой точности на наборе данных ImageNet.

4) Варианты с разной глубиной: Архитектура VGG предлагает варианты с разным количеством слоев (например, VGG-16 и VGG-19), что позволяет выбирать модель с соответствующей глубиной в зависимости от требований задачи.

5) Легко повторно использовать: Архитектура VGG может быть использована как основная модель для извлечения признаков (feature extraction) в других задачах компьютерного зрения, таких как обнаружение объектов или семантическая сегментация.

– Слабые стороны модели VGG:

1) Высокое количество параметров: Архитектура VGG имеет большое количество параметров из-за повторяющихся сверточных слоев, что требует больших вычислительных ресурсов для обучения и использования модели.

2) Высокое потребление памяти: из-за большого количества слоев и параметров, модель VGG требует значительное количество памяти для хранения и вычислений.

3) Ограниченная эффективность: В сравнении с более новыми архитектурами, такими как ResNet или Inception, VGG не обладает высокой эффективностью, особенно при работе с ограниченными вычислительными ресурсами.

4) Склонность к переобучению: из-за большого количества параметров, VGG может быть склонна к переобучению, особенно при ограниченном количестве обучающих данных. Необходимы соответствующие методы регуляризации для борьбы с этой проблемой.

Не смотря на некоторые слабости, модель VGG все еще является важным вкладом в область компьютерного зрения и имеет свои преимущества, особенно в задачах классификации изображений.

2.4 Модель YOLO (You Only Look Once) V3,V4

YOLO (You Only Look Once) V3 - модель нейронной сети для обнаружения объектов в реальном времени. Она является одной из самых популярных и эффективных архитектур для задачи обнаружения объектов.

Основные особенности модели YOLO V3:

– Одновременное обнаружение и классификация:

YOLO V3 может одновременно обнаруживать и классифицировать объекты на изображении. Вместо двух отдельных процессов - сначала обнаружение, а затем классификация, модель выполняет эти задачи параллельно, что делает ее быстрой и эффективной.

– Сетка ограничивающих рамок (bounding box grid):

YOLO V3 использует сетку ограничивающих рамок, разбивая изображение на сетку ячеек. Каждая ячейка отвечает за обнаружение объекта, если центр этого объекта попадает в данную ячейку. Каждая ячейка предсказывает ограничивающие рамки (bounding boxes) и соответствующие вероятности классов.

– Многоуровневые предсказания:

YOLO V3 строит предсказания на нескольких масштабных уровнях. Это позволяет модели обнаруживать объекты разного размера и масштаба, начиная от крупных объектов, заполняющих большую часть изображения, до маленьких объектов.

– Использование ограничивающих рамок различных размеров:

YOLO V3 использует ограничивающие рамки различных размеров для обнаружения объектов разного масштаба. Это позволяет модели лучше обрабатывать объекты разного размера и улучшает точность обнаружения.

– Использование предварительно обученных сверточных слоев:

YOLO V3 использует предварительно обученные сверточные слои, такие как DarkNet-53, для извлечения признаков из изображений.

Эти слои имеют большую глубину и хорошую способность к абстрактному представлению изображений.

YOLO V3 имеет хорошую производительность в реальном времени и способна обнаруживать объекты на изображениях с высокой скоростью. Она находит широкое применение в различных задачах, таких как автоматическое вождение, видеонаблюдение, робототехника и другие области, где требуется быстрое и точное обнаружение объектов.

Модель YOLO (You Only Look Once) v4 – последняя версия популярной модели для обнаружения объектов в реальном времени.

Она была представлена в 2020 году и представляет собой улучшение предыдущей версии YOLO v3. Конструкция модели YOLO v4 [50] имеет несколько изменений и улучшений по сравнению с предыдущими версиями.

Основные компоненты архитектуры YOLO v4 включают в себя:

– Backbone: YOLO v4 использует различные варианты предварительно обученных сверточных нейронных сетей в качестве основы (backbone), включая CSPDarknet53 и CSPResNeXt50. Они обеспечивают извлечение признаков из входного изображения.

– Feature Pyramid: В YOLO v4 внедрена модифицированная версия Feature Pyramid Network (FPN), которая помогает улавливать и использовать многомасштабные признаки из разных уровней сверточных слоев для более точного обнаружения объектов разных размеров.

– Neck: В YOLO v4 добавлено несколько дополнительных слоев, называемых Neck, для дополнительной обработки признаков и улучшения их представления перед передачей их на слои обнаружения.

– Detection Layers: аналогично YOLO v3, YOLO v4 содержит слои обнаружения, которые предсказывают границы и классы объектов в определенных ячейках сетки. Однако в YOLO v4 добавлено больше слоев обнаружения, что позволяет модели быть более точной и распознавать больше классов объектов.

– Anchor Boxes: YOLO v4 также использует якорные прямоугольники (anchor boxes), которые определяют заранее заданный набор форм объектов

разных размеров и соотношений сторон, для более точного предсказания границ объектов.

– Специфичные для YOLO v4 техники: В YOLO v4 внедрены различные техники для улучшения качества обнаружения, такие как Mish активация, PANet (Path Aggregation Network) для объединения многомасштабных признаков и SAM (Spatial Attention Module) для улучшения пространственного восприятия.

Сильные стороны модели YOLO v4:

1) Высокая скорость и точность: YOLO v4 предлагает баланс между скоростью и точностью обнаружения объектов. Она способна обрабатывать видеопотоки в реальном времени и достигает высокой точности на многих известных наборах данных.

2) Обработка объектов различных размеров: благодаря использованию FPN и множеству слоев обнаружения разного масштаба, YOLO v4 может эффективно обнаруживать объекты разных размеров на разных уровнях сверточной сети.

3) Встроенная аугментация данных: YOLO v4 включает в себя встроенную аугментацию данных, такую как случайные повороты и масштабирование, чтобы улучшить способность модели обнаруживать объекты в различных условиях.

4) Поддержка большого числа классов объектов: YOLO v4 может обнаруживать и классифицировать большое количество классов объектов одновременно, что делает его полезным в различных приложениях, требующих обнаружения множества классов.

Слабые стороны модели YOLO v4:

– Высокие вычислительные требования: YOLO v4 требует значительных вычислительных ресурсов для обучения и выполнения. Это может быть проблематично при ограниченных вычислительных мощностях или на встроенных устройствах.

– Ограничения по размеру объектов: как и предыдущие версии YOLO, YOLO v4 имеет ограничения в обнаружении малых объектов, особенно когда они находятся в контексте больших объектов.

– Зависимость от качества данных обучения: как и любая модель машинного обучения, YOLO v4 требует большого объема разнообразных и высококачественных данных для обучения, чтобы достичь оптимальной производительности. Ограниченные или некачественные данные могут привести к снижению точности модели.

В целом, модель YOLO v4 представляет собой мощный инструмент для обнаружения объектов в реальном времени, который обладает хорошим сочетанием скорости и точности. Она широко используется в различных приложениях компьютерного зрения и продолжает привлекать внимание исследователей и разработчиков.

Сравнение и выводы.

Выбор модели нейросети для распознавания дефектов при печати полиграфической продукции и дефектов изготовления печатных плат зависит от нескольких факторов. Из вариантов (VGG, ResNet, YOLOv3), наиболее оптимальной может являться модель YOLOv3.

Вот несколько причин, почему YOLOv3 может быть наилучшим выбором для задачи распознавания дефектов при печати полиграфической продукции и дефектов изготовления печатных плат:

1) Обнаружение объектов различных размеров: YOLOv3 имеет многомасштабную архитектуру, которая позволяет обнаруживать объекты разных размеров. Это важно при распознавании дефектов печати, так как дефекты могут иметь разные размеры и формы.

2) Высокая скорость обнаружения в реальном времени: YOLOv3 оптимизирована для работы в режиме реального времени и способна быстро

обрабатывать видеопотоки или изображения. Это особенно полезно при обнаружении дефектов на большом объеме полиграфической продукции.

3) Множественные классы объектов: YOLOv3 может обнаруживать и классифицировать множество классов объектов одновременно. В случае распознавания дефектов печати, модель может быть обучена на различные типы дефектов, такие как размытость, пятна, неравномерность цвета и т. д.

4) Легкость в использовании и модификации: YOLOv3 имеет относительно простую архитектуру, что делает ее легко понятной и настраиваемой. Разработчик может легко обучить и адаптировать модель под свои конкретные требования.

5) Широкое использование и сообщество: YOLOv3 является одной из самых популярных моделей для обнаружения объектов и имеет широкую поддержку и активное сообщество разработчиков. Можно найти множество пред-обученных моделей, реализаций и руководств, которые помогут разработчику в реализации решения для распознавания дефектов печати.

Важно отметить, что выбор модели нейросети также может зависеть от доступных данных для обучения и конкретных требований проекта. Рекомендуется провести предварительные эксперименты с различными моделями и оценить их производительность для конкретной задачи распознавания дефектов при печати полиграфической продукции.

3 ВАРИАНТ РЕАЛИЗАЦИИ ЗАДАЧИ

3.1 Краткое введение в фреймворк Vitis-AI

Vitis-AI – фреймворк и набор инструментов, разработанных компанией Xilinx для развертывания и оптимизации нейронных сетей на их аппаратных платформах, таких как FPGA (программируемые вентильные матрицы) и SoC (системы на кристалле).

История создания:

– В 2019 году Xilinx представила Vitis, новый программный стек и исследовательское окружение, которое предназначено для упрощения разработки и оптимизации программного обеспечения для аппаратных платформ Xilinx.

– В 2020 году Xilinx анонсировала Vitis-AI, расширение Vitis, специально разработанное для работы с нейронными сетями и их развертывания на аппаратных платформах Xilinx.

Основные особенности и возможности Vitis-AI:

– Гибкость и переносимость:

Vitis-AI обеспечивает гибкость и переносимость моделей нейронных сетей на различные аппаратные платформы Xilinx, включая FPGA и SoC. Он поддерживает различные форматы моделей, такие как TensorFlow, Caffe и ONNX, что позволяет разработчикам выбирать наиболее подходящий формат для своих нужд.

– Оптимизация и автоматическая компиляция:

Vitis-AI предлагает инструменты для оптимизации и автоматической компиляции моделей нейронных сетей для конкретных аппаратных платформ Xilinx. Он использует техники компиляции, такие как прореживание (pruning),

квантизация (quantization) и разделение модели (model partitioning), чтобы улучшить производительность и эффективность работы моделей.

– Ускорение аппаратного обеспечения:

Vitis-AI (рисунок 3.1) позволяет разработчикам использовать аппаратное ускорение с помощью FPGA и SoC для значительного увеличения

производительности моделей нейронных сетей. Он предоставляет инструменты для оптимальной конфигурации и развертывания моделей на аппаратных платформах Xilinx.

– Интеграция с другими инструментами:

Vitis-AI интегрируется с другими инструментами и фреймворками машинного обучения, такими как TensorFlow и PyTorch. Это обеспечивает удобный рабочий процесс разработчиков и позволяет им использовать привычные инструменты и библиотеки. В целом, Vitis-AI предоставляет разработчикам и исследователям мощные инструменты и возможности для развертывания и оптимизации нейронных сетей на аппаратных платформах Xilinx.

Он позволяет достичь высокой производительности и эффективности работы моделей, используя аппаратное ускорение и оптимизацию [33].

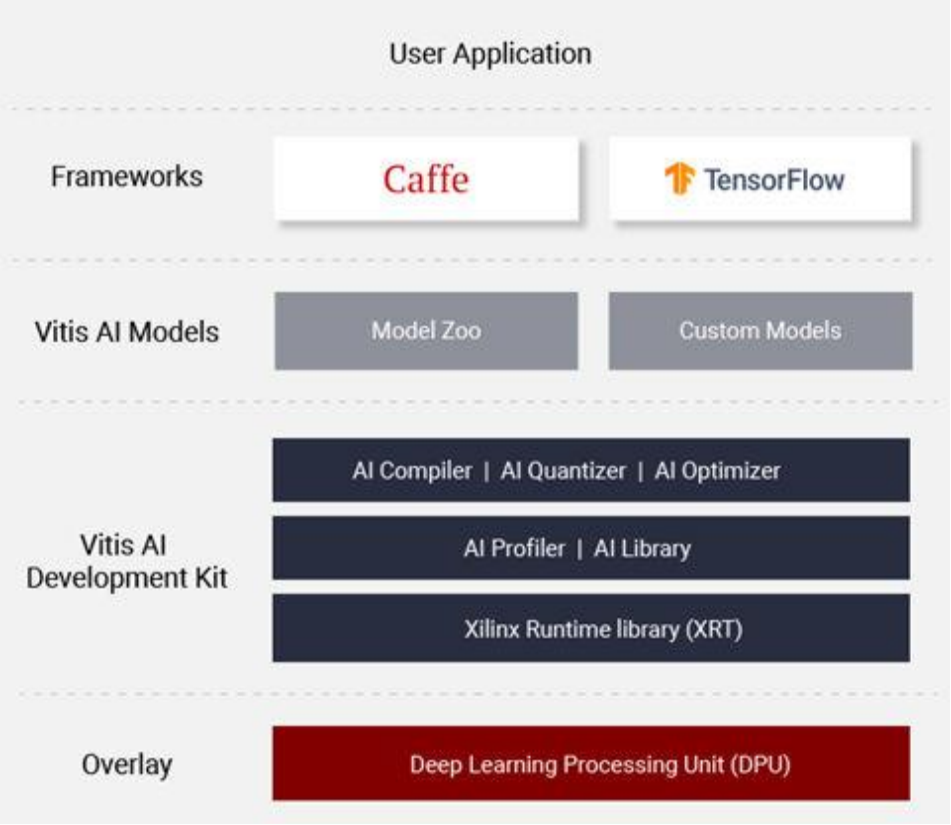

Рисунок 3.1 – Стек разработки Vitis-AI

Особенности:

Vitis-AI включает в себя следующие возможности:

– Поддержка основных фреймворков и новейших моделей, способных решать разнообразные задачи глубокого обучения задачи.

– Предоставляет полный набор предварительно оптимизированных моделей, готовых к развертыванию на устройствах Xilinx.

– Предоставляет мощный квантизатор (quantization), который поддерживает квантизацию, калибровку и тонкую настройку модели.

Для опытных пользователей Xilinx также предлагает дополнительный оптимизатор ИИ, который может сократить модель до 90%.

– Профилировщик ИИ обеспечивает послойный анализ для выявления узких мест.

– Библиотека ИИ предлагает унифицированные высокоуровневые API на C++ и Python для максимальной переносимости из оконечных устройств в облачные среды, от оконечного устройства до облака.

– Настраивает эффективные и масштабируемые IP-ядра для удовлетворения ваших потребностей для различных приложений с точки зрения пропускной способности, задержки и энергопотребления.

Компоненты:

– Процессорный блок глубокого обучения (DPU).

DPU – программируемый автомат, оптимизированный для глубоких нейронных сетей. Он представляет собой группу параметризуемых IP-ядер, предварительно реализованных на аппаратном обеспечении без необходимости трассировки в кристалле.

DPU выпускается со специализированным набором инструкций Vitis-AI, что позволяет ему эффективно реализовать множество сетей глубокого обучения.

Vitis-AI предлагает серию различных DPU для встраиваемых устройств, таких как Xilinx Zynq-7000, Zynq UltraScale+ MPSoC, так и для плат Alveo, таких как U50, U200, U250 и U280, что обеспечивает уникальную дифференциацию и

гибкость с точки зрения пропускной способности, задержки, масштабируемости и мощности.

Имеются несколько вариантов DPU, которые оптимизированы под конкретные кристаллы Soc и FPGA от фирмы Xilinx.

– AI Model Zoo (рисунок 3.2).

AI Model Zoo включает оптимизированные модели глубокого обучения для ускорения развертывания глубокого обучения на платформах Xilinx.

Эти модели охватывают различные области применения, включая ADAS/AD, видеонаблюдение, робототехника, центры обработки данных и т.д. Разработчик может начать работу с этими предварительно обученными моделями, чтобы воспользоваться преимуществами ускорения глубокого обучения.

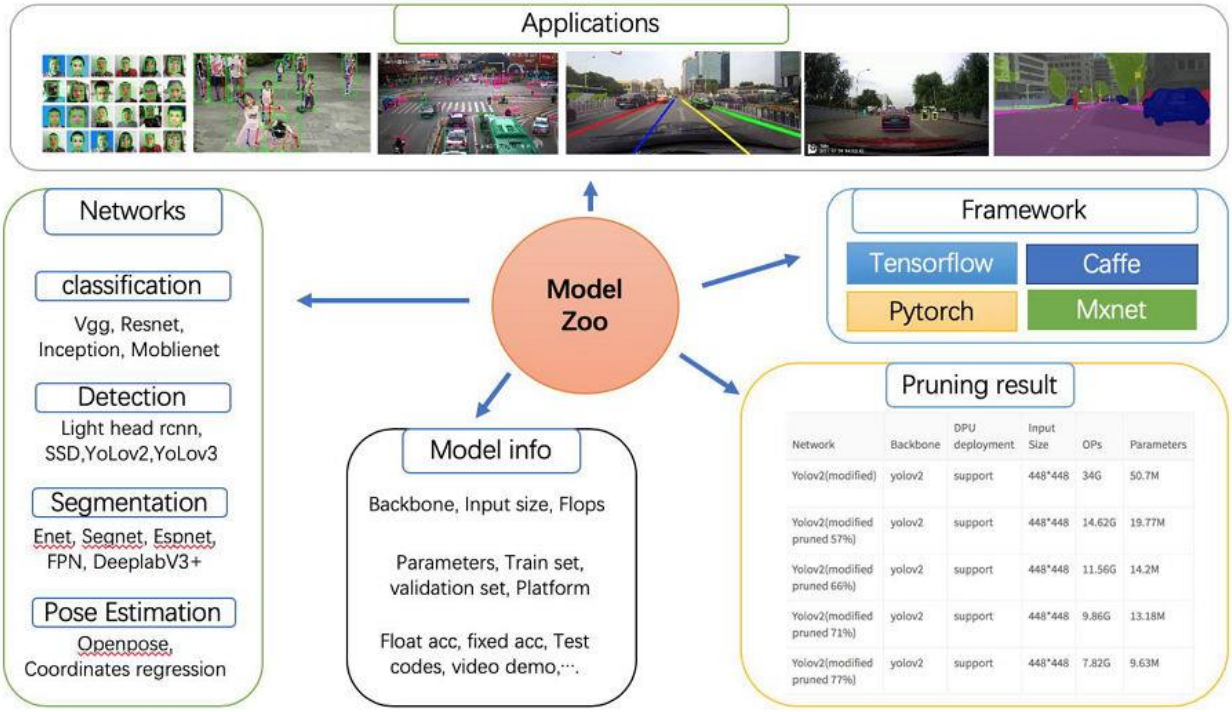

Рисунок 3.2 – AI Model Zoo

– AI Optimizer (Оптимизатор).

Позволяет сжать модель нейросети путем уменьшения сложности модели от 5 до 50 раз при минимальном влиянии на точность. Глубокое сжатие повышает производительность ИИ выводов на новый уровень. Для работы AI Optimizer

требуется коммерческая лицензия, поэтому мы не будем рассматривать этот блок в дальнейшем. В нашей работе он не используется.

– AI Quantizer (Квантизатор) (рисунок 3.3).

Путем преобразования 32-битных весов и активаций с плавающей точкой в фиксированную точку, например INT8, квантизатор ИИ может уменьшить сложность вычислений и снизить, при этом, точность предсказания. Quantizer может уменьшить сложность вычислений без потери точности предсказания, если использовать модель нейросети с фиксированной точкой. Нейросетевая модель с фиксированной точкой требует меньшей пропускной способности памяти, что обеспечивает более высокую скорость и энергоэффективность по сравнению с моделью с плавающей точкой.

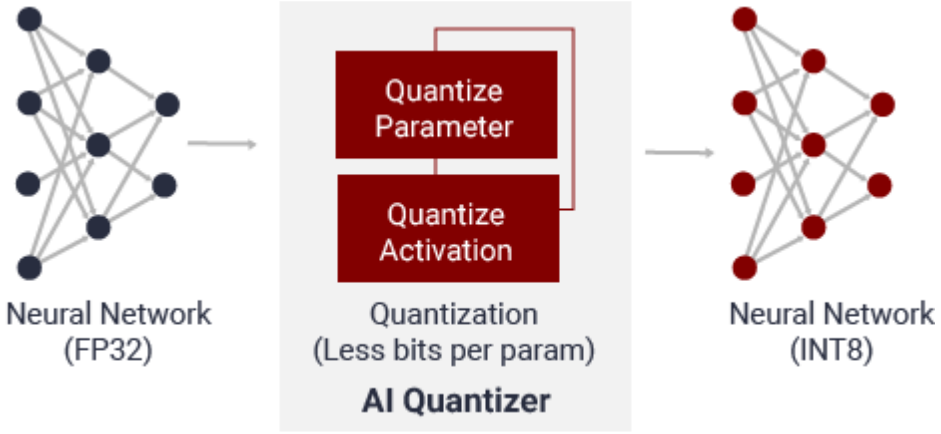

Рисунок 3.3 – AI Quantizer

– AI Compiler (Компилятор) (рисунок 3.4).

Компилятор ИИ отображает модель ИИ на высокоэффективный набор инструкций и модель потока данных. Он также выполняет сложные оптимизации, такие как объединение слоев, кэширование инструкций и повторное использование памяти на кристалле настолько, насколько это возможно.

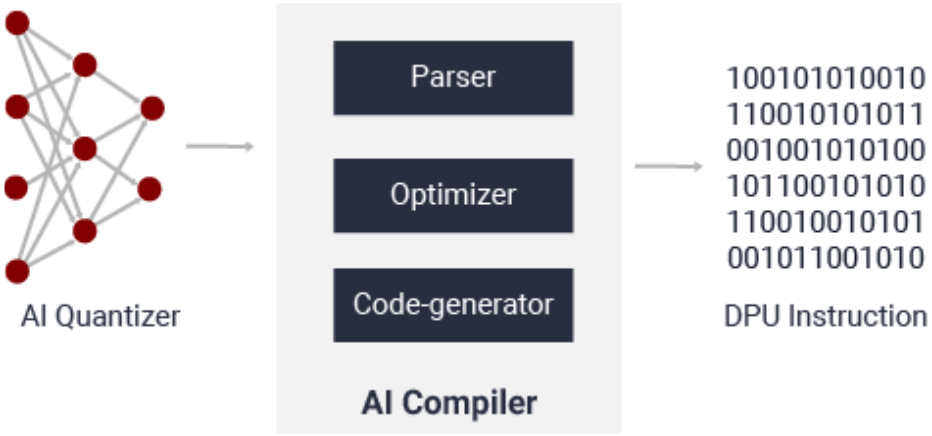

Рисунок 3.4 – AI Compiler

– AI Profiler (Профилировщик) (рисунок 3.5).

Профилировщик ИИ Vitis может помочь профилировать и визуализировать приложения ИИ, найти узкие места и помочь распределить вычислительные ресурсы между различными устройствами.

– Он прост в использовании и не требует изменения кода. Он также может отслеживать вызовы функций и время выполнения.

– Инструмент также может собирать информацию об аппаратном обеспечении, включая использование CPU/ DPU/ памяти.

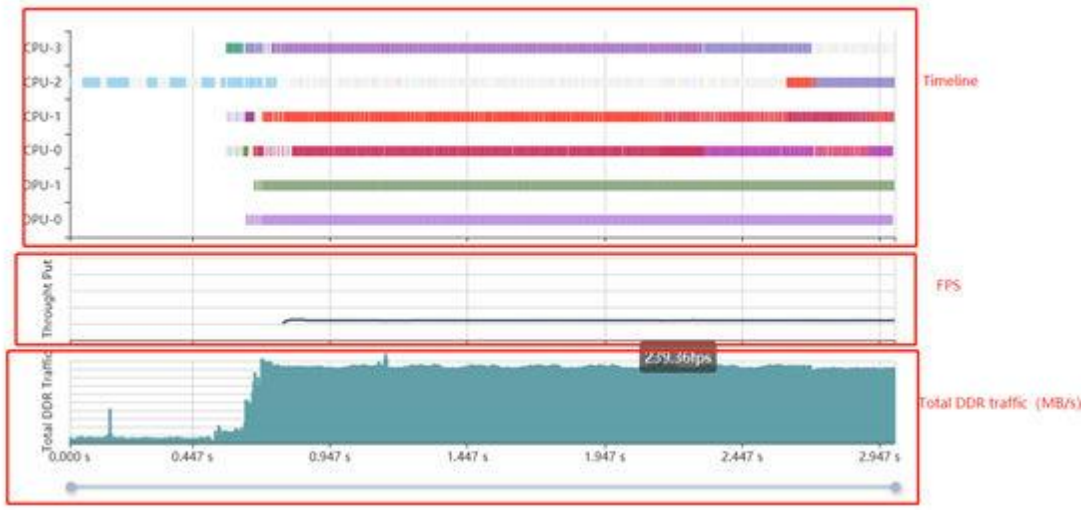

Рисунок 3.5 – AI Profiler

– AI Library (Библиотека) (рисунок 3.6)

Библиотека Vitis-AI – набор высокоуровневых библиотек и API, созданных для эффективного вывода ИИ с помощью процессорного блока глубокого обучения (DPU). Она полностью поддерживает XRT и построена на базе среды выполнения Vitis-AI с использованием унифицированных API-интерфейсов Vitis runtime.

Библиотека Vitis-AI Library обеспечивает простой в использовании и унифицированный интерфейс, инкапсулируя множество эффективных и высококачественных нейронных сетей. Это упрощает использование нейронных сетей глубокого обучения, даже для пользователей, не обладающих знаниями в области глубокого обучения или ПЛИС. Библиотека искусственного интеллекта Vitis позволяет уделять больше внимания разработке своих приложений, а не концентрироваться на базовом аппаратном обеспечении.

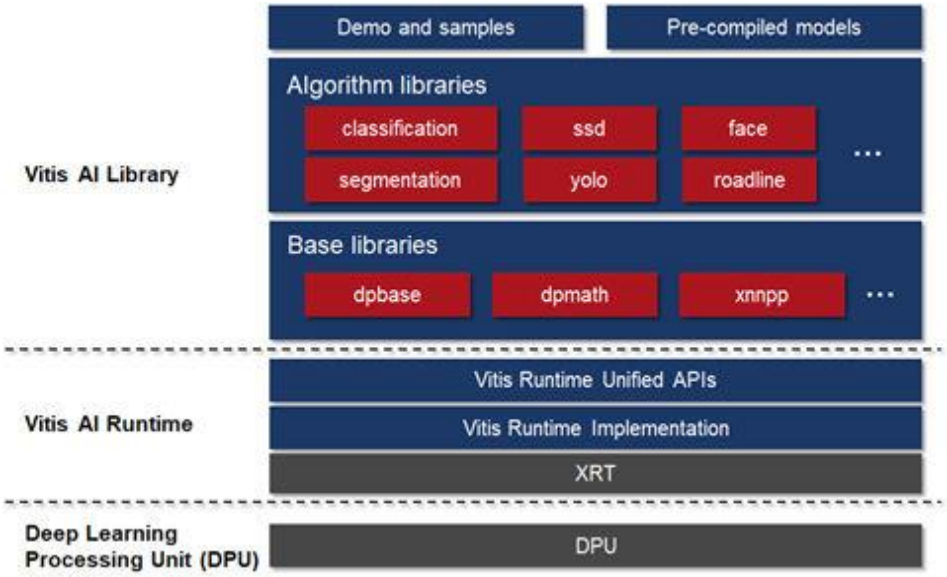

Рисунок 3.6 – AI Library

– AI Runtime (Библиотека времени исполнения).

Среда выполнения Vitis-AI позволяет приложениям использовать единый высокоуровневый API среды времени выполнения как для облака, так и на оконечном устройстве, что делает развертывание с облака на оконечное устройство беспроблемным и эффективным.

Возможности API среды выполнения Vitis-AI следующие:

– Асинхронная отправка заданий на ускоритель;

– Асинхронный получение заданий с ускорителя;

– Реализации на C++ и Python;

– Поддержка многопоточности и многопроцессного исполнения.

3.2 Постановка задачи, рабочая система и рабочее окружение

– Решаемая задача:

1) Необходимо создать систему машинного зрения с распознаванием ограниченного набора паттернов с использованием нейросети (нейросетевого ускорителя DPU Kernel (Xilinx)), расположенной в SoC фирмы Xilinx Zynq Ultrascale+. Использовать для проекта гетерогенную отладочную плату Alinx AXU2CGA. Пройти сквозной маршрут проектирования, предусмотренный стандартом разработки для Vitis-AI. Рассмотреть возможность использования нескольких известных прототипов нейросетей и Yolo-V3, как базовой модели нейросети для разработки. Ввод видеоданных будет происходить с камеры HD-USB. Распознанный паттерн будет передаваться в компьютер архитектуры PC x86_x64 с помощью известных утилит для работы с системами на основе Линукс типа PuTTY и WinSCP-5.19 для дальнейшей обработки компьютерной программой и дублироваться на мониторе, подключенном к отладочной плате Alinx AXU2CGA.

– Экспериментальная система состоит из следующих компонентов:

1) Отладочная плата Alinx AXU2CGA, с блоком питания.
2) USB видеокамера Logitech C270i.
3) Монитор с разъемом mini DP (через переходник).
4) Micro SD карта, с установленной ОС Linux (Petalinux).
5) USB-mini usb кабель для создания RS232 соединения с PC.
6) Компьютер PC с установленной системой Windows 10.

Необходимые программные компоненты на PC:

2) PuTTY 0.74
3) WinSCP-5.19

Все программные компоненты относятся к свободно распространяемому программному обеспечению и могут быть свободно скачаны из Интернета.

Собранный аппаратный комплекс показан на рисунке 3.7.

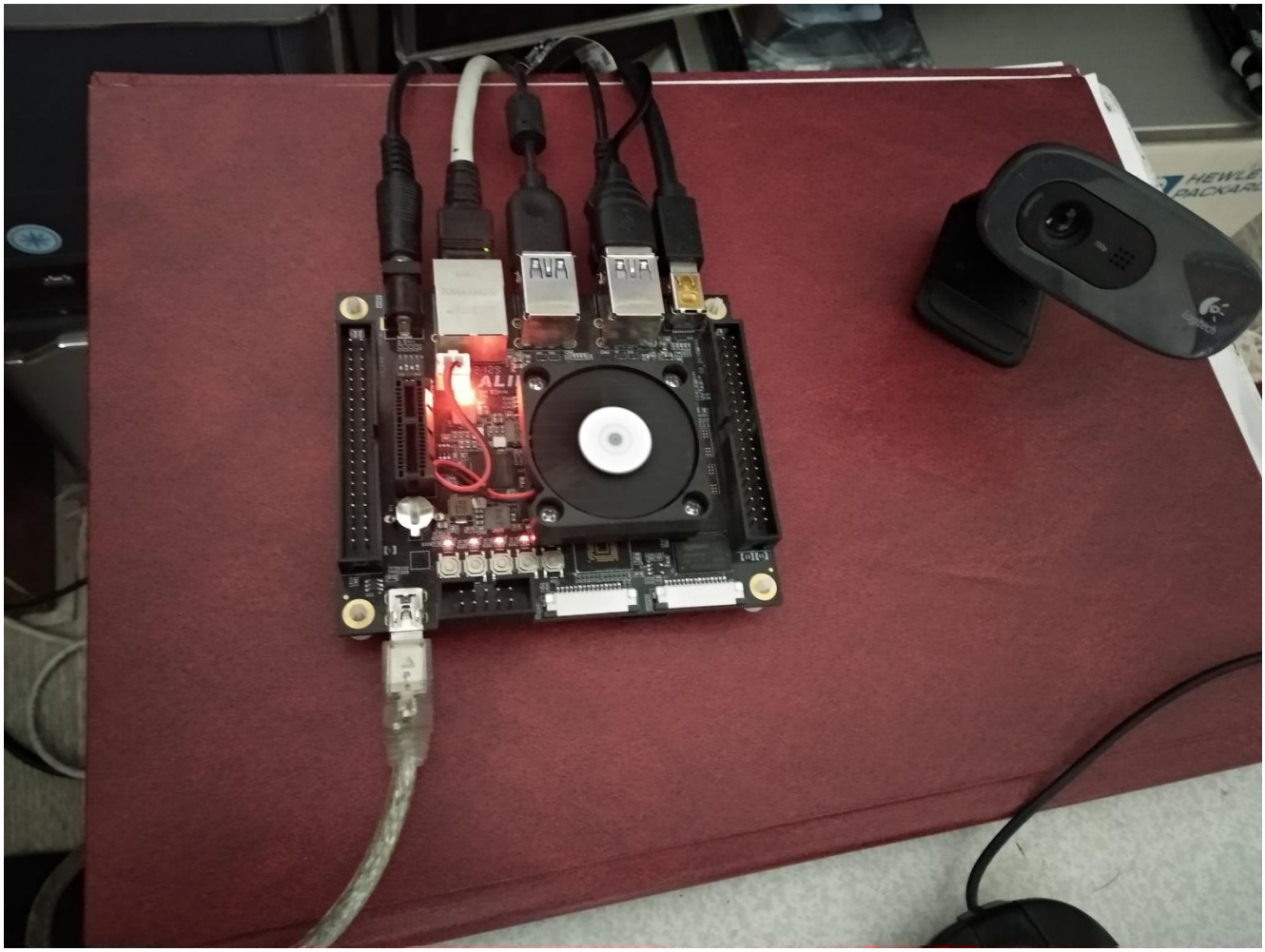

Рисунок 3.7 – Система в сборе

3.3 Реализация маршрута проектирования

Маршрут разработки можно схематично увидеть на рисунке 3.8 [37].

Рассмотрим этапы процесса разработки:

1. Создание нейросети в фреймворке TensorFlow (Keras) или PyTorch.

2. Тренировка нейросети на датасете и тестирование правильности работы.

3. Вывод нейросети в формате *.pb для дальнейшего использования в фреймворке Vitis-AI (На сегодняшний день Vitis-AI уже имеет большой список натренированных нейросетей, которые можно использовать без двух первых этапов).

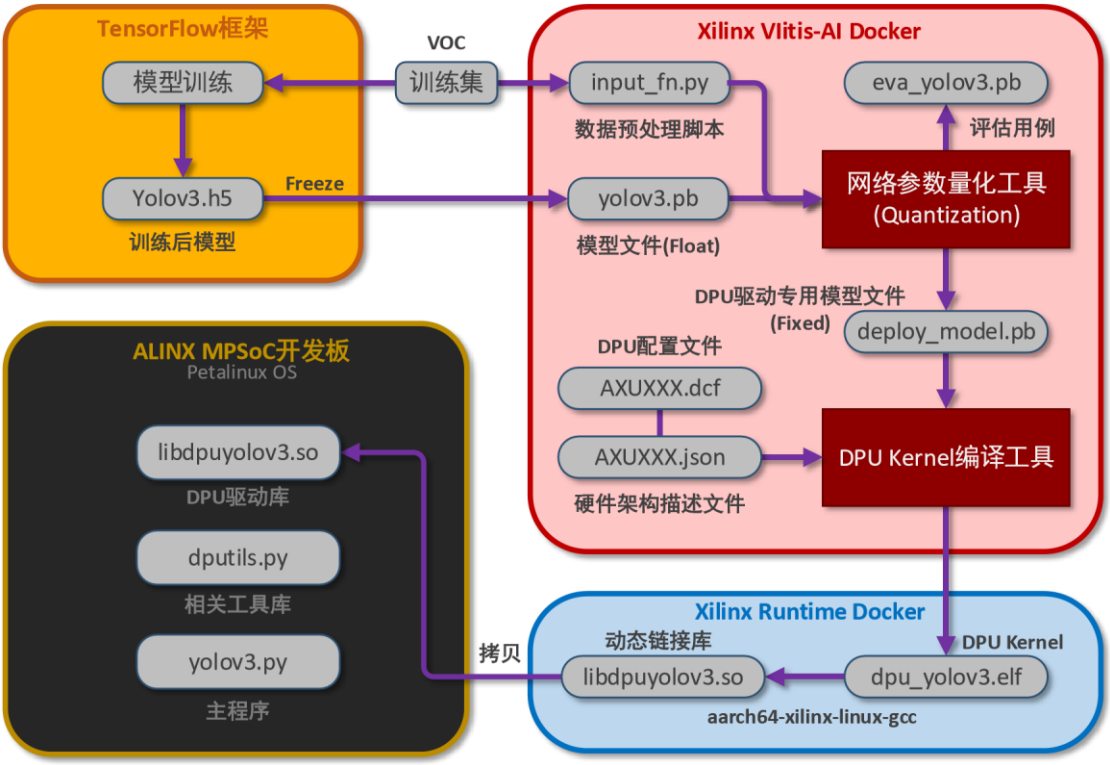

Рисунок 3.8 – Маршрут разработки в Vitis-AI

4. Квантизация нейросети. На этом этапе мы преобразуем веса сети из формата float в формат int, что сокращает ее размеры и увеличивает быстродействие.

5. Предварительная компиляция нейросети совместно с hardware описанием варианта DPU, который будет использоваться для формирования ядра сопроцессора, размещаемого в PL (FPGA) блоке SoC Zynq Ultrascale+

6. Кросс-компиляция в системе команд ARM, для создания библиотеки .SO (DLL)

7. Перенос библиотеки в целевую плату и написание программы для взаимодействия с библиотекой на C++ или Python. При этом на целевом устройстве будет находиться уже активированное в FPGA ядро DPU, предварительно подготовленное в загрузочной системе Petalinux.

8. Запуск и тестирование получившегося программно-аппаратного комплекса.

Вид ядра DPU показан на рисунке 3.9.

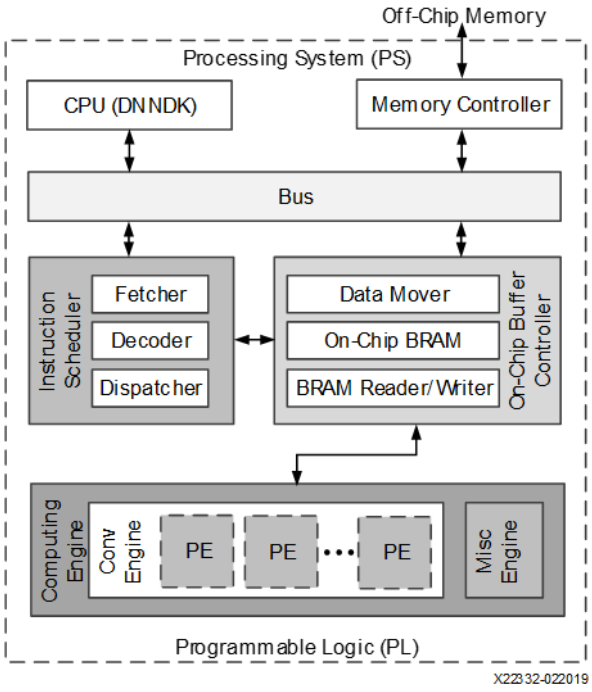

Рисунок 3.9 – Конструкция ядра DPU

Остановимся на пунктах 3–5. Они производятся в docker имидже Vitis-AI docker. Данный докер имидж содержит фреймворк tensorflow, который осуществляет необходимые операции в пакетном режиме. Кросс-компиляция ведется уже в другом докер имидже – runtime docker. На выходе мы получаем

библиотеку динамической компоновки с расширением .so, которая содержит все необходимые данные для загрузки нейросети в DPU и работы с ней.

3.4 Полученные результаты

На натренированной нейросети были протестированы 1486 черно-белых изображений [51] частей печатных плат с различными дефектами изготовления. Дефекты были разделены на 6 классов (таблица 3.1):

Таблица 3.1 – Классы дефектов, определяемых нейросетью

Класс дефекта	Описание дефекта	Комментарии
Open	Разрыв проводника	
Short	Закороченный контакт	КЗ двух разных проводников
Mousebit	Мышиное касание	Протравленная лакуна на проводнике
Spur	Шпора	Медная шпора на проводнике
Copper	Непротравленная область	
Pin-hole	Протравленная область в проводнике, не выходящая из проводника наружу	

Вид дефектов приведен на рисунке 3.10

При этом на этом рисунке мы также видим дефекты класса short и spur, которые нейросеть не обнаружила.

Для тестирования и работы с изображениями были написаны программы на Python для первичной подготовки и обработки набора тестовых изображения для приведения их в необходимый формат черно-белого изображения 8 бит с размером 640 на 640 пикселей.

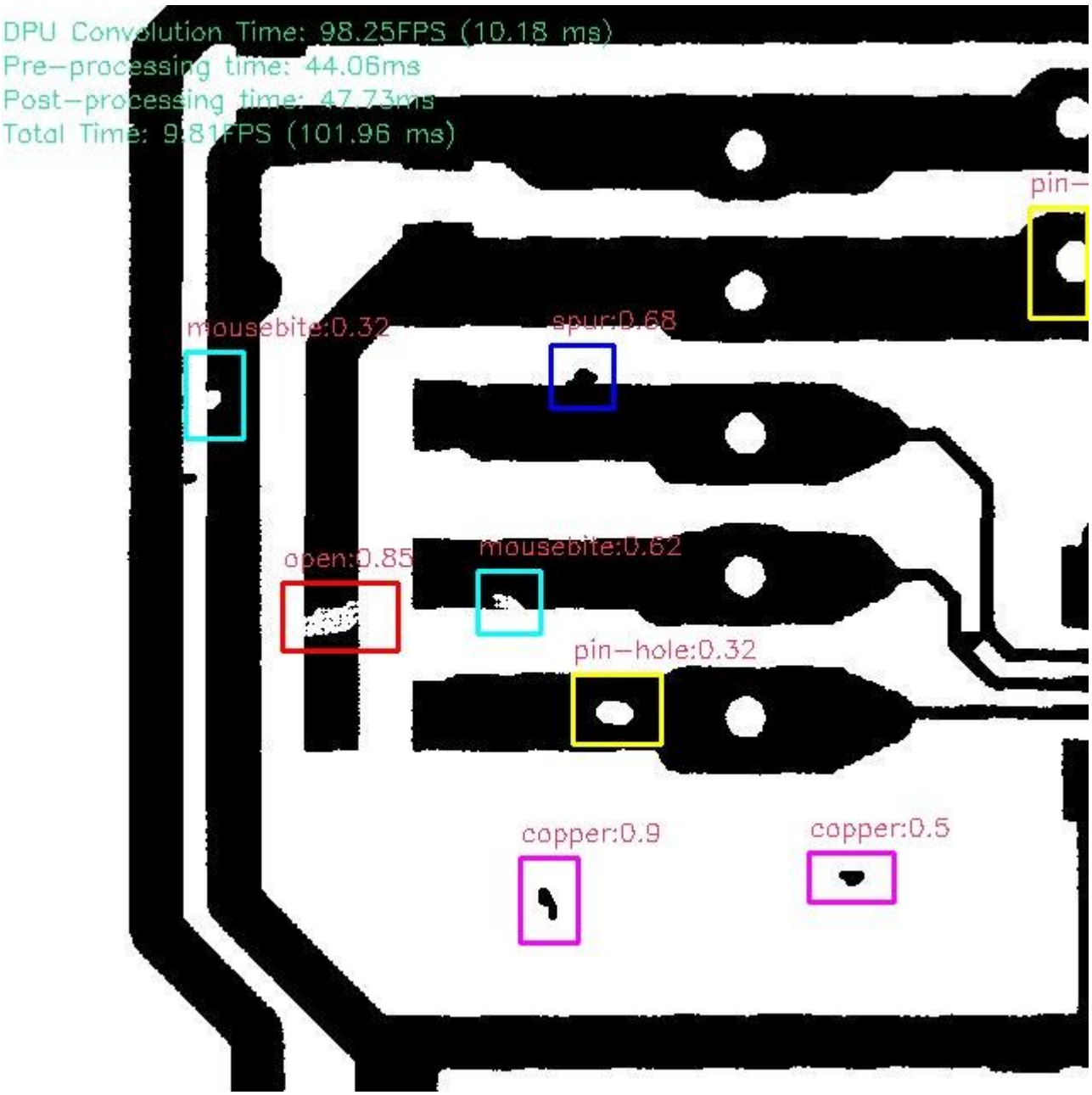

Рисунок 3.10 – Вид обработанного изображения с выделенными зонами дефектов

Программа для тестирования рассчитывает также время, которое необходимо на каждый этап обработки входного изображения. На рисунке 3.10 мы видим, что работа блока DPU занимает всего ~10 мс. В то же время время пре-процессинга ~44 мс, а пост-процессинга ~47 мс. Это можно объяснить тем, что мы работаем в режиме интерпретатора Python. Если перевести все программирование на C++, то можно ускорить обработку изображений до скорости, соизмеримой со скоростью работы DPU.

Проведена работа по подсчету качества распознавания дефектов в тестовом наборе изображений. Обработаны 1486 тестовых изображений, в которых нейросеть нашла дефекты и вычислено отношение найденных дефектов, к не найденным. Оно оказалось примерно равно 3/7 или 70%. Таким образом имеется большой потенциал для улучшения работы системы.

Ниже приведены части тестовой программы на Python для пояснения сущности работы системы.

Листинг 1:

```
# ----------------------Начало листинга ------------------------------

import time

import os

import cv2

import colorsys

import random

import numpy as np

from dnndk import n2cube

INPUT_IMAGE_PATH = 'datasets/PCB_USED/'

OBJECT_TYPE = 'pcb'

CLASSES_PATH = './model_data/classes_' + OBJECT_TYPE + '.txt'

ANCHORS_PATH = './model_data/yolo_anchors_' + OBJECT_TYPE + '.txt'

SCORE_THRESH = 0.3

NMS_THRESH = 0.3

# Изменение размера изображения, без изменения соотношения сторон

def letterbox_image(image, size):

    ih, iw, _ = image.shape

    w, h = size
```

```
    scale = min(w/iw, h/ih)
    #print("image scale:",scale)
    nw = int(iw*scale)
    nh = int(ih*scale)
    #print("image wide:",nw)
    #print("image height:",nh)
    image = cv2.resize(image, (nw,nh), interpolation=cv2.INTER_LINEAR)
    new_image = np.ones((h,w,3), np.uint8) * 128
    h_start = (h-nh)//2
    w_start = (w-nw)//2
    new_image[h_start:h_start+nh, w_start:w_start+nw, :] = image
    return new_image
# Подготовка изображения
def pre_process(image, model_image_size):
    image = image[...,::-1]
    image_h, image_w, _ = image.shape
    if model_image_size != (None, None):
        assert model_image_size[0]%32 == 0, 'Multiples of 32 required'
        assert model_image_size[1]%32 == 0, 'Multiples of 32 required'
        boxed_image = letterbox_image(image,
tuple(reversed(model_image_size)))
    else:
        new_image_size = (image_w - (image_w % 32), image_h - (image_h %
32))
        boxed_image = letterbox_image(image, new_image_size)
```

```
    image_data = np.array(boxed_image, dtype='float32')
    image_data /= 255.
    image_data = np.expand_dims(image_data, 0)
    return image_data
# Взятие модели классификации информации
def get_class(classes_path):
    with open(classes_path) as f:
        class_names = f.readlines()
    class_names = [c.strip() for c in class_names]
    return class_names

# Взятие модели величины якорей
'''Get model anchors value'''
def get_anchors(anchors_path):
    with open(anchors_path) as f:
        anchors = f.readline()
    anchors = [float(x) for x in anchors.split(',')]
    return np.array(anchors).reshape(-1, 2)

def output_fix(feats, num_classes):
    num_anchors = 3
    nu = num_classes + 5
    grid_size = np.shape(feats)[1 : 3]
    predictions = np.reshape(feats, [-1, grid_size[0], grid_size[1],
num_anchors, nu])
```

```
     # replace 0 with broken data in some output channel
    predictions[..., 0:1, :18] = np.zeros(shape=predictions[..., 0:1,
:18].shape)
    feats_fixed = np.reshape(predictions, feats.shape)

    # box_confidence = predictions[..., 4:5]
    # single_layer_box_confidence = box_confidence[..., 2:3,:]
    # print('\nOriginal output:{}'.format(feats.shape))
    # print('reshaped output:{}'.format(predictions.shape))
    # print('score output:{}'.format(box_confidence.shape))
    # print('single layer score
output:{}\n'.format(single_layer_box_confidence.shape))

    return feats_fixed

def _get_feats(feats, anchors, num_classes, input_shape):
    num_anchors = len(anchors)
    anchors_tensor = np.reshape(np.array(anchors, dtype=np.float32), [1, 1,
1, num_anchors, 2])
    grid_size = np.shape(feats)[1:3]
    nu = num_classes + 5
    predictions = np.reshape(feats, [-1, grid_size[0], grid_size[1],
num_anchors, nu])
    grid_y = np.tile(np.reshape(np.arange(grid_size[0]), [-1, 1, 1, 1]),
[1, grid_size[1], 1, 1])
```

```
    grid_x = np.tile(np.reshape(np.arange(grid_size[1]), [1, -1, 1, 1]),
[grid_size[0], 1, 1, 1])

    grid = np.concatenate([grid_x, grid_y], axis = -1)

    grid = np.array(grid, dtype=np.float32)

    box_xy = (1/(1+np.exp(-predictions[..., :2])) + grid) /
np.array(grid_size[::-1], dtype=np.float32)

    box_wh = np.exp(predictions[..., 2:4]) * anchors_tensor /
np.array(input_shape[::-1], dtype=np.float32)

    box_confidence = 1/(1+np.exp(-predictions[..., 4:5]))

    box_class_probs = 1/(1+np.exp(-predictions[..., 5:]))

    return box_xy, box_wh, box_confidence, box_class_probs

def correct_boxes(box_xy, box_wh, input_shape, image_shape):

    box_yx = box_xy[..., ::-1]

    box_hw = box_wh[..., ::-1]

    input_shape = np.array(input_shape, dtype = np.float32)

    image_shape = np.array(image_shape, dtype = np.float32)

    new_shape = np.around(image_shape * np.min(input_shape / image_shape))

    offset = (input_shape - new_shape) / 2. / input_shape

    scale = input_shape / new_shape

    box_yx = (box_yx - offset) * scale

    box_hw *= scale

    box_mins = box_yx - (box_hw / 2.)

    box_maxes = box_yx + (box_hw / 2.)

    boxes = np.concatenate([

        box_mins[..., 0:1],
```

```
        box_mins[..., 1:2],
        box_maxes[..., 0:1],
        box_maxes[..., 1:2]
    ], axis = -1)
    boxes *= np.concatenate([image_shape, image_shape], axis = -1)
    return boxes
def boxes_and_scores(feats, anchors, classes_num, input_shape,
image_shape):
    box_xy, box_wh, box_confidence, box_class_probs = _get_feats(feats,
anchors, classes_num, input_shape)
    boxes = correct_boxes(box_xy, box_wh, input_shape, image_shape)
    boxes = np.reshape(boxes, [-1, 4])
    box_scores = box_confidence * box_class_probs
    box_scores = np.reshape(box_scores, [-1, classes_num])
    return boxes, box_scores
# Отрисовка детектирующей рамки
def draw_bbox(image, bboxes, classes):
#   bboxes: [x_min, y_min, x_max, y_max, probability, cls_id] format
coordinates.
    num_classes = len(classes)
    image_h, image_w, _ = image.shape
    hsv_tuples = [(1.0 * x / num_classes, 1., 1.) for x in
range(num_classes)]
    colors = list(map(lambda x: colorsys.hsv_to_rgb(*x), hsv_tuples))
    colors = list(map(lambda x: (int(x[0] * 255), int(x[1] * 255), int(x[2]
* 255)), colors))
```

```
    random.seed(0)
    random.shuffle(colors)
    random.seed(None)
    for i, bbox in enumerate(bboxes):
        coor = np.array(bbox[:4], dtype=np.int32)
        fontScale = 0.5
        score = bbox[4]
        class_ind = int(bbox[5])
        bbox_color = colors[class_ind]
        bbox_thick = int((image_h + image_w) / 550)
        c1, c2 = (coor[0], coor[1]), (coor[2], coor[3])
        cv2.rectangle(image, c1, c2, bbox_color, bbox_thick)
    return image


def nms_boxes(boxes, scores):
    x1 = boxes[:, 0]
    y1 = boxes[:, 1]
    x2 = boxes[:, 2]
    y2 = boxes[:, 3]


    areas = (x2-x1+1)*(y2-y1+1)
    # sort the scores from minimun to maximun and invert order =>  '
maximun -> minimun '
    # output sub-number
    order = scores.argsort()[::-1]
```

```
    keep = []
    while order.size > 0:
        i = order[0] # maximun score sub-number and send it to 'keep'
        keep.append(i)
        xx1 = np.maximum(x1[i], x1[order[1:]])
        yy1 = np.maximum(y1[i], y1[order[1:]])
        xx2 = np.minimum(x2[i], x2[order[1:]])
        yy2 = np.minimum(y2[i], y2[order[1:]])
        w1 = np.maximum(0.0, xx2 - xx1 + 1)
        h1 = np.maximum(0.0, yy2 - yy1 + 1)
        inter = w1 * h1
        ovr = inter / (areas[i] + areas[order[1:]] - inter)
        # gets the index of the array within the threshold range
        inds = np.where(ovr <= NMS_THRESH)[0]
        order = order[inds + 1]
    return keep
 # Модель окончательной подготовки
'''Model post-processing'''
def eval(yolo_outputs, image_shape, class_names, anchors):
    anchor_mask = [[6, 7, 8], [3, 4, 5], [0, 1, 2]]
    boxes = []
    box_scores = []
    input_shape = np.shape(yolo_outputs[0])[1 : 3]
    input_shape = np.array(input_shape)*32
```

```
    # repare broken output data
    # yolo_outputs[2] = output_fix(yolo_outputs[2], len(class_names))
    for i in range(len(yolo_outputs)):
        _boxes, _box_scores = boxes_and_scores(yolo_outputs[i],
anchors[anchor_mask[i]], len(class_names), input_shape, image_shape)
        boxes.append(_boxes)
        box_scores.append(_box_scores)
    boxes = np.concatenate(boxes, axis = 0)
    box_scores = np.concatenate(box_scores, axis = 0)
    mask = box_scores >= SCORE_THRESH
    boxes_ = []
    scores_ = []
    classes_ = []
    for c in range(len(class_names)):
        class_boxes_np = boxes[mask[:, c]]
        class_box_scores_np = box_scores[:, c]
        class_box_scores_np = class_box_scores_np[mask[:, c]]
        nms_index_np = nms_boxes(class_boxes_np, class_box_scores_np)
        class_boxes_np = class_boxes_np[nms_index_np]
        class_box_scores_np = class_box_scores_np[nms_index_np]
        classes_np = np.ones_like(class_box_scores_np, dtype = np.int32) *
c
        boxes_.append(class_boxes_np)
        scores_.append(class_box_scores_np)
        classes_.append(classes_np)
```

```
    boxes_ = np.concatenate(boxes_, axis = 0)
    scores_ = np.concatenate(scores_, axis = 0)
    classes_ = np.concatenate(classes_, axis = 0)
    return boxes_, scores_, classes_
# Имя DPU ядра
KERNEL_CONV='tf_yolov3_'+ OBJECT_TYPE
# Имена входных/выходных слоев в DPU
CONV_INPUT_NODE="conv2d_1_convolution"
CONV_OUTPUT_NODE1="conv2d_59_convolution"
CONV_OUTPUT_NODE2="conv2d_67_convolution"
CONV_OUTPUT_NODE3="conv2d_75_convolution"

if __name__ == "__main__":

# Открытие DPU драйвера и подготовка его к запуску
  n2cube.dpuOpen()
# Загрузка DPU ядра tf_yolov3_xxx
    kernel = n2cube.dpuLoadKernel(KERNEL_CONV)
# Создание DPU задачи для tf_yolov3_xxx
    task = n2cube.dpuCreateTask(kernel, 0) # 1 = T_MODE_PROF; 0 = normal;
    # task = n2cube.dpuEnableTaskProfile(task)
# Взятие модели классификации информации
    class_names = get_class(CLASSES_PATH)
    NN_obj_info_dim = 3*(5+len(class_names))
```

```
# Взятие величин якорей
    anchors = get_anchors(ANCHORS_PATH)
    cnt=0
    while True:
        total_time_start = time.process_time() # <-------- Start total
Running Time Recording (1)
# Загрузка изображения в DPU
        cnt = cnt + 1
        image_name = 'img{:0>5d}.jpg'.format(cnt)
        image = cv2.imread(INPUT_IMAGE_PATH + image_name)
        if image is None:
            break
        preprocess_time = time.process_time() # <------ Start preprocess
time-----------
        image_ho, image_wo, _ = image.shape
        image_size = image.shape[:2]
        image_data = pre_process(image, (416, 416))
        image_data = np.array(image_data,dtype=np.float32)
        input_len = n2cube.dpuGetInputTensorSize(task, CONV_INPUT_NODE)
        preprocess_time = time.process_time() - preprocess_time # <--------
-- Stop preprocess time------
        conv_time_start = time.process_time() # <------------- Start
Convolution Time Recording (2)
# Взятие входного тензора
```

```
n2cube.dpuSetInputTensorInHWCFP32(task,CONV_INPUT_NODE,image_data,input_len
)
# Запуск модели в DPU
        n2cube.dpuRunTask(task)


        conv_sbbox_size = n2cube.dpuGetOutputTensorSize(task,
CONV_OUTPUT_NODE1)
        conv_out1 = n2cube.dpuGetOutputTensorInHWCFP32(task,
CONV_OUTPUT_NODE1, conv_sbbox_size)
        conv_out1 = np.reshape(conv_out1, (1, 13, 13, NN_obj_info_dim))
        conv_mbbox_size = n2cube.dpuGetOutputTensorSize(task,
CONV_OUTPUT_NODE2)
        conv_out2 = n2cube.dpuGetOutputTensorInHWCFP32(task,
CONV_OUTPUT_NODE2, conv_mbbox_size)
        conv_out2 = np.reshape(conv_out2, (1, 26, 26, NN_obj_info_dim))
        conv_lbbox_size = n2cube.dpuGetOutputTensorSize(task,
CONV_OUTPUT_NODE3)
        conv_out3 = n2cube.dpuGetOutputTensorInHWCFP32(task,
CONV_OUTPUT_NODE3, conv_lbbox_size)
        conv_out3 = np.reshape(conv_out3, (1, 52, 52, NN_obj_info_dim))
        conv_time = time.process_time()-conv_time_start # <------------
Stop Convolution Time Recording (2)
        #print('Convolution Speed: %s FPS'%(1/(time.process_time()-
conv_time_start)))
        yolo_outputs = [conv_out1, conv_out2, conv_out3]
# Окончательная подготовка
```

```
        out_boxes, out_scores, out_classes = eval(yolo_outputs, image_size,
class_names, anchors)
        items = []
        draws = []
        texted_image = image
        for i, c in reversed(list(enumerate(out_classes))):
            predicted_class = class_names[c]
            box = out_boxes[i]
            score = out_scores[i]
            top, left, bottom, right = box
            top = max(0, np.floor(top + 0.5).astype('int32'))
            left = max(0, np.floor(left + 0.5).astype('int32'))
            bottom = min(image_ho, np.floor(bottom + 0.5).astype('int32'))
            right = min(image_wo, np.floor(right + 0.5).astype('int32'))
            draw  = [left, top, right, bottom, score, c]
            item  = [predicted_class, score, left, top, right, bottom]
            draws.append(draw)
            items.append(item)
            texted_image = cv2.putText(image,
predicted_class+':'+str(round(score,2)), (left,top-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (92,51,255), 1)
        """ Running time calculate(over) and result print """
        run_time = time.process_time() - total_time_start # <-------------
Stop total Running Time Recording (1)
```

```
            texted_image = cv2.putText(texted_image, 'DPU Convolution Time'
+ ': '+ str(round(1/conv_time,2)) + 'FPS (' + str(round(conv_time*1000,2))
+' ms)', (0,20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (148,230,26), 1)

        texted_image = cv2.putText(texted_image, 'Pre-processing time' + ':
'+ str(round(preprocess_time * 1000,2)) + 'ms', (0,40),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (148,230,26), 1)

        texted_image = cv2.putText(texted_image, 'Post-processing time' +
': '+ str(round((run_time-preprocess_time-conv_time) * 1000,2)) + 'ms',
(0,60), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (148,230,26), 1)

        texted_image = cv2.putText(texted_image, 'Total Time' + ': '+
str(round(1/run_time,2)) + 'FPS (' + str(round(run_time*1000,2)) +' ms)',
(0,80), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (148,230,26), 1)

        image_result = draw_bbox(texted_image, draws, class_names)

        cv2.namedWindow(' YOLO-V3 ALINX '+ OBJECT_TYPE + ' Detection
',cv2.WINDOW_FREERATIO)

        cv2.imshow(' YOLO-V3 ALINX '+ OBJECT_TYPE + ' Detection
',image_result)

        k=cv2.waitKey(0)

        if k == ord("s"):

            cv2.imwrite('output_result/' + image_name, image_result)

            print('\noutput image: \n' + image_name)


        elif k == 27:

            cv2.destroyAllWindows()

            break

    cv2.destroyAllWindows()
# -------------------- Конец листинга -------------------------------
```

Вопросы и проблемы для дальнейшей разработки

Дальнейшее развитие проекта предполагает:

1. Переход с тестовых программ на python на программы на C++, cmake. Цель – ускорение работы устройства.

2. Эксперименты с пред-обработкой входного изображения, с целью усиления контрастности и выделения границ объектов с помощью методов библиотеки OpenCV. В частности, предполагается использование фильтров выделения границ типа фильтра Кэнни (Canny).

3. Наработка собственного датасета дефектов травления печатных плат и датасета дефектов печати полиграфической продукции.

4. Повышение качества работы нейросети за счет до-обучения и дополнительных методов предварительной обработки входящих изображений.

5. Считывание изображения в реальном масштабе времени и, соответственно, его предобработка и распознавание дефектов.

Нужно обратить особое внимание именно на наработку наборов для обучения нейросети. Формирование наборов данных занимает более 50% всего времени разработки и является ключевым моментов в области создания работоспособных нейросетей.

4 ПЕРСПЕКТИВЫ РАЗВИТИЯ СИСТЕМ МАШИННОГО ЗРЕНИЯ

4.1 Анализ развития систем ИИ и машинного зрения в России

Анализ развития систем искусственного интеллекта и машинного зрения в России за последнее десятилетие показывает значительные достижения и развитие в этой области. Искусственный интеллект и машинное зрение стали ключевыми направлениями развития в России, привлекая большое внимание как со стороны академического сообщества, так и бизнес-сектора.

Рассмотрим некоторые важные моменты и достижения за последние десять лет.

– Научные исследования и инновации: В России активно проводятся исследования и разработки в области искусственного интеллекта и машинного зрения. Множество научных институтов, университетов и инновационных центров активно вкладывают ресурсы и усилия в эту область. Было достигнуто значительное количество научных результатов, включая новые алгоритмы машинного обучения, разработку нейронных сетей и технологий глубокого обучения.

– Применение в промышленности и бизнесе: В последнее десятилетие Россия наблюдает активное применение систем искусственного интеллекта и машинного зрения в промышленности и бизнесе. Отрасли, такие как автомобильная промышленность, розничная торговля, медицина, робототехника и другие, используют системы машинного зрения для автоматизации процессов, улучшения качества и контроля продукции, оптимизации логистики и повышения эффективности.

– Государственная поддержка: в последние годы правительство России активно поддерживает развитие искусственного интеллекта и машинного зрения. Были разработаны и реализованы программы и инициативы, направленные на развитие и применение этих технологий в различных сферах экономики. Такие инициативы, как "Цифровая экономика" и "Стратегия развития искусственного

интеллекта", способствуют развитию инфраструктуры, финансированию исследований и обучению специалистов в этой области.

– Образование и академическая деятельность: В России развиваются образовательные программы, специализирующиеся на искусственном интеллекте и машинном зрении. Университеты и исследовательские центры предлагают специальности и курсы, позволяющие студентам углубиться в эти области и приобрести необходимые знания и навыки. Кроме того, проводятся конференции, семинары и мероприятия, на которых научные работники и специалисты обмениваются знаниями и опытом в области искусственного интеллекта и машинного зрения.

– Международное сотрудничество: Российские исследователи и компании активно сотрудничают с зарубежными партнерами в области искусственного интеллекта и машинного зрения. Участие в международных проектах и совместных исследованиях позволяет обмениваться опытом и учиться от лучших практик. Российские ученые и специалисты также регулярно представляют свои работы и результаты на международных конференциях и соревнованиях, что способствует распространению и признанию их достижений.

– Вызовы и перспективы: одновременно с достижениями и прогрессом в области искусственного интеллекта и машинного зрения в России остаются и некоторые вызовы. Одним из них является необходимость привлечения большего количества высококвалифицированных специалистов, обладающих глубокими знаниями и опытом в этой области.

– Также важно продолжать инвестировать в исследования и разработки, чтобы быть в лидирующих позициях на мировой арене искусственного интеллекта.

В целом, за последнее десятилетие Россия достигла значительного прогресса в области искусственного интеллекта и машинного зрения. Успехи в научных исследованиях, применении в промышленности и бизнесе, государственная поддержка, академическая деятельность и международное

сотрудничество свидетельствуют о перспективном развитии этой области в России.

Приведем конкретные примеры российских разработок:

– VisionLabs: Компания VisionLabs, базирующаяся в России, специализируется на разработке технологий компьютерного зрения и распознавания лиц. Они разработали продукты, такие как системы автоматической идентификации, системы видеонаблюдения и дроны с функцией распознавания объектов.

– Cognitive Technologies: Эта российская компания занимается разработкой систем машинного зрения для автомобилей. Они создали систему, способную распознавать дорожные знаки, пешеходов, другие автомобили и препятствия. Это позволяет повысить безопасность и эффективность автомобильного транспорта.

– Российские ученые активно исследуют и применяют системы ИИ и машинное зрение в медицине. Например, разработаны алгоритмы для автоматического анализа медицинских изображений, таких как снимки рентгена, КТ и МРТ, для обнаружения патологий и диагностики заболеваний.

– Автономная навигация: Российские компании работают над разработкой автономных систем, таких как беспилотные автомобили и роботы. Машинное зрение играет важную роль в этих системах, позволяя им воспринимать окружающую среду, распознавать дорожные знаки и препятствия, и принимать соответствующие решения и действия.

– Участие в международных соревнованиях: Российские исследователи активно участвуют в международных соревнованиях и конкурсах по машинному зрению, таким как соревнования по распознаванию изображений ImageNet и COCO. Российские команды и индивидуальные участники достигают заметных результатов и вносят вклад в развитие области.

Российские исследователи и компании продолжают делать значительные шаги вперед, применяя технологии машинного зрения в различных сферах и решая сложные задачи.

4.2 Перспективы внедрения ИИ и машинного зрения в производство

В последние десятилетия системы машинного зрения претерпели значительное развитие и сыграли важную роль в промышленности и науке. Машинное зрение представляет собой технологию, позволяющую компьютерным системам видеть и интерпретировать изображения или видео с использованием алгоритмов и моделей машинного обучения. Эта область имеет огромный потенциал и уже сейчас находит применение во многих отраслях, от автомобильной промышленности до медицины и робототехники. Рассмотрим некоторые перспективы развития систем машинного зрения.

– Развитие глубокого обучения и нейронных сетей: Глубокое обучение и нейронные сети сыграли ключевую роль в успехе систем машинного зрения. Однако существует потенциал для дальнейшего развития этих методов. Ожидается, что будут созданы более эффективные алгоритмы обучения и более глубокие архитектуры нейронных сетей, что приведет к улучшению точности и скорости распознавания изображений.

– Расширение области применения: в настоящее время системы машинного зрения широко применяются в промышленности для задач, таких как автоматическое распознавание и классификация объектов, контроль качества, автоматизация процессов и другие. В будущем ожидается, что машинное зрение будет использоваться в еще большем числе отраслей, включая сельское хозяйство, медицину, городское планирование, робототехнику и даже виртуальную и дополненную реальность.

– Прогнозирование отказов оборудования: Системы ИИ могут анализировать данные, собранные с производственного оборудования, и предсказывать возможные отказы или неисправности. Это позволяет про-активно планировать ремонтные работы, уменьшить простои и повысить надежность производства.

– Оптимизация производственных процессов: Нейросети и ИИ могут анализировать большие объемы данных, собранных с производственных линий, и

находить оптимальные параметры и настройки для улучшения производительности и экономии ресурсов. Это может включать оптимизацию расписания производства, управление запасами и оптимизацию энергопотребления.

– Роботизация и автоматизация: Нейросети и машинное зрение являются ключевыми компонентами роботизированных систем в производстве. Роботы, оснащенные системами машинного зрения, способны выполнять сложные задачи, такие как сортировка, сборка и упаковка, с высокой точностью и скоростью. Это позволяет снизить трудозатраты, повысить производительность и гибкость производства.

– Оптимизация логистики и складского хозяйства: Машинное зрение и нейросети могут быть использованы для автоматического распознавания, классификации и отслеживания товаров на складах и в логистических цепях. Это помогает улучшить управление запасами, минимизировать ошибки и сократить время поиска и отгрузки товаров.

– Развитие технологий дополненной реальности (AR): Машинное зрение является ключевым компонентом технологий дополненной реальности, которые уже находят широкое применение в сферах, таких как образование, развлечения и реклама. Однако существует потенциал для дальнейшего усовершенствования систем с помощью развития более точных и быстрых систем машинного зрения. Это позволит создавать еще более реалистичные и интерактивные виртуальные объекты в реальном времени.

– Автоматизация процессов: Системы машинного зрения способны обрабатывать большие объемы данных и выполнять сложные задачи автоматически. Благодаря этому они могут значительно повысить эффективность производства и сократить человеческий труд. В будущем ожидается, что системы машинного зрения будут все шире применяться в автоматизированных процессах, что приведет к увеличению производительности и снижению затрат.

– Развитие систем машинного зрения в медицине: В медицине системы машинного зрения уже находят применение в областях, таких как диагностика заболеваний, анализ медицинских изображений и мониторинг пациентов. В будущем ожидается, что системы машинного зрения будут еще шире применяться в медицине для улучшения точности диагностики, сокращения времени обработки и повышения качества ухода за пациентами.

– Безопасность и наблюдение: Системы машинного зрения играют важную роль в обеспечении безопасности и наблюдении в обществе. Они могут использоваться для обнаружения и распознавания лиц, анализа поведения людей, детектирования опасных ситуаций и многое другое. В будущем ожидается, что системы машинного зрения будут становиться более точными и эффективными, что поможет повысить общественную безопасность и снизить преступность.

Это лишь некоторые примеры применения нейросетей, ИИ и машинного зрения в производстве в России. С развитием технологий и ростом исследований в этой области, можно ожидать ещё большего расширения и применения этих технологий в различных секторах промышленности.

Системы машинного зрения имеют огромный потенциал для развития и применения в различных областях промышленности и науки. С развитием глубокого обучения, расширением области применения, усовершенствованием технологий дополненной реальности, автоматизацией процессов, применением в медицине и обеспечении безопасности, системы машинного зрения будут продолжать преображать наш мир в более дружелюбное и безопасное место.

Но все будет зависеть исключительно от того, как общество сможет взаимодействовать с этими новыми инструментами и технологиями. Насколько оно сможет употребить их себе во благо, а не во зло.

ЗАКЛЮЧЕНИЕ

В выпускной квалификационной работе были рассмотрены возможности построения системы машинного зрения на основе нейросети на отладочной плате Alinx AXU2CGA. Описан типовой пример ускорителя DCNN (Deep Convolutional Neural Network), позволяющий ускорить процесс распознавания образов. С использованием фреймворка Vitis-AI. Проведены все этапы маршрута проектирования от создания архитектуры нейросети до конечной реализации в отладочной плате. Предварительно протестирован образец устройства для распознавания дефектов травления печатных плат в условиях производства.

Предложены направления дальнейшего совершенствования работы системы.

Продемонстрирована возможность и целесообразность использования данного оборудования для решения задачи нахождения дефектов печати полиграфической продукции и дефектов при производстве печатных плат.

Наша работа показала, что нейросети, особенно глубокие сверточные нейронные сети, обладают значительным потенциалом для точного и высокоскоростного распознавания различных образов.

Были рассмотрены различные архитектуры нейросетей, такие как AlexNet, VGGNet, Inception и ResNet. Мы обнаружили, что более глубокие и сложные модели позволяют достичь лучших результатов. Эти нейросети способны автоматически изучать иерархические признаки изображений и выявлять сложные зависимости между ними, что существенно улучшает их способность к распознаванию.

Были также рассмотрены методы предварительной обработки данных, включая изменение размера изображений, нормализацию и аугментацию данных. Эти методы играют важную роль в улучшении обучения нейросетей, улучшении их обобщающей способности и снижении эффекта переобучения.

Были провели эксперименты на различных наборах данных, которые показали достаточно хорошие характеристики точности распознавания.

Эти результаты подтверждают, что нейросети являются мощным инструментом для распознавания образов и могут превзойти традиционные методы в этой области.

Однако, несмотря на хорошее качество распознавания нейросети Yolo V3, остаются некоторые вызовы и направления для дальнейших исследований.

Одно из них – повышение процента распознавания дефектов. Нам видится, что одно из решений этой проблемы, дополнительная предобработка изображения для усиления границ дефектов. Этого можно достигнуть применением встроенных фильтров библиотеки OpenCV типа фильтра Кэнни (Canny) и подобных.

Кроме того, возможна разработка более эффективных архитектур нейросетей и методов обучения, а также работа с другими наборами данных.

На подготовку наборов данных в будущем планируется обратить особое внимание. В частности, в данного время нарабатывается датасет по дефектам при изготовлении полиграфической продукции типа рекламных проспектов, открыток, и другой полиграфии. В следствии небольшого размера устройства и хороших возможностей распознавания, его можно использовать также в других областях деятельности, например для оперативного опознания лиц, в системах безопасности или наличия спецодежды у рабочих на производстве.

В целом, наша работа подтверждает, что нейросети являются мощным инструментом для распознавания образов и дает многообещающие перспективы для дальнейших исследований в этой области. С ростом доступности вычислительных ресурсов и развитием новых методов, мы можем ожидать дальнейшего прогресса в области распознавания образов и его применения в различных сферах жизни.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Рейнхард Клетте Компьютерное зрение. Теория и алгоритмы / пер. с англ. А.А. Слинкин. – М.: ДМК Пресс, 2019. – 506 с.: ил.

2 Open Computer Vision Library [Электронный ресурс] URL: http://opencv.org (дата обращения 20.04.2023)

3 Catherine Holloway. Raspberry Pi to FPGA Communication, 15 November 2016 [Электронный ресурс] URL: https://catherineh.github.io/programming/2016/11/15/raspberry-pi-to-fpga-communication-example (дата обращения 20.04.2023)

4 Что такое Zynq? Краткий обзор. 28.06.2020. Хабр. Блоги. [Электронный ресурс] URL: https://habr.com/ru/post/508292 (дата обращения 20.04.2023)

5 Интернет магазин marsohod.org [Электронный ресурс] URL: https://marsohod.org/shop/boards/marsohod2rpi (дата обращения 20.04.2023)

6 FPGA плата к Raspberry Pi. 27.11.2017. Хабр. Блоги. [Электронный ресурс] URL: https://habr.com/ru/articles/408645/ (дата обращения 20.04.2023)

7 История развития ПЛИС. Лаборатория Параллельных информационных технологий. НИВЦ МГУ [Электронный ресурс] URL: https://parallel.ru/fpga/FPGA_history.html (дата обращения 20.04.2023)

8 Процессоры Intel Xeon оснастили FPGA Altera. 27.04.2016. Хабр. Блоги. [Электронный ресурс] URL: https://habr.com/ru/company/intel/blog/282570/ (дата обращения 20.04.2023)

9 Мокеев В.В. Моделирование бизнес-процессов в среде BPWIN. Челябинск: Изд. ЮУрГУ, 2010. – 76 с.

10 Мокеев В.В. Куликов Д.С. Методология моделирование бизнес-процессов. Челябинск: Изд. ЮУрГУ, 2013. – 120 с.

11 База знаний Экспоненты - вебинары, видео, статьи и другие материалы по модельно-ориентированному проектированию. [Электронный ресурс] URL: http://exponenta.ru

12 Самые яркие проекты по созданию нейроморфных процессоров. 27.01.2022. Хабр. Блоги [Электронный ресурс] URL: https://habr.com/ru/companies/yadro/articles/648119(дата обращения 20.04.2023)

13 An OpenCL-based FPGA Accelerator for Convolutional Neural Networks (github.com) [Электронный ресурс] URL: https://github.com/doonny/PipeCNN.

14 Поллак, Г.А. Интеллектуальные информационные системы: учебное пособие / Г.А. Поллак – Челябинск: Издательский центр ЮУрГУ, 2016. – 136 с.

15 Нейронные сети [Электронный ресурс] URL: asozykin.ru (дата обращения 20.04.2023)

16 Наглядно о том, как работает свёрточная нейронная сеть. [Электронный ресурс] URL: https://habr.com/ru/companies/skillfactory/articles/565232/

17 C. Zhang, P. Li, G. Sun, Y. Guan, B. J. Xiao, and J. Cong, “Optimizing FPGA-based accelerator design for deep convolutional neural networks,” in Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA ’15), 2015.

18 N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. F. Ma, S. Vrudhula, J. S.Seo, and Y. Cao, “Throughput-Optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks,” in Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA’16), 2016.

19 J. Qiu, J. Wang, S. Yao and et al., “Going deeper with embedded FPGA platform for convolutional neural network,” in Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA’16), 2016.

20 C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie and X. Zhou, “DLAU: a scalable deep learning accelerator unit on FPGA,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2016.

21 Аппаратное ускорение глубоких нейросетей: GPU, FPGA, ASIC, TPU, VPU, IPU, DPU, NPU, RPU, NNP и другие буквы. 10.06.2019. Хабр. Блоги. [Электронный ресурс] URL: https://habr.com/ru/articles/455353/ (дата обращения 20.04.2023)

22 Kendryte K210. Описание процессора.[Электронный ресурс] URL: https://www.canaan.io/product/kendryteai (дата обращения 20.04.2023)

23 Встраиваемые системы с Jetson. Портал nVidia [Электронный ресурс] URL: https://www.nvidia.com/ru-ru/autonomous-machines/embedded-systems/ (дата обращения 20.04.2023)

24 CUVI Lib, CUDA Vision & Imaging Library. [Электронный ресурс] URL: http://www.cuvilib.com

25 NPP, NVIDIA Performance Primitives. [Электронный ресурс] URL: http://developer.nvidia.com/object/npp_home.html (дата обращения 20.04.2022)

26 NVIDIA Jetson Nano. Портал компании nVidia. [Электронный ресурс] URL: https://www.nvidia.com/ru-ru/autonomous-machines/embedded-systems/jetson-nano (дата обращения 20.04.2022)

27 NVIDIA Jetson TX2. Портал компании nVidia. [Электронный ресурс] URL: https://www.nvidia.com/ru-ru/autonomous-machines/embedded-systems/jetson-tx2/ (дата обращения 20.04.2022)

28 NVIDIA Jetson Xavier NX. Портал компании nVidia. [Электронный ресурс] URL: https://www.nvidia.com/ru-ru/autonomous-machines/embedded-systems/jetson-xavier-nx/ (дата обращения 20.04.2022)

29 NVIDIA Jetson AGX Xavier. Портал компании nVidia. [Электронный ресурс] URL: https://www.nvidia.com/ru-ru/autonomous-machines/embedded-systems/jetson-agx-xavier/ (дата обращения 20.04.2022)

30 NVIDIA Jetson Xavier NX Developer Kit. Портал компании nVidia. [Электронный ресурс] URL: https://www.nvidia.com/ru-ru/autonomous-machines/embedded-systems/jetson-xavier-nx/ (дата обращения 20.04.2022)

31 Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit. Портал компании Xilinx. [Электронный ресурс] URL: https://www.xilinx.com/products/boards-and-kits/zcu104.html (дата обращения 20.04.2022)

32 Xilinx Zynq-7000 All Programmable SoC ZC702 Evaluation Kit. Портал компании Xilinx. [Электронный ресурс] URL: https://docs.xilinx.com/v/u/1.2.1-English/UG926_Z7_ZC702_Eval_Kit (дата обращения 20.04.2022)

33 Alveo U280 Data Center Accelerator Card User Guide (UG1314). Портал компании Xilinx. [Электронный ресурс] URL: https://docs.xilinx.com/r/en-US/ug1314-alveo-u280-reconfig-accel (дата обращения 20.04.2022)

34 ALINX AXU9EG: Xilinx Zynq UltraScale+ MPSoC XCZU9EG FPGA Development board High-end Integrated Deve. Портал компании Xilinx. [Электронный ресурс] URL: https://www.xilinx.com/products/boards-and-kits/1-1ervp7b.html

35 ALINX AXU2CGB: Xilinx Zynq UltraScale+ MPSOC XCZU2CG FPGA Development board AI Artificial Intelligen. Портал компании Xilinx. [Электронный ресурс] URL: https://www.xilinx.com/products/boards-and-kits/1-1eu3go3.html

36 FZ3 Kit - for Deep Learning FZ3 Card (industrial grade). Портал компании Xilinx. [Электронный ресурс] URL: https://www.xilinx.com/products/boards-and-kits/1-1bt1jlt.html

37 Vitis-AI-foudation (V1.2.4) 2021.5.28, [Электронный ресурс] URL: https://alinx.com/en/detail/490 (дата обращения 09.06.2023)

38 OpenVINO. Техническая документация. [Электронный ресурс] URL: https://docs.openvino.ai/2022.3/home.html (дата обращения 20.04.2023)

39 Что такое OpenVINO? 15.04.2021. Хабр. Блоги. [Электронный ресурс] URL: https://habr.com/ru/companies/intel/articles/546438/ (дата обращения 20.04.2023)

40 Raspberry Pi 4. Техническая документация. Сайт разработчика. [Электронный ресурс] URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ (дата обращения 20.04.2023)

41 Сайт компании Terasic. Описание отладочной платы TR10a-HL Arria 10 FPGA Development Kit. [Электронный ресурс] URL: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=228&No=1074#contents (дата обращения 20.04.2023)

42 Intel Neural Compute Stick. Искусственный разум на флешке-2. 22.11.2018. Хабр. Блоги. [Электронный ресурс] URL: https://habr.com/ru/companies/intel/articles/430492 (дата обращения 20.04.2023)

43 Документация на фреймворк Vitis-AI. Vitis AI User Guide. UG1414 (v1.2) July 7, 2020 www.xilinx.com (дата обращения 20.04.2023)

44 Venieris S.I., Bouganis Ch-S. fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs. [Электронный ресурс]: https://steliosven10.github.io/papers/sv2016fccm.pdf (дата обращения 06.06.2023)

45 Хорошайлова М.В Реализация нейронной сети на ПЛИС с использованием аппаратных ресурсов: Вестник Воронежского государственного технического университета. Т. 17. № 3. 2021

46 Большое сравнение 400 нейронных сетей для задачи классификации на более 8000 классов. 18.05.2022. Хабр. Блоги. [Электронный ресурс] URL: https://habr.com/ru/articles/666314/ (дата обращения 20.04.2023)

47 Комплексное руководство по сверточным нейронным сетям для чайников [Электронный ресурс] URL: https://datastart.ru/blog/read/kompleksnoe-rukovodstvo-po-svertochnym-neyronnym-setyam-dlya-chaynikov (дата обращения 20.04.2023)

48 A. Krizhevsky, I. Sutskever, G. E. Hinton and et al., “ImageNet classification with deep convolutional neural networks,” in Proc. Neural Information Processing Systems (NIPS’12), 2012.

49 K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv:1409.1556, 2014.

50 Обучение YOLOv5 на кастомном датасете. 05.05.2023. Хабр. Блоги. [Электронный ресурс] URL: https://habr.com/ru/articles/733536/ (дата обращения 20.05.2023)

51 Weibo Huang, Peng Wei A PCB Dataset for Defect Detection and Classification [Электронный ресурс] URL: https/arxiv.org/abs/1901.08204 (дата обращения 6.06.2023)

52 А. Бакшеев, В. Виноградов, В. Ерухимов, К. Корняков, А.Спижевой Itseez Ltd, Использование GPU для решения задач компьютерного зрения в библиотеке OpenCV. [Электронный ресурс] URL: https://agora.guru.ru/hpc-h/files/Using_GPU_for_computer_vision_in_OpenCV_library.pdf

53 OpenVidia: Parallel GPU Computer Vision. [Электронный ресурс] URL: https://openhub.net/p/openvidia (дата обращения 20.04.2022)

54 D. Chen, D. Singh INVITED PAPER: USING OPENCL TO EVALUATE THE EFFICIENCY OF CPUS, GPUS AND FPGAS FOR INFORMATION FILTERING. Altera Toronto Technology Center Toronto, Ontario, Canada, dsingh@altera.com

55 Н. Завьялов. Графические процессоры в решении современных IT-задач. [Электронный ресурс] URL: https://selectel.ru/blog/gpu-for-business/ (дата обращения 20.04.2023)

56 Дональд Томас Логическое проектирование и верификация систем в SystemVerilog / пер. с анг. А.А. Слинкина, А.С. Камкина, М.М. Чупилко; науч. ред. А.С. Камкин, М.М. Чупилко. – М.: ДМК Пресс, 2019. – 384 с.: ил.

57 Цифровой синтез: практический курс / под общ. ред. А.Ю. Романова, Ю.В. Панчула. – М.: ДМК Пресс, 2020. – 556 с.: ил.

58 Дэвид М. Харрис, Сара Л. Харрис Цифровая схемотехника и архитектура компьютера. / пер. с англ. Imagination Technologies. – М.: ДМК Пресс, 2017. – 772 с.: ил.

59 Дэвид М. Харрис, Сара Л. Харрис Цифровая схемотехника и архитектура компьютера. Дополнения по архитектуре ARM / пер. с англ. Слинкин А. А. / науч. ред. Косолобов Д. А. – М.: ДМК Пресс, 2019. – 356 с.: ил.

60 Дейтел Пол, Дейтел Харви Python: Искусственный интеллект, большие данные и облачные вычисления. – СПб.:Питер, 2020. – 864 с.: ил.

61 Ciletti M., Advanced Digital Design with the Verilog HDL, 2nd ed. Prentice Hall, 2010