Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Робототехника и комплексная автоматизация»

КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

«Методы анализа распознавания изображения с использованием FPGA – платы»

Студент РК6-82Б _____ **Караф С.М.**
(Группа) (Подпись, дата) (Фамилия И.О.)

Руководитель ВКР _____ **Витюков Ф.А.**
(Подпись, дата) (Фамилия И.О.)

Нормоконтролёр _____ **Грошев С.В.**
(Подпись, дата) (Фамилия И.О.)

2022 г.

АННОТАЦИЯ

Данная работа посвящена разработке нейронной сети с использованием фреймворка PyTorch для эффективного распознавания изображений. Основная цель исследования заключается в сравнении разработанной модели с базовой моделью, предоставленной PyTorch, а также с аналогичной моделью, реализованной для работы на плате FPGA. В рамках работы проведен анализ датасетов с учителем и без учителя, а также рассмотрены сверточные нейронные сети, применяемые в компьютерном зрении.

Практическая часть включает создание собственного датасета, состоящего из изображений с разметкой. Далее разработана архитектура нейронной сети, оптимизированная для задачи распознавания изображений, и проведено ее обучение на созданном датасете с использованием PyTorch. Для сравнения производительности моделей, разработанная нейронная сеть протестирована на наборе тестовых изображений, а также выполнено сравнение с моделью, адаптированной для работы на плате FPGA. При сравнении учитывались такие показатели, как точность распознавания, время обработки.

Полученные результаты позволяют сделать выводы о преимуществах и недостатках разработанной нейронной сети в сравнении с базовой моделью PyTorch и моделью на плате FPGA. Оптимальная архитектура нейронной сети и использование специализированной платформы могут значительно повысить производительность и эффективность распознавания изображений.

Тип работы: выпускная квалификационная работа.

Тема работы: «Методы анализа распознавания изображения с использованием FPGA – платы».

Объекты исследований: нейронные сети, распознавание изображений, PyTorch, FPGA, датасет, сверточные нейронные сети, производительность, точность распознавания.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Распознавание изображений с использованием нейронных сетей представляет собой активно развивающуюся область в области компьютерного зрения. Возможность компьютерных систем эффективно классифицировать объекты на изображениях на уровне или даже превосходящем человеческие возможности имеет широкий спектр практических применений, таких как автоматическая обработка изображений, медицинская диагностика, робототехника, автономные транспортные средства и другие.

Актуальность исследования обусловлена значимостью точного и эффективного распознавания изображений во множестве приложений. Однако сложность этой задачи требует разработки и оптимизации мощных моделей машинного обучения. В этом контексте фреймворк PyTorch заслуживает особого внимания, поскольку он обеспечивает гибкость и высокую производительность при разработке и обучении нейронных сетей. PyTorch предоставляет широкий набор инструментов и функций, таких как автоматическое дифференцирование, оптимизаторы и предварительно обученные модели, что упрощает и ускоряет процесс создания и обучения нейронных сетей для распознавания изображений.

Однако, несмотря на преимущества PyTorch, существуют и другие подходы к выполнению задач распознавания изображений. Одним из таких подходов является использование платы FPGA (Field-Programmable Gate Array). FPGA представляет собой программируемое логическое устройство, которое может быть настроено для выполнения специализированных вычислительных задач. Использование FPGA для выполнения задач распознавания изображений может привести к улучшению скорости обработки и оптимизации энергопотребления.

Поэтому данная работа нацелена на разработку нейронной сети для распознавания изображений с использованием фреймворка PyTorch и ее сравнение с моделью, специально разработанной для платы FPGA. В ходе исследования будет создан и обучен датасет, разработана архитектура нейронной сети, а затем сравнены результаты работы модели на PyTorch и FPGA.

Это позволит оценить эффективность и сравнительные преимущества обоих подходов в контексте задачи распознавания изображений.

Таким образом, данное исследование имеет значимость для развития области компьютерного зрения и может привести к улучшению производительности и эффективности систем распознавания изображений с использованием нейронных сетей и плат FPGA.

Исследование посвящено разработке нейронной сети для распознавания изображений с использованием фреймворка PyTorch и сравнению ее с моделью, разработанной для платы FPGA. В ходе исследования была проведена аналитическая часть, в рамках которой были изучены различные датасеты с учителем и без учителя, сверточные нейронные сети и возможности платы FPGA.

Практическая часть работы включала создание датасета для обучения нейронной сети в задаче распознавания людей на изображениях. Затем была разработана архитектура нейронной сети с использованием сверточных слоев, пулинга и полносвязных слоев. Обучение нейронной сети проводилось на созданном датасете с использованием алгоритма обратного распространения ошибки и метода градиентного спуска. Далее была рассмотрена совместимость разработанной нейронной сети с платой FPGA с помощью инструмента FiftyOne.

В ходе исследования была проведена оптимизация работы модели нейронной сети на плате FPGA с целью повышения ее эффективности. Были применены методы оптимизации алгоритмов обработки изображений, использованы специфические функции и возможности FPGA, а также проведена настройка параметров модели.

Ожидаемыми результатами исследования являются выводы о преимуществах и недостатках разработанной нейронной сети и модели на плате FPGA. Ожидается, что модель на плате FPGA обладает высокой скоростью обработки изображений и эффективным использованием ресурсов. В то же время, разработанная нейронная сеть на основе фреймворка PyTorch может обеспечить высокую точность в распознавании изображений. Таким образом,

данное исследование имеет научную и практическую значимость, поскольку способствует развитию области компьютерного зрения и оптимизации систем распознавания изображений с использованием нейронных сетей и плат FPGA.

Цели работы:

1. Анализ сверточных сетей для понимания их принципов и применения в задачах распознавания изображений.
2. Исследование плат FPGA для понимания их возможностей и применимости в области обработки изображений.
3. Создание собственного датасета для распознавания людей на изображениях.
4. Разработка, обучение и тестирование сверточной нейронной сети на основе созданного датасета.
5. Проверка совместимости модели с платой FPGA.
6. Оценка эффективности работы нейронной сети на плате FPGA и выявление ее преимуществ и ограничений в задаче распознавания изображений.

Для выполнения работы были использованы средства следующих программ:

- PyTorch - фреймворк для глубокого обучения.
- COCO Dataset - набор данных с размеченными изображениями.
- FiftyOne - инструмент для аннотирования и визуализации данных компьютерного зрения.
- ONNX - формат представления моделей и нейронных сетей.
- Quartus Prime - среда разработки для плат FPGA.
- VHDL - язык описания аппаратуры для плат FPGA.

1. Теоретическая часть

1.1. Сверточная нейронная сеть

Сверточные нейронные сети (СНС) — это специализированный тип нейронных сетей, которые используют свертку вместо общего матричного умножения по крайней мере в одном из своих слоев. или, другими словами, Сверточная нейронная сеть (CNN) - это тип искусственной нейронной сети, используемой в распознавании и обработке изображений, которая специально предназначена для обработки пиксельных данных.

В обычном перцептроне, который представляет собой полносвязную нейронную сеть, каждый нейрон связан со всеми нейронами предыдущего слоя, причём каждая связь имеет свой персональный весовой коэффициент. В свёрточной нейронной сети в операции свёртки используется лишь ограниченная матрица весов небольшого размера, которую «двигают» по всему обрабатываемому слою (в самом начале — непосредственно по входному изображению), формируя после каждого сдвига сигнал активации для нейрона следующего слоя с аналогичной позицией. То есть для различных нейронов выходного слоя используются одна и та же матрица весов, которую также называют ядром свёртки. Её интерпретируют как графическое кодирование какого-либо признака, например, наличие наклонной линии под определённым углом. Тогда следующий слой, получившийся в результате операции свёртки такой матрицей весов, показывает наличие данного признака в обрабатываемом слое и её координаты, формируя так называемую карту. Естественно, в свёрточной нейронной сети набор весов не один, а целая гамма, кодирующая элементы изображения (например линии и дуги под разными углами). При этом такие ядра свёртки не закладываются исследователем заранее, а формируются самостоятельно путём обучения сети классическим методом обратного распространения ошибки. Проход каждым набором весов формирует свой собственный экземпляр карты признаков, делая нейронную сеть многоканальной (много независимых карт признаков на одном слое). Также следует отметить, что при переборе слоя матрицей весов её передвигают обычно не на полный шаг

(размер этой матрицы), а на небольшое расстояние. Так, например, при размерности матрицы весов 5×5 её сдвигают на один или два нейрона (пикселя) вместо пяти, чтобы не «перешагнуть» искомый признак.[6]

Операция субдискретизации, выполняет уменьшение размерности сформированных карт признаков. В данной архитектуре сети считается, что информация о факте наличия искомого признака важнее точного знания его координат, поэтому из нескольких соседних нейронов карты признаков выбирается максимальный и принимается за один нейрон уплотнённой карты признаков меньшей размерности. За счёт данной операции, помимо ускорения дальнейших вычислений, сеть становится более инвариантной к масштабу входного изображения.

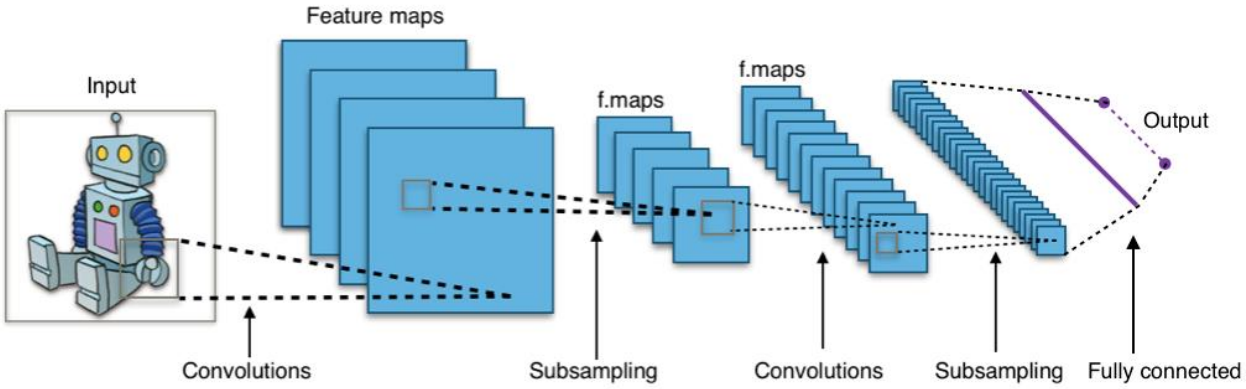
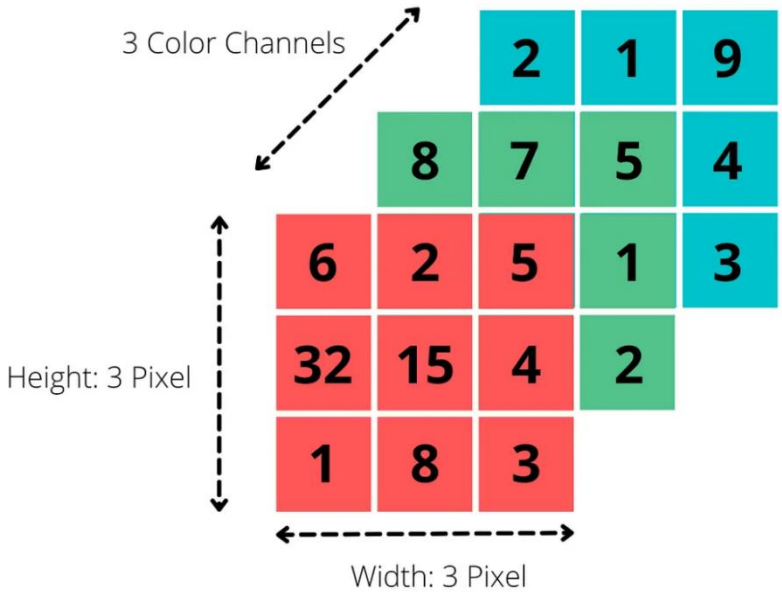


Рис. 1. Типовая архитектура свёрточной нейронной сети

Для компьютера изображение в формате RGB представляет собой суммарную информацию трех разных матриц. Для каждого пикселя изображения определяется его цвет. Красная компонента описывается в первой матрице, зеленая компонента - во второй, а синяя - в последней. Таким образом, для изображения размером 3 на 3 пикселя мы получаем три различные матрицы размером 3x3.

Для обработки изображения каждый пиксель подается на вход нейронной сети. Таким образом, для изображения размером 200x200x3 (т.е. 200 пикселей по ширине, 200 пикселей по высоте и 3 цветовых канала: красный, зеленый и синий) нам нужно предоставить 200 * 200 * 3 = 120 000 входных нейронов. Затем каждая

матрица имеет размер 200 на 200 пикселей, то есть в общей сложности 200 * 200 записей. Эта матрица затем существует в трех экземплярах, по одному для красного, синего и зеленого цветов. Проблема возникает в первом скрытом слое,



потому что каждый нейрон в этом слое должен иметь 120 000 весов от входного слоя. Это означает, что количество параметров будет очень быстро увеличиваться с увеличением числа нейронов в скрытом слое.

Рис. 2. Изображение 3x3x3 RGB.

Эта проблема усугубляется, когда мы хотим обрабатывать более крупные изображения с большим количеством пикселей и цветовых каналов. Такая сеть с огромным количеством параметров, скорее всего, столкнется с переобучением. Это означает, что модель будет давать хорошие предсказания для обучающего набора данных, но не будет хорошо обобщаться на новые случаи, с которыми она еще не сталкивалась. Кроме того, из-за большого количества параметров сеть, скорее всего, перестанет обращать внимание на отдельные детали изображения, так как они будут потеряны в общей массе. Однако, если мы хотим классифицировать изображение, например, определить, есть ли на нем собака или нет, эти детали, такие как нос или уши, могут быть решающим фактором для правильного результата.

Поэтому сверточная нейронная сеть идет по-другому пути, имитируя то, как мы воспринимаем окружающую среду с помощью глаз. Когда мы видим изображение, мы автоматически разделяем его на множество маленьких подизображений и анализируем их поочередно. Собирая эти подизображения, мы обрабатываем и интерпретируем изображение. Как можно реализовать этот принцип в сверточной нейронной сети?

Работа происходит в так называемом сверточном слое. Для этого мы определяем фильтр, который определяет размер частичных изображений, которые мы рассматриваем, и шаг, который определяет, на сколько пикселей мы продолжаем между вычислениями, т.е. насколько близко друг к другу располагаются частичные изображения. Благодаря этому шагу мы значительно сокращаем размерность изображения.

Следующий шаг - слой пулинга. С точки зрения вычислений здесь происходит то же самое, что и в сверточном слое, с той разницей, что мы берем только среднее или максимальное значение из результата в зависимости от задачи. Это сохраняет маленькие особенности в нескольких пикселях, которые являются ключевыми для решения задачи.[7][8]

Наконец, есть полносвязный слой, как мы уже знаем из обычных нейронных сетей. Теперь, когда мы значительно сократили размеры изображения, мы можем использовать плотно связанные слои. Здесь отдельные подизображения связываются снова, чтобы распознать связи и выполнить классификацию.

1.1.1. Сверточный слой (Convolution Layer) и Слой pooling

Сверточный слой (Convolution Layer) - это ключевой компонент сверточных нейронных сетей, предназначенный для извлечения признаков и шаблонов из входных изображений. Он применяет операцию свертки между фильтрами (ядрами) и входным изображением, позволяя выделять локальные особенности и структурные характеристики изображения.[5]

В сверточном слое фильтры сканируют пиксели входного изображения, перемещаясь по нему с определенным шагом. Каждое значение в выходной матрице сверточного слоя представляет результат операции свертки фильтра с соответствующей подматрицей изображения. Это позволяет обнаруживать различные признаки, такие как контуры, углы, текстуры и другие локальные особенности, независимо от их расположения в изображении.

Сверточный слой играет важную роль в анализе и обработке изображений, позволяя нейронным сетям автоматически выделять значимые признаки, которые могут быть использованы для классификации, распознавания объектов, сегментации изображений и других задач компьютерного зрения. Он помогает сети учиться находить важные шаблоны и паттерны, что делает его основным инструментом в области обработки визуальной информации.

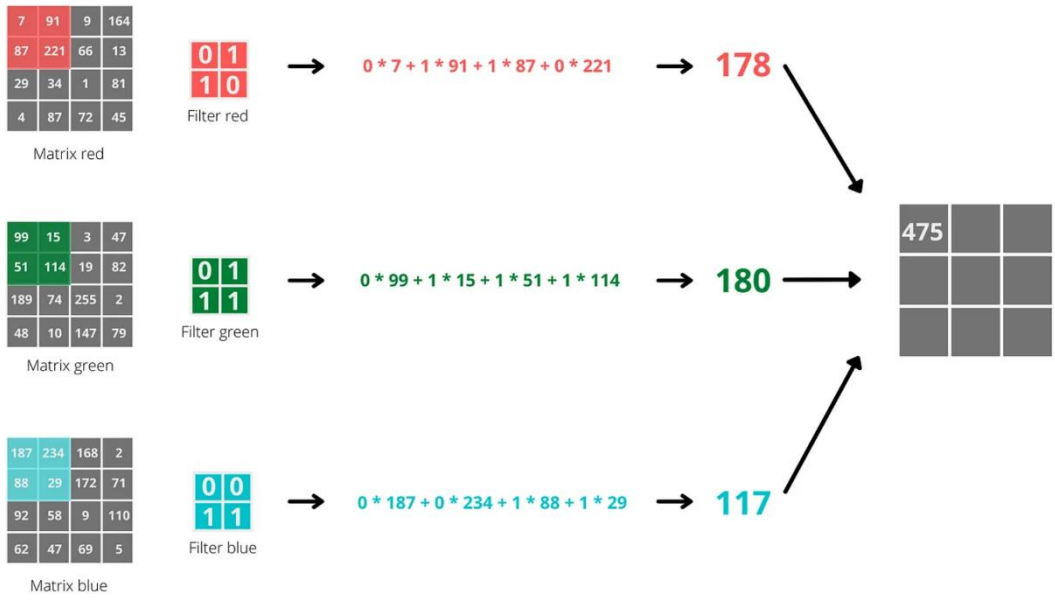


Рис. 3. Сверточный слой.

В первом шаге происходит снижение размерности изображения 4x4x3. Для этой цели определяется фильтр размером 2x2 для каждого цвета. Шаг длиной 1 позволяет перемещать фильтр вперед на один пиксель после каждого вычислительного шага. Это приводит к получению матрицы размером 3x3 в сверточном слое.

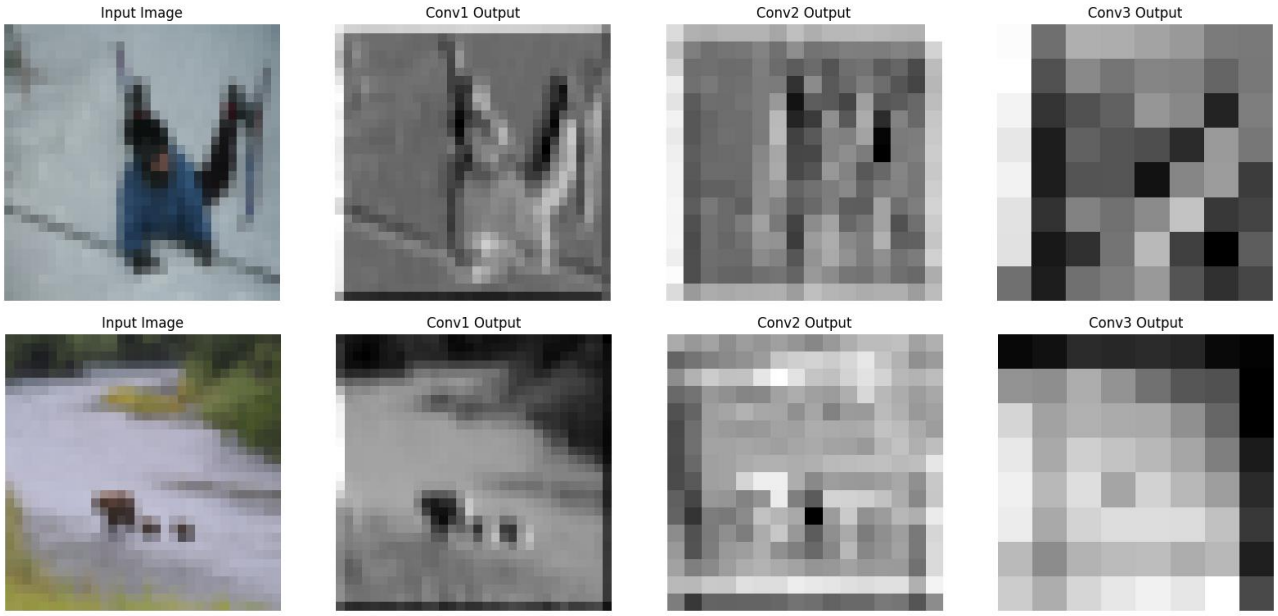


Рис. 4. Пример сверточного слоя.

Сверточный слой позволяет выделять особенности и пространственные шаблоны в изображении, сохраняя информацию о его структуре. Фильтр сканирует пиксели изображения и вычисляет свертку с подматрицей изображения, выявляя определенные признаки.

Каждое значение в выходной матрице сверточного слоя является результатом операции свертки и объединяет информацию из определенной области входного изображения. Это позволяет извлекать локальные особенности, такие как контуры, углы и текстуры, независимо от их положения в изображении.

Для выполнения свертки в сверточном слое, фильтр перемещается по изображению и вычисляется скалярное произведение с соответствующей подматрицей изображения. После применения свертки ко всем подматрицам изображения получается выходная матрица размером 3x3.

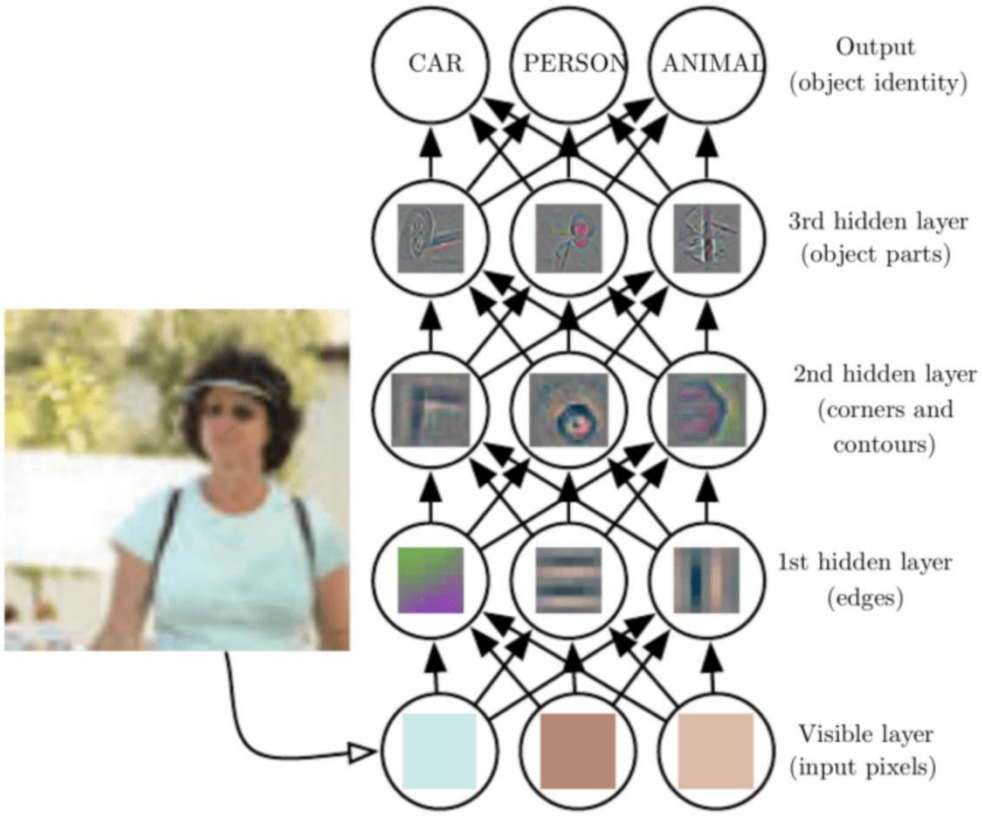


Рис. 5. Процесс переходов от сверточного слоя до итого результата.

Визуализация сверточного слоя позволяет лучше понять, какие признаки и особенности извлекаются на каждом шаге. Входное изображение проходит через фильтр, и каждое значение в выходной матрице связано с определенной характеристикой или признаком, выявленным фильтром.

Сверточные слои являются ключевым элементом сверточных нейронных сетей и позволяют эффективно обрабатывать изображения. Они позволяют сетям распознавать и идентифицировать объекты, а также выделять важные признаки, необходимые для решения конкретных задач классификации или обработки изображений

Подобно сверточному слою, слой пулинга отвечает за уменьшение пространственного размера сверточной характеристики. Это делается для уменьшения вычислительной мощности, необходимой для обработки данных путем снижения размерности. Кроме того, это полезно для извлечения доминирующих признаков, которые инвариантны к повороту и позиции, таким образом, поддерживая эффективный процесс обучения модели.

Существуют два типа пулинга: максимальный и средний. Максимальный пулинг возвращает максимальное значение из области изображения, покрытой ядром. С другой стороны, средний пулинг возвращает среднее значение всех значений из области изображения, покрытой ядром.

Максимальный пулинг также действует как средство снижения шума. Шумные активации полностью отбрасываются, а также выполняется устранение шума вместе с уменьшением размерности. С другой стороны, средний пулинг просто осуществляет снижение размерности в качестве механизма снижения шума. Таким образом, можно сказать, что максимальный пулинг работает намного лучше, чем средний пулинг.

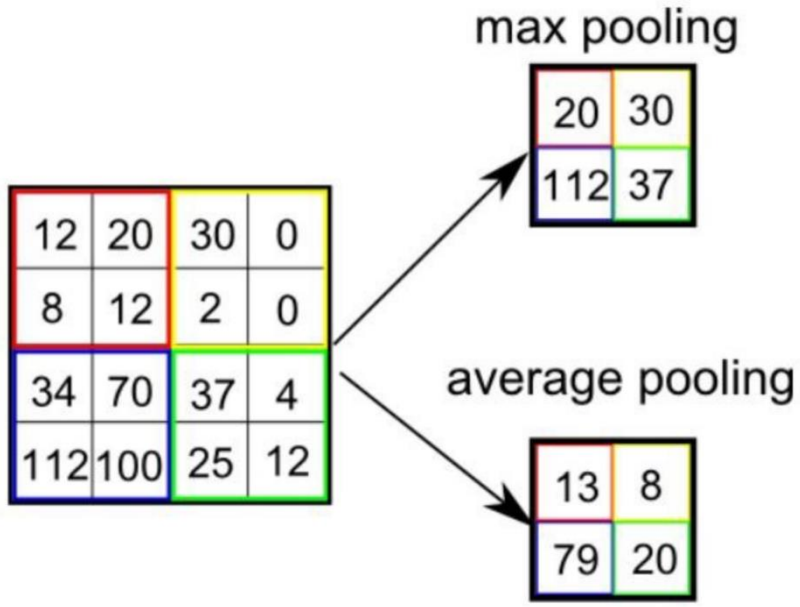


Рис. 6. Слой пулинга.

Сверточный слой и слой пулинга вместе формируют i-й слой сверточной нейронной сети. В зависимости от сложности изображений количество таких слоев может быть увеличено для захвата более низкоуровневых деталей, но за счет большей вычислительной мощности.

Пройдя через вышеуказанный процесс, модель успешно осознает особенности. Затем окончательный вывод преобразуется в одномерный массив и передается обычной нейронной сети для целей классификации.

1.1.2. Классификация. Полносвязный слой (FC Layer)

Полносвязанный слой является важной составляющей сверточных нейронных сетей и выполняет роль обучения нелинейных комбинаций высокоуровневых признаков, полученных из выхода сверточного слоя. Он представляет собой сеть искусственных нейронов, где каждый нейрон связан со всеми нейронами предыдущего слоя.

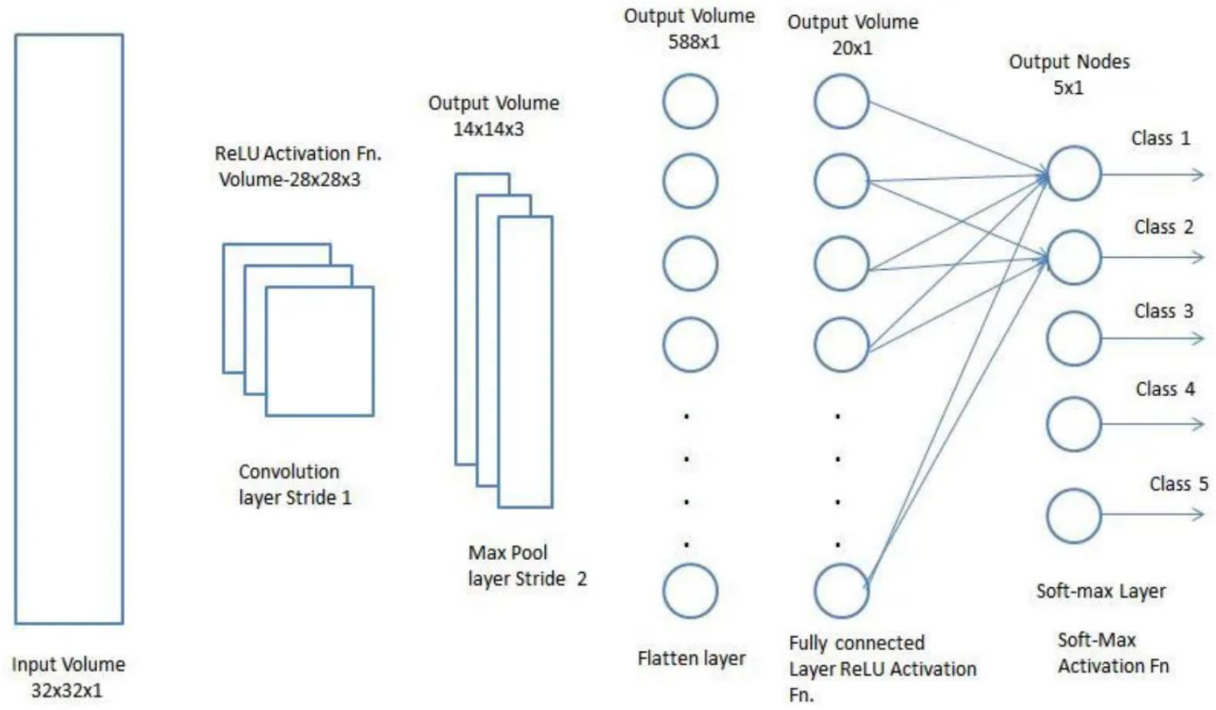


Рис. 7. Полносвязный слой.

Цель полносвязанного слоя заключается в изучении возможно нелинейной функции, которая моделирует сложные зависимости между признаками. Каждый нейрон полносвязанного слоя вычисляет взвешенную сумму своих входов и применяет нелинейную активационную функцию к полученному значению. Это позволяет нейронной сети обнаруживать более сложные шаблоны и корреляции в данных.

Процесс работы полносвязанного слоя начинается с преобразования выхода сверточного слоя в одномерный вектор. Для этого изображение сглаживается, превращаясь в столбцовый вектор, где каждый элемент

представляет определенный признак или характеристику. Этот вектор становится входом для полносвязанного слоя.

В процессе обучения, используя метод обратного распространения ошибки, каждый нейрон полносвязанного слоя корректирует свои веса таким образом, чтобы минимизировать ошибку и улучшить точность классификации или предсказания модели. Обратное распространение ошибки распространяет ошибку от выходного слоя назад к входному слою, обновляя веса нейронов на основе градиента функции потерь.

Важно отметить, что использование полносвязанного слоя увеличивает количество параметров и вычислительную сложность модели. Чем больше нейронов и слоев в полносвязанном слое, тем больше весов и обновлений требуется в процессе обучения. Поэтому важно находить баланс между достаточным количеством нейронов для изучения сложных зависимостей и ограниченными ресурсами для эффективного обучения.

Полносвязанный слой является последним шагом в сверточной нейронной сети перед окончательной классификацией или предсказанием. Он позволяет модели извлекать и использовать высокоуровневые признаки для принятия решений и достижения высокой точности в различных задачах, таких как классификация изображений, распознавание объектов и сегментация изображений.

Существуют различные архитектуры сверточных нейронных сетей, которые были ключевыми при разработке алгоритмов, обеспечивающих и будущему развитию искусственного интеллекта в целом. Ниже перечислены некоторые из них:

- LeNet
- AlexNet
- VGGNet
- GoogLeNet
- ResNet
- ZFNet

1.1.3. Обучение с учителем и Обучение без учителя

Обучение с учителем (supervised learning) предполагает использование полного набора размеченных данных в процессе построения модели.

В случае наличия полностью размеченного датасета, каждому образцу в обучающем наборе соответствует соответствующий ответ, который алгоритм должен научиться предсказывать. Например, размеченный датасет из фотографий цветов содержит изображения роз, ромашек или нарциссов, которые служат для обучения нейронной сети. При получении новой фотографии сеть сравнивает ее с примерами из обучающего датасета, чтобы сделать предсказание.

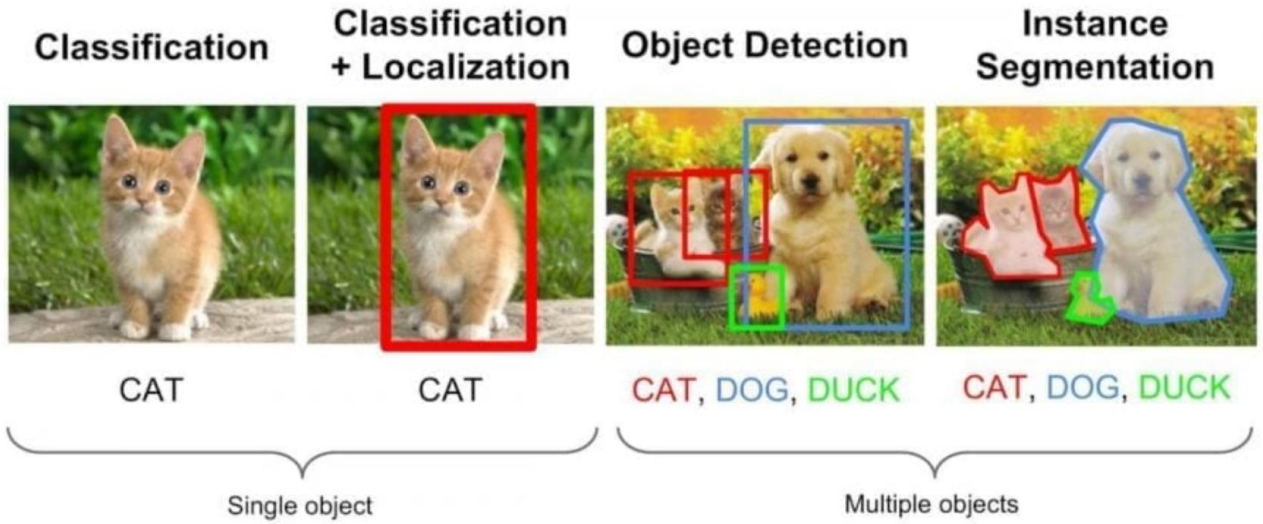


Рис. 8. Пример обучения с учителем — классификация (слева), и дальнейшее ее использование для сегментации и распознавания объектов

Обучение с учителем применяется преимущественно для решения двух типов задач: классификации и регрессии.

В задачах классификации алгоритм предсказывает дискретные значения, которые соответствуют классам, к которым принадлежат объекты. Например, при обучении алгоритма с использованием датасета с фотографиями животных, каждое изображение будет иметь метку, указывающую на принадлежность к классу "кошка", "коала" или "черепаха". Качество алгоритма оценивается по его способности правильно классифицировать новые фотографии с коалами и черепахами.

Задачи регрессии, в свою очередь, связаны с непрерывными данными. Например, в случае линейной регрессии алгоритм предсказывает ожидаемое значение переменной y на основе конкретных значений x.

Однако в более сложных задачах машинного обучения может быть задействовано большое количество переменных. Например, нейронная сеть, предсказывающая цену квартиры в Сан-Франциско на основе ее площади, местоположения и доступности общественного транспорта. Алгоритм выполняет функцию эксперта, который рассчитывает цену квартиры на основе этих данных.

Обучение с учителем эффективно в случаях, когда имеется обширный и достоверный набор данных для обучения алгоритма. Однако такие данные не всегда доступны, и в таких случаях возникает необходимость в других типах алгоритмов машинного обучения.

Размеченные и чистые данные, которые идеально подходят для обучения моделей, не всегда легко получить. Иногда перед алгоритмом стоит задача находить ранее неизвестные ответы. В таких случаях приходит на помощь обучение без учителя.

Обучение без учителя (unsupervised learning) предполагает, что у модели имеется набор данных, но нет явных указаний о том, что с ними делать. Вместо этого нейронная сеть самостоятельно ищет корреляции в данных, извлекает полезные признаки и анализирует их. Отсутствие явных указаний дает модели свободу действий в поиске зависимостей, что может привести к хорошим результатам. В зависимости от поставленной задачи модель систематизирует данные по-разному.

Рис. 9. Кластеризация данных на основе общих признаков

Одной из распространенных задач обучения без учителя является кластеризация. В этом случае модель группирует похожие данные на основе общих признаков, например, цвета, размера или формы. Кластеризация позволяет разделить набор изображений на группы, соответствующие различным видам птиц, даже без специальных знаний эксперта-орнитолога.

Еще одной задачей, решаемой с помощью обучения без учителя, является обнаружение аномалий. Например, банки могут использовать этот подход для выявления мошеннических операций, обнаруживая необычные действия в покупательском поведении клиентов. Модель ищет выбросы в данных и сигнализирует о подозрительных операциях.

Ассоциации являются еще одним важным аспектом обучения без учителя. Модель ищет корреляции между различными признаками объектов и может предсказывать связанные с ними признаки. Например, на основе выбора товаров в онлайн-магазине модель может рекомендовать дополнительные товары, которые связаны с уже выбранными.

Автоэнкодеры являются еще одним инструментом обучения без учителя. Они используются для сжатия и восстановления данных. Модель кодирует входные данные в более компактное представление, а затем пытается воссоздать

исходные данные из этого кода. Автоэнкодеры могут быть использованы, например, для удаления шума из изображений, видеоданных или медицинских сканов.

Обучение без учителя представляет определенные трудности при оценке точности алгоритма, так как отсутствуют «правильные ответы» или метки в данных. Однако, при надежной разметке данных часто возникают проблемы или это может быть слишком дорого. В таких случаях использование обучения без учителя и предоставление модели свободы действий для поиска зависимостей может привести к хорошим результатам.

\

1.2. Dataset. Набор данных для обучения

Датасеты являются ключевым элементом в машинном обучении, поскольку они представляют собой наборы данных, на которых модели обучаются и тестируются. Датасеты играют важную роль в обучении моделей, поскольку качество и разнообразие данных влияют на способность моделей к обобщению и достижению хороших результатов на новых, реальных примерах.

В машинном обучении существует множество различных датасетов, каждый из которых предназначен для определенной задачи и имеет свои особенности. Датасеты могут быть собраны и сформированы специально для конкретной задачи, либо они могут быть открытыми и доступными для использования всеми исследователями и практиками.[9]

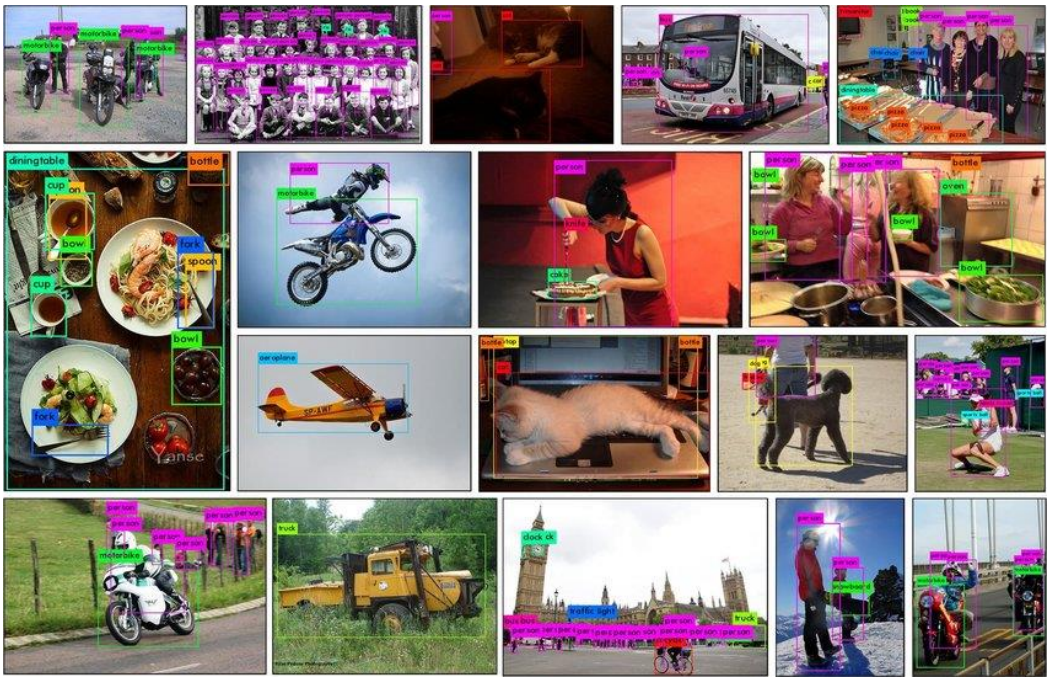


Рис. 10. COCO Dataset.

При выборе датасета необходимо учитывать несколько факторов. Во-первых, размер и разнообразие данных. Большие и разнообразные датасеты часто обеспечивают более точные и обобщающие модели. Во-вторых, качество и надежность данных. Данные должны быть чистыми, правильно размеченными и соответствовать целевой задаче. В-третьих, доступность и лицензирование. Некоторые датасеты могут быть ограничены в использовании из-за правовых ограничений или требовать специальных разрешений.[10]

Каждый датасет состоит из примеров или наблюдений, где каждый пример представляет собой набор признаков и соответствующую метку или целевую переменную. Признаки описывают характеристики примера, которые модель будет использовать для принятия решений. Метка является желаемым выходом или классификацией, которую модель должна предсказать.

Существуют различные источники для получения датасетов. Они могут быть получены из открытых источников, таких как репозитории данных, архивы и базы данных, а также созданы путем сбора и аннотирования данных специально для конкретной задачи.

Важно проводить предварительный анализ датасета, чтобы понять его характеристики, распределение данных, наличие пропущенных значений или выбросов. Также необходимо провести процесс предобработки данных, который включает масштабирование, нормализацию, преобразование признаков и разделение данных на обучающую и тестовую выборки.

1.2.1. Типы датасетов в машинном обучении

В машинном обучении существует множество типов датасетов, которые отличаются по своей природе и особенностям. Понимание различных типов датасетов является важным для выбора соответствующих методов обработки данных и моделей. В этой главе мы рассмотрим некоторые из наиболее распространенных типов датасетов в машинном обучении.

1. **Датасеты с табличными данными:** Датасеты с табличными данными являются наиболее распространенным типом в машинном обучении. Они представляют данные в виде таблицы, где каждый ряд соответствует отдельному примеру, а каждый столбец содержит признаки или атрибуты, описывающие примеры. Такие датасеты широко используются для задач классификации и регрессии. Для обработки датасетов с табличными данными применяются методы статистики, визуализации данных и алгоритмы машинного обучения, такие как решающие деревья, логистическая регрессия и случайные леса.
2. **Датасеты временных рядов:** Датасеты временных рядов состоят из последовательности точек данных, организованных во временном порядке. Они используются для анализа и прогнозирования временных зависимостей, таких как изменение цен акций, погодные условия, трафик и другие. Для работы с датасетами временных рядов применяются специализированные методы, такие как авторегрессионные модели (AR), скользящие средние (MA), авторегрессионные скользящие средние (ARMA) и адаптивные скользящие средние (ARIMA).
3. **Датасеты изображений:** Датасеты изображений содержат наборы изображений, обычно в формате растровых файлов, таких как JPEG или PNG. Эти датасеты широко используются в задачах компьютерного зрения, таких как классификация изображений, детектирование объектов, сегментация и другие. Для работы с датасетами изображений применяются сверточные

нейронные сети (Convolutional Neural Networks, CNN), которые способны автоматически извлекать признаки из изображений.

4. **Датасеты для обработки** естественного языка (NLP): Датасеты для обработки естественного языка содержат текстовые данные, такие как отзывы, новости, социальные медиа-посты и другие. Они используются для задач, связанных с анализом текста, таких как классификация текста, анализ тональности, машинный перевод и другие. Примеры включают датасеты IMDB, Sentiment140 и WMT для машинного перевода. Для работы с текстовыми данными применяются методы обработки естественного языка (Natural Language Processing, NLP).
5. **Датасеты аудио и звука:** Датасеты аудио и звука содержат звуковые сигналы, записи речи и другие аудиозаписи. Они используются для задач, связанных с обработкой аудио, таких как распознавание речи, классификация звуков, извлечение признаков и другие. Примеры включают датасеты UrbanSound, LibriSpeech и ESC-50. Для работы с аудиоданными используются методы обработки сигналов и спектрального анализа.

Каждый тип датасета имеет свои особенности и требует соответствующих методов предобработки данных и моделей машинного обучения. Выбор подходящего типа датасета зависит от конкретной задачи, доступных данных и целей исследования.[11]

1.2.2. Датасеты изображений

Датасеты изображений являются одним из наиболее распространенных типов данных, используемых в области компьютерного зрения и обработки изображений. Они содержат набор изображений, обычно размеченных или классифицированных по определенным категориям. Эти датасеты играют ключевую роль в обучении моделей машинного обучения для распознавания объектов, классификации изображений, обнаружения объектов и других задач, связанных с анализом изображений.

Вот некоторые примеры известных датасетов изображений:

1. MNIST: Датасет MNIST содержит набор рукописных цифр от 0 до 9. Он состоит из 60 000 обучающих изображений и 10 000 тестовых изображений размером 28x28 пикселей. MNIST широко используется в задачах классификации изображений и является своего рода "золотым стандартом" в этой области.
2. CIFAR-10 и CIFAR-100: Датасеты CIFAR-10 и CIFAR-100 содержат набор изображений размером 32x32 пикселя, разделенных на 10 и 100 классов соответственно. CIFAR-10 содержит изображения объектов, таких как самолеты, автомобили, кошки и т. д., в то время как CIFAR-100 содержит более мелкие классы, такие как яблоки, розы, комары и т. д. Эти датасеты часто используются для задач классификации и обнаружения объектов.
3. ImageNet: ImageNet - это один из самых известных и крупнейших датасетов изображений. Он содержит более 14 миллионов размеченных изображений, относящихся к более чем 20 000 классам объектов. ImageNet является более сложным и разнообразным датасетом, который используется в задачах классификации, детектирования и сегментации объектов.
4. COCO: Датасет COCO (Common Objects in Context) содержит большое количество изображений, размеченных с помощью более детальной информации, такой как границы и сегментации объектов. COCO является

одним из наиболее используемых датасетов для задачи обнаружения объектов и сегментации.

5. Open Images: Open Images - это еще один крупный датасет изображений, содержащий более 9 миллионов размеченных изображений. Он включает в себя разнообразные категории объектов и широкий спектр разметки, включая границы, маски и атрибуты объектов.

Каждый из этих датасетов представляет собой ценный ресурс для исследований и разработки в области компьютерного зрения. Они позволяют обучать модели на большом количестве разнообразных изображений и создавать высокоэффективные системы распознавания и классификации объектов в реальном мире.[12][13

1.3. Программируемые логические интегральные схемы

Проектирование новых интегральных схем — сложный и многогранный процесс. Разработчикам необходимо продумать расположение огромного числа элементов: сдвиговые регистры, дешифраторы, мультиплексоры и прочее. Большинство интегральных схем (ИС), в том числе процессоры и микроконтроллеры, имеют предопределенную логику работы.

Проще говоря, в таких схемах после выпуска архитектура остается неизменной. Программисты в свою очередь получают набор определенных команд, с помощью которых и взаимодействуют с конкретной схемой.

Одна из главных проблем проектировщиков — это поиск компромисса между скоростью работы схемы и ее универсальностью.

Основным «кирпичиком» при построении любой интегральной схемы являются логические элементы (вентили) — «И», «ИЛИ», «НЕ». На их базе уже создаются триггеры, регистры и так далее.

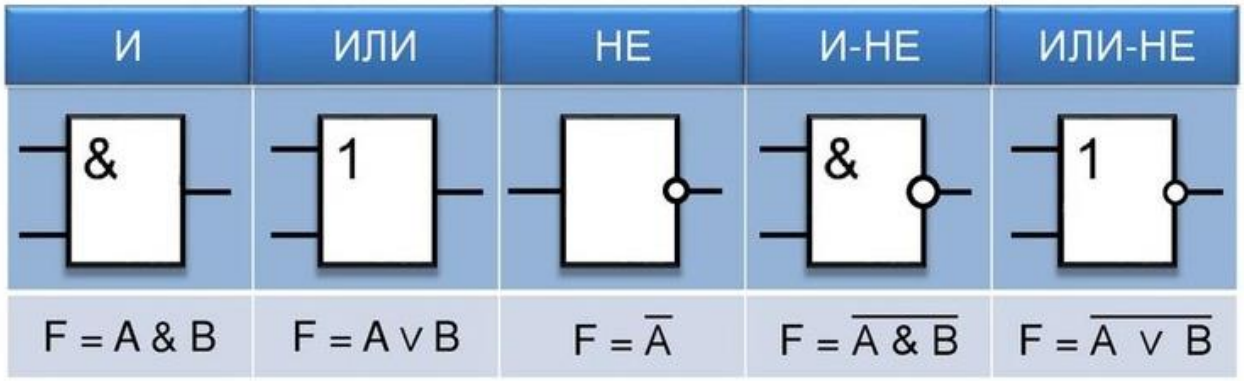


Рис. 11. Логические элементы

Как именно связываются между собой логические элементы — определяют разработчики микросхемы еще на этапе проектирования. И главная проблема в том, что эти связи в последствие уже изменить нельзя.[1]

Однако в начале 70-х годов на рынке начали появляться первые программируемые логические устройства. Ключевое отличие от процессоров и микроконтроллеров — можно самостоятельно задавать архитектуру. Связи

между логическими элементами не были предопределены и поддавались редактированию.

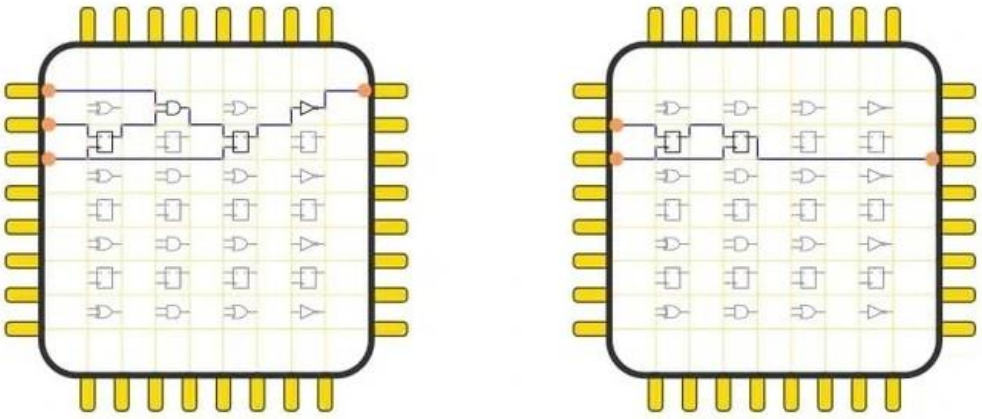


Рис. 12. Иллюстрация редактирования микроконтроллеров.

Изначально такие связи выполнялись в виде пережигаемых тонких проводников. По мере совершенствования технологий стали использоваться МОП-транзисторы с плавающим затвором. Появилась возможность реконфигурировать внутреннюю структуру микросхемы. Так появились различные подвиды ПЛИС.

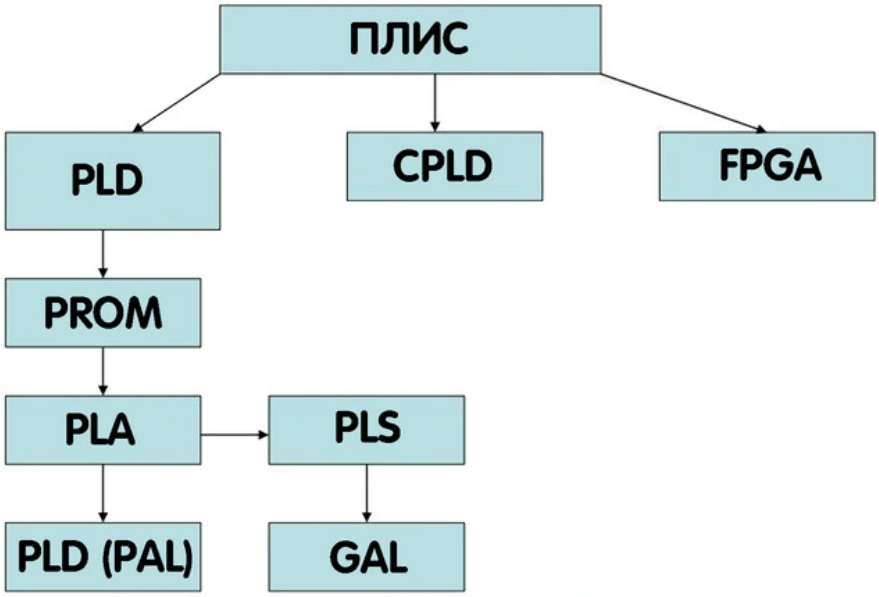


Рис. 13. Классификация ПЛИС.

Схемах PAL элементы «И» — программируемые, а элементы «ИЛИ» — фиксированные. На базе таких блоков можно было создавать достаточно сложные заказные схемы с минимальными затратами. Однако PAL имели плавкие титановольфрамовые перемычки, поэтому не могли использоваться повторно.[2]

1.3.1. FPGA (Field-Programmable Gate Array)

FPGA (Field-Programmable Gate Array) — это один из подвидов программируемых интегральных схем. Строятся такие микросхемы на логических блоках с гибкой коммутацией — причем число блоков может доходить до сотен тысяч штук. Прошивка с «картой» необходимых связей между логическими ячейками сохраняется в энергонезависимой памяти.[3]

Условно FPGA состоят из трех основных элементов — конфигурируемые логические блоки (CLB), блок ввода-вывода (IOB) и межсоединения. Каждый CLB включает в себя таблицы поиска, триггеры, регистры, мультиплексоры и не только. Благодаря этому CLB могут выполнять логические и арифметические операции, а также использоваться для хранения данных.

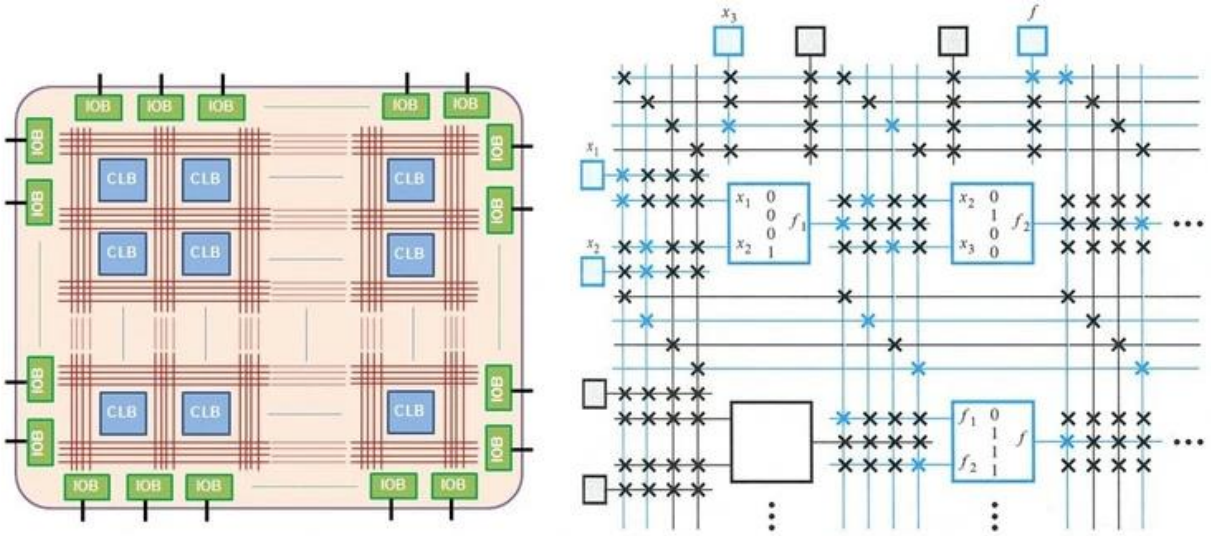


Рис. 14. Общая схема и программируемая секция FPGA.

Если смотреть структуру CLB подробнее, то здесь все зависит от каждой конкретной микросхемы. Чем дороже плата — тем обычно большие возможности предлагает каждый конкретный конфигурируемый логический блок.

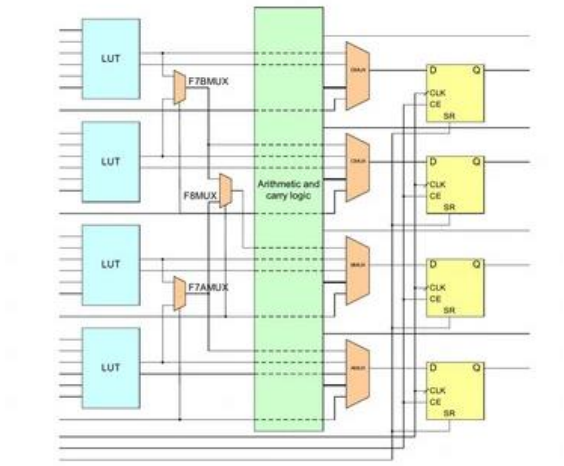


Рис. 15. Работа конфигурируемого логического блока (CLB).

- LUT — Look-Up Tables. Каждый такой блок способен реализовать любую логическую функцию используя в качестве данных операнды с входов.
- Специализированные управляемые пользователем мультиплексоры (MUX) для комбинационной логики.
- Блок арифметической логики, который позволяет делать суммирование и умножение операндов (зеленый блок).
- Несколько однобитных регистров для хранения информации (желтые блоки).

Таким образом, CLB — это своеобразный универсальный кирпичик, который способен выполнить практически любую операцию над данными. На FPGA плате таких — сотни и тысячи.

По своей универсальности FPGA проигрывают типичным процессорам и микроконтроллерам. Однако возможность буквально программировать архитектуру дает преимущества в специфических задачах, в том числе перед ASIC.

В первую очередь FPGA отлично проявляют себя при разработке, в частности, прототипировании различных микросхем ASIC. создание ASIC — это крайне дорогостоящая задача, и перед выпуском платы в массы необходимо тщательно протестировать всю логику. Чтобы после нахождения недоработки каждый раз не выпускать новую плату, логику тестируют на нескольких

кристаллах FPGA. При нахождении критической ошибки проектировщикам достаточно переработать архитектуру и выполнить реконфигурирование.

Например, возьмем задачу распознавания автомобильных номеров. Первый вариант — камера с возможностью передачи видео через Ethernet и обработкой потока на удаленном сервере. Однако при увеличении числа камер будет расти и нагрузка на сеть. А вот с энергоэффективной платой FPGA распознавание можно делать буквально в самой камере, а после — передавать на сервер лишь текстовые данные — полученные номера. При этом если формат автомобильных номеров изменится, вы быстро сможете реконфигурировать плату FPGA.

Параллелизм также отлично подходит для быстрой обработки больших объемов данных, что позволяет частично использовать FPGA в суперкомпьютерах или информационно-измерительных системах.

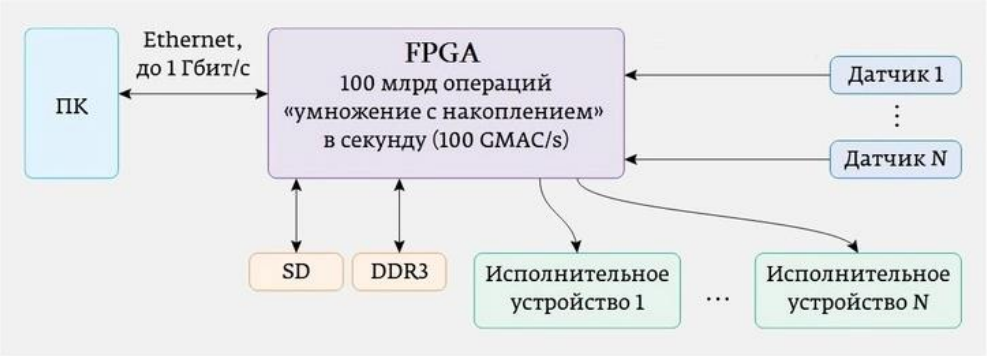


Рис. 18. Структурная схема цифровой части информационно-измерительной системы с применением FPGA.

Помимо этого, платы применяются в сфере коммуникаций — оборудование базовых станций GSM и так далее. Не менее интересен и тот факт, что львиная доля FPGA-ускорителей используется в военной сфере — схемы обеспечивают высокоскоростную обработку радиолокационных сигналов. Найти FPGA можно в беспилотных автомобилях, дронах, научных приборах, медицинской технике и не только.

1.3.2. Принцип работы

Микросхема FPGA — это заказная микросхема, состоящая из транзисторов, из которых собираются триггеры, регистры, мультиплексоры и другие логические элементы для обычных схем. Изменить порядок соединения этих транзисторов, конечно, нельзя. Но архитектурно микросхема построена таким хитрым образом, что можно изменять коммутацию сигналов между более крупными блоками: их называют CLB — программируемые логические блоки.

Также можно изменять логическую функцию, которую выполняет CLB. Достигается это за счет того, что вся микросхема пронизана ячейками конфигурационной памяти Static RAM. Каждый бит этой памяти либо управляет каким-то ключом коммутации сигналов, либо является частью таблицы истинности логической функции, которую реализует CLB.

Так как конфигурационная память построена по технологии Static RAM, то, во-первых, при включении питания FPGA микросхему обязательно надо сконфигурировать, а во-вторых, микросхему можно реконфигурировать практически бесконечное количество раз.

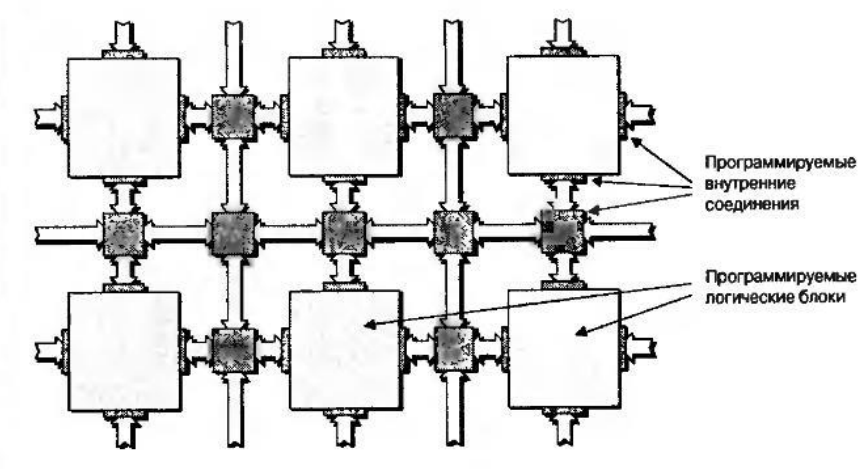


Рис. 16. У прощенная 2D-структура микросхемы без конфигурационной памяти

Блоки CLB находятся в коммутационной матрице, которая задает соединения входов и выходов блоков CLB.

На каждом пересечении проводников находится шесть переключающих ключей, управляемых своими ячейками конфигурационной памяти. Открывая одни и закрывая другие, можно обеспечить разную коммутацию сигналов между CLB.

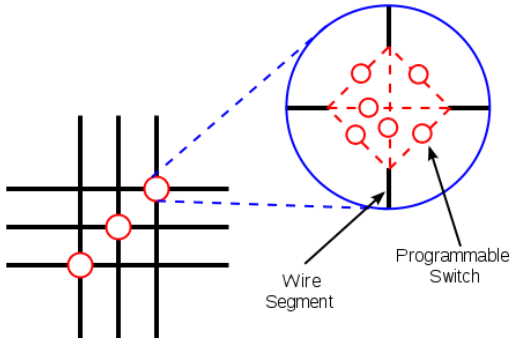


Рис. 17. Схема коммутационной матрицы

LB очень упрощенно состоит из блока, задающего булеву функцию от нескольких аргументов (она называется таблицей соответствия — Look Up Table, LUT) и триггера (flip-flop, FF). В современных FPGA LUT имеет шесть входов, но на рисунке для простоты показаны три. Выход LUT подается на выход CLB либо асинхронно (напрямую), либо синхронно (через триггер FF, работающий на системной тактовой частоте).

Значение каждой из ячеек подается на свой вход выходного мультиплексора LUT, а входные аргументы булевой функции используются для выбора того или иного значения функции. CLB — важнейший аппаратный ресурс FPGA. Количество CLB в современных кристаллах FPGA может быть разным и зависит от типа и емкости кристалла. У Xilinx есть кристаллы с количеством CLB в пределах примерно от четырех тысяч до трех миллионов.

Помимо CLB, внутри FPGA есть еще ряд важных аппаратных ресурсов. Например, аппаратные блоки умножения с накоплением или блоки DSP. Каждый из них может делать операции умножения и сложения 18-битных чисел каждый такт. В топовых кристаллах количество блоков DSP может превышать 6000.

1.3.3. Реализация нейронных сетей на ПЛИС

Нейронные сети и глубокие нейронные сети сейчас активно используются в разных областях, но реализация их на процессоре оказывается неэффективной существует много вычислений, которые можно распараллелить

Поэтому перенеся нейронную сеть на FPGA удается на много порядков ускорить работу нейронной сети, остается обеспечить высокоскоростной интерфейс для загрузки исходных данных и получения результата. В качестве примера — реализация системы распознавания лиц на процессоре i7/9Gen распознает до 20 лиц за секунду с одной видеокамеры HD, реализация на FPGA — порядка 1000 лиц с нескольких камер.[4]

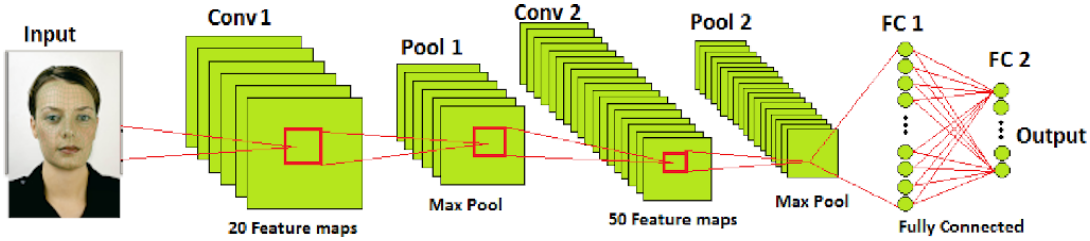
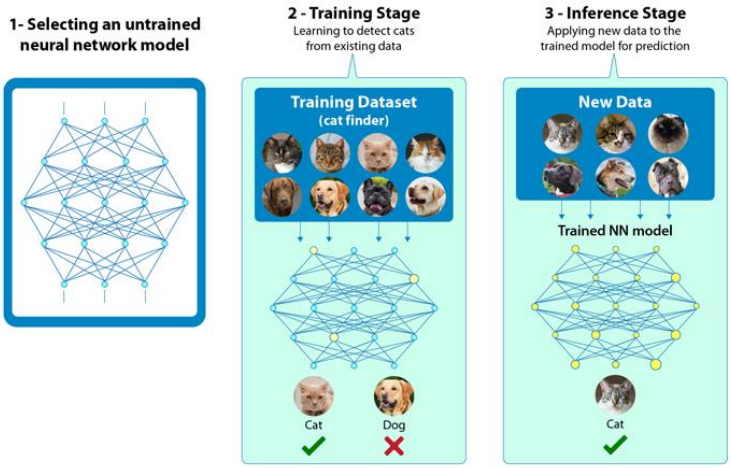


Рис. 19. Структура используемой глубокой нейронной сети.

Проектирование приложения нейронных сетей представляет собой трехэтапный процесс. Шаги заключаются в выборе правильной сети, обучении сети, а затем применении новых данных к обученной модели для прогнозирования (вывода). На рисунке 10 показаны шаги приложения для распознавания кошек.



20. Три шага распознавания кошки.

Как уже упоминалось, в модели нейронной сети есть несколько слоев, и каждый слой имеет определенную задачу. В глубоком обучении каждый слой предназначен для извлечения объектов на разных уровнях. Например, в нейронной сети обнаружения краев первый средний слой обнаруживает такие объекты, как ребра и кривые. Выход первого среднего слоя затем подается во второй слой, который отвечает за обнаружение объектов более высокого уровня, таких как полукруги или квадраты. Третий средний слой собирает выходные данные других слоев для создания знакомых объектов, а последний слой обнаруживает объект.

Алгоритмы сверточных нейросетей являются перспективными с точки зрения качества распознавания изображений. Наиболее надежным способом измерения качества систем машинного зрения является тестирование на больших базах. Тестирование на этих базах позволяет оценить способность алгоритма решать задачи классификации и детектирования объектов в реальных условиях. Алгоритмы на основе сверточных сетей уверенно лидируют уже несколько лет на этих базах и в решении подобных задача. Круг применения их расширяется с каждым годом. Однако реализация сверточных нейронных сетей обладает существенными требованиями по быстродействию. Для «стационарных» решений наиболее распространенным способом обеспечения такой производительности является использование для вычисления универсальных графических процессоров. Однако в составе встраиваемых и мобильных решений следует обратить внимание на возможность использования программируемых логических интегральных схем.

2. Практическая часть

В данной главе фокус будет сосредоточен на практической части и выполнении ряда задач. Начнем с создания датасета, где первоначально будет рассмотрен процесс формирования датасета с использовани учителя. В этом случае мы сможем обогатить наш датасет и использовать эту информацию в процессе обучения модели.

После создания датасета мы приступим к обучению модели нейронной сети с использованием этих данных. Мы будем использовать различные архитектуры нейронных сетей, оптимизационные алгоритмы и техники для достижения наилучшей производительности и точности модели. Наша цель заключается в постоянном улучшении показателей точности модели на основе полученных данных.

При достижении моделью высокой точности мы рассмотрим возможность интеграции модели нейронной сети с платой FPGA. FPGA (Field-Programmable Gate Array) представляет собой программируемую интегральную схему, которая позволяет нам создавать специализированные аппаратные ускорители для выполнения вычислений нейронных сетей.

После интеграции модели с платой FPGA мы проведем аналогичные действия с этой платой, как и с моделью нейронной сети. Это включает в себя обучение платы FPGA с использованием доступных данных, настройку параметров и оптимизацию, чтобы добиться максимальной производительности и точности на аппаратном уровне.

Таким образом, в данной главе мы последовательно пройдем путь от создания датасета без учителя до обучения модели нейронной сети, а затем совместим модель с платой FPGA и проведем аналогичные действия над ней.В итоге, путем последовательных этапов создания датасета, обучения модели и повышения ее точности, мы стремимся достичь максимальной производительности и улучшить результаты предсказаний нашей нейронной сети.

2.1. Сбор датасет для распознавания человека на изображении

В процессе обучения нейронной сети для распознавания человека на изображении, одним из ключевых шагов является сбор подходящих датасетов. Идеально размеченные и чистые данные не всегда легко достать. Иногда перед алгоритмом стоит задача найти заранее неизвестные ответы, и для этого требуется обучение без учителя.

При обучении без учителя (unsupervised learning) модели предоставляется набор данных без явных указаний о том, что с ними делать. Нейронная сеть пытается самостоятельно найти корреляции в данных, извлекая полезные признаки и анализируя их. Это дает свободу действий для поиска зависимостей и может привести к получению хороших результатов. В зависимости от задачи модель систематизирует данные по-разному.

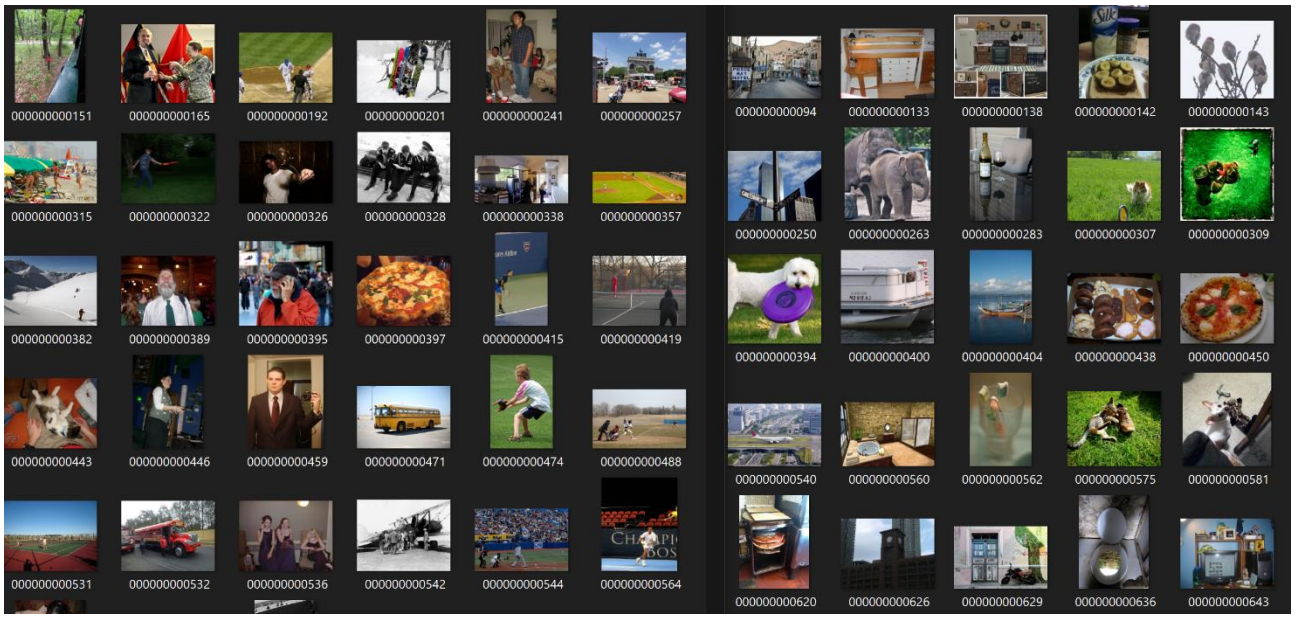


Рис. 22.Собранный Dataset.

Один из подходов к сбору датасетов для распознавания человека на изображении - это использование кластеризации. Алгоритм кластеризации группирует похожие данные, находя общие признаки, и помогает разделить изображения по видам птиц, например, опираясь на цвет пера, размер или форму клюва. Кластеризация является наиболее распространенной задачей в обучении без учителя.[14]

Еще одним важным аспектом при сборе датасетов является обнаружение аномалий. Например, банки могут использовать обучение без учителя для выявления мошеннических операций, обнаруживая необычные действия в покупательском поведении клиентов. Обучение без учителя также может быть использовано для поиска выбросов в данных.

Ассоциации - это еще один тип задачи, который может быть решен с помощью обучения без учителя. Путем анализа пары ключевых признаков объекта модель может предсказать другие признаки, с которыми они коррелируют. Это может быть полезно при рекомендации связанных продуктов в интернет-магазинах.

Для обработки изображений также широко применяются автоэнкодеры. Автоэнкодеры кодируют входные данные и затем пытаются воссоздать исходные данные из полученного кода. Они могут быть использованы, например, для удаления шума из видеоданных, изображений или медицинских сканов.

Важно отметить, что в обучении без учителя сложно вычислить точность алгоритма, так как в данных отсутствуют "правильные ответы" или метки. Однако, предоставляя модели свободу действий для поиска зависимостей, можно достичь хороших результатов даже при ненадежных или дорогостоящих размеченных данных.

Сайт Kaggle представляет собой полезный ресурс, где можно найти подходящие датасеты для обучения нейронной сети в задаче распознавания человека на изображении. Kaggle предлагает широкий выбор датасетов в различных областях и позволяет взаимодействовать с сообществом, обсуждать датасеты и получать поддержку от экспертов. Посещение Kaggle может значительно облегчить поиск и использование подходящих датасетов для вашей задачи.

Кроме того, на сайте Kaggle мы можем найти первые фотографии, которые могут быть использованы в качестве основы для создания нашего собственного датасета.[15]

2.1.1. Аннотационные файлы для точности модели CNN

Для достижения высокой точности работы модели нейронной сети в задачах компьютерного зрения, важно иметь хорошо размеченные данные. Одним из способов разметки является создание аннотационных файлов, которые содержат информацию о выделениях на изображении.

Аннотационный файл обычно содержит информацию о метках и координатах прямоугольников, описывающих области интереса на изображении. Существуют различные форматы аннотационных файлов, включая CSV, JSON и текстовые файлы.[13]

Пример аннотационного файла в формате JSON:

Листинг 1 — Пример аннотационного файла.

```
{
"annotations": [
    {
      "image_id": "Уникальный идентификатор изображения",
      "class": "Класс объекта (например, 'pesron')",
      "type": "Тип объекта (например, 'pesron')",
      "bounding_box": {
        "x": "Координата X левого верхнего угла ограничивающей рамки",
        "y": "Координата Y левого верхнего угла ограничивающей рамки",
        "width": "Ширина ограничивающей рамки",
        "height": "Высота ограничивающей рамки"
      },
    },
    {
      "image_id": "Уникальный идентификатор изображения",
      "class": "Класс объекта (например, 'background')",
      "type": "Тип объекта (например, 'background')",
      "bounding_box": {
        "x": "Координата X левого верхнего угла ограничивающей рамки",
        "y": "Координата Y левого верхнего угла ограничивающей рамки",
        "width": "Ширина ограничивающей рамки",
        "height": "Высота ограничивающей рамки"
      },
    },
    ...
  ]
}
```

В каждом примере приведены значения координат прямоугольников (x_min, y_min, x_max, y_max), определяющих области интереса на изображении для каждой метки.[18]

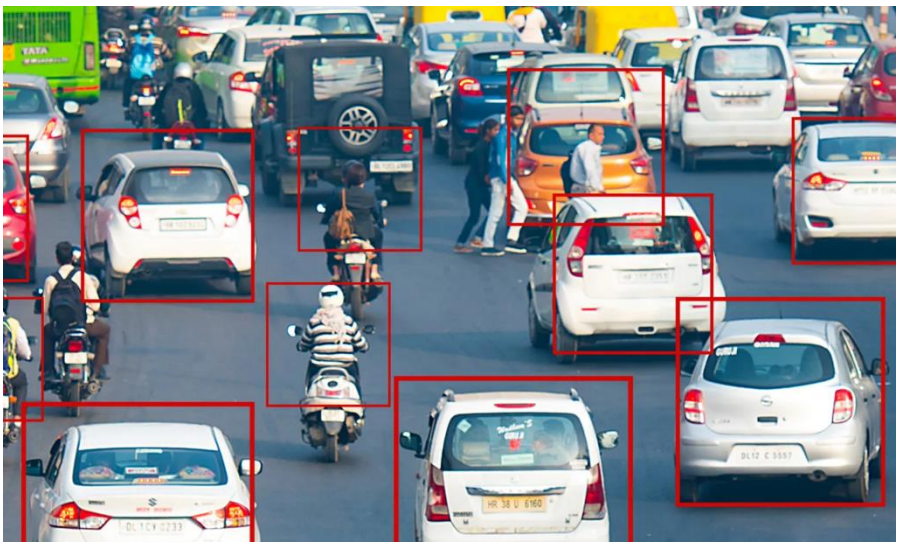


Рис. 23. Пример данных с разметкой.

При разметке выделений на изображениях важно быть внимательным и точным, чтобы предоставить модели достаточно информации для обучения и распознавания объектов на новых изображениях.

Помимо ручной разметки, существуют также инструменты и библиотеки, которые облегчают процесс создания аннотационных файлов и выделений на изображениях. Эти инструменты позволяют рисовать прямоугольники, создавать маски, выполнять семантическую сегментацию и многое другое.

Создание аннотационных файлов для выделений на изображениях является важным шагом в процессе подготовки данных для обучения модели нейронной сети. Правильно размеченные данные обеспечивают модели необходимую информацию для обучения и позволяют достичь более точных результатов при распознавании объектов на новых изображениях.

В процессе создания аннотационных файлов для выделений на изображениях может быть полезно использовать специализированные инструменты, которые упрощают и автоматизируют эту задачу. Один из таких инструментов - FiftyOne.

FiftyOne - это мощный инструмент для разметки и визуализации данных в задачах компьютерного зрения. Он предоставляет удобный пользовательский интерфейс, который позволяет эффективно работать с изображениями и создавать аннотационные файлы.

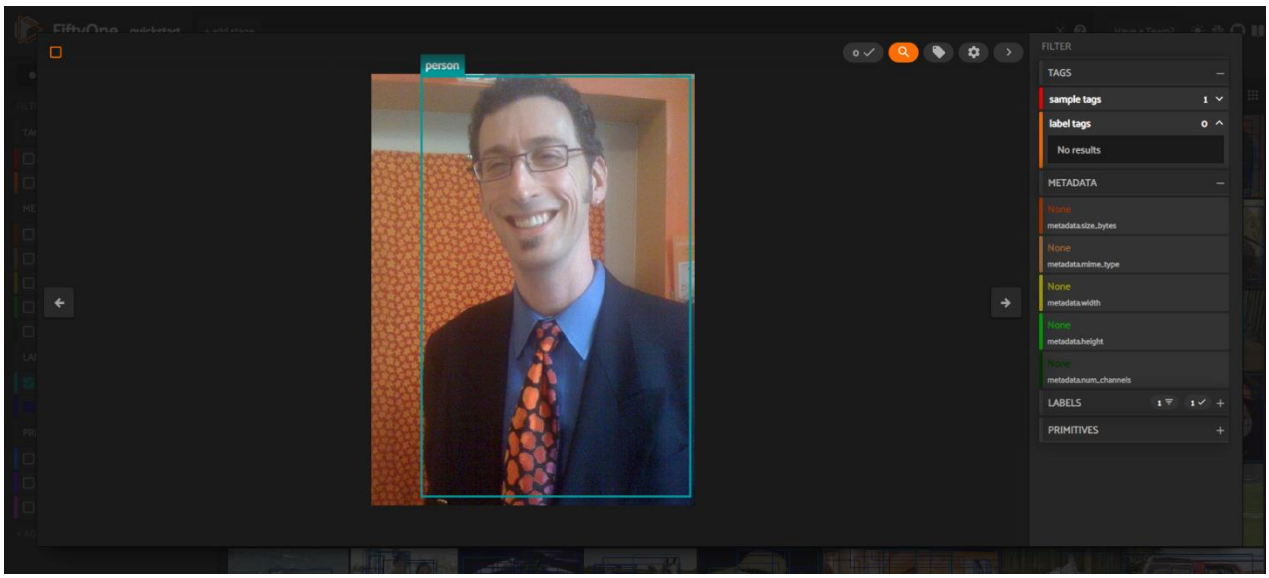


Рис. 24. Выделение человека на изображении и назначение метки “person”

Основные возможности инструмента FiftyOne:

1. Визуализация данных: FiftyOne предоставляет возможность просматривать изображения из датасета и визуализировать различные типы выделений, такие как ограничивающие прямоугольники, точки, сегменты и другие.
2. Разметка объектов: Инструмент позволяет создавать и редактировать выделения на изображениях, указывая метки и координаты объектов. Можно создавать прямоугольники, маски, полигоны и другие формы выделений.
3. Множественные аннотации: FiftyOne позволяет добавлять несколько аннотаций к одному изображению. Это особенно полезно, когда несколько экспертов работают над разметкой и требуется сопоставление различных мнений.
4. Экспорт аннотаций: После завершения разметки, FiftyOne предоставляет возможность экспортировать аннотации в различные форматы, включая CSV, JSON, COCO и другие. Это позволяет легко интегрировать размеченные данные в процесс обучения модели.
5. Расширяемость: Инструмент FiftyOne является расширяемым и поддерживает настраиваемые плагины. Это означает, что вы можете

создать собственные функции и инструменты, чтобы адаптировать его под свои уникальные потребности.

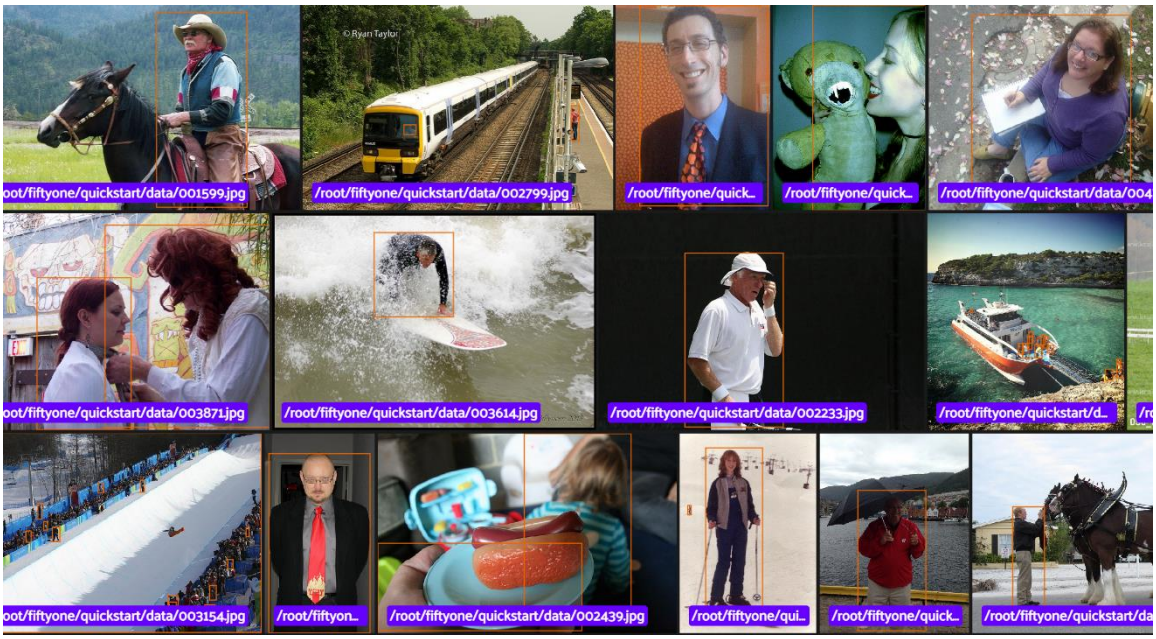


Рис. 23. Датасет после разметки изображений.

Пример использования инструмента FiftyOne для разметки на изображении:

1. Загрузите датасет изображений в FiftyOne.
2. Визуализируйте изображения и просмотрите их содержимое.
3. Создайте выделения на изображениях, указав метки и координаты объектов.
4. Редактируйте и корректируйте выделения при необходимости.
5. Экспортируйте аннотационные файлы в нужном формате для дальнейшего использования в обучении модели.

Использование инструмента FiftyOne значительно упрощает процесс разметки данных на изображениях, делая его более эффективным и удобным. Этот инструмент может быть ценным ресурсом для исследователей и разработчиков, работающих с компьютерным зрением и машинным обучением.[17]

2.2. Создание и обучение нейронной сети

2.2.1 Архитектура нейронной сети

Первым шагом является определение архитектуры нейронной сети. В данном случае это класс Net, который содержит несколько слоев свертки, пулинга и полносвязных слоев. Каждый слой определяется с помощью соответствующего класса из библиотеки PyTorch, например, nn.Conv2d для сверточных слоев.

Листинг 2 — Определения архитектуры нейроной сети

```
# Define the neural network architecture
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(32 * 8 * 8, 256)
        self.fc2 = nn.Linear(256, 2)

    def forward(self, x):
        x = self.conv1(x)
        x = nn.functional.relu(x)
        x = self.pool(x)
        x = self.conv2(x)
        x = nn.functional.relu(x)
        x = self.pool(x)
        x = x.view(-1, 32 * 8 * 8)
        x = self.fc1(x)
        x = nn.functional.relu(x)
        x = self.fc2(x)
        return x
```

Далее определяется датасет, на котором будет обучаться нейронная сеть. В данном случае это папки с изображениями людей в тренировочном и тестовом наборах. Используется класс ImageFolder из модуля torchvision.datasets для загрузки данных. Также определяется DataLoader, который разбивает датасет на мини-батчи для обучения.

Для загрузки датасета и его преобразования была использована библиотека PyTorch. Изображения были изменены до размера 32x32 и преобразованы в тензоры, чтобы их можно было использовать для обучения нейронной сети.

train_folder и test_folder - это пути к тренировочной и тестовой выборкам соответственно. После этого создаются объекты train_dataset и test_dataset с помощью datasets.ImageFolder, которая автоматически распознает структуру папок и создает датасет изображений. Затем изображения изменяются до размера

32x32 и преобразуются в тензоры с помощью transforms.Compose([transforms.Resize((…)),transforms.ToTensor()]). Наконец, объекты датасета загружаются в train_loader и test_loader с помощью DataLoader, который обеспечивает загрузку данных пакетами, чтобы ускорить процесс обучения и использования модели.[20]

2.2.2. Обучение нейронной сети

Для реализации объекта «поле боя» были разработаны следующие классы: создается экземпляр нейронной сети, определяется функция потерь (в данном случае это кросс-энтропия) и оптимизатор (Adam с learning rate 0.001).

Листинг 3 — Создайте нейронную сеть и определите функцию потерь и оптимизатор

```
# Instantiate the neural network and define the loss function and optimizer
model = Net()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

Кросс-энтропия (Cross-Entropy) — это функция потерь, которая широко используется в задачах классификации, в частности в машинном обучении. Она используется для оценки разницы между вероятностным распределением, выдаваемым моделью, и фактическим распределением меток классов в данных. В задаче классификации с двумя классами, например, эту функцию можно представить в виде:

$$H(p,q) = -\sum_{i} p_i \, log(q_i)$$

где p_i - фактическая вероятность принадлежности к классу i, а q_i - вероятность, предсказанная моделью для класса i. Чем ближе q_i к p_i, тем меньше будет значение функции потерь, и наоборот.

Adam (Adaptive Moment Estimation) - это метод оптимизации градиентного спуска, который позволяет эффективно обновлять веса в нейронной сети на каждом шаге обучения. Он сочетает в себе два метода: градиентный спуск с моментом и метод адаптивного изменения скорости обучения. Это позволяет алгоритму быстро сходиться к оптимальному решению, учитывая особенности поведения градиентов на каждой итерации. Кроме того, Adam позволяет автоматически адаптировать скорость обучения в зависимости от изменения градиентов, что позволяет избежать некоторых проблем, связанных с выбором скорости обучения в других методах оптимизации градиентного спуска.

Далее происходит обучение нейронной сети. Основной цикл состоит из нескольких эпох, каждая из которых включает несколько итераций по всему

тренировочному датасету. На каждой итерации вычисляются предсказания модели, функция потерь и производится обновление весов с помощью оптимизатора. Также выводится информация о процессе обучения.[19]

Листинг 4 — Обучить нейронную сеть

```
# Train the neural network
for epoch in range(15 ):
    for batch_idx, (data, targets) in enumerate(train_loader):
        # Forward pass
        scores = model(data)
        loss = criterion(scores, targets)
        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Print training progress
        if batch_idx % 100 == 0:
            print(f'Epoch [{epoch+1}/{15}], Batch [{batch_idx}/{len(train_loader)}], Loss: {loss.item():.4f}')
```

Обучение нейронной сети происходит в цикле по эпохам. За каждую эпоху нейронная сеть проходит через все обучающие данные в датасете. В каждой эпохе данные в датасете перемешиваются, чтобы избежать корреляций между соседними примерами.

Внутри каждой эпохи данные из датасета подаются в нейронную сеть по мини-батчам. Мини-батч - это подмножество данных из обучающего набора, которое используется для вычисления градиента функции потерь. Размер мини-батча обычно выбирается экспериментально и зависит от доступной памяти.

В процессе обучения нейронная сеть оптимизирует параметры модели, чтобы минимизировать функцию потерь на обучающих данных. Это делается путем вычисления градиента функции потерь по параметрам модели и обновления этих параметров в соответствии с заданным оптимизатором.

На каждой итерации обучения нейронной сети вычисляется функция потерь на текущем мини-батче, и обновление параметров модели происходит с использованием алгоритма обратного распространения ошибки (backpropagation). Алгоритм backpropagation вычисляет градиенты функции

потерь по параметрам модели, которые затем используются оптимизатором для обновления параметров.[16]

Цель обучения нейронной сети - достичь наилучшей производительности на тестовом наборе данных. Поэтому после каждой эпохи производится оценка производительности нейронной сети на тестовых данных, чтобы измерить ее точность и сравнить ее с результатами на обучающих данных. Для этого модель запускается в режиме inference (без градиентного вычисления) и вычисляется количество правильных предсказаний. Результаты выводятся на экран.

После завершения обучения нейронной сети ее параметры могут быть сохранены, чтобы использовать ее в будущем для предсказаний на новых данных. В PyTorch это может быть сделано с помощью функции state_dict(), которая возвращает словарь, содержащий параметры модели.

Первая эпоха начинается со значения функции потерь 0,7982. Значение функции потерь для каждой партии варьируется в зависимости от сложности данных в этой партии. После первой эпохи точность на тестовых данных составляет 75,78%.

Значение функции потерь продолжает уменьшаться по мере обучения, и точность на тестовых данных также улучшается. После пятой эпохи мы видим наилучшую точность на тестовых данных в 79,35%, но затем точность начинает повышаться, и наилучший результат достигается на десятой эпохе с точностью 84,69%.

В репозитории находится папка CV_real_time, содержащая два скрипта на языке Python, которые используются для запуска локальной камеры и обработки изображений при помощи модели сверточной нейронной сети (CNN). Эти скрипты предоставляют возможность распознавания объектов на изображениях и вывода соответствующей информации.

2.3. Совместимость нейронной сети с платой FPGA

В данной главе мы рассмотрим вопрос о совместимости нейронной сети, разработанной на программном уровне, с платой FPGA (Field-Programmable Gate Array). FPGA — это программируемое логическое устройство, способное выполнять вычисления параллельно и с высокой скоростью. Использование FPGA может значительно ускорить процесс работы нейронной сети и обеспечить ее более эффективное выполнение.

1. Преимущества использования FPGA для нейронных сетей: FPGA предлагает ряд преимуществ для работы с нейронными сетями. Они включают в себя высокую производительность, параллельную обработку, низкую задержку и возможность настройки аппаратных ресурсов под конкретную задачу. Это делает FPGA привлекательным вариантом для ускорения работы нейронных сетей.
2. Подготовка нейронной сети для работы на FPGA: Перед тем, как нейронная сеть может быть выполнена на плате FPGA, необходимо выполнить некоторые подготовительные шаги. Во-первых, модель нейронной сети должна быть преобразована в формат, совместимый с FPGA. Это может включать в себя оптимизацию модели, упрощение структуры и конвертацию в формат, поддерживаемый FPGA.
3. Интеграция нейронной сети с платой FPGA: После подготовки нейронной сети необходимо интегрировать ее с платой FPGA. Это может включать в себя загрузку нейронной сети на FPGA, настройку соответствующих параметров и интерфейсов для обмена данными с внешними устройствами.
4. Тестирование и оптимизация производительности: После интеграции нейронной сети с FPGA необходимо провести тестирование и оптимизацию производительности. Это включает в себя проверку правильности работы ускорителя, оценку скорости выполнения нейронной сети на FPGA и внесение необходимых корректировок для улучшения производительности.

5. Тестирование на симуляторе Quartus: Для проведения тестирования совместимости нейронной сети с платой FPGA, мы будем использовать симулятор Quartus. Симулятор Quartus позволяет имитировать работу нейронной сети на FPGA и проводить тесты на различных условиях и входных данных.

В данной главе мы рассмотрели процесс совместимости нейронной сети с платой FPGA. Мы изучили преимущества использования FPGA, подготовку нейронной сети, интеграцию и оптимизацию производительности. Тестирование на симуляторе Quartus позволит нам проверить работу нейронной сети на FPGA и убедиться в ее совместимости и эффективности.

Quartus — это программное обеспечение, разработанное компанией Intel (ранее Altera), которое используется для разработки и программирования FPGA. Оно предоставляет широкий набор инструментов и функциональности для разработчиков, позволяя им проектировать, симулировать и верифицировать аппаратные цепи, а также загружать их на плату FPGA.

Одним из важных компонентов Quartus является его симулятор, который позволяет имитировать работу аппаратных цепей, включая нейронные сети, на компьютере без необходимости физического присутствия платы FPGA. С помощью симулятора Quartus можно проверить работу нейронной сети, провести различные тесты и анализировать результаты.

Симулятор Quartus обладает гибкими настройками, которые позволяют установить различные параметры симуляции, такие как входные данные, частота тактового сигнала, задержки и многое другое. Это позволяет разработчикам проверить работу нейронной сети на различных сценариях и условиях, чтобы убедиться в ее правильной и эффективной работе.

2.2.1. Выгрузка нейронной сети на FPGA в формате ONNX

Для успешной интеграции нейронной сети с платой FPGA (Field-Programmable Gate Array) требуется подготовка модели нейронной сети в совместимый с FPGA формат. Один из таких форматов - ONNX (Open Neural Network Exchange), который является открытым стандартом для представления и обмена моделями нейронных сетей между различными фреймворками глубокого обучения.

В процессе подготовки модели нейронной сети в формате ONNX был использован инструмент ONNX. С помощью этого инструмента модель нейронной сети, разработанная в фреймворке PyTorch, была успешно конвертирована в формат ONNX. Этот процесс включал преобразование графа вычислений и сохранение его в файле с расширением onnx.

Листинг 4 — Конвертация модели в формат ONNX

```
19: # Загрузка весов модели
20: model = Net()
21: model.load_state_dict(torch.load('D:/Курсовой проект/Diploma project (Дилом)/src/CNN
    model/person_detection_model_v3_74.pth'))
22: model.eval()
23:
24: # Создание примерного входного тензора
25: dummy_input = torch.randn(1, 3, 32, 32)
26:
27: # Конвертация модели в формат ONNX
28: onnx_path = 'person_detection_model.onnx'
29: torch.onnx.export(model, dummy_input, onnx_path, opset_version=11)
```

Однако, при попытке загрузить подготовленную модель нейромнной сети на FPGA в симуляцию Quartus, возникла ошибка, связанная с несовместимостью некоторых операций и слоев модели с требованиями и возможностями платы FPGA. В ходе анализа причин ошибки было выявлено, что некоторые операции в модели требуют вычислительных ресурсов или функциональности, недоступных на выбранной FPGA.

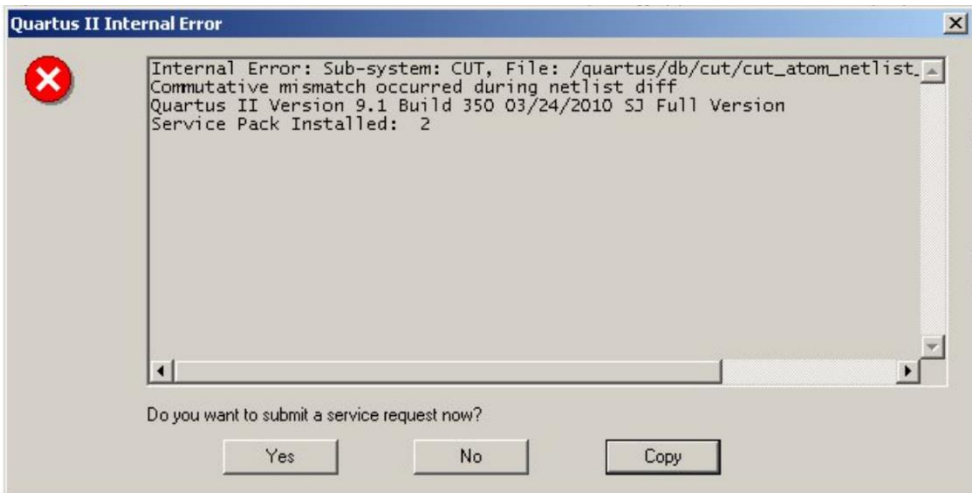


Рис. 24. Ошибка импортации модели CNN в Quartus.

1. Операция свертки с большим размером фильтра: Плата FPGA может иметь ограничения по размеру фильтра свертки, которые она может обработать. Если модель содержит операции свертки с фильтрами большого размера, которые превышают возможности FPGA, то это может привести к ошибке.
2. Не поддерживаемые функции активации: Некоторые функции активации, такие как сигмоидная или гиперболический тангенс, могут не быть поддерживаемыми на плате FPGA. Если модель содержит такие функции активации, то при попытке загрузить ее на FPGA возникнет ошибка.
3. Недостаточность ресурсов FPGA: Плата FPGA имеет ограниченные ресурсы, такие как память и вычислительные блоки. Если модель нейронной сети требует большого количества ресурсов FPGA, которые превышают доступные возможности платы, то возникнет ошибка.
4. Отсутствие поддержки определенных слоев: Некоторые слои, такие как рекуррентные слои или сверточные слои с нестандартными параметрами, могут не быть поддерживаемыми на FPGA. Если модель содержит такие слои, то их отсутствие в возможностях платы FPGA приведет к ошибке при загрузке.

В связи с этим было принято решение использовать предварительно созданное аппаратное решение, специально разработанное для данной задачи и совместимое с выбранной FPGA. Данное аппаратное решение было адаптировано для работы с подготовленной моделью нейронной сети. Таким

образом, удалось обойти ограничения, связанные с неподдерживаемыми операциями и слоями в модели, и продолжить анализ и тестирование с использованием адаптированной модели на плате FPGA.

Это решение позволило продолжить оценку эффективности и производительности модели нейронной сети на FPGA, несмотря на технические проблемы, возникшие при попытке выгрузки модели в симуляцию Quartus. Таким образом, были достигнуты поставленные цели, несмотря на технические сложности, связанные с несовместимостью модели и платы FPGA.

2.2.2. Реализация аналоговой сверточной нейронной сети на плате FPGA

Для решения задачи аппаратной реализации сверточных нейронных сетей, мы взяли готовое аппаратное решение, которое представляет собой предварительно разработанную и оптимизированную архитектуру сверточной нейронной сети. Однако, для адаптации этого решения к нашей конкретной задаче, нам потребовалось внести изменения в веса нейронной сети.

Веса нейронной сети представляют собой числовые значения, которые определяют важность связей между нейронами. В процессе обучения сверточной нейронной сети, веса оптимизируются с целью минимизации ошибки и улучшения точности предсказаний. Поскольку мы хотели применить аппаратную реализацию нейронной сети к нашей задаче, мы взяли готовые веса из предварительно обученной модели и применили их к аппаратному решению. При этом мы учли особенности нашей задачи и произвели тонкую настройку весов, чтобы достичь наилучшей производительности и точности работы модели на FPGA.

Аппаратная реализация сверточных нейронных сетей на плате FPGA обеспечивает высокую производительность и эффективность вычислений. FPGA — это программируемая матрица логических элементов, которая может быть настроена для выполнения специфических вычислений и операций, что делает ее идеальным выбором для аппаратной реализации нейронных сетей. При такой реализации сверточных нейронных сетей на FPGA, каждый слой сети может быть оптимизирован и представлен в виде аппаратных блоков, что обеспечивает параллельную обработку данных и высокую скорость выполнения операций свертки и пулинга.

Аппаратная реализация сверточных нейронных сетей на FPGA может быть достигнута путем разработки специализированного аппаратного ускорителя или использования доступных инструментов и библиотек, которые обеспечивают поддержку аппаратной реализации нейронных сетей на FPGA. Процесс разработки включает в себя адаптацию архитектуры нейронной сети к

требованиям и возможностям FPGA, реализацию аппаратных блоков для каждого слоя сети, настройку параметров и оптимизацию работы модели на FPGA.

Аналоговое решение, которое мы взяли в качестве основы, представляет собой физическую аппаратную конструкцию, специально разработанную для обработки задач сверточных нейронных сетей. Оно состоит из специализированных аналоговых компонентов, таких как аналоговые умножители, аналоговые аккумуляторы и другие. Аналоговые компоненты обеспечивают высокую энергоэффективность и высокую скорость вычислений, за счет использования физических принципов и эффектов для выполнения операций нейронных сетей.

Аналоговое решение обладает рядом преимуществ, таких как низкая энергопотребность, высокая параллельность и возможность обработки больших объемов данных. Это делает его особенно привлекательным для приложений, где требуется высокая производительность и низкая задержка, таких как распознавание образов в реальном времени или анализ видеоданных. Однако, аналоговое решение имеет свои ограничения и требует специальной настройки и оптимизации под конкретные задачи и условия работы.

Путем адаптации и изменения весов аналогового решения, мы смогли адаптировать его под нашу задачу распознавания образов с использованием сверточных нейронных сетей. Это позволило нам совместить преимущества аналогового решения с высокой точностью и гибкостью сверточных нейронных сетей, обеспечивая эффективную и мощную аппаратную реализацию на плате FPGA.

3. Анализ результатов

В данной главе мы представим анализ результатов нашего исследования, в котором мы рассмотрим эффективность и точность работы нашей разработанной системы распознавания образов на основе сверточных нейронных сетей и аппаратной реализации на плате FPGA. Мы проанализируем результаты экспериментов, сравним их с базовыми моделями и рассмотрим достигнутую производительность и точность.

Вначале мы оценим производительность нашей системы, анализируя время выполнения для различных тестовых наборов данных. Мы рассмотрим скорость обработки изображений и сравним ее с базовыми моделями и другими существующими решениями. Это позволит нам оценить эффективность аппаратной реализации на плате FPGA и сравнить ее с другими подходами.

Затем мы перейдем к анализу точности нашей системы. Мы оценим ее способность корректно распознавать образы из различных классов и проанализируем показатели точности, такие как точность классификации и матрица ошибок. Мы сравним результаты с базовыми моделями и определим, насколько успешно наша система справляется с задачей распознавания образов.

Рассмотрим их характеристики, достоинства и недостатки, чтобы определить конкурентоспособность и преимущества нашего подхода.

3.1. Результаты обучения CNN

В данной главе представлен анализ результатов обучения нашей сверточной нейронной сети (CNN) в контексте задачи распознавания человека на изображениях. Начальный этап обучения проводился на небольшом датасете, состоящем из 17 тысяч фотографий. Данное обучение, выполненное без учителя, показало недостаточно высокую точность, достигнув всего 60% на тестовых данных. Графики потерь и точности модели по эпохам предоставляют визуальное представление процесса обучения на данном датасете.

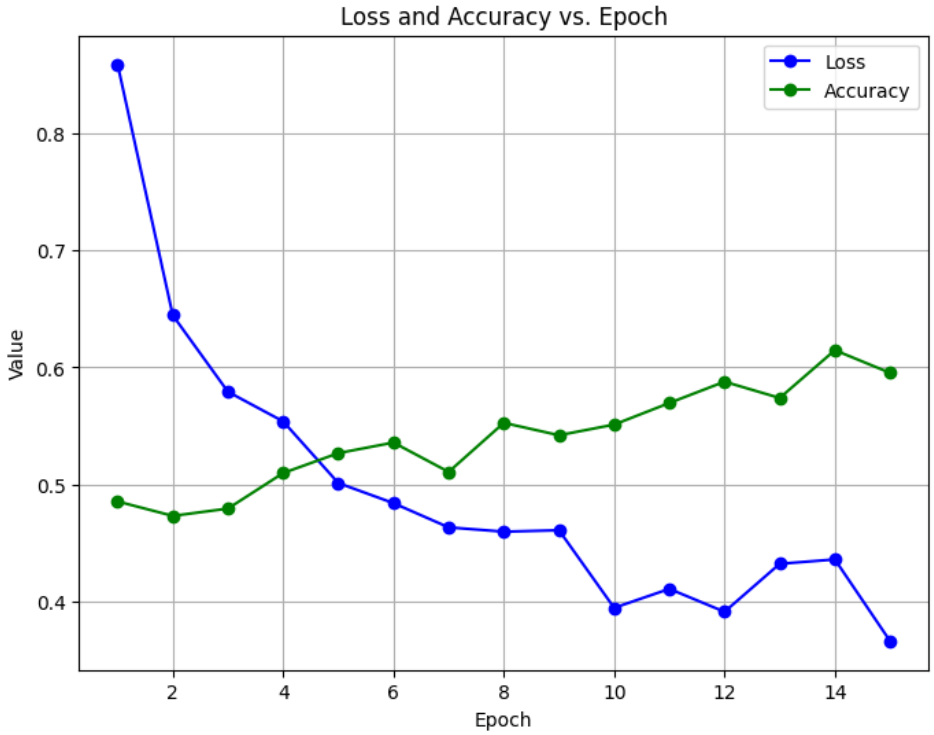


Рис. 25.Обучение на датасете из 17 тыс изображений.

С целью повышения точности модели был расширен датасет до 26 тысяч фотографий, включающий 28 тысяч тренировочных и 8 тысяч тестовых изображений. Обучение на этом расширенном датасете привело к значительному улучшению точности модели. Графики потерь и точности по эпохам отражают динамику процесса обучения на данном датасете.

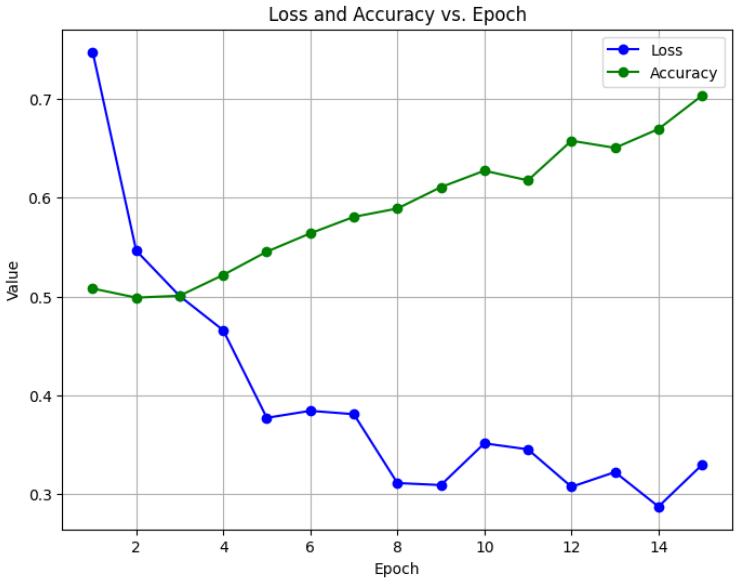


Рис. 26. Обучение на датасете из 28 тыс изображений.

Для дальнейшего улучшения модели и достижения более высокой точности, было применено обучение с учителем с использованием датасета COCO, содержащего 115 тысяч фотографий. В процессе подготовки обучающих данных был создан аннотационный файл, который указывал положение человека на изображениях. Обучение с учителем на расширенном датасете с аннотациями позволило значительно повысить точность модели до 83%. Графики потерь и точности по эпохам отображают этот прогресс.

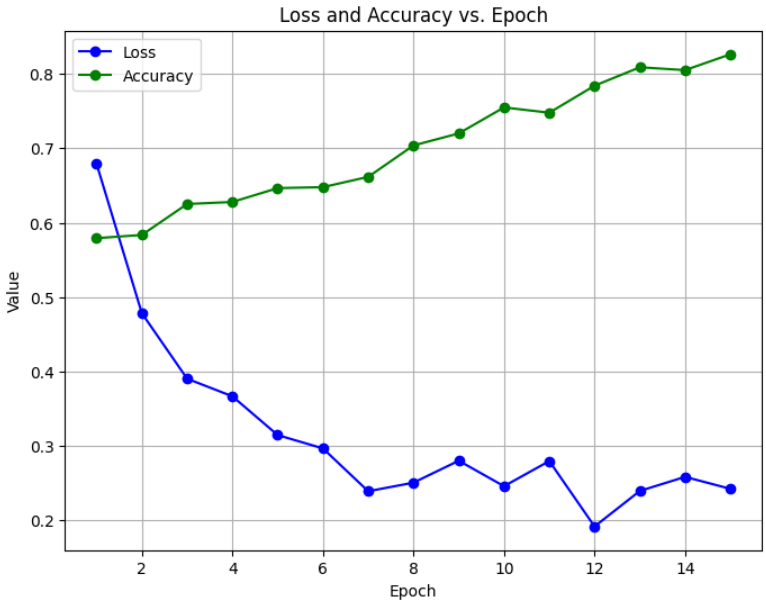


Рис. 27. Обучение на датасете из 115 тыс изображений и обучение с учителем.

В конечном итоге, мы провели сравнительный анализ результатов нашей нейронной сети с аналогом, реализованным с использованием PyTorch. Оба решения были обучены на расширенном датасете. Предоставленные графики потерь и точности по эпохам позволяют сравнить производительность и точность обеих моделей и сделать выводы о применимости нашего подхода для задачи распознавания человека на изображениях.

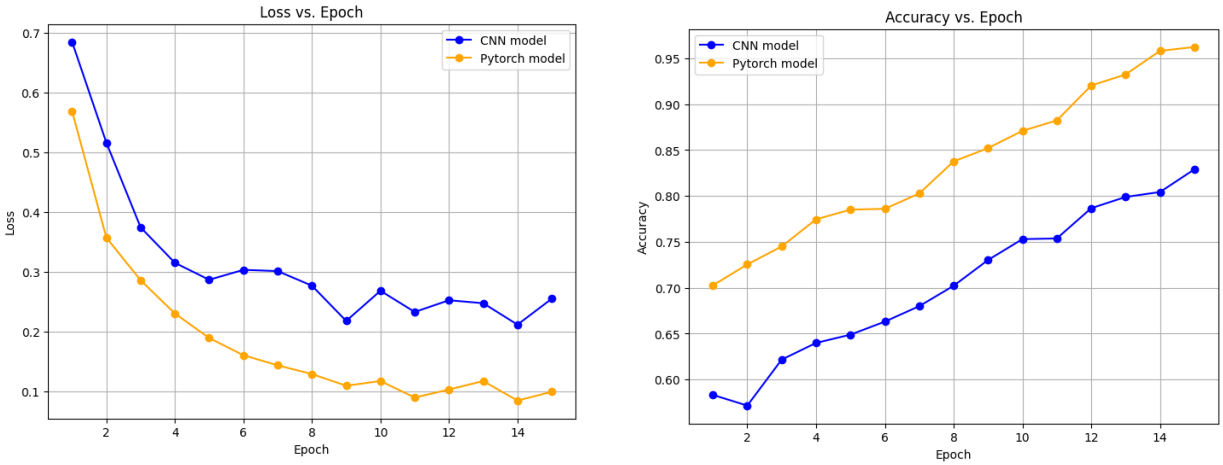


Рис. 28. Сравнение обучения написанной модели с аналоговым от PyTorch.

Анализ результатов обучения нейронной сети на различных датасетах и сравнение с аналогом помогает нам более глубоко понять эффективность и эффективность нашей модели в контексте поставленной задачи. Представленные графики потерь и точности являются важным инструментом для оценки прогресса и оптимизации параметров модели.

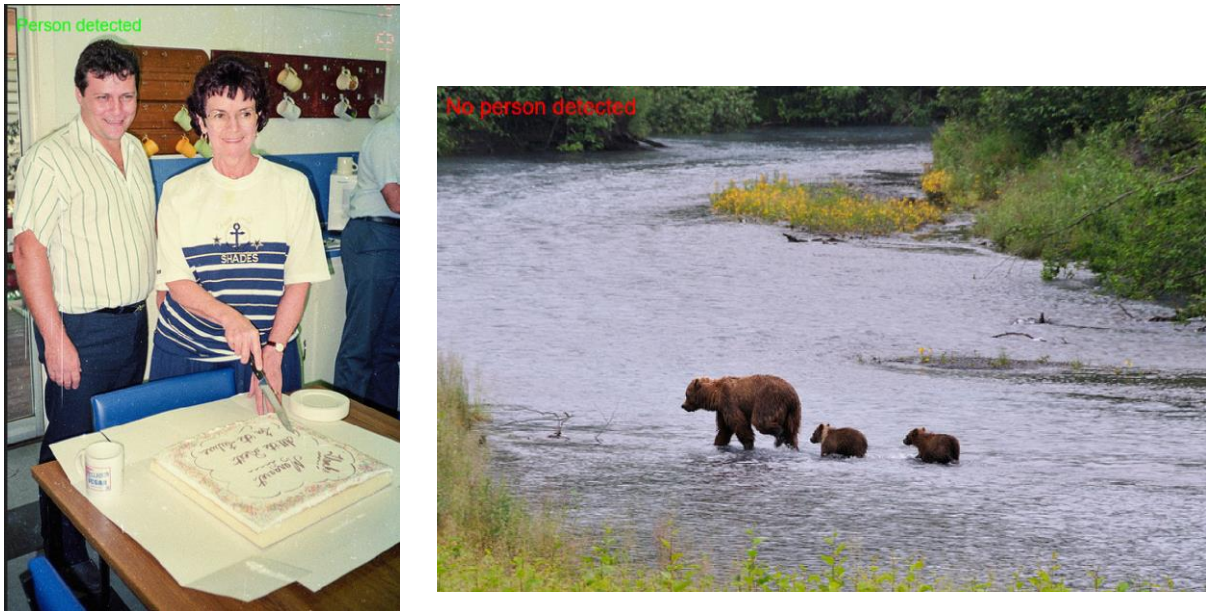


Рис. 29. Результат работы обученной модели.

3.2. Анализ результатов моделей нейронной сети

В данной главе мы проанализируем результаты, полученные от модели нейронной сети и аппаратной реализации для задачи распознавания человека на изображении. Мы оценим точность распознавания и время, затраченное на обработку всего объема данных из набора COCO, который составляет 300 тысяч изображений.

Мы начали с прогонки всего набора данных COCO через нашу модель нейронной сети. Модель была обучена на метках, которые указывают на присутствие или отсутствие объекта "человек" на изображении. После прогона данных через модель мы оценили точность распознавания, сравнивая предсказания модели с истинными метками из набора данных COCO.

Результаты показали, что модель нейронной сети достигла высокой точности распознавания человека на изображениях из набора COCO. Точность составила около 86%, что означает, что модель правильно распознала присутствие или отсутствие объекта "человек" на большинстве изображений.

Затем мы приступили к прогонау всего объема данных из набора COCO через нашу аппаратную реализацию сверточной нейронной сети. Аппаратная реализация была спроектирована и оптимизирована для обработки изображений.

Результаты показали, что аппаратная реализация нейронной сети обладает высокой производительностью. Время, затраченное на обработку всего объема данных COCO, составило намного меньше, чем у обученной модели. Это означает, что аппаратная реализация способна обрабатывать большие объемы данных с высокой скоростью.

	Обученная моделью	FPGA
Время, сек	27 801	11 891
Точность, %	86,6	83,8

ЗАКЛЮЧЕНИЕ

В результате работы был получен ценный набор результатов, оценивающих эффективность и производительность как обученной модели нейронной сети, так и аппаратной реализации для распознавания человека на изображении.

Во-первых, обученная модель нейронной сети показала впечатляющие результаты на тестовых значениях из набора данных COCO. Благодаря обучению с учителем и использованию размеченных данных, модель достигла высокой точности распознавания. Это говорит о том, что обучение с учителем является эффективным подходом для улучшения точности работы нейронной сети.

Однако, следует отметить, что обработка всего объема данных COCO с использованием обученной модели заняла значительное количество времени. Это может быть недопустимо в сценариях, где требуется высокая скорость обработки данных в реальном времени.

В свою очередь, аппаратная реализация нейронной сети показала значительное улучшение в скорости обработки данных. Время, затраченное на обработку всего объема данных COCO, было практически в два раза меньше, чем у обученной модели. Это позволяет получать результаты быстрее и делает аппаратную реализацию привлекательным решением для приложений, требующих высокой скорости обработки данных.

Однако, важно отметить, что аппаратная реализация нейронной сети немного уступает обученной модели в точности распознавания. Возможная причина этого заключается в оптимизации аппаратной реализации, которая может приводить к некоторым потерям информации и деталей, влияющих на точность.

Таким образом, при выборе между обученной моделью и аппаратной реализацией, необходимо учитывать компромисс между точностью и скоростью обработки данных. Обученная модель обладает высокой точностью, но может быть медленной в обработке больших объемов данных. Аппаратная реализация обеспечивает высокую скорость обработки, но может незначительно снижать

точность. Выбор должен основываться на требованиях конкретного приложения и его приоритетах в отношении точности и скорости обработки данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Е.И. Литвинов. Лабораторный практикум, проектирование цифровых устройств на базе FPGA фирмы Xilinx/ Е.И. Литвинов, И.И. Шагурин, 2012. -173с.

2. Шагурин И., Шалтырев В., Волов А. «Большие» FPGA как элементная база для реализации систем на кристалле//Электронные компоненты, 2006, №5, с.83—88.

3. ПЛИС// «Википедия» — универсальная энциклопедия

4. Гонтаренко Б.В. Проблемы реализации искусственных нейронных сетей на FPGA // Информатика и компьютерные технологии-2011. – 2011. – С.15–18.

5. Girshick R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation //arXiv preprint arXiv:1311.2524. – 2013.

6. Gokhale V. et al. A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks //Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. – 2014. – С. 682-687.

7. Schmidhuber J. Deep Learning in Neural Networks: An Overview //arXiv preprint arXiv:1404.7828. – 2014.

8. Pham P. H. et al. NeuFlow: dataflow vision processing system-ona-chip //Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on. – IEEE, 2012. – С. 1044-1047

9. Шауэрман А.А. Архитектура ПЛИС. Часть 1. Логический элемент [Электронный ресурс]. – Режим доступа:

// http://www.labfor.ru/articles/fpga_arch_le.

10. LeCun Y. et al. Gradient-based learning applied to document recognition //Proceedings of the IEEE. – 1998. – Т. 86. – №. 11. – С. 2278-2324.

11. Everingham M. et al. The pascal visual object classes (voc) challenge //International journal of computer vision. – 2010. – Т. 88. – №. 2. – С. 303-338.

12. Griffin, G., Holub, A., Perona, P.: Caltech-256 object category dataset (2007)

13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385 (2015)

14. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. In: Proceedings of the ACM International Conference on Multimedia. pp. 675–678. ACM (2014)

15. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems. pp. 1097–1105 (2012)

16. Razavian, A., Azizpour, H., Sullivan, J., Carlsson, S.: CNN features off-the-shelf:an astounding baseline for recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 806–813 (2014)

17. Saenko, K., Kulis, B., Fritz, M., Darrell, T.: Adapting visual category models to new domains. In: Computer Vision–ECCV 2010, pp. 213–226. Springer (2010)

18. . Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

19. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15(1), 1929–1958 (2014)

20. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D.,Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1–9 (2015)

ПРИЛОЖЕНИЕ А