

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
Вычислительная техника

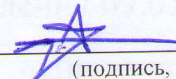
Направление подготовки 09.03.01 «Информатика и вычислительная техника»
Профиль Вычислительные машины, комплексы, системы и сети

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ
РАБОТА БАКАЛАВРА**

на тему

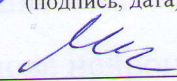
**СИСТЕМА ОБНАРУЖЕНИЯ И ОТСЛЕЖИВАНИЯ ОБЪЕКТОВ
В ВИДЕОПОТОКЕ НА ОСНОВЕ НЕЙРОННОЙ СЕТИ**

Студент


(подпись, дата)

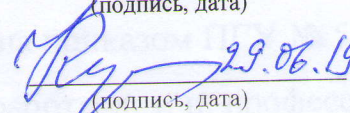
Буряков Артём Вадимович

Руководитель


(подпись, дата)

Митрохин М. А.

Нормоконтролёр


(подпись, дата) 29.06.19

Кучин А. В.

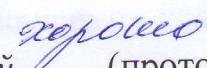
Работа допущена к защите (протокол заседания кафедры от 31.05.2019
№38-09-14)

Заведующий кафедрой


(подпись)

Митрохин М. А.

Работа защищена с отметкой



(протокол заседания ГЭК от 01.07.19, №28)

Секретарь ГЭК


(подпись)

Гурин Е. И.

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

«Утверждаю»

Заведующий кафедрой

 М. А. Митрохин
«23»  2019 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Буряков Артём Вадимович, гр. 15ВВ1, факультета вычислительной техники, направления 09.03.01.
2. Тема работы: «Система обнаружения и отслеживания объектов в видеопотоке на основе нейронной сети».

Тема утверждена приказом ПГУ № 534/о от 29.04.2019.

3. Руководитель работы д. т. н. профессор, зав. каф. ВТ ПГУ Митрохин М. А.

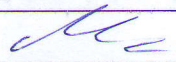
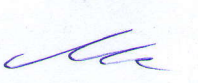
4. Задание на работу (назначение разработки, исходные данные и т.п.):

Разработать систему трекинга, слежения и распознавания объектов в видеопотоке на основе сверточной нейронной сети. Система реализуется в виде программного продукта. Данная система должна осуществлять распознавание образов объектов на каждом кадре, их классификацию и локализацию, трекинг (отслеживание) каждого объекта на последовательности кадров из видеопотока. Источником видеопотока являются видео файлы и потоковая трансляция с камер видеонаблюдения.

5. Перечень подлежащих разработке вопросов:

Требуется произвести исследование и анализ предметной области, существующих алгоритмов трекинга объектов, основных принципов построения искусственных нейронных сетей, алгоритмов фильтрации данных. Выбрать технологии для проектирования и разработки системы. Реализовать архитектуру системы. Разработать и описать алгоритмы функционирования системы.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области	40%	12.05.2019	
2	Выбор средств разработки	10%	15.05.2019	
3	Поиск реализаций алгоритмов	10%	18.04.2019	
4	Проектирование системы	20%	24.04.2019	
5	Оформление выпускной работы	20%	31.05.2019	

Дата выдачи задания: "29" апреля 2019 г.

Руководитель бакалаврской работы


(подпись, дата)

Митрохин М. А.

Задание к исполнению принял студент


(подпись, дата)

Буряков А. В.

Работу к защите допустить
Декан факультета


(подпись, дата)

Фионова Л. Р.

РЕФЕРАТ

Пояснительная записка содержит 109 листов, 38 рисунков, 14 источников, 2 приложения.

ИСКУССТВЕННАЯ НЕЙРОННАЯ СЕТЬ, СВЕРТОЧНАЯ НЕЙРОННАЯ СЕТЬ, КОМПЬЮТЕРНОЕ ЗРЕНИЕ, МАШИННОЕ ОБУЧЕНИЕ, ТРЕКИНГ, КЛАССИФИКАЦИЯ, ЛОКАЛИЗАЦИЯ, ТОПОЛОГИЯ.

Цель работы – разработка системы, выполняющей задачу детектирования и классификации различных видов транспорта в кадрах видеопотока на основе сверточной нейронной сети.

В результате проведенной работы была разработана система, способная находить, захватывать и отслеживать различные виды транспорта на любом видеоисточнике, таком как видеопоток, захваченный с камеры или видеофайл.

Разработанная программа в дальнейшем может стать модулем системы распознавания автомобильных номеров, определения ДТП и анализа городского трафика. Графическая часть содержит диаграмму системы и алгоритмов работы программных модулей.

					ПГУ 2.090301.10.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.	Буряков А.В.				Система обнаружения и отслеживания объектов в видеопотоке на основе нейронной сети. Пояснительная записка.	Лит.	Лист
Провер.	Митрохин М.А.						
						4	109
Н. контр.	Кучин А.В.					ФВТ, Кафедра ВТ, гр. 15ВВ1	
Утвер.							

Содержание

ПЕРЕЧЕНЬ ПРИНЯТЫХ СОКРАЩЕНИЙ	7
ВВЕДЕНИЕ	8
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ.....	10
1.1 Биологические предпосылки к созданию нейронных сетей ...	11
1.2 Введение в искусственные нейронные сети.....	15
1.2.1 Нейрон.....	16
1.2.2 Перцептрон	19
1.2.3 Обучение	24
1.3 Компьютерное зрение.....	26
1.4 Сверточные нейронные сети.....	29
1.4.1 Базовые слои	31
1.4.2 Архитектуры сверточных нейронных сетей	42
1.4.3 Алгоритмы детектирования объектов	50
1.5 Фильтрация	55
1.5.1 Фильтр Калмана.....	56
1.5.2 Алгоритм SORT	60
2. ТРЕБОВАНИЯ К РАЗРАБОТКЕ СИСТЕМЫ ТРЕКИНГА.....	62
3. ВЫБОР СРЕДСТВ ПРОЕКТИРОВАНИЯ И РАЗРАБОТКИ	63
3.1 Python	63
3.1.1 Numpy.....	64
3.1.2 PyTorch	65
3.1.3 OpenCV.....	66
3.2 YOLO.....	67

3.3	SORT.....	69
3.4	PyCharm.....	69
4.	ПРОГРАММНАЯ РЕАЛИЗАЦИЯ.....	70
4.1	Структура скрипта.....	71
4.2	Описание классов и файлов	73
4.3	Диаграмма классов UML.....	74
4.4	Результаты работы системы.....	75
5.	ЗАКЛЮЧЕНИЕ.....	76
	СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.....	77
	ПРИЛОЖЕНИЕ А: ЛИСТИНГИ ПРОГРАММ	79
	Приложение А.1 – Файл «tracker.py»	79
	Приложение А.2 – Файл «sort.py»	82
	Приложение А.3 – Файл «models.py».....	88
	Приложение А.4 – Файл «utils.py»	94
	Приложение А.5 – Файл «torch_utils.py»	100
	Приложение А.6 – Файл «parse_config.py»	101
	ПРИЛОЖЕНИЕ Б: ПРЕЗЕНТАЦИЯ.....	102

ПЕРЕЧЕНЬ ПРИНЯТЫХ СОКРАЩЕНИЙ

ИНС – Искусственная нейронная сеть

CNN – Сверточная нейронная сеть

IDE – Интегрированная среда разработки

MOT – Multiple Object Tracking, Трекинг множества объектов

Open-source – Программное обеспечение с открытым исходным кодом

Convolution layer – Сверточный слой

Pooling, subsampling layer – Слой объединения, подвыборки

GPU – Графический ускоритель

FPS – Frames per second, кадры в секунду

ИИ – Искусственный интеллект

					ПГУ 2.090301.10.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

ВВЕДЕНИЕ

Одними из важнейших перспективных направлений исследований и разработок являются области машинного обучения, распознавания образов, трекинга и компьютерного зрения. Ускоряющиеся темпы развития технологий информационного общества, развитие робототехники, беспилотных автомобилей и летательных аппаратов, развитие интернета вещей и систем искусственного интеллекта выводят эту область в авангард развития информационных технологий.

В нашу жизнь все больше проникают технологии, использующие методы сбора и анализа данных, кластеризации и классификации, статистического вывода. В корпоративной, промышленной и повседневной жизни повсеместно внедряются технологии распознавания, перешедшие не так давно на новый качественный уровень.

Прогресс развивается, считающиеся сложнейшими ранее задачи сейчас способны выполняться на современных компьютерах и мобильных устройствах. Развитие виртуальной реальности, дополненной реальности, интернета вещей, усложнение искусственного интеллекта в играх и программном обеспечении требуют новых идей и подходов, нового уровня скорости и точности распознавания образов, а развитие вычислительных мощностей позволяет их реализовать.

Задача трекинга, а именно определение местоположения одного или нескольких объектов в видеопотоке кадров во времени на данный момент является одной из наиболее распространенных и необходимых из задач компьютерного зрения. Она находит применение в составе систем контроля скорости, слежения за территорией, обнаружения объектов, распознавания

					ПГУ 2.090301.10.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

лиц, беспилотных автомобилях, системах контроля качества на промышленных предприятиях и многих других.

Трекинг принято разделять в зависимости от необходимости отслеживания одного или нескольких объектов. Для отслеживания одиночного объекта на данный момент разработано множество алгоритмов, однако задача трекинга множества объектов (MOT) гораздо более универсальна и применима для широкого спектра прикладных задач.

Отслеживание нескольких объектов является более сложной задачей, особенно в случае их частичного перекрытия (окклюзии), или их значительного сходства, но MOT гораздо более универсальна и применима для широкого спектра прикладных задач.

Любая система трекинга состоит из двух основных частей:

- Представление и Локализация Целевого Объекта (Target Representation and Localization)
- Фильтрация и Объединение Данных (Filtering and Data Association)

Классические алгоритмы представления и локализации одиночного объекта, например каскадные классификаторы Виолы-Джонса, проигрывают методам, основанным на сверточных нейронных сетях (CNN), особенно в случае MOT. CNN лучше справляются с необработанными изображениями, выдают более точные результаты и не так сильно подвержены влиянию перспективы изображения.

Фильтрация и Объединение данных традиционно реализуется с помощью фильтра Калмана и позволяет предсказать будущее состояние динамической системы и каждого ее объекта на основе предыдущего и текущего кадра.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Искусственная нейронная сеть – это математическая модель и ее программная реализация, построенная по принципу функционирования биологических нейронных сетей – нервных клеток живого организма. Идея создать ИНС появились очень давно, еще в 40-х годах, она была достаточно простой. Лучшая машина для принятия решений – человеческий мозг. Почему бы не сделать компьютерную симуляцию мозга? Однако, на практике все оказалось несколько сложнее. Нейронов в мозге огромное количество, необходимо было также выяснить по каким законам они функционируют, как взаимодействуют друг с другом. К тому же вычислительные мощности того времени и отсутствие развитых технологий не позволили сразу реализовать сложный алгоритм для использования в прикладных задачах.

В полном соответствии с законом Мура, с тех времен постоянно возрастали мощности вычислительных машин, появлялись новые инструменты, более высокоуровневые языки программирования, различные фреймворки, библиотеки и прочие инструменты разработки. Развитие интернета сыграло важную роль как в улучшении исследователями структур ИНС, так и в накоплении различного рода данных, размеченных и подготовленных для обучения – датасетов. Без них нейросети невозможно обучить для дальнейшего использования в прикладных задачах.

Со временем ИНС постепенно развивались исследователями, учитывались накопленные ошибки и как результат в 2012 году ежегодное соревнование ImageNet по распознаванию и классификации изображений выигрывает сверточная нейронная сеть AlexNet, с отрывом от ближайшего

					ПГУ 2.090301.10.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

соперника более чем в два раза (с количеством ошибок 15,4% против 26,2% у второго места).

Через год после публикации AlexNet все участники конкурса ImageNet начали использовать сверточные нейронные сети для решения задачи классификации. С тех пор подходы, не основанные на сверточных нейронных сетях, больше не побеждали в конкурсе, а процент ошибок последующих моделей с каждым годом становился все ниже. Успех AlexNet в 2012 году, архитектуры CNN, которая ошеломила конкурентов в конкурсе ImageNet, стал свидетельством революции, которая де-факто произошла в этой области.

В дальнейшем многочисленные исследователи начали использовать системы на основе CNN, которые стали традиционной технологией в компьютерном зрении. AlexNet была первой реализацией сверточных нейронных сетей, она новую эру исследований. Сейчас реализовать AlexNet и другие, более развитые архитектуры нейросетей, стало проще с помощью библиотек глубокого обучения: PyTorch, TensorFlow, Keras и других.

1.1 Биологические предпосылки к созданию нейронных сетей

Прототипом для создания искусственных нейронных сетей послужили биологические нейронные сети, а точнее зрительная система нашего мозга. Более чем две трети сенсорной информации, которая попадает в человеческий мозг, поступает со зрительных органов восприятия. Изображение попадает к нам из сетчатки глаза, проходит череду зрительных зон и заканчивается в височной зоне. Каждая зрительная зона отвечает за

					ПГУ 2.090301.10.001 ПЗ	Лист
						11
Изм.	Лист	№ докум.	Подпись	Дата		

свою строго определенную функцию. Именно на этом принципе и основаны искусственные нейронные сети, поэтому для их понимания стоит рассмотреть работу биологических зрительных зон подробнее.

Первая зрительная зона была исследована Дэвидом Хьюбелем и Торстеном Визелем в 1962 году. Они проводили эксперименты на кошках. Так как тогда невозможно было применить метод магнитно-резонансной томографии (тогда он попросту не был изобретен), исследователям приходилось вживлять электроды напрямую в мозг животным, точнее в зрительную кору головного мозга. В ходе эксперимента кошкам показывались различные движущиеся объекты. То, на что реагировали клетки мозга и было тем объектом, которое распознавало животное. В результате многочисленных опытов и анализа данных было открыто множество любопытных фактов.

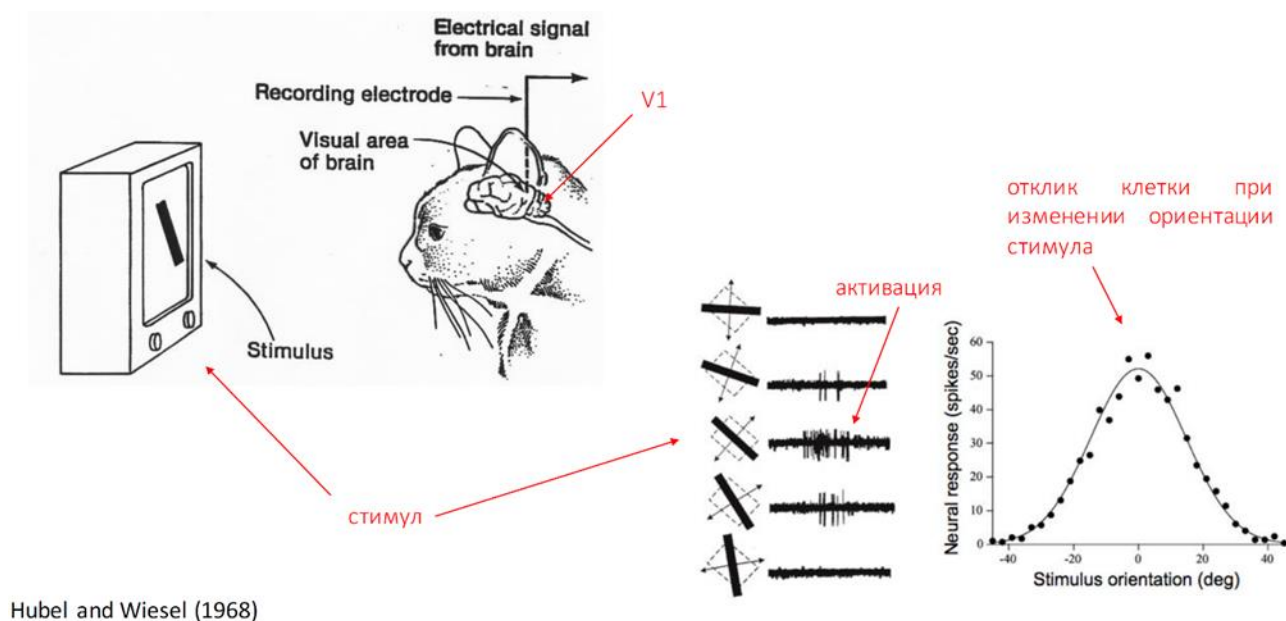


Рисунок 1 - Демонстрация опыта Хьюбеля и Торстена

Во-первых, индивидуальные клетки в зрительной коре кошачьего мозга реагируют на специфические паттерны визуальных сигналов, например, края или линии, расположенные под определенными углами. До этого открытия нам было известно лишь то, колбочки и палочки в глазу реагируют на свет, и что вся входящая визуальная информация в конечном счете направляется к зрительной коре. В каждом из наших глаз имеется около 120 млн палочек и около 6 млн колбочек. Это означает, что в любой момент времени они генерируют 252 млн различных нервных импульсов. Но, однако, мы воспринимаем мир в виде целостного образа, а не состоящим из единичных сигналов с колбочек и палочек. Оставалось большой загадкой, как же наш мозг собирает всю эту визуальную информацию воедино, что позволяет нам в итоге воспринимать окружающий мир в виде целостной картинки. Значит, где-то в зрительной системе все зрительные импульсы каким-то образом объединяются и суммируются.

Второй вывод из опыта относится к нюансам бинокулярного зрения, что нам в данной работе не интересно. Отметим лишь, что наличие двух глаз и двойная обработка похожей информации позволяет мозгу ощущать глубину пространства.

Третье открытие говорит нам, в мозге корковые клетки (Индивидуальные клетки внешней поверхности головного мозга, которая называется корой) группируются в виде маленьких колоннообразных микроструктур. Эти структуры расположены перпендикулярно к поверхностям мозга, а клетки внутри каждой из них реагируют на одни и те же простые паттерны, которые называли рецептивными полями. Рецептивное поле – это та часть изображения, которую обрабатывает каждая такая клетка нашего мозга, те самые паттерны, в простейшем случае представляющие собой линии под разными углами. После этого открытия стало понятно, почему мы не воспринимаем мир как 252 млн точек. Но тогда почему мы не

воспринимаем мир как смешение линий и краев? Поскольку теперь нам известно, что информация о линиях и краях передается к корковым клеткам от палочек и колбочек, разумно предположить, что если корковые клетки суммируют результаты с нескольких палочек и колбочек в виде паттернов, почему какие-нибудь клетки высшего уровня не могут выполнять аналогичную функцию с сигналами от нескольких корковых клеток и объединять их в более сложные образы?

Представим, что клетка высшего уровня, назовем ее сложной, принимает сигналы от двух клеток более низкого уровня, которые мы по аналогии назовем простыми. Одна из простых клеток интенсивнее всего реагирует на вертикальную линию, а другая на горизонтальную. В этом случае, если в поле зрения присутствуют обе линии, каждая из двух простых клеток интенсивно отреагирует, а сложная клетка просуммирует эти сигналы. Тогда мы можем сказать, что данная сложная клетка реагирует на сочетание двух перпендикулярных линий, или, говоря проще, на крест. Можно сказать, что в данном случае образ крестов будет являться рецептивным полем этой сложной клетки.

Эту логику можно продолжить: группы сложных клеток связаны с группами еще более сложных клеток, назовем их сверхсложными. Реакция сверхсложной клетки говорит о наличии в поле зрения паттерна еще более высокого уровня, соответственно, ее рецептивное поле увеличивается. Это может быть не крест, а, например, какая-нибудь геометрическая фигура. Итак, проходя все далее и далее, от простейших паттернов к сложным, мы в конце концов получаем способность зрительно воспринимать все изображение целиком, образы из которых оно состоит и весь окружающий мир как комбинацию этих образов.

Результаты этого исследования принесли Хьюбелу и Визелю нобелевскую премию в области физиологии и медицины, а в дальнейшем

					ПГУ 2.090301.10.001 ПЗ	Лист
						14
Изм.	Лист	№ докум.	Подпись	Дата		

позволили построить модели искусственной нейронной сети, во многом повторяющие описанную выше биологическую структуру зрительной системы. Важное свойство, сохраненное в нейронных сетях – при возрастании рецептивных полей увеличивается и сложность объектов, которые она способна распознать.

1.2 Введение в искусственные нейронные сети

Искусственная нейронная сеть представляет собой математическую модель, построенную и организованную по принципам функционирования биологических нейронных сетей, состоящих из нервных клеток.

После изучения базовых принципов работы головного мозга, в частности рассмотренных выше процессов, протекающих в зрительной системе, начались попытки моделирования этих процессов, которые совпали с началом стремительного развития мощностей ЭВМ. Искусственная нейронная сеть представляет собой систему, состоящую из соединенных в особой последовательности и взаимодействующих друг с другом простых процессоров (искусственных нейронов). Каждый нейрон способен только принимать сигналы и формировать выходной сигнал в зависимости от поступающих значений. При создании достаточно большой и сложной сети, состоящей из множества нейронов, они способны решать довольно сложные задачи.

Отличительной особенностью нейросетей от других программ, моделей и алгоритмов является то, что они обладают способностью обучаться. Технически обучение представляет из себя процесс нахождения необходимых коэффициентов связей между нейронами. В процессе обучения нейросеть способна находить сложные зависимости между входными данными, а также обладает возможностью выполнять обобщение данных.

					ПГУ 2.090301.10.001 ПЗ	Лист
						15
Изм.	Лист	№ докум.	Подпись	Дата		

Это означает способность находить закономерности, схожие с теми что присутствовали в обучающей выборке, при поступлении новых данных.

Современные ИНС прошли эволюцию от простейшего перцептрона до сложных многослойных сетей, способных решать различные задачи, такие как классификация, кластеризация, распознавание образов, прогнозирование и многие другие.

1.2.1 Нейрон

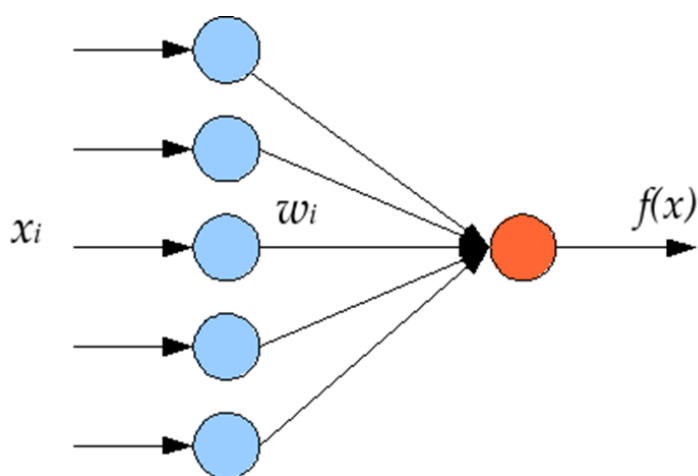


Рисунок 2 - Схема искусственного нейрона

Простейший элемент любой искусственной нейронной сети, напоминающий клетку мозга — нейрон. Он имеет входные элементы, которые по умолчанию располагаются слева и выходные справа, а также связи с ними, по аналогии с биологическими синапсами. Каждая связь имеет определенный вес.

Математически выход нейрона равен $y = f(u)$, где $u = \sum_{i=1}^n w_i x_i + w_0$. Здесь $f(u)$ называется передаточной функцией, $x_i w_i$ — соответственно сигналы на входах и веса входов. Дополнительный вес w_0 используется для инициализации нейрона при создании модели и часто называется

смещением. Таким образом, только передаточная функция $f(u)$ и значения весов определяют выход нейрона, а каждый отдельный нейрон полностью характеризуется своей передаточной функцией. Она также называется функцией активации. Выход любого нейрона лежит в диапазоне $[0,1]$ или $[-1,1]$. Таким образом функция активации нормализует выходные значения нейрона так, чтобы те находились в выходном диапазоне. Однако при этом поступающее на вход число может находиться вне этих диапазонов, т.к. оно может быть суммой с входов нескольких нейронов. Использование различных передаточных функций позволяет нейронной сети быть нелинейной, поэтому обычно при построении модели нейросети функции комбинируют по слоям.

Необходимо подробнее разобраться в видах передаточных функций. Их существует достаточно много, основные: линейная, сигмоид, пороговая и гиперболический тангенс.

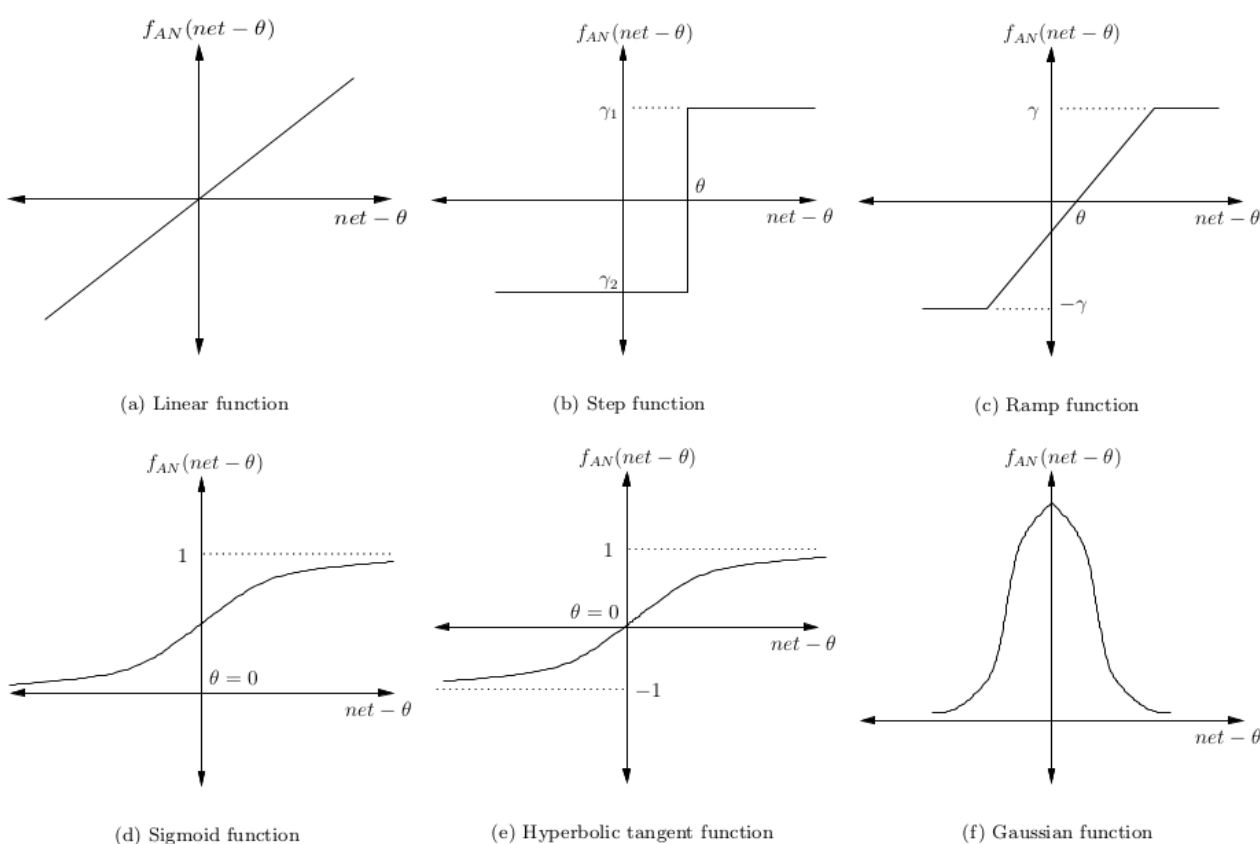


Рисунок 3 - различные виды передаточных (активационных) функций

Отличия в основном заключаются в диапазоне выходных значений. Линейная функция не производит никаких преобразований и чаще всего используется во входном, то есть первом слое нейросети, в который поступают данные. Пороговая выдает дискретный результат – 0 либо 1 после превышения на входе указанного уровня. Сигмоид выдает результат в диапазоне $[0,1]$, а гиперболический тангенс используют, когда значения могут быть отрицательными, диапазон выходных значений $[-1,1]$.

Существуют и другие активационные функции. Важный нюанс заключается в том, что для современных алгоритмов обучения необходимо, чтобы функция была дифференцируемой на всей числовой оси. К ним не относится пороговая функция, хотя именно нейроны такого типа составляли первую ИНС – перцептрон, который до сих пор не теряет актуальности и используется повсеместно в составе моделей, решающих различные задачи. О нем и об алгоритмах обучения будет рассказано далее.

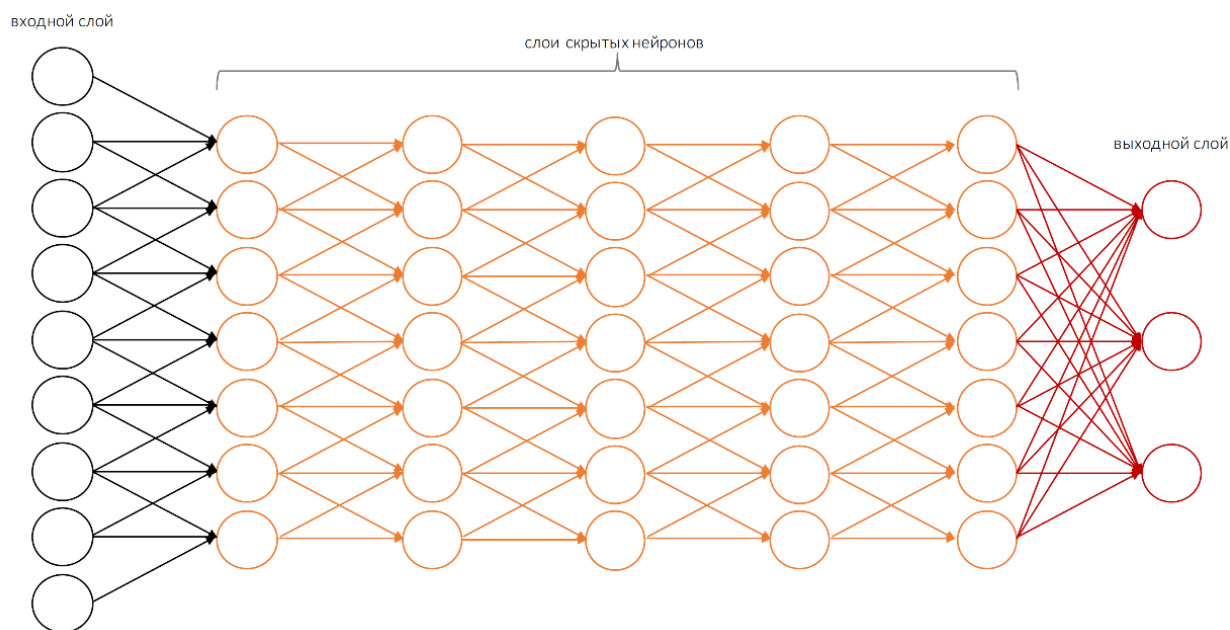


Рисунок 4 – общая топология нейронной сети.

В искусственных нейронных сетях нейросетях принято выделять входной слой, один или множество промежуточных слоев, которые принято называть скрытыми и выходной слой. Простейший перцептрон, содержащий один скрытый слой, способен выполнять только самые простые операции. Для того, чтобы выполнять более сложные вычисления, необходима структура с большим количеством скрытых слоёв. С другой стороны, в многослойных нейронных сетях при возникает проблема поиска необходимых коэффициентов. Необходимо, основываясь на ответе нейросети, продвинутся в обратном направлении и найти те нейроны, из-за которых был дан неверный результат.

1.2.2 Перцептрон

Перцептрон – одна из первых математических моделей восприятия информации мозгом, реализованная на базе вычислительной машины. Он получил свое название от латинского слова *perception* – восприятие. Перцептрон стал одной из первых моделей нейросетей, и был предложен еще в 1957 году Фрэнком Розенблаттом, а затем реализован им же в 1960 на базе первого в мире нейрокомпьютера «Марк-1», который был способен распознавать некоторые буквы английского алфавита.

Перцептроны стали активно исследовать, возлагая на них большие надежды. Однако на тот момент они оказались недостаточно совершенны и имели серьезные ограничения. На волне роста популярности искусственных нейросетей в 1969 году вышла книга за авторством Марвина Минского и Сеймура Паперта, которая объясняла недостатки перцептронов и надолго охладила интерес к искусственным нейронным сетям. Позднее Минский сожалел, что книга оказала такое сильное влияние на эту область. Но со

временем были открыты новые виды нейронных сетей, а также алгоритмы обучения, что снова возродило интерес к ИНС.

Для обучения данной модели нейронной сети первоначально был разработан «метод коррекции ошибки», который основан на многократной итеративной коррекции весов перцептрона, основываясь на сравнении выходных данных и ожидаемых результатов. Однако данный метод обучения испытывал сложности при увеличении количества скрытых слоев, так как при их увеличении становится сложно найти веса, требующие коррекции. Во многом именно после открытия метода обратного распространения ошибки возобновился интерес к искусственным нейронным сетям, потому что при увеличении количества слоев нейросеть становится способной решать более сложные задачи.

В современной классификации нейросетей перцептрон с одним скрытым слоем представляет собой простейшую искусственную нейронную сеть. Заложенные в нем принципы применяются в более сложных сетях. Рассмотрим его поподробнее.

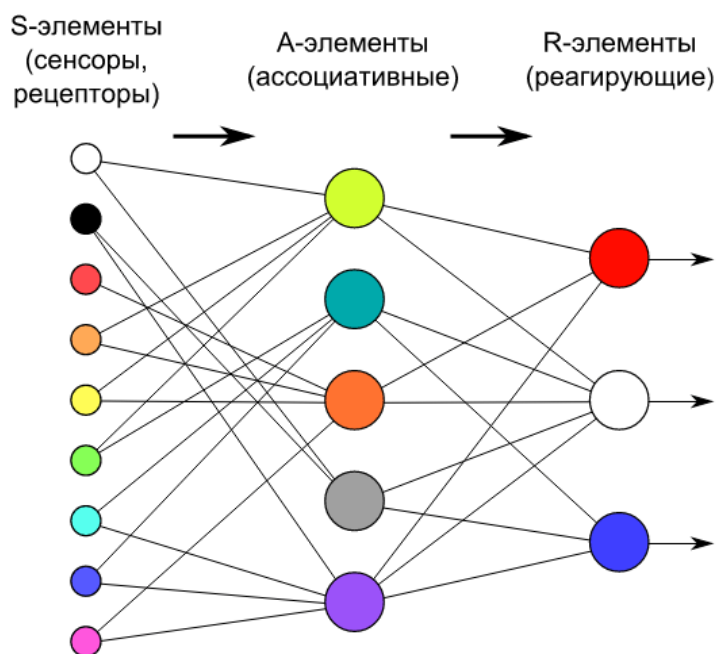


Рисунок 5 - схема перцептрона с одним скрытым слоем

В общем своем виде перцептрон представляет собой систему из элементов трех разных типов: сенсоры S, ассоциативные элементы A и реагирующие элементы R.

Первыми начинают работать сенсоры. Они могут находиться либо в состоянии покоя (сигнал равен 0), либо в состоянии возбуждения (сигнал равен 1).

Далее сигналы от сенсоров передаются на ассоциативные элементы с помощью специальных S-A связей, которые имеют дискретные веса, равные -1, 0, либо 1. Пройдя через них сигналы от S-элементов попадают в A-элементы, причем одному ассоциативному может соответствовать несколько сенсорных элементов. Как было упомянуто выше, активационная функция нейронов (в данном случае ассоциативных элементов) в перцептроне представляет собой пороговую функцию, то есть она даст на выходе сигнал, равный 1, если входные сигналы превысят некий порог θ . В таком случае элемент возбуждается, иначе генерируется нулевой сигнал.

A-элементы называли ассоциативными, потому что они агрегируют сигналы от сенсорных элементов, также как и в человеческом мозге постепенно повышается рецептивное поле клеток. Представим себе несколько сенсоров, каждый из которых распознает отдельную часть буквы «Р». Только их совокупность способна возбудить A-элемент, ассоциирующийся с этой буквой. На другие буквы этот элемент реагировать не будет, благодаря чему он и получил такое название.

Чтобы добраться до R элемента, сигналы проходят по A-R связям, у которых также имеются свои веса. Как и ассоциативные, результирующий элемент суммирует значения входных сигналов, помноженные на веса по формуле $\sum_{i=1}^n w_i x_i - \theta$, где θ – порог передаточной функции. В итоге перцептрон дает на выходе 1 либо -1, зависит от того, был ли превышен порог. Эти значения и представляют собой ответ – найдена или нет искомая буква.

Понятно, что при создании модели и ее инициализации (связям даны случайные веса) перцептрон не знает, что от него требуется и на какие именно объекты он должен реагировать. Говоря другими словами, сеть необходимо обучить. Классический метод обучения для перцептрона – метод коррекции ошибки, являющейся разновидностью метода обучения с учителем. Его суть состоит в изменении веса связи при неправильной реакции перцептрона на входные данные.

Перцептроны хорошо подходят для задач классификации. Упростим рассмотренный выше перцептрон с одним скрытым слоем до однослойного перцептрона и попробуем решить задачу разделения объектов на два класса методом коррекции ошибки. Для этого:

1. Будем считать, что A-R связи однослойного перцептрона будут принимать только целые значения.
2. С каждым S элементом входа ассоциируется только один A-элемент.
3. На выходе у нас будет только 1 R-элемент.

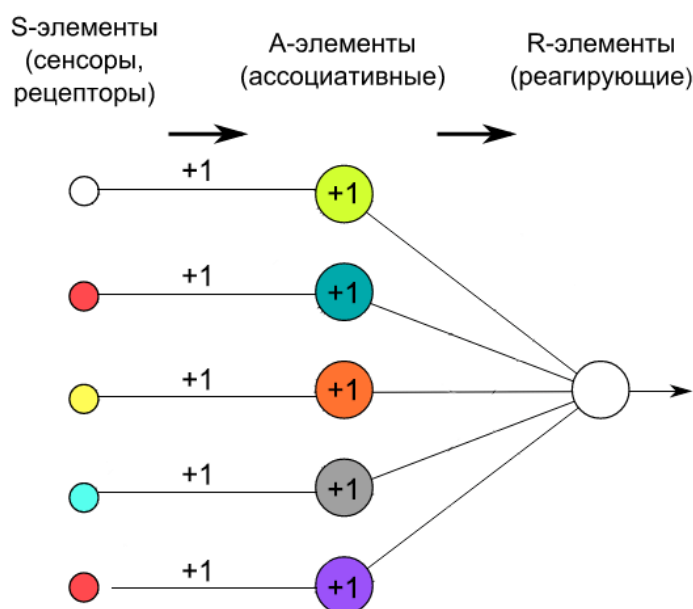


Рисунок 6 - модель однослойного перцептрона

Допустим, мы хотим обучить данный перцептрон разделять два класса объектов так, чтобы при предъявлении объектов первого класса выход перцептрона был положителен (+1), а при предъявлении объектов второго класса — отрицательным (−1). Для этого выполним следующий:

1. Случайным образом инициализируем веса связей S-A и выбираем пороги для ассоциативных элементов. В дальнейшем они останутся неизменными.
2. Начальные коэффициенты A-R весов w_i полагаем равными нулю.
3. Показываем обучающую выборку с указанием класса, к которому принадлежит объект.
 - Показываем объект первого класса. При этом некоторые A-элементы возбуждятся. Коэффициенты w_i , соответствующие этим возбужденным элементам, увеличиваем на 1.
 - Аналогичным образом показываем объект второго класса и уменьшаем веса на 1.
4. Повторяем шаг 3 для всей обучающей выборки. В результате обучения сформируются значения весов связей w_i .

После определенного количества итераций перцептрон будет успешно справляться с поставленной задачей. В данном примере в шаге 3 мы увеличиваем либо уменьшаем веса на 1, что в данном случае является шагом обучения. Значение может быть другим, а правильно подобранный баланс числа обучающих итераций и шага отвечает за скорость обучения нейронной сети.

1.2.3 Обучение

Без конкретных весов нейросеть бесполезна, она будет выдавать неверный результат. Процесс нахождения весов называется обучением. Наша цель состоит в обучении сети таким образом, чтобы достичь баланса между способностью сети давать верный отклик на входные данные, и способностью выдавать правильные результаты в ответ на входные данные схожие, но неидентичные тем, что были использованы при обучении (принцип обобщения).

При инициализации нейронной сети, каждый нейрон получает случайный вес, который в дальнейшем корректируется при обучении.

Обучение происходит на тренировочных сетах. Один сет — это последовательность данных, которыми оперирует нейронная сеть. В случае обучения на изображениях, одно изображение можно назвать одним тренировочным сетом.

Итерация представляет из себя счетчик, который инкрементируется каждый раз, когда через нейронную сеть проходит один тренировочный сет. Другими словами, это общее количество пройденных сетов.

Эпоха представляет из себя величину, установленную в 0 в момент инициализации и растущей каждый раз, когда нейронная сеть проходит весь набор тренировочных сетов. Она имеет потолок, до которого будет возрастать, установленный вручную.

Результат нейросети по началу будет неправильным, относительно ожидаемого. Это называется ошибкой, которая представляет собой процентную величину, отражающую расхождение между ожидаемыми и полученными ответами. Ошибка должна идти на спад после каждой эпохи

					ПГУ 2.090301.10.001 ПЗ	Лист
						24
Изм.	Лист	№ докум.	Подпись	Дата		

обучения. Таким образом разработчик нейронной сети добивается наименьшей ошибки, причем ее величина никогда не будет равна нулю.

Обучение нейронной сети методом обратного распространения ошибки используется в многослойных нейронных сетях, со множеством скрытых слоев, и разбивается на 3 этапа: подачу данных на вход, с последующим распространением в направлении выходов, вычисление и обратное распространение ошибки и корректировку весов. После обучения будет использоваться только подача данных на вход. Для нейронов выходного слоя известны их фактические и желаемые значения выходов и скорректировать их не составляет труда. Однако для нейронов предыдущих слоев настройка не настолько проста. Вес скрытого нейрона должен изменяться прямо пропорционально ошибке тех нейронов, с которыми данный нейрон связан.

Метод обратного распространения ошибки (Backpropagation) позволяет найти место в архитектуре нейросети, которое повлияло на ошибку и скорректировать веса именно там. Он был предложен в 1986 г. Румельхартом, Макклеландом и Вильямсом.

Этот алгоритм определяет два «потока» в сети. Входные сигналы двигаются в прямом направлении, в результате будет получен выходной сигнал, из которого мы получаем значение ошибки. Величина ошибки двигается в обратном направлении, в результате происходит корректировка весов.

					ПГУ 2.090301.10.001 ПЗ	Лист
						25
Изм.	Лист	№ докум.	Подпись	Дата		

1.3 Компьютерное зрение



Рисунок 7 - пример изображения

Мы различаем отдельные предметы на изображении: дом, дерево или гору. Мы понимаем, какой из предметов находится ближе к нам, а какой — дальше. Мы осознаём, что крыша дома — красная, а листья на дереве — зелёные. Наконец, мы можем с уверенностью заявить, что наша картинка — это пейзаж, а не портрет или натюрморт. Все эти выводы мы делаем за считанные секунды.

Человек обладает способностью абстрактного мышления, в то время как машина такой способности лишена, зато способна производить сложные численные расчёты гораздо быстрее любого человека. К тому же компьютер не будет жаловаться на усталость, не требует еды и не зависим от многих человеческих проблем. Зато он также привносит свои.

Для нас посмотреть на картинку и сказать, что на ней изображено, является достаточно тривиальной задачей, однако до 2010 года для компьютерного зрения это было невероятно сложно. Необходимо

геометрическим образом описать объект, его составные части, как они относятся друг к другу. Потом найти похожий объект на изображении, который сравнивается с описанными условиями. При этом точность подобной системы распознавания оставляла желать лучшего, особенно на сильно отличающихся от обучающей выборки входных данных.

Компьютерное зрение – сравнительно молодая область, тесно связанная с обработкой изображений, машинным зрением, визуализацией сцен и распознаванием образов. Британская ассоциация машинного зрения (BMVA) определяет компьютерное зрение как «автоматическое извлечение, анализ и понимание полезной информации из изображения или их последовательности».

Эта область появилась из-за желания адаптировать способности вычислительных машин под возникающие прикладные задачи: распознавание номерных знаков автомобилей, чтение штрихкодов на товарах в супермаркете, анализ записей с камер наблюдения, поиск лиц на фото, создание роботов, умеющих находить (и обходить) препятствия, — всё это задачи, которые требуют от компьютера способности «видеть» и интерпретировать увиденное. Этот процесс обычно включает в себя оценку одного или нескольких изображений, или видео.

Вместо решения одной конкретной проблемы, существует масса методов для решения различных задач компьютерного зрения, где методы часто подбираются исходя из поставленной задачи. Многие из них все ещё находятся в стадии фундаментальных исследований, но всё большее их число находит применение в коммерческих продуктах, где эти методы применяются в составе большей системы, способной решать сложные задачи, зачастую комбинируя различные методики (например, в области медицинских изображений или измерения и контроля качества в процессах изготовления различных задач).

					ПГУ 2.090301.10.001 ПЗ	Лист
						27
Изм.	Лист	№ докум.	Подпись	Дата		

К традиционным задачам компьютерного зрения относят:

- Classification — классификация изображения по типу объекта, которое оно содержит;
- Semantic segmentation — определение всех пикселей объектов определённого класса или фона на изображении. Если несколько объектов одного класса перекрываются, их пиксели никак не отделяются друг от друга;
- Object detection — обнаружение всех объектов указанных классов и определение охватывающей рамки для каждого из них;
- Instance segmentation — определение пикселей, принадлежащих каждому объекту каждого класса по отдельности;

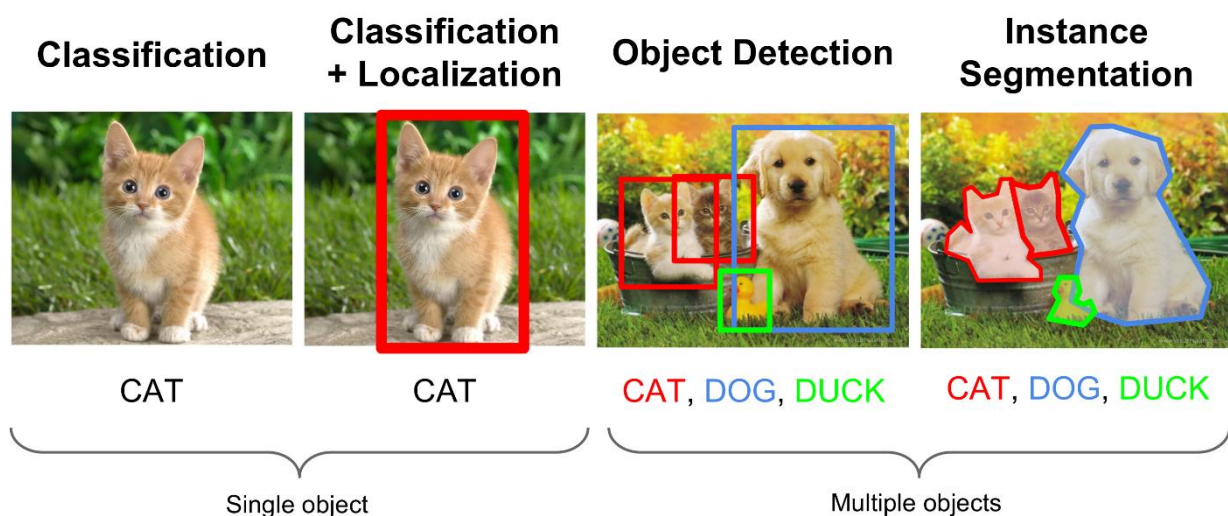


Рисунок 8 – основные задачи компьютерного зрения

1.4 Сверточные нейронные сети

Сверточные нейронные сети (CNN, Convolutional Neural Network) – это особый тип искусственных многослойных нейронных сетей, который на данный момент широко используется для распознавания образов и решения задач компьютерного зрения.

Они были построены на основе биологических процессов, протекающих внутри коры головного мозга, и используют похожие идеи в своей архитектуре:

- Локальные рецептивные поля для каждого слоя
- Иерархическая организация слоев и принцип обобщения образов. Каждый последующий слой выделяет все более высокоуровневые признаки, его рецептивное поле возрастает относительно предыдущего слоя
- общие синаптические коэффициенты (обеспечивают детектирование некоторых черт в любом месте изображения и уменьшают общее число весовых коэффициентов);

CNN являются сетями прямого распространения, то есть в них реализована последовательная связь между слоями, отсутствует обратная связь между нейронами, а выходной результат однозначно определяется входом и не зависит от предыдущего состояния сети.

Сверточные нейронные сети показывают большую точность чем стандартные полносвязные нейронные сети, такие как перцептрон, при решении задач классификации изображения – нахождение объектов на изображении, относящихся к множеству предобученных классов. Они также обходят любые другие классические алгоритмы решения задач

компьютерного зрения как минимум на 15 – 20 % по точности. По сути, CNN это вариация многослойного перцептрона, который был рассмотрен выше.

Дальнейшее развитие сверточных нейронных сетей продвинуло их еще дальше. Их модификации способны также производить локализацию классифицируемого объекта, то есть указывать его местоположение, размер на изображении и даже решать задачу определения границ различных классов для каждого пикселя изображения (instance segmentation).

CNN более устойчивы к изменениям масштаба, поворотам, смене ракурса и прочим искажениям перспективы входного изображения, чем остальные алгоритмы распознавания образов компьютерного зрения. На данный момент CNN и ее модификации являются самыми точными алгоритмами локализации и классификации объектов на сцене. В последнее время развитие архитектуры сверточных нейросетей позволило также увеличить и скорость распознавания образов при сохранении хороших показателей точности.

К минусам сверточных нейронных сетей можно отнести достаточно долгое обучение, но во временные затраты на обучение не влияют на систему в ее рабочем режиме. К тому же, современные архитектуры CNN благодаря развитию сообщества предлагают предобученные веса, которые можно переобучать и дообучать под свои нужды. Нет смысла учить заново нейросеть как минимум на первых слоях, отвечающих за простейшие признаки объекта. Обучение чаще всего производится с помощью метода обратного распространения ошибки, которое является подмножеством метода обучения с учителем, то есть на предварительно размеченной выборке данных.

В данной работе нас интересуют алгоритмы реализации сверточных нейронных сетей, выполняющие задачу Object detection, как часть системы

					ПГУ 2.090301.10.001 ПЗ	Лист
						30
Изм.	Лист	№ докум.	Подпись	Дата		

трекинга, отвечающая за представление и локализацию объектов. Необходимо провести анализ существующих алгоритмов, выбрать из них подходящий под требования в скорости и удовлетворяющий точностью распознавания алгоритм. Также, необходимо разобраться как именно реализованы CNN.

1.4.1 Базовые слои

Любая сверточная нейронная сеть имеет в своем составе три слоя:

- Сверточные слои. По сути, это матричный фильтр, который трансформирует исходное изображение в какое-то другое.
- Слои подвыборки (субдискретизации) необходимы для уменьшения размера изображения;
- Полносвязные слои, такие же как в обычном перцептроне, используемые для классификации полученных в предыдущих слоях данных на выходе модели.

Сверточный слой - то, чему сверточные нейронные сети обязаны своим названием. Производит математическую операцию свертки над пикселями изображения.

Каждый пиксель изображения представлен в памяти в виде числа, обозначающего интенсивность цвета. Например, черно-белое одноканальное изображение цифры “3” в представлении компьютера можно представить как двумерный массив чисел, где каждому пикселю соответствует число от 0 (черный) до 1 (белый). Выглядит это следующим образом:

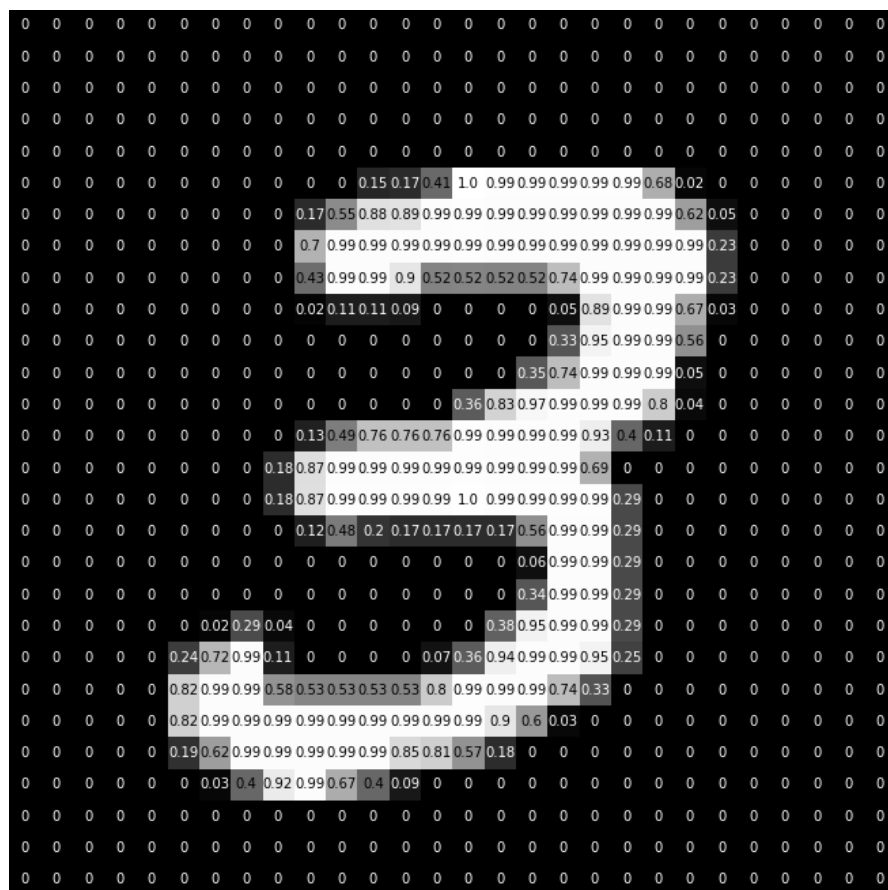


Рисунок 9 – Представление одноканального черно-белого изображения в памяти компьютера, взятое из датасета MNIST

Если в полносвязных многослойных ИНС каждый нейрон подключается к каждому пикселю входного изображения, образуя огромное число связей для обучения, то в сверточных сетях нейрон получает на вход лишь ограниченный участок изображения небольшого размера (1x1, 3x3, 5x5) и производит над ним операцию свертки.

Свертка представляет собой передвижение ядра или фильтра по входному изображению, где ядро имеет такую же размерность, как и подаваемый на вход нейрону участок изображения. При передвижении ядро выполняет поэлементную операцию умножения над той областью, над которой оно скользит в данный момент для каждого пикселя, а затем суммирует все полученные значения в один выходной пиксель.

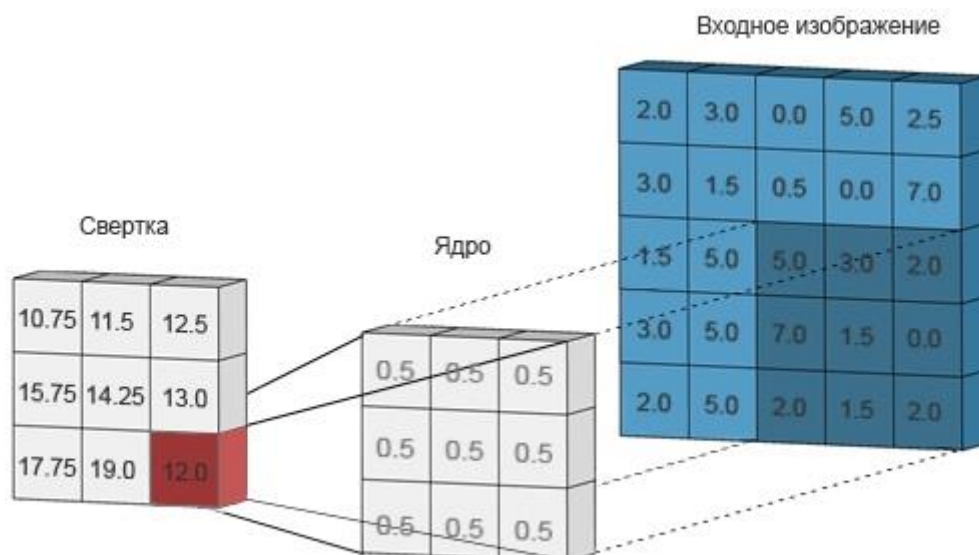


Рисунок 10 – Иллюстрация операции свертки

Легко заметить, что размер ядра определяет количество признаков, которые будут объединены для получения нового признака на выходе. В примере выше мы имеем $5 \times 5 = 25$ признаков на входе и $3 \times 3 = 9$ признаков на выходе. Таким образом, сверточный слой свернул входные данные с 25 до 9 признаков. Для стандартного полносвязного слоя перцептрона в данном случае мы бы имели весовую матрицу $25 \times 9 = 225$ параметров, а каждый выходной признак являлся бы взвешенной суммой всех признаков на входе. Свертка позволяет уменьшать количество параметров, необходимых для обучения и работы модели на порядок, тем самым повысить скорость и уменьшить общую сложность сети.

То число, на которое ядро умножает каждый пиксель участка изображения и является параметром, который изменяется при обучении. Коэффициенты ядра могут быть обучены на нахождение какого-либо специфического признака в этом изображении.

Процесс выделения этого специфического признака называется *признаковым отображением (feature mapping)*. Это название отражает идею,

что каждый сверточный фильтр может быть обучен на отображение различных признаков изображения, для того чтобы нейросеть могла их использовать в дальнейшем для классификации изображения.

Конечно, для дальнейшей классификации нам понадобится отобразить множество признаков, присущих различным классам. Этот процесс называется построением *карт признаков*.

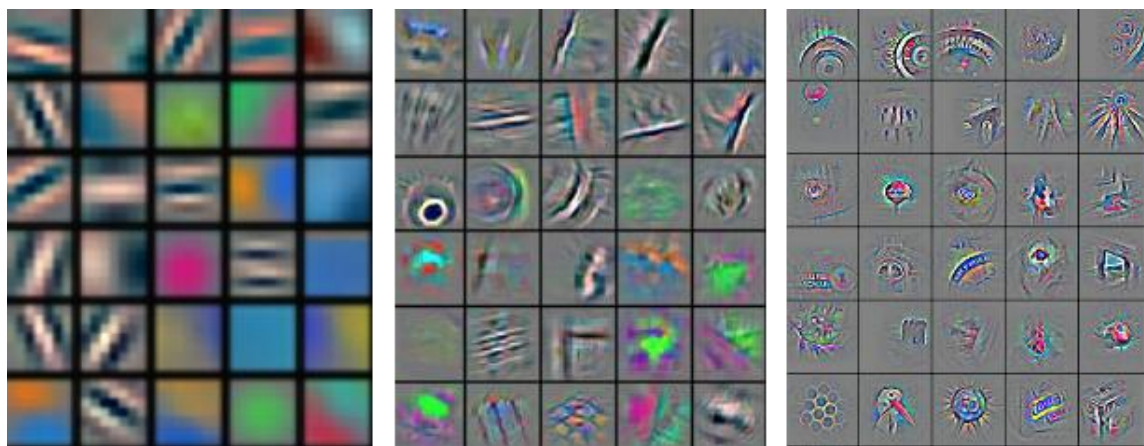


Рисунок 11 – Карты признаков с иерархическим усложнением признаков при продвижении вглубь нейросети

Для отображения множества признаков нам необходимо обучить множество ядер, обученных на нахождение различных признаков. В сверточном слое эти ядра складываются в некий стек.

На выходе мы имеем 3 измерения ширина, высота и карты признаков: Структура будет представлена следующим образом (рисунок 12):

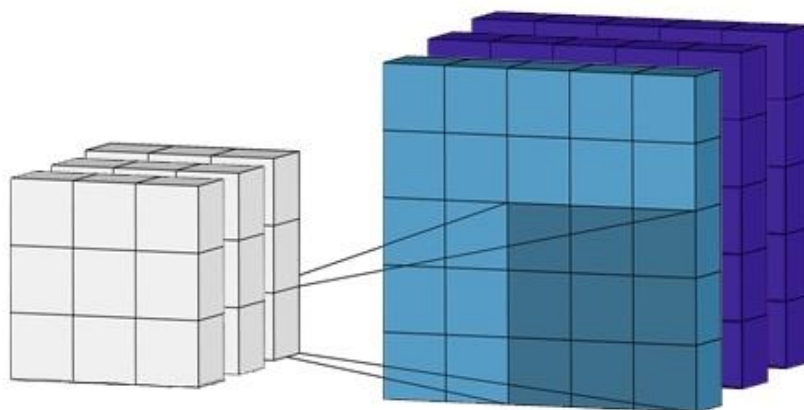


Рисунок 12 – структура выхода сверточного слоя для одноканального изображения

При этом следует учитывать, что при использовании 3-х канальных изображений в формате RGB, которому соответствуют все используемые форматы цветных изображений, добавляется 4-е измерение, определяющее канал. В таком случае, понятия ядро и фильтр уже не взаимозаменяемы. Ядро будет представлять собой 4-х мерный тензор: ширина, высота, канал, карта признаков (фильтр).

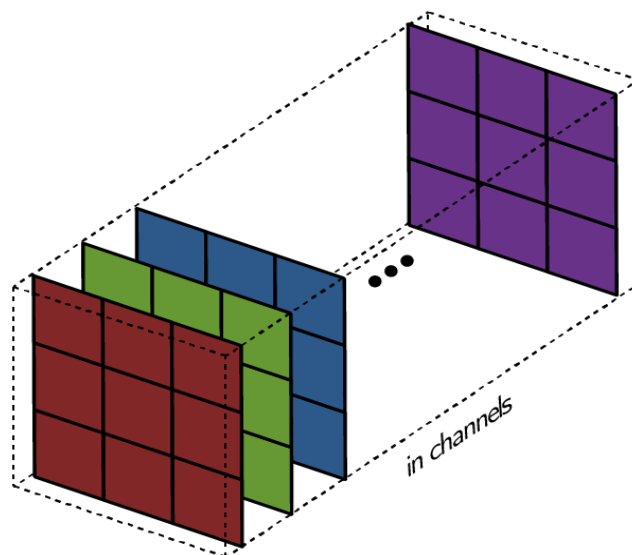


Рисунок 13 – Отдельное ядро для каждого из каналов составляет фильтр

Изм.	Лист	№ докум.	Подпись	Дата

Каждый фильтр в сверточном слое создает только один выходной канал: каждое из ядер фильтра «скользит» по их соответствующим входным каналам, создавая обработанную версию каждого из них. Некоторые ядра могут иметь больший вес, чем другие, для того чтобы уделять больше внимания определенным входным каналам (например, фильтр может задать красному каналу ядра больший вес, чем другим каналам, и, следовательно, больше реагировать на различия в образах из красного канала).

Обработанные каналы суммируются вместе для формирования одного выходного канала: ядра каждого фильтра генерируют одну версию каждого канала, а фильтр в целом создает один общий выходной канал.

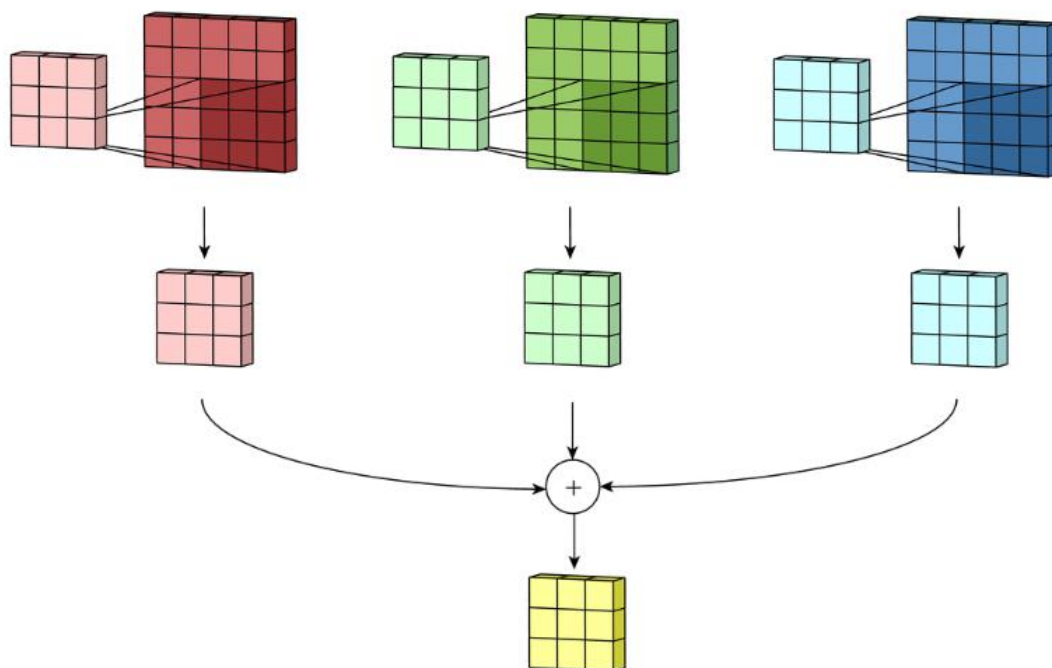


Рисунок 14 – Формирование общего выходного канала

Наконец, каждый выходной файл имеет свое смещение. Смещение добавляется к выходному каналу для создания конечного выходного канала:

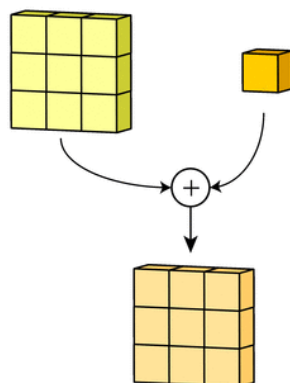


Рисунок 15 – Добавление смещения к выходному каналу

Выходной результат слоя для любого количества фильтров будет одинаковым. Каждый фильтр обрабатывает вход со своим, отличающимся от других набором ядер и скалярным смещением по описанному выше процессу, создавая один выходной канал. Затем они объединяются вместе для получения общего выхода, который представляет из себя 3-х мерный тензор размерностью

$$c \times \left(\frac{w - k_x + 1}{s}\right) \times \left(\frac{h - k_y + 1}{s}\right),$$

где c – количество фильтров в сверточном слое, w – ширина входного изображения, h – высота, k_x и k_y – ширина и высота ядра фильтра, s – шаг, на который скользит ядро при одном проходе.

Следующий шаг работы сверточного слоя CNN – прохождение выхода операции свертки через нелинейную активационную функцию. Если мы не будем применять никакую функцию активации к выходу слоя, тогда вся нейронная сеть может быть описана линейной функцией.

В данный момент чаще всего используют ReLu (рисунок 14) и ее подвиды. Эта функция отличается простотой вычисления градиента ошибки при обучении методом обратного распространения.

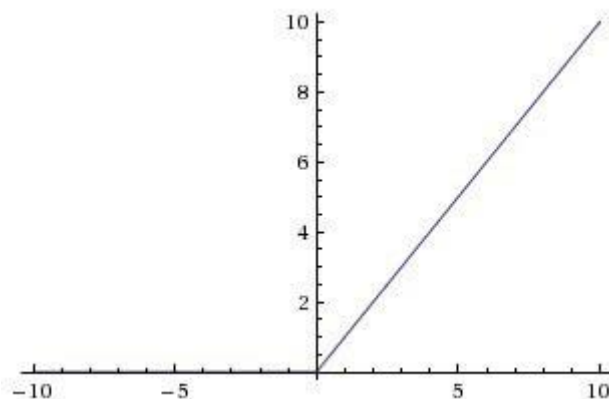


Рисунок 16 – активационная функция ReLu

Функция записывается как $f(x) = \max(0, x)$. Это означает, что ReLu возвращает значение x , если x является положительным и 0 в противном случае.

На первый взгляд кажется, что она очень похожа на линейную и ступенчатую функцию, а также обладает их недостатками – линейностью. Но на самом деле, ReLu нелинейна по своей природе, а комбинация ReLu также нелинейна. Такая функция является хорошим аппроксиматором, так как любая функция может быть аппроксимирована комбинацией ReLu. Это означает, что мы можем паковать слои в стек. Область допустимых значений ReLu — $[0, +\infty)$, то есть активация может “взорваться”. Кроме того, выбор именно этой функции обусловлен некоторыми возможностями оптимизации работы нейросетей, таких как отключение лишних нейронов при обучении, экономия времени и предотвращение переобучения.

Слой подвыборки – также слой объединения (*pooling, subsampling layer*), объединяет и уменьшает размерность данных. Проще говоря, он уменьшает размер изображения. Обычно применяется сразу после сверточного слоя, а вся сверточная нейросеть состоит из чередующихся комбинаций сверточного и подвыборочного слоя.

Применение слоя подвыборки обусловлено следующими факторами:

- Уменьшение количества параметров модели, уменьшение размерности изображения;
- Детектирование признаков на изображении становится более чувствительным к изменению ориентации, размера и угла объекта в пространстве;
- Увеличение рецептивного поля следующего сверточного слоя для нахождения свойств более высокого уровня

Слой объединения всегда следует после слоя свертки. На вход ему поступает не сырое изображение, а карты признаков (рисунок 11) первого сверточного слоя. Последующие пары слоев действуют аналогично.

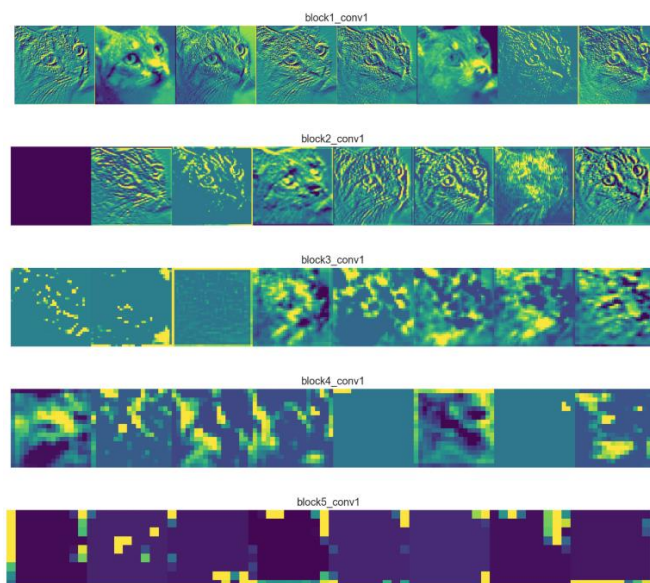


Рисунок 17 – Результаты прохождения 5 пар слоев сверток и объединения

Каждая карта признаков описывает, где на исходном изображении встречаются определенные базовые признаки. Пулинговый слой, как и сверточный использует технику скользящего окна, но обучаемых параметров у этого слоя нет. Вместо этого исходное изображение разбивается на блоки фиксированного размера (ядро), для каждого блока вычисляется некоторая функция. Чаще всего используется функция максимума (*max pooling*) или (взвешенного) среднего (*average pooling*). Важно отметить, что шаг в данном случае обычно больше единицы, что позволяет кратно уменьшить размер карт признаков на выходе.

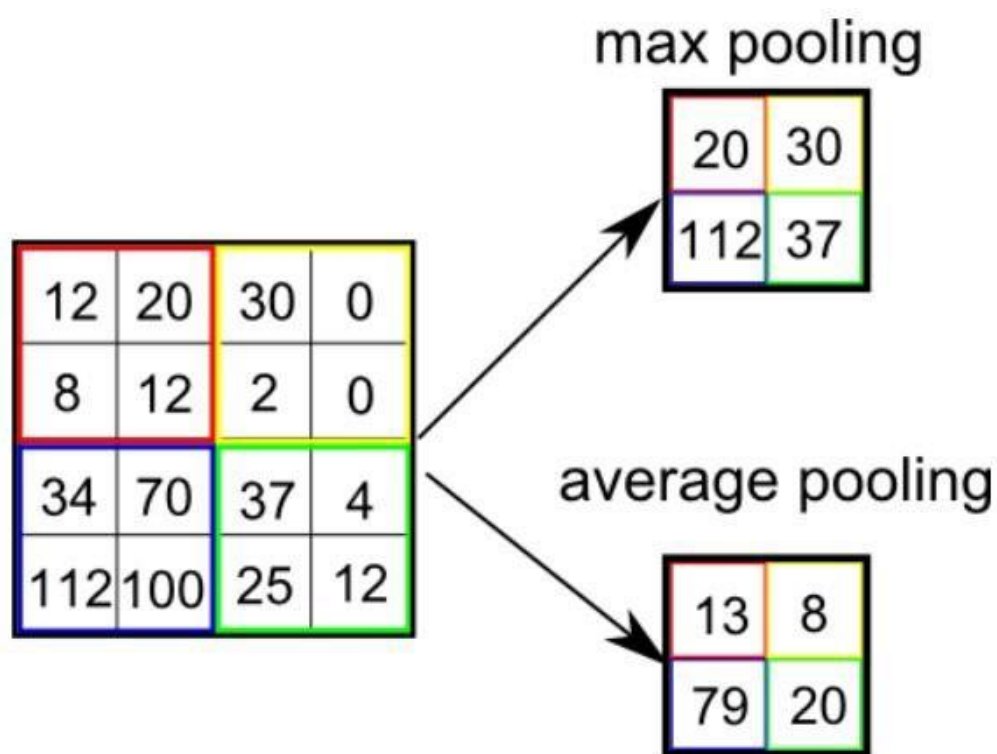


Рисунок 18 – Выход карты признаков размером 4x4, ядро размером 2x2 с шагом равным 2 позволяет уменьшить размер карты на выходе до 2x2 пикселей

Необходимо отметить, что в случае несовпадения размерности входного изображения применяются отступы (*padding*), дополняющие отсутствующие столбец или строку нулями. Например, у нас есть входное изображение размером 5x5, ядро размером 3x3 с шагом равным 2. В таком

случае необходимо преобразовать входное изображение до размера 6×6 , кратному размеру ядра, путем добавления дополнительной строки и столбца, заполненного нулями. При таком подходе не возникнет ошибки размерности, а *max pooling* просто проигнорирует значения, равные 0.

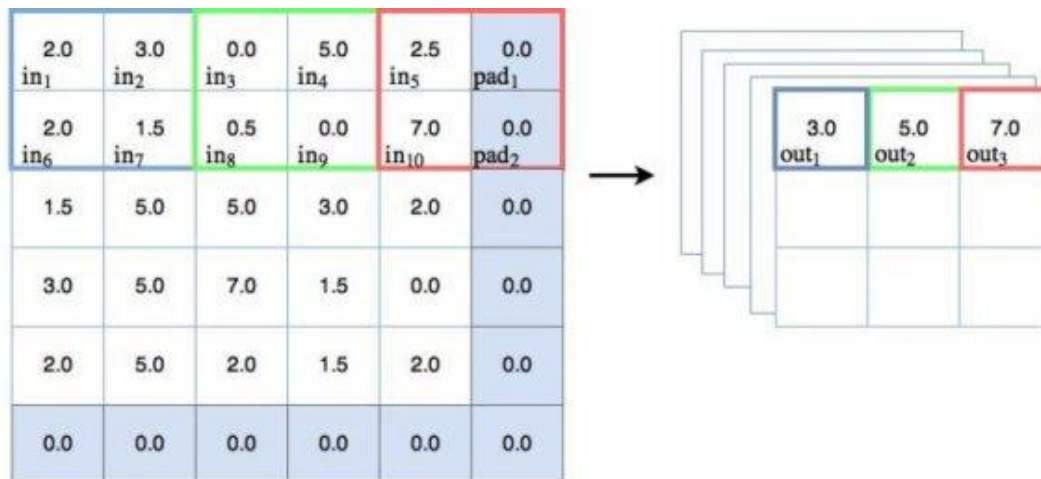


Рисунок 19 – Использование отступов (голубые ячейки) при операции пуллинга.

Полносвязный слой – находится в конце сверточной нейронной сети и предназначен для классификации выделенных предыдущими слоями признаков. Представляет собой обычный перцептрон. Когда данные попадают в полносвязный слой, они представляют собой вектор карт признаков размером 1×1 . Значения каждого элемента вектора обозначают, насколько изображение активировало эту карту. Этот слой выводит N -пространственный вектор, где N – число классов, которые способна распознать сверточная нейросеть. Например, если в задаче требуется распознавать рукописные цифры от 0 до 9, значение N будет равным 10.

Число N_i в выходном векторе обозначает уверенность нейросети в том, что на входной картинке изображен класс с номером i . В зависимости от датасета, на котором производилось обучение ИНС, и классов в этом датасете, можно сопоставить номер i с названием класса в списке классов датасета под этим номером и понять, что нашла нейросеть на изображении.

1.4.2 Архитектуры сверточных нейронных сетей

Развитие open-source сообщества и постоянно растущие требования к точности распознавания объектов на изображениях привели к эволюции архитектур CNN для решения задач классификации изображений. Зачастую такие сети находятся в открытом доступе, включая предварительно обученные веса (коэффициенты нейронов).

Крупные компании, такие как Google, Microsoft и Facebook, применяют методы машинного обучения для перевода веб-страниц на другие языки, обработки огромных массивов изображений, поиска по картинке, создания персонализированной рекламы, и многих других задач. У них есть необходимые наборы накопленных за долгие годы данных, вычислительные мощности и команды исследователей, а также заинтересованность в продвижении сферы deep learning и стимулировании прогресса. Благодаря этому в открытом доступе существуют созданные ими архитектуры искусственных нейронных сетей для различных задач.

Кроме того, в последние несколько лет появились удобные инструменты разработки, фреймворки глубокого обучения, позволяющие свободно создавать собственные архитектуры различного рода ИНС, сохранять их, распространять, измерять эффективность, передавать часть вычислений на GPU и многое другое.

Прежде чем перейти к рассмотрению алгоритмов, реализующих МОТ, необходимо рассмотреть существующие архитектуры CNN для классификации изображений. Для классификации необходимо, чтобы на изображении был всего один объект. Но если на картинке необходимо найти несколько объектов, нужно решить две задачи:

					ПГУ 2.090301.10.001 ПЗ	Лист
						42
Изм.	Лист	№ докум.	Подпись	Дата		

1. Сегментация - выделение участков изображения, которые относятся к разным объектам.
2. Классификация, то есть определение типа объекта, для каждого выделенного сегмента отдельно.

Поэтому в состав любой нейросети, решающей задачу MOT, входят сверточные нейронные сети для классификации изображений. Рассмотрим их подробнее:

LeNet5 - Одна из первых свёрточных нейросетей для распознавания рукописных цифр из датасета MNIST. Она была разработана в 1994-м Яном Лекуном. Архитектура LeNet5 стала фундаментальной для глубокого обучения, особенно с точки зрения распределения свойств изображения по всей картинке. Свёртки с обучаемыми параметрами позволяли с помощью нескольких параметров эффективно извлекать одинаковые свойства из разных мест. В те годы ещё не было видеокарт, способных ускорить процесс обучения, центральные процессоры также были довольно медленными, что не способствовало росту популярности машинного обучения.

Ключевым преимуществом архитектуры оказалась возможность сохранять параметры и результаты вычислений, в отличие от использования каждого пикселя в качестве отдельных входных данных для полносвязного перцептрона. Особенности LeNet5:

- Используется последовательность из трёх слоёв: слои свёртки (*convolution*), слои объединения (*pooling*) и слои нелинейности (*non-linearity*). С момента публикации работы Лекуна это, пожалуй, одна из главных особенностей всех последующих CNN;
- Использует свёртку для извлечения пространственных свойств изображения;
- Подвыборка с использованием пространственного усреднения карт;

- Нелинейность обеспечивается активационными функциями в виде гиперболического тангенса или сигмоид;
- Финальный классификатор в виде многослойной нейросети;

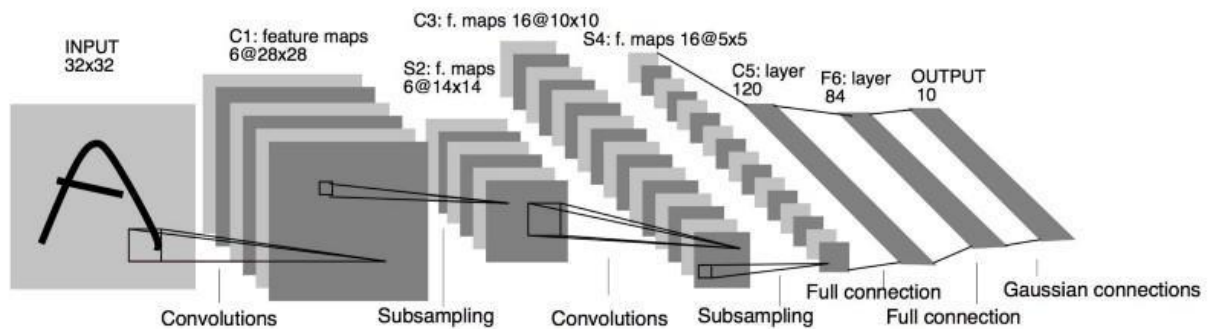


Рисунок 20 – Архитектура LeNet5

Эта нейросеть легла в основу многих последующих архитектур и вдохновила множество исследователей.

AlexNet – Углубленная и сильно модифицированная версия LeNet5, представленная в 2012 году Алексеем Крижевским. Прославилась тем, что с большим отрывом выиграла соревнование по классификации изображений ImageNet (точность 84.6%) и стимулировала область к развитию более точных и быстрых архитектур CNN, а сами сверточные нейросети стали повсеместно использоваться для решения практических задач. Особенности:

- Использование активационной функции линейной ректификации (*ReLU*) для обеспечения нелинейности.
- Использование методики отбрасывания для выборочного игнорирования отдельных нейронов в ходе обучения, что позволяет избежать переобучения модели.
- Перекрытие *max-pooling* в слоях пуллинга вместо *average-pooling*, что позволяет избежать эффектов усреднения.
- Использование NVIDIA GTX 580 для ускорения обучения.

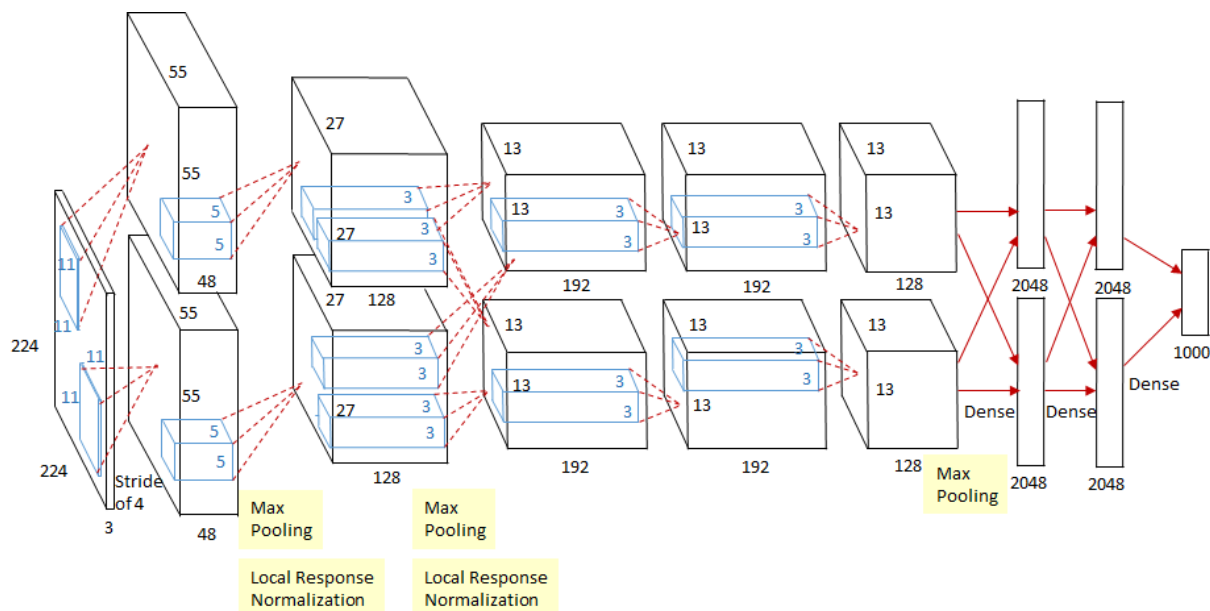


Рисунок 21 – Архитектура AlexNet

К тому времени мощности GPU сильно возросли, что позволило усложнить архитектуру и ускорить обучение как минимум в 10 раз, использовать картинки и сами датасеты большего размера.

VGG – Разработана в стенах университета Оксфорда командой исследователей. Включает в себя VGG-11, VGG-13, VGG-16 и VGG-19, где цифры после названия обозначают количество слоев и косвенно указывают на размер нейросети. Победителем соревнования ImageNet 2013-ого года была VGG-16, набравшая точность 92.7%.

Одной из отличительных особенностей является использование ядер свертки небольшого размера (3x3, в отличие от больших ядер размера 7x7 или 11x11, применяемых AlexNet). Было открыто, что несколько таких сверток, объединенных в последовательность, могут заменить сверточный слой с более крупными ядрами. Количество параметров для обучения при этом заметно сокращается. Позднее, эти идеи были использованы в архитектурах Inception и ResNet.

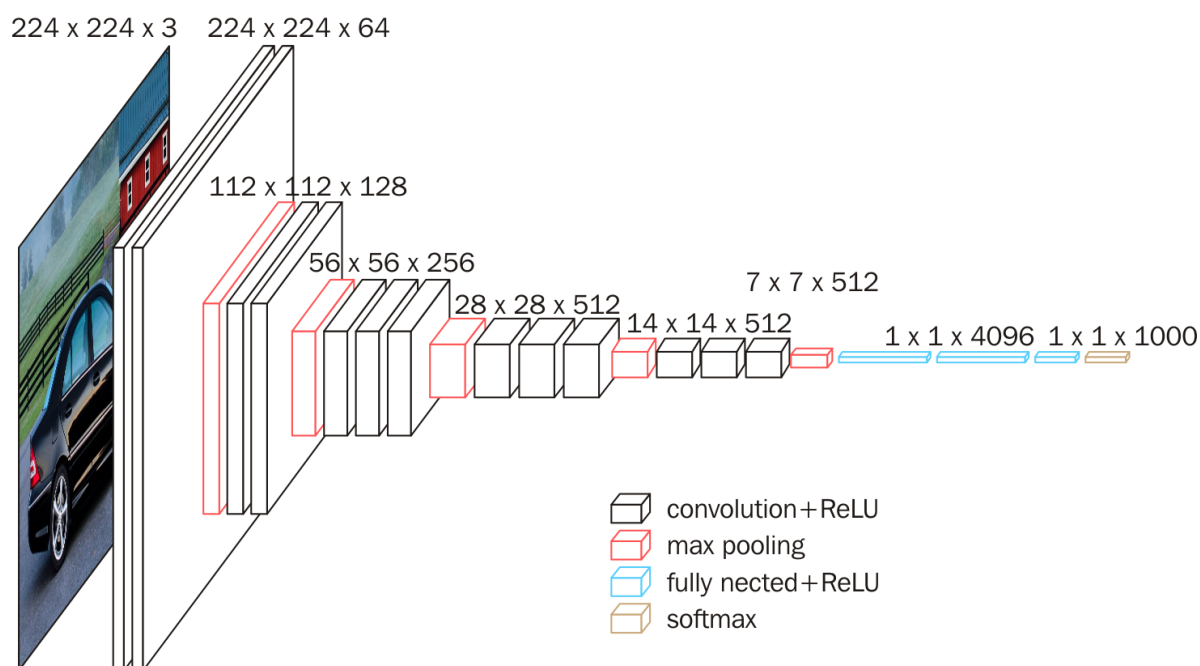


Рисунок 22 – Архитектура VGG-16

Однако учить такие сети из-за множества слоев было сложно. VGG в слоях использует большое количество параметров, поэтому обучение сети требует больших вычислительных затрат. Снизить нагрузку можно, уменьшив количество свойств, как это сделано было в дальнейшем в слоях архитектуры Inception.

GoogLeNet – Первая версия архитектуры, основанной на *inception* слоях, благодаря которым получило название семейство Inception сетей. GoogLeNet является первой в этом семействе. Победитель ILSVRC-2014 с результатом точности 93.3%, разработана в Google Кристианом Жегеди.

К 2014 скептики признали пользу глубокого обучения и нейросетей в практических задачах, многие крупные компании заинтересовались развертыванием глубоких ИНС на собственных мощностях. Кристиан искал способ уменьшить вычислительную нагрузку и количество обучаемых параметров нейросети, при этом не потеряв в эффективности и точности.

Результатом его работы стало применение *inception* слоя в качестве строительных блоков нейросети. Особенности:

- *Network-in-network (NiN)* слой с размером ядра 1x1, сворачивающий данные перед подачей в остальные слои в блоке
- Комбинация параллельно выполняющихся сверточных фильтров с ядрами различной размерности.

NiN слой уменьшает количество параметров перед подачей в параллельные блоки, что позволяет повысить скорость и при этом сохранить высокий уровень точности. После свертки в таком блоке применяется внутренний маленький полносвязный слой, чтобы лучше скомбинировать полученные признаки перед отправкой в следующий слой.

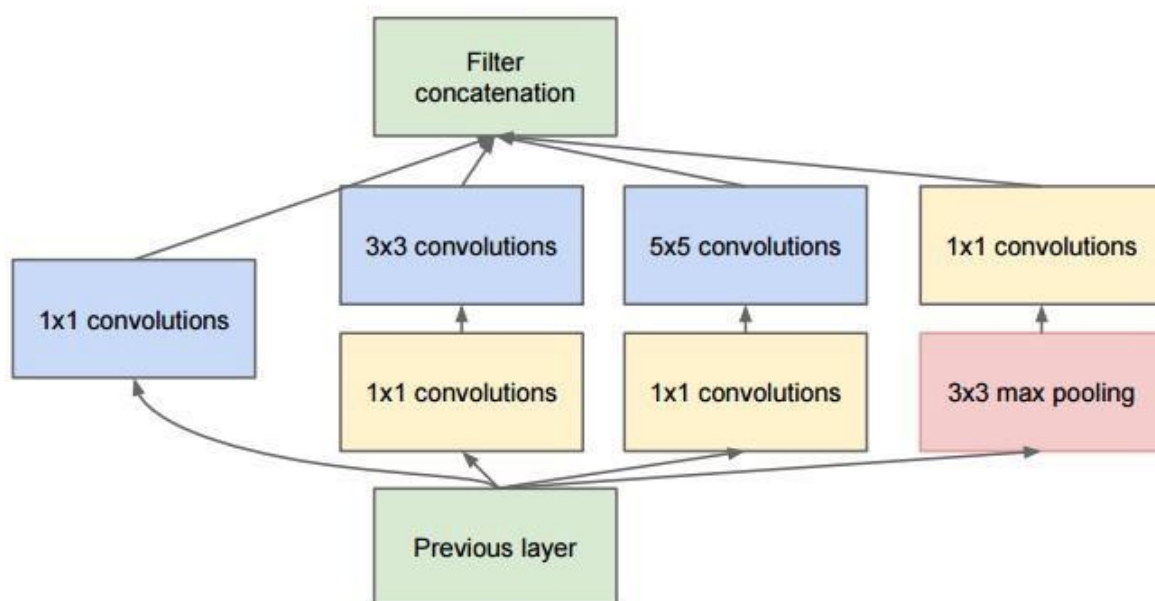


Рисунок 23 – Inception-модуль, из нескольких таких состоит GoogLeNet

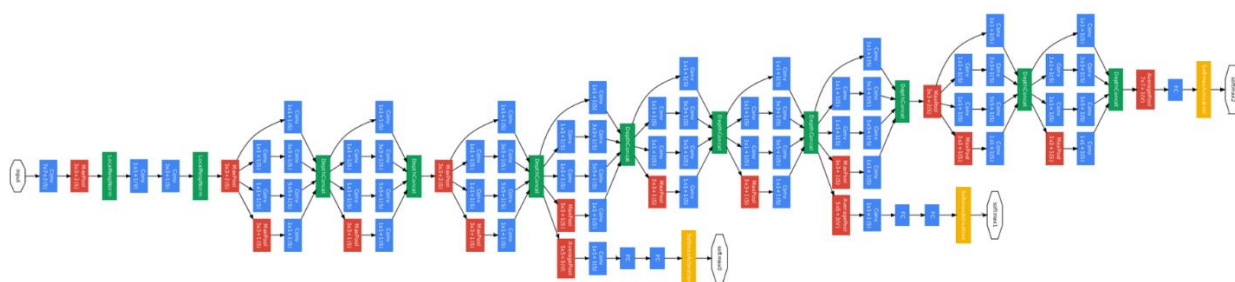


Рисунок 24 – Архитектура GoogLeNet

ResNet - это сокращенное название для *Residual Network* (остаточная сеть). Данная сеть обошла усовершенствованную GoogLeNet, Inception v2 в соревновании ImageNet 2015. ResNet создана командой китайского подразделения Microsoft Research. Ее архитектура при большей точности не так сложна, как в Inception, с ее составными блоками параллельных сверток, хотя и используется идея со сверткой 1x1.

Исследователи давно заметили, если ставить больше и больше сверточных слоев, точность такой модели будет расти до некоторого предела, после чего резко оно резко падает. Идея состоит в том, чтобы использовать соединения быстрого доступа через каждые несколько сверток напрямую к следующим слоям (рисунок 25).

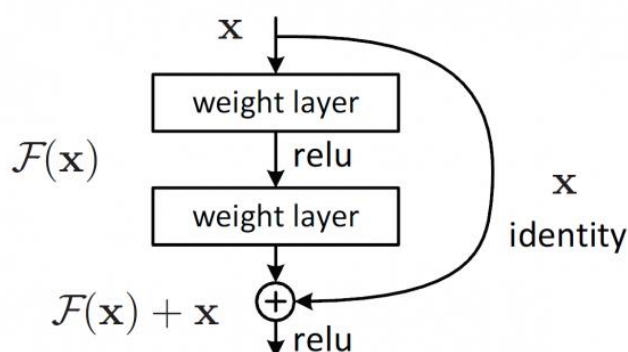


Рисунок 25 – соединение быстрого доступа

Здесь $F(x)$ – остаточная функция, $F(x) + x$ – целевая функция.

Модель, с которой выиграла ImageNet 2015 содержит меньше параметров, чем 19-слойный VGG, при глубине 152 слоя, чем наглядно демонстрирует развитие эффективности и эволюцию CNN.

Кроме того, в 2015 году впервые искусственная нейросеть, а именно ResNet, по точности распознавания объектов превысила возможности человека.

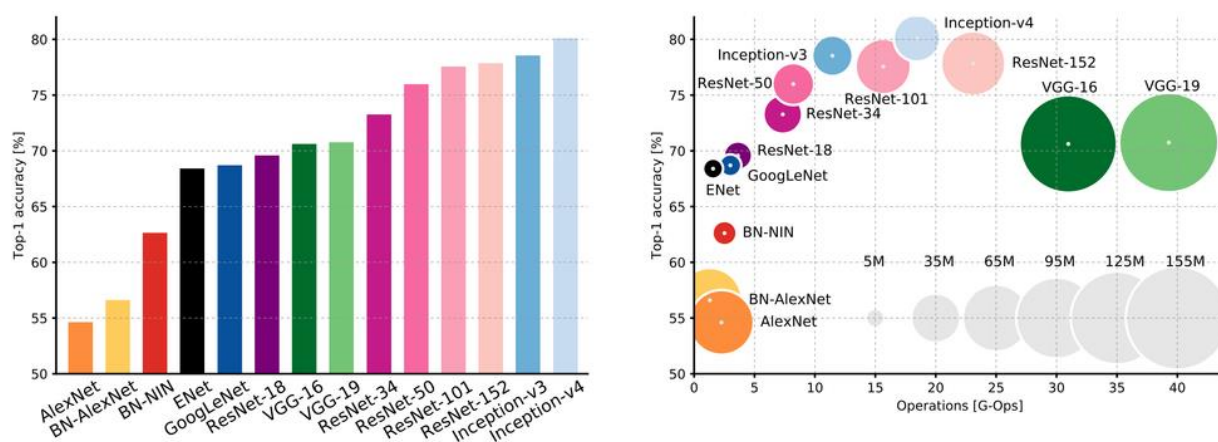


Рисунок 26 – Графики точности и количества операций для рассмотренных архитектур CNN

1.4.3 Алгоритмы детектирования объектов

Так как в данной работе необходимо использовать не просто классификацию одного объекта на изображении, а классификацию и локализацию множества объектов (детектирование, MOT) для их дальнейшего отслеживания, необходимо рассмотреть существующие архитектуры сверточных нейронных сетей, предназначенные для решения данной задачи. Под задачей детектирования обычно понимается определение ограничивающих областей и класса для каждого объекта на анализируемых изображениях.

Алгоритмы применения CNN к задаче object detection делятся на два подхода:

- Алгоритмы на основе предположений регионов
- Алгоритмы, обрабатывающие изображение целиком

Алгоритмы R-CNN, Fast R-CNN, Faster R-CNN и большинство других основаны на подходе *region proposals*, т.е. предположении регионов. Они состоят из двух частей: первая часть генерирует наборы участков, на которых вероятно изображен объект, отличный от фона. Вторая часть обрабатывает эти предположения и классифицирует объект на каждом участке. Исторически они появились раньше чем второй тип алгоритмов, являются более точными но также значительно более медленными.

R-CNN (*Regions with CNN features*) - один из первых подходов, в которых использовались региональные предложения. Здесь впервые использовали алгоритм, называемый выборочным поиском (*selective search*), для обнаружения 2000 региональных предложений на изображении, которые представляют из себя зоны, где вероятно может быть изображен какой-либо объект. Данный подход конечно не так затратен по вычислительным

операциям, как метод скользящего окна, но все равно такая сеть была довольно медленной (18-20 секунд на изображение на GPU). Точность при этом была достаточно высока, в качестве сверточной нейросети использовался AlexNet.

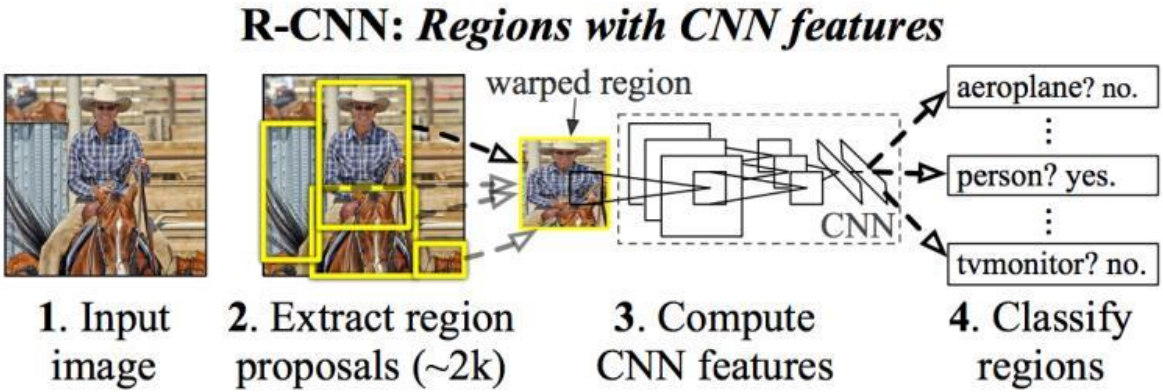


Рисунок 27 – Алгоритм работы R-CNN

Каждый из 2000 регионов, найденных алгоритмом селективного поиска, масштабировался до 224x224 и подавался на вход нейросети. При неквадратном соотношении сторон недостающие пиксели оставались пустыми, а изображение преобразовывалось с помощью аффинных трансформаций.



Рисунок 28 – Пример региона и его модификация перед подачей в нейросеть

Fast R-CNN – Дальнейшее развитие R-CNN. Этот алгоритм выполняется быстрее за счет нескольких модификаций предыдущей архитектуры:

- Теперь через CNN пропускается все изображение целиком, а предложенные алгоритмом селективного поиска регионы накладываются на общую карту признаков.
- Вся процедура тренировки модели совмещенная, без необходимости отдельно тренировать различные части.

Faster R-CNN считается одним из самых точных нейросетевых алгоритмов [5] обнаружения объектов на изображении, но для достижения высокой точности приходится жертвовать скоростью обработки, что не подходит для использования в режиме реального времени.

После улучшений в Fast R-CNN была развита идея вычислять регионы не по входному изображению, а по карте признаков, полученных из выходного слоя сверточной нейронной сети. Для этого был создан отдельный модуль RPN (*Region Proposal Network*).

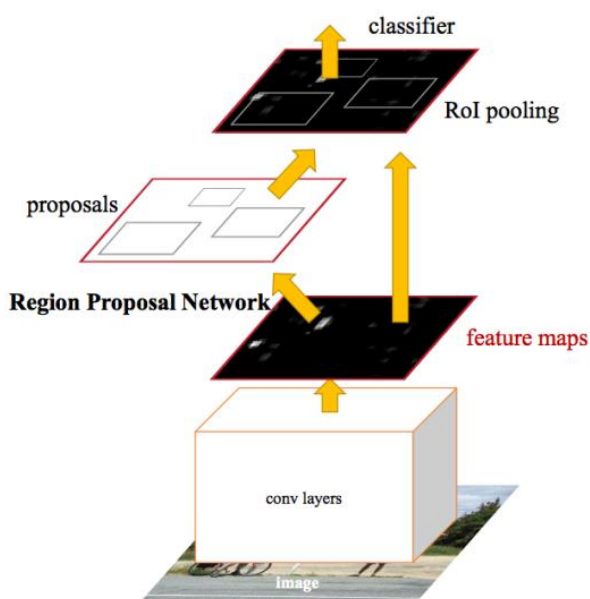


Рисунок 29 – Алгоритм работы Faster R-CNN

Фактически RPN представляет из себя маленькую сверточную нейросеть, которая скользит по извлеченным CNN признакам ядром размером 3×3 . Ее результат направляется в два параллельных полносвязных слоя, которые выдают предположения регионов, в которых может находиться объект. Те регионы, которые имеют высокую вероятность нахождения объекта, передаются далее для классификации объектов и уточнения охватывающей рамки.

Кроме общей сложности, невысокой скорости работы и большого количества параметров в такой сети, она фактически представляет из себя две отдельных ИНС, которые необходимо также обучать независимо друг от друга.

Алгоритмы **YOLO** (*You Only Look Once*) и **SSD** (*single-shot-detector*) работают по принципу схожему принципу - за раз обрабатывается все изображение целиком [6,7,8]. Из-за этого их скорость работы в разы превосходит алгоритмы на основе предположений регионов, однако они немного проигрывают в точности. Впрочем, в YOLO 3 версии точность фактически приближена к Faster R-CNN.

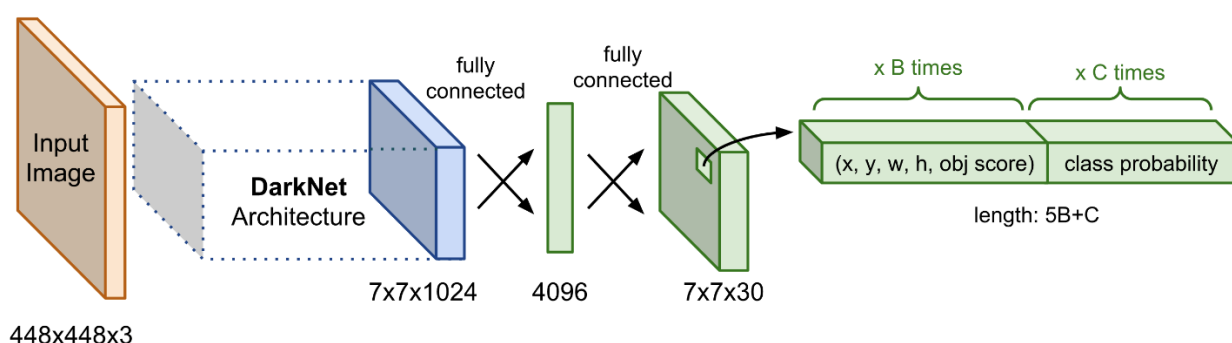


Рисунок 30 – Архитектура YOLO

Yolo построена на базе модифицированного варианта сети GoogLeNet (на базе фреймворка Darknet и одноименной CNN для классификации изображений), из архитектуры которого удалены два полносвязных слоя на выходе, что позволяет не использовать метод dropout на этапе обучения для предотвращения переобучения.

В ходе работы алгоритма на изображение накладывается сетка, разделяя его на ячейки. Каждая ячейка пытается предсказать координаты зоны обнаружения с оценкой уверенности для этих полей и вероятностью классов. Затем оценка уверенности для каждой зоны обнаружения умножается на вероятность класса, чтобы получить окончательную оценку.

Чтобы детектировать объекты разного масштаба карты признаков перед последним слоем подвыборки объединяются с выходными картами признаков выходного слоя. Для определения ограничивающих областей объектов используются смещения относительно якорных значений (*anchor boxes*), которые вычисляются на из обучающей выборки с использованием метода K средних.

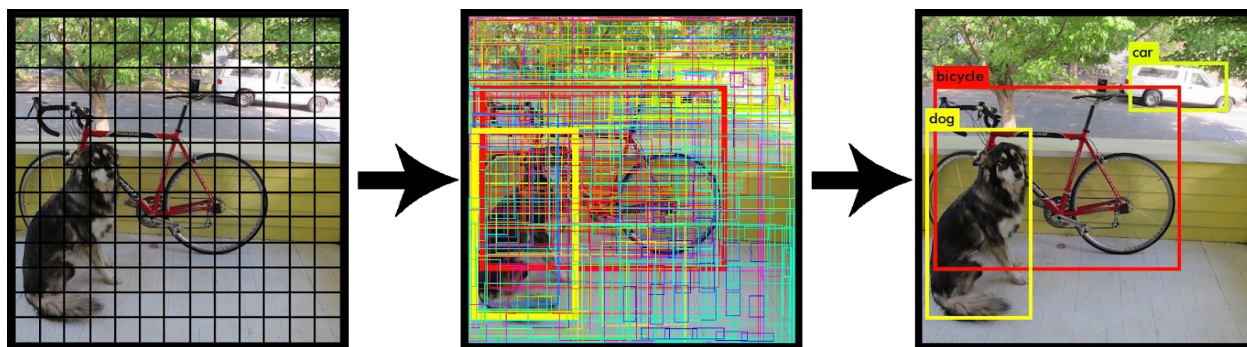


Рисунок 31 – Алгоритм работы YOLO

Скорость работы YOLO доходит до 45 кадров при использовании GPU, чего достаточно для трекинга объектов в реальном времени, большинство камер снимает видео с fps от 24 до 30.

1.5 Фильтрация

В процессе автоматизации технологических процессов для управления механизмами и агрегатами приходится сталкиваться с измерениями различных физических величин. Это может быть давление и расход жидкости или газа, частота вращения, температура и многое другое. Измерение физических величин осуществляется с помощью аналоговых датчиков. Аналоговый сигнал - сигнал данных, у которого каждый из представляющих параметров описывается функцией времени и непрерывным множеством возможных значений. Из непрерывности пространства значений следует, что любая помеха, внесенная в сигнал, неотличима от полезного сигнала. Поэтому на аналоговый вход управляющего устройства будет поступать неверное значение требуемой физической величины. Следовательно, необходимо производить фильтрацию сигнала, поступающего с датчика.

Для устранения шумов и другой лишней информации и существуют специальные алгоритмы цифровой обработки данных, называемые фильтрами. Нелишним будет отметить, что задача фильтрации — это не задача сглаживания показаний, стремление получить наиболее близкое значение к реальному значению.

Кроме того, фильтры можно использовать для прогнозирования будущего значения сигнала, получаемого с датчиков. В нашем случае вместо показания датчиков используются координаты x и y центра объекта на изображении, получаемые через постоянные промежутки времени, так что к задаче трекинга объектов применимы цифровые дискретные фильтры для прогнозирования местоположения объекта через некоторый момент времени.

1.5.1 Фильтр Калмана

Фильтр Калмана используется в сложных динамических системах, где известны управляющие воздействия для формирования оптимальной оценки состояния, примерами которых являются непрерывные производственные процессы, такие как полет самолета, корабля и космического аппарата. Для того чтобы управлять системой, необходимо в первую очередь определить ее текущее состояние. Для этих целей не всегда имеется возможность измерить каждый интересующий нас параметр управления достаточно точно, тогда как фильтр Калмана обеспечивает получение отсутствующей информации при неизвестных параметрах косвенных изменений.

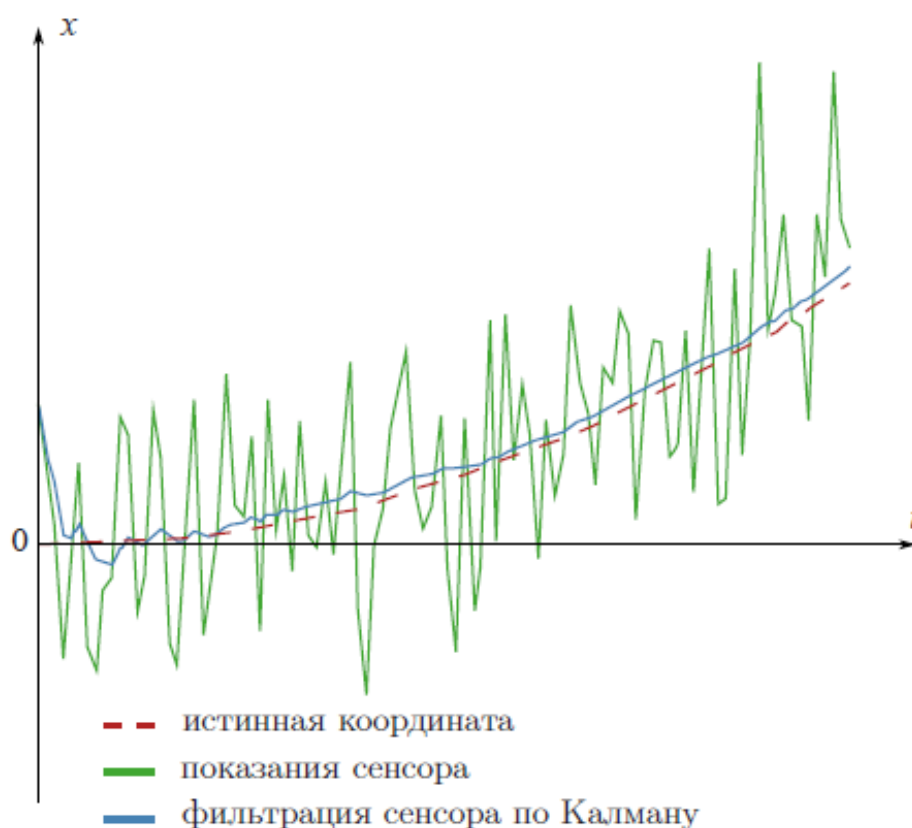


Рисунок 32 – Пример работы фильтра при получении зашумленных данных с сенсора

Изм.	Лист	№ докум.	Подпись	Дата

Фильтр Калмана также используется для предсказания следующего состояния динамических систем, что интересно применительно к задаче трекинга: необходимо определить, где будет находиться на следующем кадре найденный объект. Алгоритм состоит из двух повторяющихся фаз: предсказание и корректировка.

На первом рассчитывается предсказание состояния в следующий момент времени (с учетом неточности их измерения). На втором, новая информация с датчика корректирует предсказанное значение (также с учетом неточности и зашумленности этой информации).

Например, стоит задача получить координату объекта на картинке в данный момент времени k , обозначим ее x_k . Допустим также, что мы знаем закон, по которому можно вычислить позицию объекта в момент времени

$$x_{k+1} = x_k + l_k + e_k,$$

где l_k – закон, по которому изменяется система, e_k – ошибка модели, то есть случайная величина. Значение координаты объекта, которое на самом деле возвращает нейросеть в данный момент времени k мы обозначим как

$$z_k = x_k + n_k,$$

где n_k – ошибка, которая также является случайной величиной.

Таким образом, задача состоит в том, что исходя из неточных показаний z_k , используя ошибку n_k необходимо найти максимально близкое приближение к реальному местоположению координаты объекта x_k , обозначим его как $x_{k, opt}$.

Отметим, что нам известно:

- l_k – известная величина, контролирующая изменение системы.

- Ошибка модели e_k и ошибка полученных данных n_k – случайные величины. Их законы распределения не зависят от времени k .
- Средние значения обеих ошибок равны нулю
- Закон распределения случайных величин ошибок может быть неизвестен, но известны их дисперсии σ_e^2 и σ_n^2 . Дисперсии не зависят от времени k .
- Предполагается, что все случайные ошибки независимы друг от друга: какая ошибка будет в момент времени k совершенно не зависит от ошибки в другой момент времени k' .

Представим, что мы уже нашли значение $x_{k,opt}$. Мы можем предположить, что на шаге $k+1$ значение z_{k+1} будет близко к $x_{k,opt} + l_k$.

Идея фильтра Калмана состоит в том, что чтобы получить наилучшее приближение к истинной координате x_{k+1} , мы должны выбрать золотую середину между показаниями нейросети z_{k+1} и $x_{k,opt} + l_k$ — нашим предсказанием того, что мы ожидали от него увидеть. Показанию сенсора мы дадим вес K , а на предсказанное значение останется вес $(1 - K)$:

$$x_{k+1, opt} = K \times z_{k+1} + (1 - K) \times (x_{k,opt} + l_k)$$

Коэффициент K принято называть коэффициентом Калмана. Он зависит от шага итерации. Необходимо выбрать K таким, чтобы получившееся оптимальное значение координаты $x_{k+1,opt}$ было наиболее близко к истинной координате x_{k+1} . В общем случае, чтобы найти точное значение коэффициента K , необходимо минимизировать ошибку:

$$\varepsilon_{k+1} = x_{k+1} - x_{k+1, opt}$$

Если проследить, как с шагом итерации изменяется коэффициент Калмана K , то можно показать, что он всегда стабилизируется к определенному значению K_{stab} .

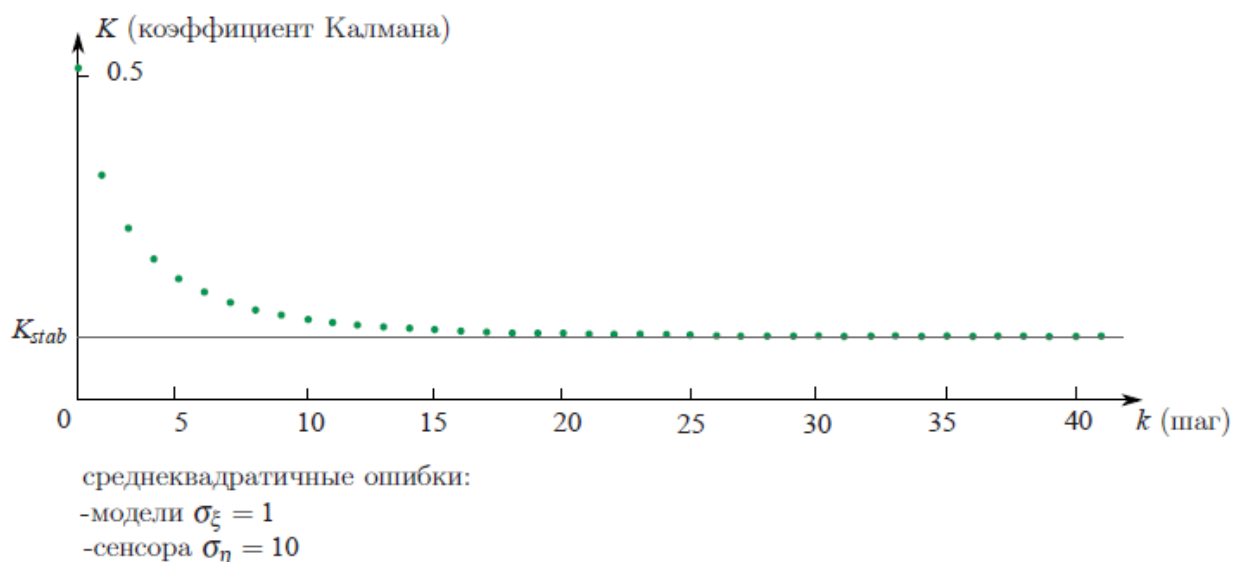


Рисунок 33 – График коэффициента K в зависимости от итераций

В нашем случае мы не знаем ничего о законе изменения системы l_k , но зато можно подобрать коэффициент K вручную в виде константы K_{stab} , что позволит подставить его в итерационную формулу:

$$x_{k+1, opt} = K_{stab} \times z_{k+1} + (1 - K_{stab}) \times x_{k, opt}$$

Таким образом мы можем предсказать координату объекта в будущем $x_{k+1, opt}$, а формулы можно модифицировать для применения в случае получения двух координат объекта x и y на двумерном изображении.

1.5.2 Алгоритм SORT

Алгоритм SORT – Simple Online and Realtime Tracking, был представлен в 2016 году группой исследователей из университета Сиднея [9] для решения проблемы MOT. Каждый найденный сверточной нейросетью объект представлен в виде обрамляющей рамки (bounding box). Проблема состоит в необходимости ассоциирования этих объектов в последовательности кадров, учитывая найденные объекты на каждом кадре, возможность пропажи объекта на некоторое время, его передвижения в пространстве, перекрытия другим объектом, а также передвижение камеры по сцене.

В отличие от других подходов к трекингу, данный алгоритм использует только предыдущий и текущий кадр. Используя комбинацию существующих ранее методов, таких как фильтр Калмана и венгерский алгоритм, разработчикам удалось достичь крайне высокой скорости и точности работы.

Ранее данную задачу решали с помощью алгоритма MHT - Multiple Hypothesis Tracking, либо Joint Probabilistic Data Association (JPDA) фильтрами, которые отличались большей вычислительной сложностью и меньшей точностью. Это накладывало также ограничения на использование в приложениях детектирования объектов в реальном времени.

Состояние каждой цели моделируется как $x = [u, v, s, r, u_1, v_1, s_1]^T$, где u и v – центр модели по горизонтали и вертикали, s – масштаб, r – представляет масштаб (область) и соотношение сторон ограничивающего прямоугольника цели соответственно.

Когда трек (обнаружение) связан с целью, обнаруженная ограничивающая рамка используется для обновления целевого состояния, в котором компоненты скорости рассчитываются оптимально с помощью

					ПГУ 2.090301.10.001 ПЗ	Лист
						60
Изм.	Лист	№ докум.	Подпись	Дата		

Калмана. Если трек не связан с целью, его следующее состояние просто прогнозируется без коррекции с помощью линейной модели скорости.

При назначении треков для существующих целей, геометрия ограничительной рамки каждой цели оценивается путем прогнозирования ее нового места в текущем кадре. Матрица затрат назначения затем вычисляется как расстояние пересечения над объединением (IOU) между каждым треком и всеми предсказанными ограничивающими рамками существующих целей, используя венгерский алгоритм. Кроме того, минимальный IOU накладывается чтобы отклонить случаи, где целевое перекрытие меньше IOU_{min} .

Когда новый объект найден или потерян, соответственно создаются либо удаляются уникальные идентификаторы для его отслеживания. Для создания идентификатора, мы считаем что любой трек с перекрытием меньше IOU_{min} означает существование еще не отслеживаемого системой объекта. Его трек инициализируется с использованием геометрии охватывающей рамки со значением скорости, равным нулю. Так как скорость равна нулю, в этот момент ожидание ковариации скорости больше, чем в последующих случаях.

Дополнительно, новый отслеживаемый объект с идентификатором затем проходит испытательный период, когда цель должна быть обнаружена еще несколько раз, чтобы накопить достаточно доказательств истинности ее существования.

Объекты удаляются, если они не найдены за T_{lost} кадров. Это предотвращает неограниченный рост числа объектов и ошибки локализации, вызванные предсказаниями объектов с длинной продолжительностью без их новых местоположений от детектора.

2. ТРЕБОВАНИЯ К РАЗРАБОТКЕ СИСТЕМЫ ТРЕКИНГА

По результатам анализа предметной области система и исходя из задачи, система трекинга объектов должна реализовывать следующие требования:

- Локализация и классификация следующих категорий объектов на изображении: человек, автомобиль, автобус, грузовик, мотоцикл, велосипед, дорожный знак, светофор.
- Детектирование объектов должно происходить в реальном времени, либо с незначительными пропусками кадров.
- Источником видеопотока может быть видеофайл, либо потоковая трансляция с видеокамеры.
- Система должна отслеживать объекты в видеопотоке на протяжении последовательности кадров, при условии их обнаружения сверточной нейросетью.
- Система должна предсказывать местоположение каждого захваченного объекта на следующих кадрах и удалять объекты, долго не появляющиеся в поле зрения.

Для предсказания местоположения объекта необходимо применять методы фильтрации данных для координат x и y каждого объекта на изображении.

Система трекинга должна иметь легко настраиваемые пользователем параметры перед запуском.

Успешность работы нейросети зависит также от качества видеоисточника.

					ПГУ 2.090301.10.001 ПЗ	Лист
						62
Изм.	Лист	№ докум.	Подпись	Дата		

3. ВЫБОР СРЕДСТВ ПРОЕКТИРОВАНИЯ И РАЗРАБОТКИ

После анализа всех требований системы трекинга объектов необходимо определить сверточную нейросеть для детектирования объектов, язык программирования, фреймворк глубокого обучения для реализации нейросети, алгоритм фильтрации данных и библиотеку, позволяющую работать с изображениями и видео. По результатам анализа предметной области были выбраны следующие технологии для проектирования и разработки системы:

3.1 Python

В качестве языка программирования был выбран **Python** версии 3.7 – это высокоуровневый язык программирования общего назначения с динамической типизацией, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Он поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

					ПГУ 2.090301.10.001 ПЗ	Лист
						63
Изм.	Лист	№ докум.	Подпись	Дата		

Python — активно развивающийся язык программирования с большим сообществом. Он имеет свой собственный стандарт написания кода PEP 8. Важно отметить, что он является интерпретируемым языком, а сам интерпретатор доступен на всех основных используемых операционных системах на сегодняшний день.

Ключевые особенности:

- Динамическая типизация
- Полная поддержка ООП
- Хорошая стандартная библиотека
- Легкость разработки и чтения программ
- Повышенная по сравнению с большинством классических статически типизированных языков (C++, Java, C#) скорость разработки
- Кроссплатформенность

Стоит отдельно отметить тот факт, что выбор именно этого языка обусловлен наличием огромного количества библиотек глубокого обучения и машинного обучения, а также одним из самых больших и быстрорастущих сообществ ученых и программистов в области обработки Big-data, data-science, машинного обучения и ИИ. Рассмотрим использованные библиотеки и фреймворки:

3.1.1 Numpy

NumPy — это библиотека языка Python с открытым исходным кодом, добавляющая поддержку многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых математических функций для операций с

					ПГУ 2.090301.10.001 ПЗ	Лист
						64
Изм.	Лист	№ докум.	Подпись	Дата		

этим массивами. Python имеет в своем распоряжении массивы в виде списков (list) и коллекции (tuple), но numpy работает гораздо быстрее них, предоставляя скорость, сравнимую с компилируемыми языками программирования. Этот пакет является базовым для многих других библиотек.

3.1.2 PyTorch

PyTorch — современная библиотека глубокого обучения с открытым исходным кодом для языка Python, разработанная в компании Facebook на базе библиотеки torch для Lua. Распространяется по лицензии BSD, предоставляет две основные парадигмы:

- Тензорные вычисления (по аналогии с NumPy) с развитой поддержкой ускорения на GPU с помощью CUDA
- Построение глубоких нейронных сетей

PyTorch отличается от других библиотек глубокого обучения простотой установки, легкостью написания кода при проектировании нейросетей, развитой поддержкой, частыми полезными обновлениями и большим сообществом, в том числе ее используют такие компании как Twitter, Facebook, NVIDIA.

Также в отличие от большинства других библиотек глубокого обучения, граф вычислений создается и выполняется динамически, что позволяет быстро его модифицировать без необходимости перекомпилировать модель. Благодаря технике автоматического дифференцирования (automatic differentiation, AD) мы можем взять и продифференцировать граф в любой момент времени в любом его состоянии.

					ПГУ 2.090301.10.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

Тензоры могут вычисляться как на центральном процессоре, так на GPU с поддержкой CUDA, где скорость вычислений заметно повышается за счет распараллеливания вычислений. Тензор может находиться как в оперативной памяти, так и в области видеопамати.

3.1.3 OpenCV

OpenCV (Open Source Computer Vision Library) - Это открытая кроссплатформенная библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения, написанная на C и C++, также поставляется для многих других языков программирования. Распространяется под лицензией BSD.

Библиотека имеет более 2500 оптимизированных алгоритмов, включающих в себя полный набор как классических, так и современных алгоритмов компьютерного зрения и машинного обучения. Эти алгоритмы могут использоваться для обнаружения и распознавания лиц, идентификации объектов, классификации действий человека в видео, отслеживания движений камеры, отслеживания движущихся объектов, извлечения 3D-моделей объектов, создания 3D-облаков точек со стереокамер, сшивания изображений вместе для получения изображения с высоким разрешением всей сцены, поиска аналогичных изображений из базы данных изображений и многих других задач. Библиотека широко используется компаниями, исследовательскими группами и государственными органами.

OpenCV имеет интерфейсы C++, Python, Java и MATLAB и поддерживает операционные системы Windows, Linux, Android и Mac OS. OpenCV использует преимущества MMX и инструкций SSE при возможности. В настоящее время активно разрабатываются

					ПГУ 2.090301.10.001 ПЗ	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

полнофункциональные интерфейсы CUDA и OpenCL. Существует более 500 алгоритмов и примерно в 10 раз больше функций, которые составляют или поддерживают эти алгоритмы. OpenCV написан изначально на C++ и имеет шаблонный интерфейс, который легко работает с контейнерами STL.

Кроме того, OpenCV позволяет удобно работать с различными видеоисточниками, включая потоковую трансляцию с видеокамеры и обычные видеофайлы в различных форматах, а также оперировать изображениями в виде numpy массивов, рисовать обрамляющие рамки найденных объектов и выводить текстовую информацию.

3.2 YOLO

В качестве сверточной нейросети для детектирования объектов была выбрана сеть **YOLO v3** [5], реализованная на фреймворке PyTorch [12], обученная на наборе данных COCO (Common Objects in Context), предоставляющим 80 классов изображений, включая необходимые для распознавания транспорта классы: автомобиль, автобус, грузовик, дорожный знак, светофор, человек, мотоцикл, велосипед. Она позволяет выполнять детектирование объектов в реальном времени, либо с небольшими пропусками кадров при повышении точности распознавания путем увеличения размера входного изображения. На видеокарте Titan X она позволяет обрабатывать изображения с fps равным 30 и точностью 57.9% mAP на тестовой выборке COCO.

					ПГУ 2.090301.10.001 ПЗ	Лист
						67
Изм.	Лист	№ докум.	Подпись	Дата		

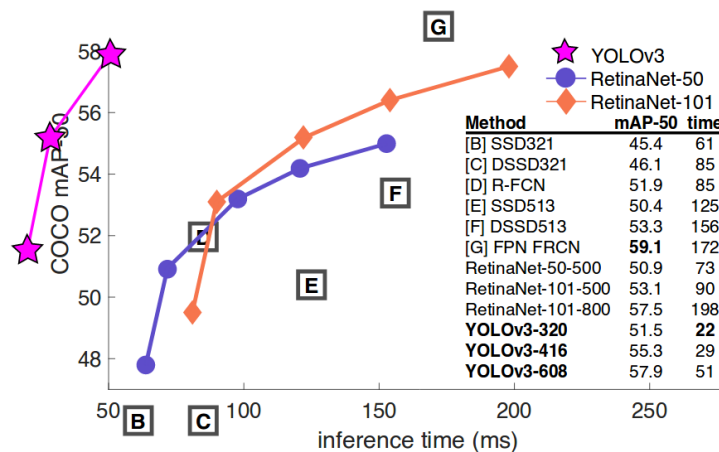


Рисунок 34 – Сравнение точности и скорости моделей YOLO с другими распространенными детекторами реального времени

Сеть принимает в качестве параметров размер входного изображения, путь к файлу с весами, порог уверенности от 0 до 1 для отсеивания объектов с низкой уверенностью и *nms-threshold* (порог не-максимального подавления) также от 0 до 1, позволяющий с применением данного коэффициента объединить несколько находжений одного и того же объекта в близких местах.

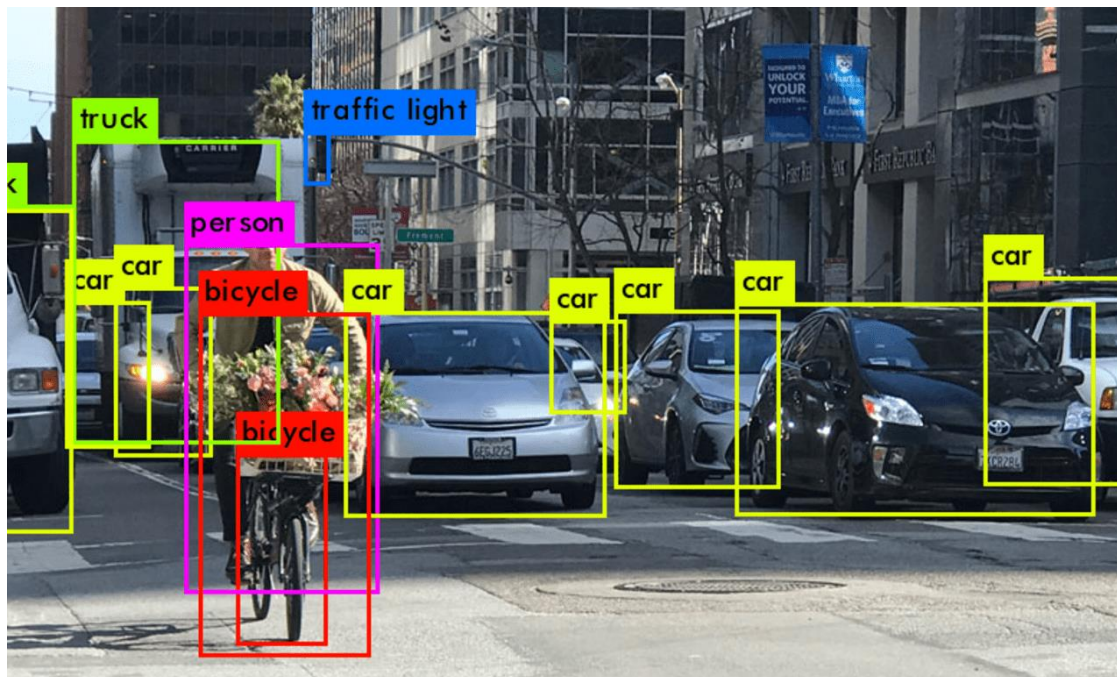


Рисунок 35 – Пример работы YOLO

3.3 SORT

SORT – алгоритм, выполняющий трекинг детектированных YOLO объектов на последовательности видеок кадров. Главным образом он основан на принципе работы фильтра Калмана. Алгоритм был подробнее рассмотрен в разделе 1.5.2. На момент публикации, SORT был оценен как лучший трекер множества объектов в MOT Benchmark. Программная реализация на языке Python была взята из оригинального документа [9].

3.4 PyCharm

Для более быстрой и удобной разработки системы подобного уровня необходима IDE, сочетающая в себе возможности синтаксического анализа кода, рефакторинга, удобной навигации по проекту, интеграцию с системами контроля версий, инструменты отладки и тестирования. Все это предоставляет кроссплатформенный программный продукт PyCharm 2018 Community Edition, распространяемый под лицензией Apache 2.

Кроме стандартных требований к любой современной IDE, PyCharm отличается удобным интерфейсом, встроенным терминалом и интерпретатором с возможностью подключения по ssh, поддержкой юнит-тестов, расширяемостью с помощью плагинов, а также интеграцией с Docker и всеми популярными системами контроля версий.

					ПГУ 2.090301.10.001 ПЗ	Лист
						69
Изм.	Лист	№ докум.	Подпись	Дата		

4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Для работы системы необходимо загрузить предобученные веса “yolov3.weights” и файл конфигурации “yolov3.cfg”, описывающей архитектуру восстановления вычислительного графа из файла весов, с официального сайта разработчика алгоритма YOLO [13]. Затем нужно импортировать их в папку проекта “yolofiles”.

Далее необходимо загрузить файл “coco.names”, описывающий названия классов в датасете COCO, и также импортировать их в эту папку. Путем сопоставления числа на выходе сверточной нейросети и номера класса в файле, можно получить название найденного класса по его номеру. В этой же папке стоит расположить файл “approved.names”, для того чтобы отсеять не интересные для детектирования классы объектов. Из 80 базовых классов нам нужны только 8, но в любой момент можно добавить остальные, либо продолжить обучение YOLO на большем числе классов и добавить их в оба файла.

Допустимо также использовать и другие варианты нейросети YOLO с официального сайта, такие как Tiny-YOLO. Для этого необходимо также скачать с официального сайта веса и конфигурационный файл описания, по аналогии поместив их в папку “yolofiles”.

Для работы системы необходим интерпретатор языка python 3.7 с установленными зависимостями: PyTorch $\geq 1.1.0$, numpy, OpenCV ≥ 3.5 , filterpy ≥ 1.4 . Допускается использование готовой сборки Anaconda и других поставляемых в сборе интерпретаторов, при условии удовлетворения ими зависимости пакетов. Установить отсутствующие зависимости можно с помощью пакетного менеджера pip или conda.

					ПГУ 2.090301.10.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

На запускаемой системе также может быть установлена видеокарта NVIDIA для ускорения тензорных вычислений путем параллельного выполнения. Для того чтобы применить GPU, необходима версия CUDA, совместимая с PyTorch версии $\geq 1.1.10$ и имеющимся GPU. Подробнее о совместимости на официальном сайте [10]. Если CUDA в системе не найдена, либо видеокарта недоступна, приложение выполняется последовательно на CPU.

4.1 Структура скрипта

Скрипт **tracker.py** запускается интерпретатором Python через командную строку, либо с использованием средств отладки IDE. Настройка параметров производится при запуске, с использованием командной строки.

Аргументы:

- `--source` – задает путь к источнику. При установке значения “0” будет использовать локальную web-камеру устройства.
- `--cfg` – задает путь к файлу конфигурации сети “.cfg”. Значение по умолчанию: “yolofiles/yolov3.cfg”
- `--weights` – задает путь к файлу весов “.weights”. Значение по умолчанию: “yolofiles/yolov3.weights”
- `--coco` – задает путь к файлу “.names”, содержащему названия классов, которые нейросеть обучена распознавать. Значение по умолчанию: “yolofiles/coco.names”
- `--approved` – задает путь к файлу “.names|”, содержащему интересующие нас при данном запуске названия классов (одобренные классы). Все названия в этом файле должны также

находится в файле coco, а нейросеть должна уметь распознавать классы одобренных объектов. Значение по умолчанию: “yolofiles/approved.names”

- --img-size – задает размер изображения, поступающего на вход YOLO. Должен быть кратен 32. Значение по умолчанию: 416.
- --conf-thres – задает порог уверенности определения объекта. Принимает значения от 0 до 1. Значение по умолчанию: 0.8
- --nms-thres – задает порог не-максимального подавления. Значение по умолчанию: 0.4
- --skip-frames – при установке параметра как “true” система пропускает кадры в видео, при условии если она не успевает обрабатывать каждый кадр в реальном времени. Значение по умолчанию: “false”
- --fourcc – задает параметры кодировщика выходного видеофайла. Значение по умолчанию: “XVID”
- --output – задает путь для записи выходного видеофайла. Может быть задан как “true”, в таком случае выходной файл запишется в ту же папку, что и входной, с тем же названием с приставкой “-det”. Значение по умолчанию: отсутствует.
- --help – вывод справки по командам

4.2 Описание классов и файлов

Скрипт **models.py** содержит класс Darknet, создающий в памяти один экземпляр вычислительного графа YOLO и классы, описывающие составляющие слои сети YOLO: UpsampleLayer, EmptyLayer, YOLOLayer.

Darknet принимает в качестве параметра файл конфигурации сети с расширением “.cfg” и размер входного изображения. Этот класс унаследован от базового класса библиотеки PyTorch – torch.nn.modules.module.Module

По информации конфигурационного файла из классов слоев строится вычислительный граф. Слои YOLO также наследуются от базового модуля PyTorch.

Скрипт **sort.py** описывает создание трекера, методы его обновления и глобальные параметры в классе Sort, а также класс KalmanBoxTracker, представляющий внутреннее состояние отдельных отслеживаемых объектов, наблюдаемых в виде ограничительной рамки (*bounding box*).

Скрипт **tracker.py** обращается к остальным скриптам при создании и использовании модели и фильтра, а также отвечает за весь алгоритм работы программы. На вход принимает аргументы командной строки.

					ПГУ 2.090301.10.001 ПЗ	Лист
						73
Изм.	Лист	№ докум.	Подпись	Дата		

4.3 Диаграмма классов UML

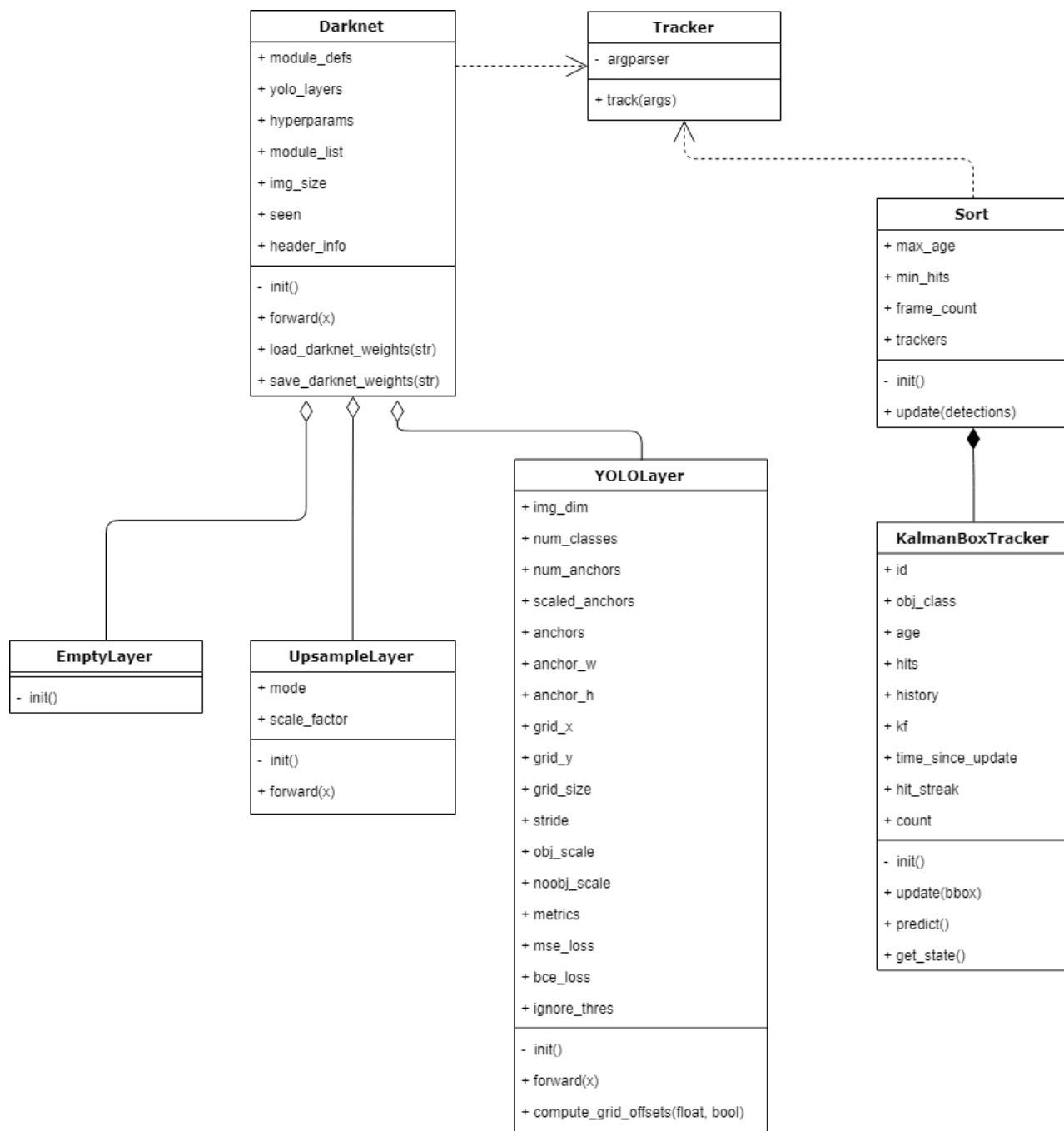


Рисунок 36 – Зависимости системы трекинга в виде диаграммы классов UML

4.4 Результаты работы системы

Система успешно определяет одни и те же объекты на видео с разницей во времени, при условии их обнаружения нейросетью и нахождении в поле зрения. Также система справляется с кратковременным скрыванием объекта за препятствием либо перекрытием одного объекта другим.

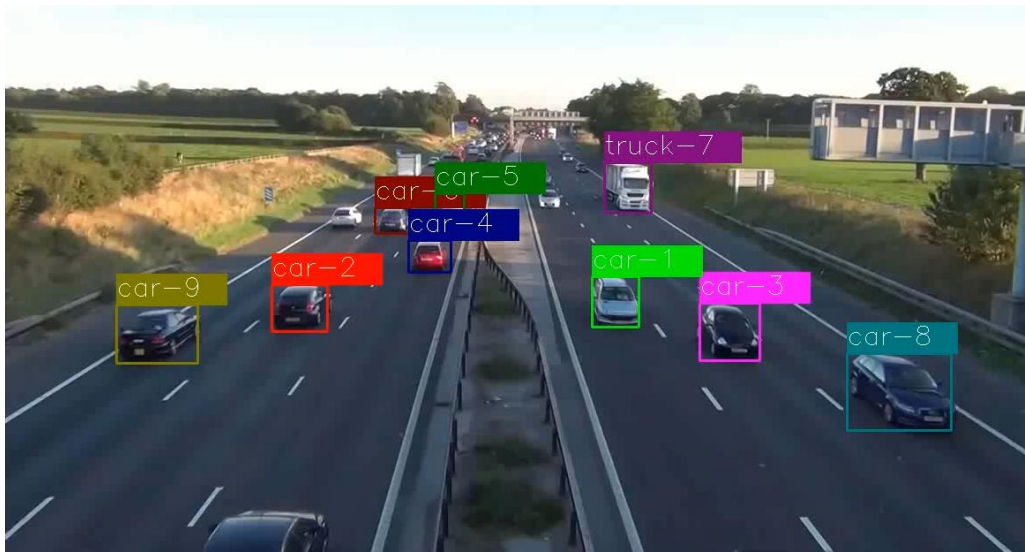


Рисунок 37- Обнаружение объектов в начале работы системы

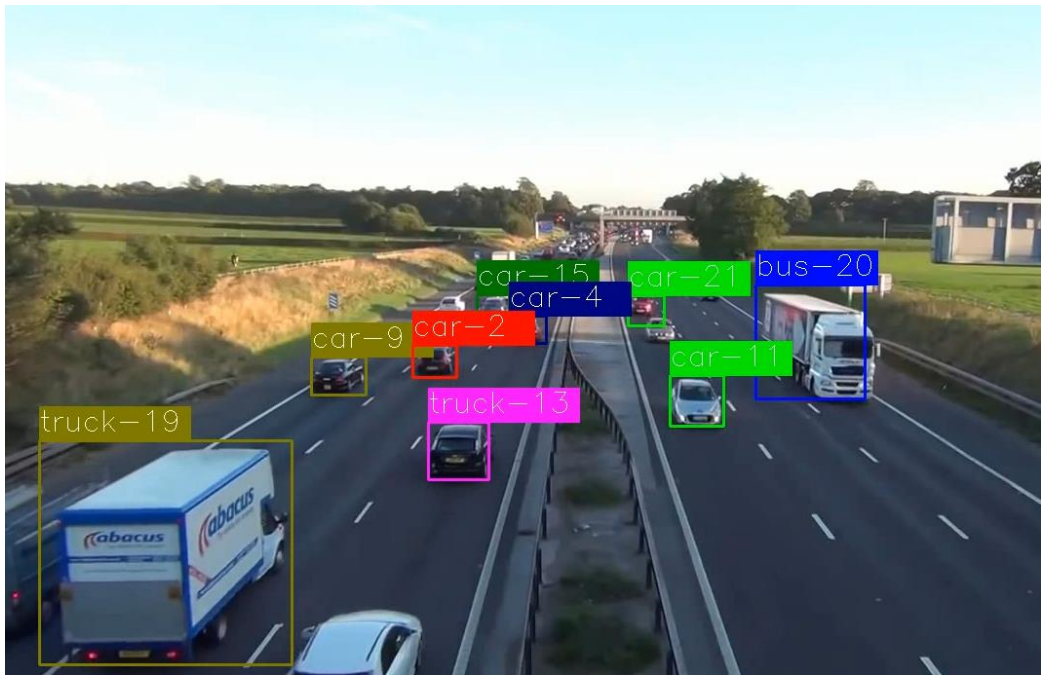


Рисунок 38 – Результат спустя 5 секунд

5. ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работ была создана система трекинга объектов, работающая с любым видеоисточником, на основе сверточной нейронной сети YOLO версии 3. Система успешно определяет местонахождение уникальных объектов, при условии их корректного распознавания нейросетью. Точность и скорость работы системы в будущем можно повысить, используя более совершенную архитектуру сверточной нейронной сети.

Текущий уровень точности распознавания и трекинга дает возможность использовать данную систему в составе любого программного комплекса для решения прикладных задач отслеживания объектов. Сеть может быть дополнительно обучена для определения классов, отсутствующих в базовом датасете COCO. Данная система готова к использованию в комплексе определения нарушений ПДД, контроля скорости, обеспечения контроля безопасности против проникновения на охраняемую территорию, подсчета объектов с помощью управляемого дрона или БПЛА и многим другим применениям.

Требования технического задания выполнены в полном объеме.

					ПГУ 2.090301.10.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

- [1] Способы классификации движущихся объектов на видео / Исаев А. Л., Газаров Д. А., Евсеев С. Д. // Молодой ученый. — 2016. — №18. — С. 44-47. [Электронный ресурс]: <https://moluch.ru/archive/122/33749/>
- [2] Stanford University CS231n: Convolutional Neural Networks for Visual Recognition. [Электронный ресурс]: <http://cs231n.stanford.edu>
- [3] Real-Time Grasp Detection Using Convolutional Neural Networks / Joseph Redmon, Anelia Angelova // IEEE International Conference on Robotics and Automation (ICRA). — 2015 [Электронный ресурс]: <http://arxiv.org/abs/1412.3128>
- [4] Fast R-CNN / Ross Girshick // IEEE International Conference on Computer Vision (ICCV) — 2015. [Электронный ресурс]: <https://arxiv.org/abs/1504.08083>
- [5] Faster R-CNN: Towards real-time object detection with region proposal networks / Shaoqing Ren [и др.] // IEEE Transactions on Pattern Analysis and Machine Intelligence. — 2017. — Vol. 39, № 6. — Pp.1137-1149. [Электронный ресурс]: <https://arxiv.org/abs/1506.01497>
- [6] You Only Look Once: Unified, Real-Time Object Detection / Joseph Redmon [и др.] // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). — 2016. [Электронный ресурс]: <https://arxiv.org/abs/1506.02640>
- [7] YOLOv3: An Incremental Improvement / Joseph Redmon, Ali Farhadi // Computing Research Repository (CoRR). — 2018. [Электронный ресурс]: <https://arxiv.org/abs/1804.02767>
- [8] SSD: Single Shot MultiBox Detector / Wei Liu [и др.] // The 14th European Conference on Computer Vision (ECCV). — 2016.

- [9] Simple Online and Realtime Tracking / Alex Bewley, Zongyuan Ge [и др.] // IEEE International Conference on Image Processing (ICIP). – 2017.
[Электронный ресурс]: <https://arxiv.org/abs/1602.00763>
- [10] PyTorch [Электронный ресурс]: <https://pytorch.org>
- [11] OpenCV [Электронный ресурс]: <https://opencv.org>
- [12] PyTorch implementation of YOLOv3. [Электронный ресурс]:
<https://github.com/ultralytics/yolov3>
- [13] YOLO: Real-Time Object Detection [Электронный ресурс]:
<https://pjreddie.com/darknet/yolo/>
- [14] MS COCO Dataset [Электронный ресурс]: <http://cocodataset.org/#home>

ПРИЛОЖЕНИЕ А: ЛИСТИНГИ ПРОГРАММ

Приложение А.1 – Файл «tracker.py»

```
from models import *
from utils import utils

import argparse
import os, sys, time, datetime, random
import torch
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from torch.autograd import Variable

from PIL import Image
import cv2
from sortnew import *
from math import ceil

def tracker(
    source,
    cfg,
    weights,
    coco_classes,
    approved_classes,
    img_size,
    conf_thres,
    nms_thres,
    skip_frames,
    fourcc,
    output
):
    with open(approved_classes) as file:
        approved_classes = [row.strip() for row in file]

    model = Darknet(cfg, img_size=img_size)
    model.load_darknet_weights(weights)
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
    model.to(device)
    model.eval()

    classes = utils.load_classes(coco_classes)
    Tensor = torch.cuda.FloatTensor if torch.cuda.is_available() else torch.FloatTensor

    def detect_image(img):
        # scale and pad image
        ratio = min(img_size/img.size[0], img_size/img.size[1])
        imw = round(img.size[0] * ratio)
        imh = round(img.size[1] * ratio)
        img_transforms = transforms.Compose([
            transforms.Resize((imh, imw)),
            transforms.Pad(
                (max(int((imh-imw)/2),0), max(int((imw-imh)/2),0), max(int((imh-imw)/2),0), max(int((imw-imh)/2),0)),
                (128,128,128)
            ),
            transforms.ToTensor(),
```

					ПГУ 2.090301.10.001 ПЗ	Лист
						79
Изм.	Лист	№ докум.	Подпись	Дата		

```

    ])
    # convert image to Tensor
    image_tensor = img_transforms(img).float()
    image_tensor = image_tensor.unsqueeze_(0)
    input_img = Variable(image_tensor.type(Tensor))
    # run inference on the model and get detections
    with torch.no_grad():
        detections = model(input_img)
        detections = utils.non_max_suppression(detections, conf_thres, nms_thres)
    return detections[0]

```

```

colors=[(255,0,0),(0,255,0),(0,0,255),(255,0,255),(128,0,0),(0,128,0),(0,0,128),(128,0,128),(128,128,0),(0,128,128)]

```

```

cap = cv2.VideoCapture(source)
mot_tracker = Sort(max_age=60, min_hits=5)
fps = cap.get(cv2.CAP_PROP_FPS)

ret, frame = cap.read()
if ret is True:
    vw = frame.shape[1]
    vh = frame.shape[0]
    print("Video ", source, " size ", vw, 'x', vh)
else:
    print("Can't read video ", source)
    exit()

if output is not "":
    if output is 'true':
        video_output_path = source.replace(".mp4", "-detected.avi")
        fourcc = cv2.VideoWriter_fourcc(*fourcc)
        outvideo = cv2.VideoWriter(video_output_path, fourcc, fps, (vw, vh))
    else:
        outvideo = False

cv2.namedWindow('Stream', cv2.WINDOW_NORMAL)
cv2.resizeWindow('Stream', (vw, vh))

frames = 0
start_time = time.time()
while True:
    frame_detection_time = time.time()
    ret, frame = cap.read()
    if not ret:
        break
    frames += 1
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    pilimg = Image.fromarray(frame)
    detections = detect_image(pilimg)

    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
    img = np.array(pilimg)
    pad_x = max(img.shape[0] - img.shape[1], 0) * (img_size / max(img.shape))
    pad_y = max(img.shape[1] - img.shape[0], 0) * (img_size / max(img.shape))
    unpad_h = img_size - pad_y
    unpad_w = img_size - pad_x

```

					ПГУ 2.090301.10.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

```

objects_in_frame = 0
if detections is not None:
    tracked_objects = mot_tracker.update(detections.cpu())
    objects_in_frame = tracked_objects.shape[0]

    unique_labels = detections[:, -1].cpu().unique()
    scores = detections[:, 4].cpu().numpy()
    for x1, y1, x2, y2, obj_id, cls_pred in tracked_objects:
        cls = classes[int(cls_pred)]
        if cls not in approved_classes:
            tracked_objects -= 1
            break

    box_h = int(((y2 - y1) / unpad_h) * img.shape[0])
    box_w = int(((x2 - x1) / unpad_w) * img.shape[1])
    y1 = int(((y1 - pad_y // 2) / unpad_h) * img.shape[0])
    x1 = int(((x1 - pad_x // 2) / unpad_w) * img.shape[1])
    color = colors[int(obj_id) % len(colors)]
    cv2.rectangle(frame, (x1, y1), (x1+box_w, y1+box_h), color, 2)
    cv2.rectangle(frame, (x1, y1-35), (x1+len(cls)*15+80, y1), color, -1)
    cv2.putText(frame, cls + "-" + str(int(obj_id)) + ' ', (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255,255,255), 1)
    cv2.imshow('Stream', frame)
    total_time = time.time() - start_time
    print("frame", frames, ",", objects_in_frame, "tracked objects, time", time.time() - frame_detection_time)
    if outvideo is not False:
        outvideo.write(frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    if skip_frames is not 'false' and frames < total_time * fps:
        while frames < total_time * fps:
            cap.read()
            frames += 1

totaltime = time.time() - start_time
print(frames, "frames", totaltime/frames, "average sec/frame")
cv2.destroyAllWindows()
cap.release()
if output is not "":
    outvideo.release()
return

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--source', type=str, default='traffic.mp4',
        help='path or url to source video file, set 0 for local webcam')
    parser.add_argument('--cfg', type=str, default='yolofiles/yolov3.cfg', help='cfg file path')
    parser.add_argument('--weights', type=str, default='yolofiles/yolov3.weights', help='path to weights file')
    parser.add_argument('--coco', type=str, default='yolofiles/coco.names', help='coco.names file path')
    parser.add_argument('--approved', type=str, default='yolofiles/approved.names',
        help='approved.names file path to show only approved classes')
    parser.add_argument('--img-size', type=int, default=416, help='inference size (pixels): 320 416 512. Must be %
32')
    parser.add_argument('--conf-thres', type=float, default=0.8, help='object confidence threshold')
    parser.add_argument('--nms-thres', type=float, default=0.4, help='iou threshold for non-maximum suppression')

```

```

parser.add_argument('--skip-frames', type=str, default='false', help='skip frames if cant work in realtime')
parser.add_argument('--fourcc', type=str, default='XVID',
                    help='specifies the fourcc code for output video encoding (make sure ffmpeg supports specified
fourcc codec)')
parser.add_argument('--output', type=str, default='', help='specifies the output path folder for video')

try:
    opt = parser.parse_args()
except:
    parser.print_help()
    sys.exit(0)

print(opt)

with torch.no_grad():
    tracker(
        opt.source,
        opt.cfg,
        opt.weights,
        opt.coco,
        opt.approved,
        opt.img_size,
        opt.conf_thres,
        opt.nms_thres,
        opt.skip_frames,
        opt.fourcc,
        opt.output
    )

```

Приложение А.2 – Файл «sort.py»

""""

SORT: A Simple, Online and Realtime Tracker
 Copyright (C) 2016 Alex Bewley alex@dynamicdetection.com

This program is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 GNU General Public License for more details.

You should have received a copy of the GNU General Public License
 along with this program. If not, see <<http://www.gnu.org/licenses/>>.

""""

```
from __future__ import print_function
```

```

from numba import jit
import os.path
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches

```

					ПГУ 2.090301.10.001 ПЗ	Лист
						82
Изм.	Лист	№ докум.	Подпись	Дата		

```

from skimage import io
# from sklearn.utils.linear_assignment_ import linear_assignment
from scipy.optimize import linear_sum_assignment
import glob
import time
import argparse
from filterpy.kalman import KalmanFilter

@jit(forceobj=True)
def iou(bb_test,bb_gt):
    """
    Computes IUO between two bboxes in the form [x1,y1,x2,y2]
    """
    xx1 = np.maximum(bb_test[0], bb_gt[0])
    yy1 = np.maximum(bb_test[1], bb_gt[1])
    xx2 = np.minimum(bb_test[2], bb_gt[2])
    yy2 = np.minimum(bb_test[3], bb_gt[3])
    w = np.maximum(0., xx2 - xx1)
    h = np.maximum(0., yy2 - yy1)
    wh = w * h
    o = wh / ((bb_test[2]-bb_test[0])*(bb_test[3]-bb_test[1])
    + (bb_gt[2]-bb_gt[0])*(bb_gt[3]-bb_gt[1]) - wh)
    return(o)

def convert_bbox_to_z(bbox):
    """
    Takes a bounding box in the form [x1,y1,x2,y2] and returns z in the form
    [x,y,s,r] where x,y is the centre of the box and s is the scale/area and r is
    the aspect ratio
    """
    w = bbox[2]-bbox[0]
    h = bbox[3]-bbox[1]
    x = bbox[0]+w/2.
    y = bbox[1]+h/2.
    s = w*h #scale is just area
    r = w/float(h)
    return np.array([x,y,s,r]).reshape((4,1))

def convert_x_to_bbox(x,score=None):
    """
    Takes a bounding box in the centre form [x,y,s,r] and returns it in the form
    [x1,y1,x2,y2] where x1,y1 is the top left and x2,y2 is the bottom right
    """
    w = np.sqrt(x[2]*x[3])
    h = x[2]/w
    if(score==None):
        return np.array([x[0]-w/2.,x[1]-h/2.,x[0]+w/2.,x[1]+h/2.]).reshape((1,4))
    else:
        return np.array([x[0]-w/2.,x[1]-h/2.,x[0]+w/2.,x[1]+h/2.,score]).reshape((1,5))

class KalmanBoxTracker(object):
    """
    This class represents the internal state of individual tracked objects observed as bbox.
    """
    count = 0
    def __init__(self,bbox):

```

					ПГУ 2.090301.10.001 ПЗ	Лист
						83
Изм.	Лист	№ докум.	Подпись	Дата		

```

"""
Initialises a tracker using initial bounding box.
"""

#define constant velocity model
self.kf = KalmanFilter(dim_x=7, dim_z=4)
self.kf.F = np.array([[1,0,0,0,1,0,0],[0,1,0,0,0,1,0],[0,0,1,0,0,0,1],[0,0,0,1,0,0,0],
[0,0,0,0,1,0,0],[0,0,0,0,0,1,0],[0,0,0,0,0,0,1]])
self.kf.H = np.array([[1,0,0,0,0,0,0],[0,1,0,0,0,0,0],[0,0,1,0,0,0,0],[0,0,0,1,0,0,0]])

self.kf.R[2:,2:] *= 10.
self.kf.P[4:,4:] *= 1000. #give high uncertainty to the unobservable initial velocities
self.kf.P *= 10.
self.kf.Q[-1,-1] *= 0.01
self.kf.Q[4:,4:] *= 0.01

self.kf.x[:4] = convert_bbox_to_z(bbox)
self.time_since_update = 0
self.id = KalmanBoxTracker.count
KalmanBoxTracker.count += 1
self.history = []
self.hits = 0
self.hit_streak = 0
self.age = 0
self.objclass = bbox[6]

def update(self,bbox):
    """
    Updates the state vector with observed bbox.
    """
    self.time_since_update = 0
    self.history = []
    self.hits += 1
    self.hit_streak += 1
    self.kf.update(convert_bbox_to_z(bbox))

def predict(self):
    """
    Advances the state vector and returns the predicted bounding box estimate.
    """
    if((self.kf.x[6]+self.kf.x[2])<=0):
        self.kf.x[6] *= 0.0
    self.kf.predict()
    self.age += 1
    if(self.time_since_update>0):
        self.hit_streak = 0
    self.time_since_update += 1
    self.history.append(convert_x_to_bbox(self.kf.x))
    return self.history[-1]

def get_state(self):
    """
    Returns the current bounding box estimate.
    """
    return convert_x_to_bbox(self.kf.x)

def associate_detections_to_trackers(detections,trackers,iou_threshold = 0.3):
    """

```

					ПГУ 2.090301.10.001 ПЗ	Лист
						84
Изм.	Лист	№ докум.	Подпись	Дата		

Assigns detections to tracked object (both represented as bounding boxes)

Returns 3 lists of matches, unmatched_detections and unmatched_trackers

```
"""
if(len(trackers)==0):
    return np.empty((0,2),dtype=int), np.arange(len(detections)), np.empty((0,5),dtype=int)
iou_matrix = np.zeros((len(detections),len(trackers)),dtype=np.float32)

for d,det in enumerate(detections):
    for t,trk in enumerate(trackers):
        iou_matrix[d,t] = iou(det,trk)
row_ind, col_ind = linear_sum_assignment(-iou_matrix)
matched_indices = np.column_stack((row_ind, col_ind))
matched_indices.shape = (-1, 2)

unmatched_detections = []
for d,det in enumerate(detections):
    if(d not in matched_indices[:,0]):
        unmatched_detections.append(d)
unmatched_trackers = []
for t,trk in enumerate(trackers):
    if(t not in matched_indices[:,1]):
        unmatched_trackers.append(t)

#filter out matched with low IOU
matches = []
for m in matched_indices:
    if(iou_matrix[m[0],m[1]]<iou_threshold):
        unmatched_detections.append(m[0])
        unmatched_trackers.append(m[1])
    else:
        matches.append(m.reshape(1,2))
if(len(matches)==0):
    matches = np.empty((0,2),dtype=int)
else:
    matches = np.concatenate(matches,axis=0)

return matches, np.array(unmatched_detections), np.array(unmatched_trackers)
```

```
class Sort(object):
    def __init__(self, max_age=1, min_hits=3):
        """
        Sets key parameters for SORT
        """
        self.max_age = max_age
        self.min_hits = min_hits
        self.trackers = []
        self.frame_count = 0

    def update(self,dets):
        """
        Params:
            dets - a numpy array of detections in the format [[x1,y1,x2,y2,score],[x1,y1,x2,y2,score],...]
        Requires: this method must be called once for each frame even with empty detections.
        Returns the a similar array, where the last column is the object ID.
```

					ПГУ 2.090301.10.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

NOTE: The number of objects returned may differ from the number of detections provided.

"""

self.frame_count += 1

#get predicted locations from existing trackers.

trks = np.zeros((len(self.trackers),5))

to_del = []

ret = []

for t, trk in enumerate(trks):

pos = self.trackers[t].predict()[0]

trk[:] = [pos[0], pos[1], pos[2], pos[3], 0]

if(np.any(np.isnan(pos))):

to_del.append(t)

trks = np.ma.compress_rows(np.ma.masked_invalid(trks))

for t in reversed(to_del):

self.trackers.pop(t)

matched, unmatched_dets, unmatched_trks = associate_detections_to_trackers(dets, trks)

#update matched trackers with assigned detections

for t, trk in enumerate(self.trackers):

if(t not in unmatched_trks):

d = matched[np.where(matched[:,1]==t)[0],0]

trk.update(dets[d,:][0])

#create and initialise new trackers for unmatched detections

for i in unmatched_dets:

trk = KalmanBoxTracker(dets[i,:])

self.trackers.append(trk)

i = len(self.trackers)

for trk in reversed(self.trackers):

d = trk.get_state()[0]

if((trk.time_since_update < 1) and (trk.hit_streak >= self.min_hits or self.frame_count <= self.min_hits)):

ret.append(np.concatenate((d,[trk.id+1], [trk.objclass])).reshape(1,-1)) # +1 as MOT benchmark requires

positive

i -= 1

#remove dead tracklet

if(trk.time_since_update > self.max_age):

self.trackers.pop(i)

if(len(ret)>0):

return np.concatenate(ret)

return np.empty((0,5))

def parse_args():

"""Parse input arguments."""

parser = argparse.ArgumentParser(description='SORT demo')

parser.add_argument('--display', dest='display', help='Display online tracker output (slow)

[False]', action='store_true')

args = parser.parse_args()

return args

if __name__ == '__main__':

all train

sequences = ['PETS09-S2L1', 'TUD-Campus', 'TUD-Stadtmitte', 'ETH-Bahnhof', 'ETH-Sunnyday', 'ETH-Pedcross2', 'KITTI-13', 'KITTI-17', 'ADL-Rundle-6', 'ADL-Rundle-8', 'Venice-2']

args = parse_args()

display = args.display

phase = 'train'

					ПГУ 2.090301.10.001 ПЗ	Лист
						86
Изм.	Лист	№ докум.	Подпись	Дата		

```

total_time = 0.0
total_frames = 0
colours = np.random.rand(32,3) #used only for display
if(display):
    if not os.path.exists('mot_benchmark'):
        print('\n\tERROR: mot_benchmark link not found!\n\n  Create a symbolic link to the MOT benchmark\n
(https://motchallenge.net/data/2D_MOT_2015/#download). E.g.: \n\n  $ ln -s
/path/to/MOT2015_challenge/2DMOT2015 mot_benchmark\n\n')
        exit()
    plt.ion()
    fig = plt.figure()

if not os.path.exists('output'):
    os.makedirs('output')

for seq in sequences:
    mot_tracker = Sort() #create instance of the SORT tracker
    seq_dets = np.loadtxt('data/%s/det.txt'%(seq),delimiter=',') #load detections
    with open('output/%s.txt'%(seq),'w') as out_file:
        print("Processing %s."%(seq))
        for frame in range(int(seq_dets[:,0].max()+1)):
            frame += 1 #detection and frame numbers begin at 1
            dets = seq_dets[seq_dets[:,0]==frame,2:7]
            dets[:,2:4] += dets[:,0:2] #convert to [x1,y1,w,h] to [x1,y1,x2,y2]
            total_frames += 1

            if(display):
                ax1 = fig.add_subplot(111, aspect='equal')
                fn = 'mot_benchmark/%s/%s/img1/%06d.jpg'%(phase,seq,frame)
                im = io.imread(fn)
                ax1.imshow(im)
                plt.title(seq+' Tracked Targets')

            start_time = time.time()
            trackers = mot_tracker.update(dets)
            cycle_time = time.time() - start_time
            total_time += cycle_time

            for d in trackers:
                print('%d,%d,%0.2f,%0.2f,%0.2f,%0.2f,1,-1,-1,-1'%(frame,d[4],d[0],d[1],d[2]-d[0],d[3]-d[1]),file=out_file)
                if(display):
                    d = d.astype(np.int32)
                    ax1.add_patch(patches.Rectangle((d[0],d[1]),d[2]-d[0],d[3]-d[1],fill=False,lw=3,ec=colours[d[4]%32,:]))
                    ax1.set_adjustable('box-forced')

            if(display):
                fig.canvas.flush_events()
                plt.draw()
                ax1.cla()

print("Total Tracking took: %.3f for %d frames or %.1f FPS"%(total_time,total_frames,total_frames/total_time))
if(display):
    print("Note: to get real runtime results run without the option: --display")

```

					ПГУ 2.090301.10.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.3 – Файл «models.py»

```

from __future__ import division

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
import numpy as np

from utils.parse_config import *
from utils.utils import build_targets, to_cpu, non_max_suppression

import matplotlib.pyplot as plt
import matplotlib.patches as patches

def create_modules(module_defs):
    """
    Constructs module list of layer blocks from module configuration in module_defs
    """
    hyperparams = module_defs.pop(0)
    output_filters = [int(hyperparams["channels"])]
    module_list = nn.ModuleList()
    for module_i, module_def in enumerate(module_defs):
        modules = nn.Sequential()

        if module_def["type"] == "convolutional":
            bn = int(module_def["batch_normalize"])
            filters = int(module_def["filters"])
            kernel_size = int(module_def["size"])
            pad = (kernel_size - 1) // 2
            modules.add_module(
                f"conv_{module_i}",
                nn.Conv2d(
                    in_channels=output_filters[-1],
                    out_channels=filters,
                    kernel_size=kernel_size,
                    stride=int(module_def["stride"]),
                    padding=pad,
                    bias=not bn,
                ),
            )
            if bn:
                modules.add_module(f"batch_norm_{module_i}", nn.BatchNorm2d(filters, momentum=0.9, eps=1e-5))
            if module_def["activation"] == "leaky":
                modules.add_module(f"leaky_{module_i}", nn.LeakyReLU(0.1))

        elif module_def["type"] == "maxpool":
            kernel_size = int(module_def["size"])
            stride = int(module_def["stride"])
            if kernel_size == 2 and stride == 1:
                modules.add_module(f"_debug_padding_{module_i}", nn.ZeroPad2d((0, 1, 0, 1)))
                maxpool = nn.MaxPool2d(kernel_size=kernel_size, stride=stride, padding=int((kernel_size - 1) // 2))
                modules.add_module(f"maxpool_{module_i}", maxpool)

        elif module_def["type"] == "upsample":

```

					ПГУ 2.090301.10.001 ПЗ	Лист
						88
Изм.	Лист	№ докум.	Подпись	Дата		

```

        upsample = Upsample(scale_factor=int(module_def["stride"]), mode="nearest")
        modules.add_module(f"upsample_{module_i}", upsample)

    elif module_def["type"] == "route":
        layers = [int(x) for x in module_def["layers"].split(",")]
        filters = sum([output_filters[1:][i] for i in layers])
        modules.add_module(f"route_{module_i}", EmptyLayer())

    elif module_def["type"] == "shortcut":
        filters = output_filters[1:][int(module_def["from"])]
        modules.add_module(f"shortcut_{module_i}", EmptyLayer())

    elif module_def["type"] == "yolo":
        anchor_idxs = [int(x) for x in module_def["mask"].split(",")]
        # Extract anchors
        anchors = [int(x) for x in module_def["anchors"].split(",")]
        anchors = [(anchors[i], anchors[i + 1]) for i in range(0, len(anchors), 2)]
        anchors = [anchors[i] for i in anchor_idxs]
        num_classes = int(module_def["classes"])
        img_size = int(hyperparams["height"])
        # Define detection layer
        yolo_layer = YOLOLayer(anchors, num_classes, img_size)
        modules.add_module(f"yolo_{module_i}", yolo_layer)
    # Register module list and number of output filters
    module_list.append(modules)
    output_filters.append(filters)

return hyperparams, module_list

class Upsample(nn.Module):
    """ nn.Upsample is deprecated """

    def __init__(self, scale_factor, mode="nearest"):
        super(Upsample, self).__init__()
        self.scale_factor = scale_factor
        self.mode = mode

    def forward(self, x):
        x = F.interpolate(x, scale_factor=self.scale_factor, mode=self.mode)
        return x

class EmptyLayer(nn.Module):
    """Placeholder for 'route' and 'shortcut' layers"""

    def __init__(self):
        super(EmptyLayer, self).__init__()

class YOLOLayer(nn.Module):
    """Detection layer"""

    def __init__(self, anchors, num_classes, img_dim=416):
        super(YOLOLayer, self).__init__()
        self.anchors = anchors
        self.num_anchors = len(anchors)

```

					ПГУ 2.090301.10.001 ПЗ	Лист
						89
Изм.	Лист	№ докум.	Подпись	Дата		

```

self.num_classes = num_classes
self.ignore_thres = 0.5
self.mse_loss = nn.MSELoss()
self.bce_loss = nn.BCELoss()
self.obj_scale = 1
self.noobj_scale = 100
self.metrics = {}
self.img_dim = img_dim
self.grid_size = 0 # grid size

def compute_grid_offsets(self, grid_size, cuda=True):
    self.grid_size = grid_size
    g = self.grid_size
    FloatTensor = torch.cuda.FloatTensor if cuda else torch.FloatTensor
    self.stride = self.img_dim / self.grid_size
    # Calculate offsets for each grid
    self.grid_x = torch.arange(g).repeat(g, 1).view([1, 1, g, g]).type(FloatTensor)
    self.grid_y = torch.arange(g).repeat(g, 1).t().view([1, 1, g, g]).type(FloatTensor)
    self.scaled_anchors = FloatTensor([(a_w / self.stride, a_h / self.stride) for a_w, a_h in self.anchors])
    self.anchor_w = self.scaled_anchors[:, 0:1].view((1, self.num_anchors, 1, 1))
    self.anchor_h = self.scaled_anchors[:, 1:2].view((1, self.num_anchors, 1, 1))

def forward(self, x, targets=None, img_dim=None):

    # Tensors for cuda support
    FloatTensor = torch.cuda.FloatTensor if x.is_cuda else torch.FloatTensor
    LongTensor = torch.cuda.LongTensor if x.is_cuda else torch.LongTensor
    ByteTensor = torch.cuda.ByteTensor if x.is_cuda else torch.ByteTensor

    self.img_dim = img_dim
    num_samples = x.size(0)
    grid_size = x.size(2)

    prediction = (
        x.view(num_samples, self.num_anchors, self.num_classes + 5, grid_size, grid_size)
        .permute(0, 1, 3, 4, 2)
        .contiguous()
    )

    # Get outputs
    x = torch.sigmoid(prediction[..., 0]) # Center x
    y = torch.sigmoid(prediction[..., 1]) # Center y
    w = prediction[..., 2] # Width
    h = prediction[..., 3] # Height
    pred_conf = torch.sigmoid(prediction[..., 4]) # Conf
    pred_cls = torch.sigmoid(prediction[..., 5:]) # Cls pred.

    # If grid size does not match current we compute new offsets
    if grid_size != self.grid_size:
        self.compute_grid_offsets(grid_size, cuda=x.is_cuda)

    # Add offset and scale with anchors
    pred_boxes = FloatTensor(prediction[..., :4].shape)
    pred_boxes[..., 0] = x.data + self.grid_x
    pred_boxes[..., 1] = y.data + self.grid_y
    pred_boxes[..., 2] = torch.exp(w.data) * self.anchor_w
    pred_boxes[..., 3] = torch.exp(h.data) * self.anchor_h

```

```

output = torch.cat(
    (
        pred_boxes.view(num_samples, -1, 4) * self.stride,
        pred_conf.view(num_samples, -1, 1),
        pred_cls.view(num_samples, -1, self.num_classes),
    ),
    -1,
)

if targets is None:
    return output, 0
else:
    iou_scores, class_mask, obj_mask, noobj_mask, tx, ty, tw, th, tcls, tconf = build_targets(
        pred_boxes=pred_boxes,
        pred_cls=pred_cls,
        target=targets,
        anchors=self.scaled_anchors,
        ignore_thres=self.ignore_thres,
    )

    # Loss : Mask outputs to ignore non-existing objects (except with conf. loss)
    loss_x = self.mse_loss(x[obj_mask], tx[obj_mask])
    loss_y = self.mse_loss(y[obj_mask], ty[obj_mask])
    loss_w = self.mse_loss(w[obj_mask], tw[obj_mask])
    loss_h = self.mse_loss(h[obj_mask], th[obj_mask])
    loss_conf_obj = self.bce_loss(pred_conf[obj_mask], tconf[obj_mask])
    loss_conf_noobj = self.bce_loss(pred_conf[noobj_mask], tconf[noobj_mask])
    loss_conf = self.obj_scale * loss_conf_obj + self.noobj_scale * loss_conf_noobj
    loss_cls = self.bce_loss(pred_cls[obj_mask], tcls[obj_mask])
    total_loss = loss_x + loss_y + loss_w + loss_h + loss_conf + loss_cls

    # Metrics
    cls_acc = 100 * class_mask[obj_mask].mean()
    conf_obj = pred_conf[obj_mask].mean()
    conf_noobj = pred_conf[noobj_mask].mean()
    conf50 = (pred_conf > 0.5).float()
    iou50 = (iou_scores > 0.5).float()
    iou75 = (iou_scores > 0.75).float()
    detected_mask = conf50 * class_mask * tconf
    precision = torch.sum(iou50 * detected_mask) / (conf50.sum() + 1e-16)
    recall50 = torch.sum(iou50 * detected_mask) / (obj_mask.sum() + 1e-16)
    recall75 = torch.sum(iou75 * detected_mask) / (obj_mask.sum() + 1e-16)

    self.metrics = {
        "loss": to_cpu(total_loss).item(),
        "x": to_cpu(loss_x).item(),
        "y": to_cpu(loss_y).item(),
        "w": to_cpu(loss_w).item(),
        "h": to_cpu(loss_h).item(),
        "conf": to_cpu(loss_conf).item(),
        "cls": to_cpu(loss_cls).item(),
        "cls_acc": to_cpu(cls_acc).item(),
        "recall50": to_cpu(recall50).item(),
        "recall75": to_cpu(recall75).item(),
        "precision": to_cpu(precision).item(),
        "conf_obj": to_cpu(conf_obj).item(),
    }

```

					ПГУ 2.090301.10.001 ПЗ	Лист
						91
Изм.	Лист	№ докум.	Подпись	Дата		

```

        "conf_noobj": to_cpu(conf_noobj).item(),
        "grid_size": grid_size,
    }

```

```

    return output, total_loss

```

```

class Darknet(nn.Module):

```

```

    """YOLOv3 object detection model"""

```

```

    def __init__(self, config_path, img_size=416):

```

```

        super(Darknet, self).__init__()

```

```

        self.module_defs = parse_model_config(config_path)

```

```

        self.hyperparams, self.module_list = create_modules(self.module_defs)

```

```

        self.yolo_layers = [layer[0] for layer in self.module_list if hasattr(layer[0], "metrics")]

```

```

        self.img_size = img_size

```

```

        self.seen = 0

```

```

        self.header_info = np.array([0, 0, 0, self.seen, 0], dtype=np.int32)

```

```

    def forward(self, x, targets=None):

```

```

        img_dim = x.shape[2]

```

```

        loss = 0

```

```

        layer_outputs, yolo_outputs = [], []

```

```

        for i, (module_def, module) in enumerate(zip(self.module_defs, self.module_list)):

```

```

            if module_def["type"] in ["convolutional", "upsample", "maxpool"]:

```

```

                x = module(x)

```

```

            elif module_def["type"] == "route":

```

```

                x = torch.cat([layer_outputs[int(layer_i)] for layer_i in module_def["layers"].split(",")], 1)

```

```

            elif module_def["type"] == "shortcut":

```

```

                layer_i = int(module_def["from"])

```

```

                x = layer_outputs[-1] + layer_outputs[layer_i]

```

```

            elif module_def["type"] == "yolo":

```

```

                x, layer_loss = module[0](x, targets, img_dim)

```

```

                loss += layer_loss

```

```

                yolo_outputs.append(x)

```

```

            layer_outputs.append(x)

```

```

        yolo_outputs = to_cpu(torch.cat(yolo_outputs, 1))

```

```

        return yolo_outputs if targets is None else (loss, yolo_outputs)

```

```

    def load_darknet_weights(self, weights_path):

```

```

        """Parses and loads the weights stored in 'weights_path'"""

```

```

        # Open the weights file

```

```

        with open(weights_path, "rb") as f:

```

```

            header = np.fromfile(f, dtype=np.int32, count=5) # First five are header values

```

```

            self.header_info = header # Needed to write header when saving weights

```

```

            self.seen = header[3] # number of images seen during training

```

```

            weights = np.fromfile(f, dtype=np.float32) # The rest are weights

```

```

        # Establish cutoff for loading backbone weights

```

```

        cutoff = None

```

```

        if "darknet53.conv.74" in weights_path:

```

```

            cutoff = 75

```

```

        ptr = 0

```

```

        for i, (module_def, module) in enumerate(zip(self.module_defs, self.module_list)):

```

```

            if i == cutoff:

```

					ПГУ 2.090301.10.001 ПЗ	Лист
						92
Изм.	Лист	№ докум.	Подпись	Дата		

```

        break
    if module_def["type"] == "convolutional":
        conv_layer = module[0]
        if module_def["batch_normalize"]:
            # Load BN bias, weights, running mean and running variance
            bn_layer = module[1]
            num_b = bn_layer.bias.numel() # Number of biases
            # Bias
            bn_b = torch.from_numpy(weights[ptr : ptr + num_b]).view_as(bn_layer.bias)
            bn_layer.bias.data.copy_(bn_b)
            ptr += num_b
            # Weight
            bn_w = torch.from_numpy(weights[ptr : ptr + num_b]).view_as(bn_layer.weight)
            bn_layer.weight.data.copy_(bn_w)
            ptr += num_b
            # Running Mean
            bn_rm = torch.from_numpy(weights[ptr : ptr + num_b]).view_as(bn_layer.running_mean)
            bn_layer.running_mean.data.copy_(bn_rm)
            ptr += num_b
            # Running Var
            bn_rv = torch.from_numpy(weights[ptr : ptr + num_b]).view_as(bn_layer.running_var)
            bn_layer.running_var.data.copy_(bn_rv)
            ptr += num_b
        else:
            # Load conv. bias
            num_b = conv_layer.bias.numel()
            conv_b = torch.from_numpy(weights[ptr : ptr + num_b]).view_as(conv_layer.bias)
            conv_layer.bias.data.copy_(conv_b)
            ptr += num_b
            # Load conv. weights
            num_w = conv_layer.weight.numel()
            conv_w = torch.from_numpy(weights[ptr : ptr + num_w]).view_as(conv_layer.weight)
            conv_layer.weight.data.copy_(conv_w)
            ptr += num_w

def save_darknet_weights(self, path, cutoff=-1):
    """
    @:param path - path of the new weights file
    @:param cutoff - save layers between 0 and cutoff (cutoff = -1 -> all are saved)
    """
    fp = open(path, "wb")
    self.header_info[3] = self.seen
    self.header_info.tofile(fp)

    # Iterate through layers
    for i, (module_def, module) in enumerate(zip(self.module_defs[:cutoff], self.module_list[:cutoff])):
        if module_def["type"] == "convolutional":
            conv_layer = module[0]
            # If batch norm, load bn first
            if module_def["batch_normalize"]:
                bn_layer = module[1]
                bn_layer.bias.data.cpu().numpy().tofile(fp)
                bn_layer.weight.data.cpu().numpy().tofile(fp)
                bn_layer.running_mean.data.cpu().numpy().tofile(fp)
                bn_layer.running_var.data.cpu().numpy().tofile(fp)
            # Load conv bias
            else:

```

					ПГУ 2.090301.10.001 ПЗ	Лист
						93
Изм.	Лист	№ докум.	Подпись	Дата		

```

        conv_layer.bias.data.cpu().numpy().tofile(fp)
    # Load conv weights
    conv_layer.weight.data.cpu().numpy().tofile(fp)

fp.close()

```

Приложение А.4 – Файл «utils.py»

```

from __future__ import division
import math
import time
import tqdm
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches

def to_cpu(tensor):
    return tensor.detach().cpu()

def load_classes(path):
    """
    Loads class labels at 'path'
    """
    fp = open(path, "r")
    names = fp.read().split("\n")[:-1]
    return names

def weights_init_normal(m):
    classname = m.__class__.__name__
    if classname.find("Conv") != -1:
        torch.nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find("BatchNorm2d") != -1:
        torch.nn.init.normal_(m.weight.data, 1.0, 0.02)
        torch.nn.init.constant_(m.bias.data, 0.0)

def rescale_boxes(boxes, current_dim, original_shape):
    """ Rescales bounding boxes to the original shape """
    orig_h, orig_w = original_shape
    # The amount of padding that was added
    pad_x = max(orig_h - orig_w, 0) * (current_dim / max(original_shape))
    pad_y = max(orig_w - orig_h, 0) * (current_dim / max(original_shape))
    # Image height and width after padding is removed
    unpad_h = current_dim - pad_y
    unpad_w = current_dim - pad_x
    # Rescale bounding boxes to dimension of original image
    boxes[:, 0] = ((boxes[:, 0] - pad_x // 2) / unpad_w) * orig_w
    boxes[:, 1] = ((boxes[:, 1] - pad_y // 2) / unpad_h) * orig_h
    boxes[:, 2] = ((boxes[:, 2] - pad_x // 2) / unpad_w) * orig_w

```

					ПГУ 2.090301.10.001 ПЗ	Лист
						94
Изм.	Лист	№ докум.	Подпись	Дата		

```
boxes[:, 3] = ((boxes[:, 3] - pad_y // 2) / unpad_h) * orig_h
return boxes
```

```
def xywh2xyxy(x):
```

```
    y = x.new(x.shape)
    y[..., 0] = x[..., 0] - x[..., 2] / 2
    y[..., 1] = x[..., 1] - x[..., 3] / 2
    y[..., 2] = x[..., 0] + x[..., 2] / 2
    y[..., 3] = x[..., 1] + x[..., 3] / 2
    return y
```

```
def ap_per_class(tp, conf, pred_cls, target_cls):
```

```
    """ Compute the average precision, given the recall and precision curves.
```

```
    Source: https://github.com/rafaelpadilla/Object-Detection-Metrics.
```

```
    # Arguments
```

```
    tp: True positives (list).
```

```
    conf: Objectness value from 0-1 (list).
```

```
    pred_cls: Predicted object classes (list).
```

```
    target_cls: True object classes (list).
```

```
    # Returns
```

```
    The average precision as computed in py-faster-rcnn.
```

```
    """
```

```
    # Sort by objectness
```

```
    i = np.argsort(-conf)
```

```
    tp, conf, pred_cls = tp[i], conf[i], pred_cls[i]
```

```
    # Find unique classes
```

```
    unique_classes = np.unique(target_cls)
```

```
    # Create Precision-Recall curve and compute AP for each class
```

```
    ap, p, r = [], [], []
```

```
    for c in tqdm.tqdm(unique_classes, desc="Computing AP"):
```

```
        i = pred_cls == c
```

```
        n_gt = (target_cls == c).sum() # Number of ground truth objects
```

```
        n_p = i.sum() # Number of predicted objects
```

```
        if n_p == 0 and n_gt == 0:
```

```
            continue
```

```
        elif n_p == 0 or n_gt == 0:
```

```
            ap.append(0)
```

```
            r.append(0)
```

```
            p.append(0)
```

```
        else:
```

```
            # Accumulate FPs and TPs
```

```
            fpc = (1 - tp[i]).cumsum()
```

```
            tpc = (tp[i]).cumsum()
```

```
            # Recall
```

```
            recall_curve = tpc / (n_gt + 1e-16)
```

```
            r.append(recall_curve[-1])
```

```
            # Precision
```

```
            precision_curve = tpc / (tpc + fpc)
```

```
            p.append(precision_curve[-1])
```

					ПГУ 2.090301.10.001 ПЗ	Лист
						95
Изм.	Лист	№ докум.	Подпись	Дата		

```

        # AP from recall-precision curve
        ap.append(compute_ap(recall_curve, precision_curve))

    # Compute F1 score (harmonic mean of precision and recall)
    p, r, ap = np.array(p), np.array(r), np.array(ap)
    f1 = 2 * p * r / (p + r + 1e-16)

    return p, r, ap, f1, unique_classes.astype("int32")

def compute_ap(recall, precision):
    """ Compute the average precision, given the recall and precision curves.
    Code originally from https://github.com/rbgirshick/py-faster-rcnn.

    # Arguments
        recall: The recall curve (list).
        precision: The precision curve (list).
    # Returns
        The average precision as computed in py-faster-rcnn.
    """
    # correct AP calculation
    # first append sentinel values at the end
    mrec = np.concatenate(([0.0], recall, [1.0]))
    mpre = np.concatenate(([0.0], precision, [0.0]))

    # compute the precision envelope
    for i in range(mpre.size - 1, 0, -1):
        mpre[i - 1] = np.maximum(mpre[i - 1], mpre[i])

    # to calculate area under PR curve, look for points
    # where X axis (recall) changes value
    i = np.where(mrec[1:] != mrec[:-1])[0]

    # and sum (\Delta recall) * prec
    ap = np.sum((mrec[i + 1] - mrec[i]) * mpre[i + 1])
    return ap

def get_batch_statistics(outputs, targets, iou_threshold):
    """ Compute true positives, predicted scores and predicted labels per sample """
    batch_metrics = []
    for sample_i in range(len(outputs)):

        if outputs[sample_i] is None:
            continue

        output = outputs[sample_i]
        pred_boxes = output[:, :4]
        pred_scores = output[:, 4]
        pred_labels = output[:, -1]

        true_positives = np.zeros(pred_boxes.shape[0])

        annotations = targets[targets[:, 0] == sample_i[:, 1], 1:]
        target_labels = annotations[:, 0] if len(annotations) else []
        if len(annotations):

```

					ПГУ 2.090301.10.001 ПЗ	Лист
						96
Изм.	Лист	№ докум.	Подпись	Дата		

```

detected_boxes = []
target_boxes = annotations[:, 1:]

for pred_i, (pred_box, pred_label) in enumerate(zip(pred_boxes, pred_labels)):

    # If targets are found break
    if len(detected_boxes) == len(annotations):
        break

    # Ignore if label is not one of the target labels
    if pred_label not in target_labels:
        continue

    iou, box_index = bbox_iou(pred_box.unsqueeze(0), target_boxes).max(0)
    if iou >= iou_threshold and box_index not in detected_boxes:
        true_positives[pred_i] = 1
        detected_boxes += [box_index]
    batch_metrics.append([true_positives, pred_scores, pred_labels])
return batch_metrics

```

```

def bbox_wh_iou(wh1, wh2):
    wh2 = wh2.t()
    w1, h1 = wh1[0], wh1[1]
    w2, h2 = wh2[0], wh2[1]
    inter_area = torch.min(w1, w2) * torch.min(h1, h2)
    union_area = (w1 * h1 + 1e-16) + w2 * h2 - inter_area
    return inter_area / union_area

```

```

def bbox_iou(box1, box2, x1y1x2y2=True):
    """
    Returns the IoU of two bounding boxes
    """
    if not x1y1x2y2:
        # Transform from center and width to exact coordinates
        b1_x1, b1_x2 = box1[:, 0] - box1[:, 2] / 2, box1[:, 0] + box1[:, 2] / 2
        b1_y1, b1_y2 = box1[:, 1] - box1[:, 3] / 2, box1[:, 1] + box1[:, 3] / 2
        b2_x1, b2_x2 = box2[:, 0] - box2[:, 2] / 2, box2[:, 0] + box2[:, 2] / 2
        b2_y1, b2_y2 = box2[:, 1] - box2[:, 3] / 2, box2[:, 1] + box2[:, 3] / 2
    else:
        # Get the coordinates of bounding boxes
        b1_x1, b1_y1, b1_x2, b1_y2 = box1[:, 0], box1[:, 1], box1[:, 2], box1[:, 3]
        b2_x1, b2_y1, b2_x2, b2_y2 = box2[:, 0], box2[:, 1], box2[:, 2], box2[:, 3]

    # get the corrdinates of the intersection rectangle
    inter_rect_x1 = torch.max(b1_x1, b2_x1)
    inter_rect_y1 = torch.max(b1_y1, b2_y1)
    inter_rect_x2 = torch.min(b1_x2, b2_x2)
    inter_rect_y2 = torch.min(b1_y2, b2_y2)
    # Intersection area
    inter_area = torch.clamp(inter_rect_x2 - inter_rect_x1 + 1, min=0) * torch.clamp(
        inter_rect_y2 - inter_rect_y1 + 1, min=0
    )
    # Union Area
    b1_area = (b1_x2 - b1_x1 + 1) * (b1_y2 - b1_y1 + 1)
    b2_area = (b2_x2 - b2_x1 + 1) * (b2_y2 - b2_y1 + 1)

```

```
iou = inter_area / (b1_area + b2_area - inter_area + 1e-16)
```

```
return iou
```

```
def non_max_suppression(prediction, conf_thres=0.5, nms_thres=0.4):
```

```
    """
```

```
    Removes detections with lower object confidence score than 'conf_thres' and performs  
    Non-Maximum Suppression to further filter detections.
```

```
    Returns detections with shape:
```

```
        (x1, y1, x2, y2, object_conf, class_score, class_pred)
```

```
    """
```

```
    # From (center x, center y, width, height) to (x1, y1, x2, y2)
```

```
    prediction[..., :4] = xywh2xyxy(prediction[..., :4])
```

```
    output = [None for _ in range(len(prediction))]
```

```
    for image_i, image_pred in enumerate(prediction):
```

```
        # Filter out confidence scores below threshold
```

```
        image_pred = image_pred[image_pred[:, 4] >= conf_thres]
```

```
        # If none are remaining => process next image
```

```
        if not image_pred.size(0):
```

```
            continue
```

```
        # Object confidence times class confidence
```

```
        score = image_pred[:, 4] * image_pred[:, 5:].max(1)[0]
```

```
        # Sort by it
```

```
        image_pred = image_pred[(-score).argsort()]
```

```
        class_confs, class_preds = image_pred[:, 5:].max(1, keepdim=True)
```

```
        detections = torch.cat((image_pred[:, :5], class_confs.float(), class_preds.float()), 1)
```

```
        # Perform non-maximum suppression
```

```
        keep_boxes = []
```

```
        while detections.size(0):
```

```
            large_overlap = bbox_iou(detections[0, :4].unsqueeze(0), detections[:, :4]) > nms_thres
```

```
            label_match = detections[0, -1] == detections[:, -1]
```

```
            # Indices of boxes with lower confidence scores, large IOUs and matching labels
```

```
            invalid = large_overlap & label_match
```

```
            weights = detections[invalid, 4:5]
```

```
            # Merge overlapping bboxes by order of confidence
```

```
            detections[0, :4] = (weights * detections[invalid, :4]).sum(0) / weights.sum()
```

```
            keep_boxes += [detections[0]]
```

```
            detections = detections[~invalid]
```

```
        if keep_boxes:
```

```
            output[image_i] = torch.stack(keep_boxes)
```

```
    return output
```

```
def linear_assignment(X):
```

```
    """Solve the linear assignment problem using the Hungarian algorithm.
```

```
    The problem is also known as maximum weight matching in bipartite graphs.
```

```
    The method is also known as the Munkres or Kuhn-Munkres algorithm.
```

```
    Parameters
```

```
    -----
```

```
    X : array
```

```
        The cost matrix of the bipartite graph
```

					ПГУ 2.090301.10.001 ПЗ	Лист
						98
Изм.	Лист	№ докум.	Подпись	Дата		

Returns

indices : array

The pairs of (row, col) indices in the original array giving the original ordering.

"""

```
indices = _hungarian(X).tolist()
indices.sort()
# Re-force dtype to ints in case of empty list
indices = np.array(indices, dtype=int)
# Make sure the array is 2D with 2 columns.
# This is needed when dealing with an empty list
indices.shape = (-1, 2)
return indices
```

```
def build_targets(pred_boxes, pred_cls, target, anchors, ignore_thres):
```

```
ByteTensor = torch.cuda.ByteTensor if pred_boxes.is_cuda else torch.ByteTensor
FloatTensor = torch.cuda.FloatTensor if pred_boxes.is_cuda else torch.FloatTensor
```

```
nB = pred_boxes.size(0)
nA = pred_boxes.size(1)
nC = pred_cls.size(-1)
nG = pred_boxes.size(2)
```

```
# Output tensors
obj_mask = ByteTensor(nB, nA, nG, nG).fill_(0)
noobj_mask = ByteTensor(nB, nA, nG, nG).fill_(1)
class_mask = FloatTensor(nB, nA, nG, nG).fill_(0)
iou_scores = FloatTensor(nB, nA, nG, nG).fill_(0)
tx = FloatTensor(nB, nA, nG, nG).fill_(0)
ty = FloatTensor(nB, nA, nG, nG).fill_(0)
tw = FloatTensor(nB, nA, nG, nG).fill_(0)
th = FloatTensor(nB, nA, nG, nG).fill_(0)
tcls = FloatTensor(nB, nA, nG, nG, nC).fill_(0)
```

```
# Convert to position relative to box
target_boxes = target[:, 2:6] * nG
gxy = target_boxes[:, :2]
gwh = target_boxes[:, 2:]
# Get anchors with best iou
ious = torch.stack([bbox_wh_iou(anchor, gwh) for anchor in anchors])
best_ious, best_n = ious.max(0)
# Separate target values
b, target_labels = target[:, :2].long().t()
gx, gy = gxy.t()
gw, gh = gwh.t()
gi, gj = gxy.long().t()
# Set masks
obj_mask[b, best_n, gj, gi] = 1
noobj_mask[b, best_n, gj, gi] = 0
```

```
# Set noobj mask to zero where iou exceeds ignore threshold
for i, anchor_ious in enumerate(ious.t()):
```

					ПГУ 2.090301.10.001 ПЗ	Лист
						99
Изм.	Лист	№ докум.	Подпись	Дата		

```

noobj_mask[b[i], anchor_iou > ignore_thres, gj[i], gi[i]] = 0

# Coordinates
tx[b, best_n, gj, gi] = gx - gx.floor()
ty[b, best_n, gj, gi] = gy - gy.floor()
# Width and height
tw[b, best_n, gj, gi] = torch.log(gw / anchors[best_n][:, 0] + 1e-16)
th[b, best_n, gj, gi] = torch.log(gh / anchors[best_n][:, 1] + 1e-16)
# One-hot encoding of label
tcls[b, best_n, gj, gi, target_labels] = 1
# Compute label correctness and iou at best anchor
class_mask[b, best_n, gj, gi] = (pred_cls[b, best_n, gj, gi].argmax(-1) == target_labels).float()
iou_scores[b, best_n, gj, gi] = bbox_iou(pred_boxes[b, best_n, gj, gi], target_boxes, x1y1x2y2=False)

tconf = obj_mask.float()
return iou_scores, class_mask, obj_mask, noobj_mask, tx, ty, tw, th, tcls, tconf

```

Приложение А.5 – Файл «torch_utils.py»

```

import torch

def init_seeds(seed=0):
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)

def select_device(force_cpu=False):
    cuda = False if force_cpu else torch.cuda.is_available()
    device = torch.device('cuda:0' if cuda else 'cpu')

    if not cuda:
        print('Using CPU')
    if cuda:
        c = 1024 ** 2 # bytes to MB
        ng = torch.cuda.device_count()
        x = [torch.cuda.get_device_properties(i) for i in range(ng)]
        print("Using CUDA device0 _CudaDeviceProperties(name='%s', total_memory=%dMB)" %
              (x[0].name, x[0].total_memory / c))
        if ng > 0:
            # torch.cuda.set_device(0) # OPTIONAL: Set GPU ID
            for i in range(1, ng):
                print("      device%g _CudaDeviceProperties(name='%s', total_memory=%dMB)" %
                      (i, x[i].name, x[i].total_memory / c))

    print("") # skip a line
    return device

def fuse_conv_and_bn(conv, bn):
    # https://tehnokv.com/posts/fusing-batchnorm-and-conv/
    with torch.no_grad():
        # init

```

					ПГУ 2.090301.10.001 ПЗ	Лист
						100
Изм.	Лист	№ докум.	Подпись	Дата		

```

fusedconv = torch.nn.Conv2d(
    conv.in_channels,
    conv.out_channels,
    kernel_size=conv.kernel_size,
    stride=conv.stride,
    padding=conv.padding,
    bias=True
)

# prepare filters
w_conv = conv.weight.clone().view(conv.out_channels, -1)
w_bn = torch.diag(bn.weight.div(torch.sqrt(bn.eps + bn.running_var)))
fusedconv.weight.copy_(torch.mm(w_bn, w_conv).view(fusedconv.weight.size()))

# prepare spatial bias
if conv.bias is not None:
    b_conv = conv.bias
else:
    b_conv = torch.zeros(conv.weight.size(0))
b_bn = bn.bias - bn.weight.mul(bn.running_mean).div(torch.sqrt(bn.running_var + bn.eps))
fusedconv.bias.copy_(b_conv + b_bn)

return fusedconv

```

Приложение А.6 – Файл «parse_config.py»

```

def parse_model_config(path):
    """Parses the yolo-v3 layer configuration file and returns module definitions"""
    file = open(path, 'r')
    lines = file.read().split('\n')
    lines = [x for x in lines if x and not x.startswith('#')]
    lines = [x.rstrip().lstrip() for x in lines] # get rid of fringe whitespaces
    module_defs = []
    for line in lines:
        if line.startswith('['): # This marks the start of a new block
            module_defs.append({})
            module_defs[-1]['type'] = line[1:-1].rstrip()
            if module_defs[-1]['type'] == 'convolutional':
                module_defs[-1]['batch_normalize'] = 0
            else:
                key, value = line.split("=")
                value = value.strip()
                module_defs[-1][key.rstrip()] = value.strip()

    return module_defs

def parse_data_config(path):
    """Parses the data configuration file"""
    options = dict()
    options['gpus'] = '0,1,2,3'
    options['num_workers'] = '10'

```

					ПГУ 2.090301.10.001 ПЗ	Лист
						101
Изм.	Лист	№ докум.	Подпись	Дата		

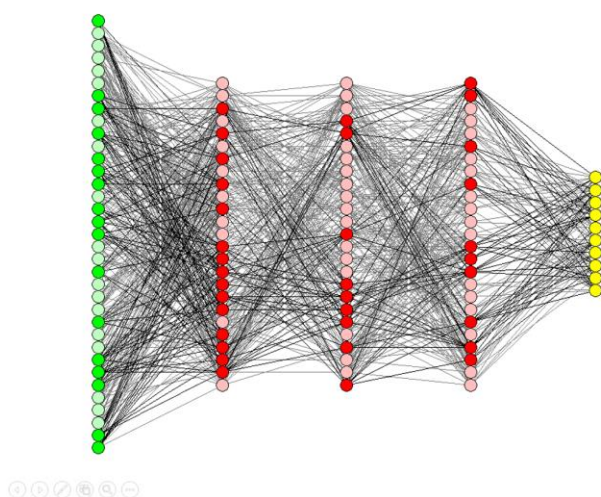
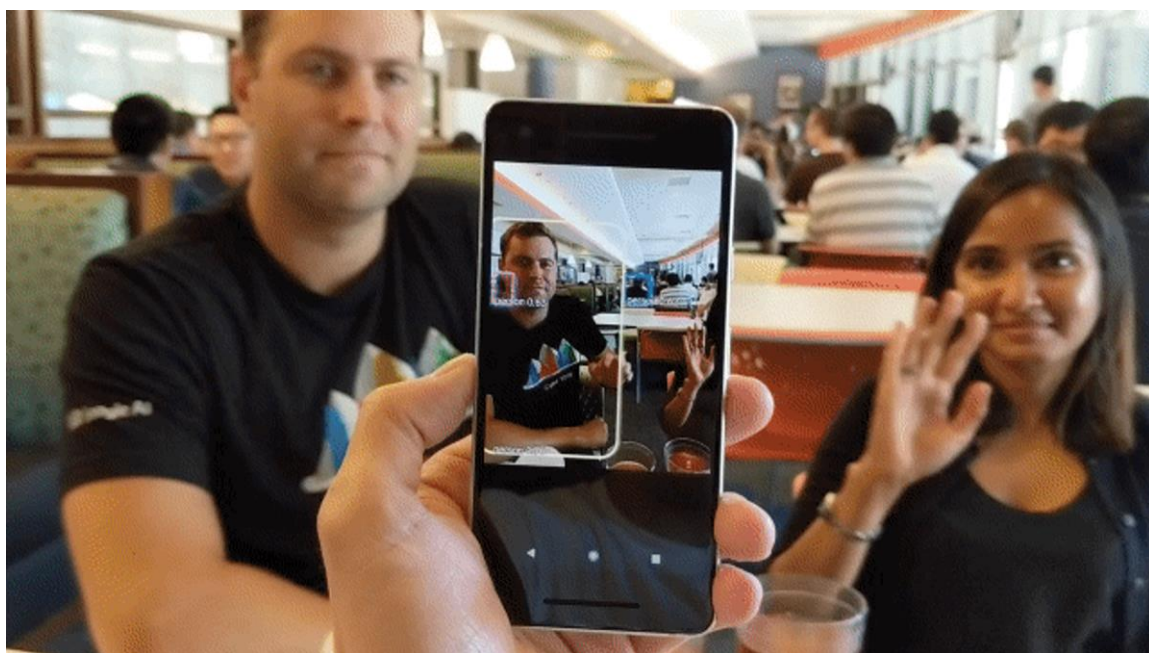
```

with open(path, 'r') as fp:
    lines = fp.readlines()
for line in lines:
    line = line.strip()
    if line == "" or line.startswith('#'):
        continue
    key, value = line.split('=')
    options[key.strip()] = value.strip()
return options

```

ПРИЛОЖЕНИЕ Б: ПРЕЗЕНТАЦИЯ

Глубокая нейросеть

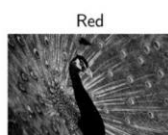
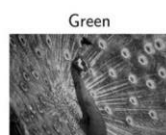
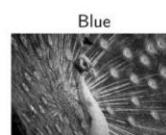
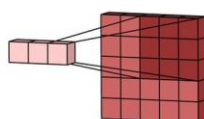
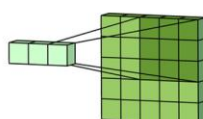
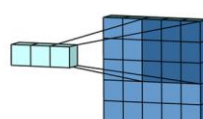
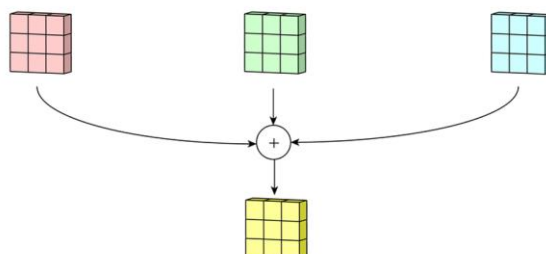


Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.10.001 ПЗ

Лист

102



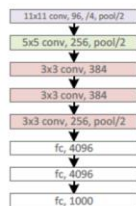
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.10.001 ПЗ

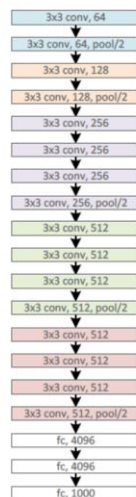
Лист

103

AlexNet, 8 layers
(ILSVRC 2012)

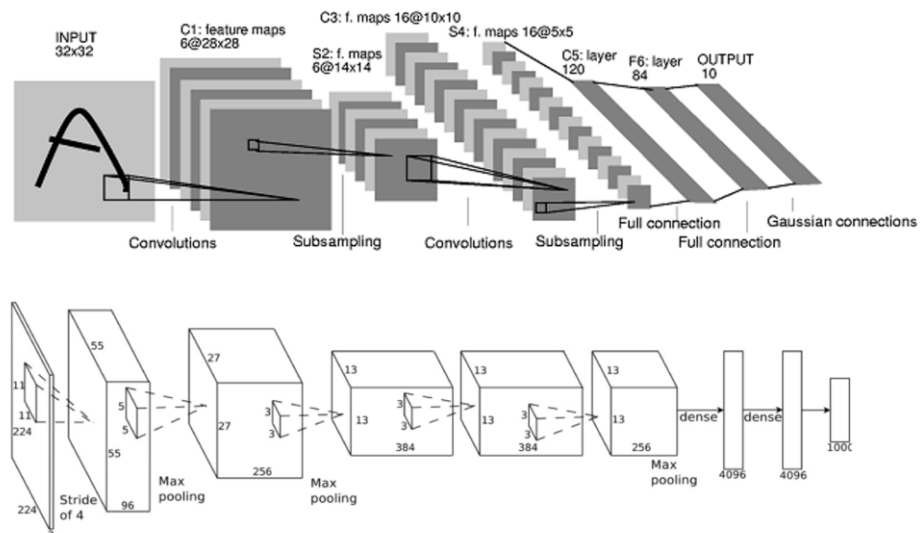


VGG, 19 layers
(ILSVRC 2014)



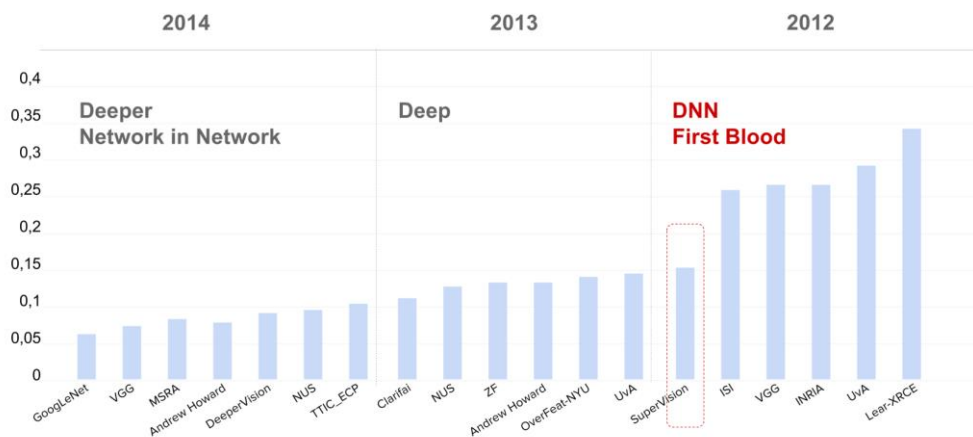
GoogleNet, 22 layers
(ILSVRC 2014)

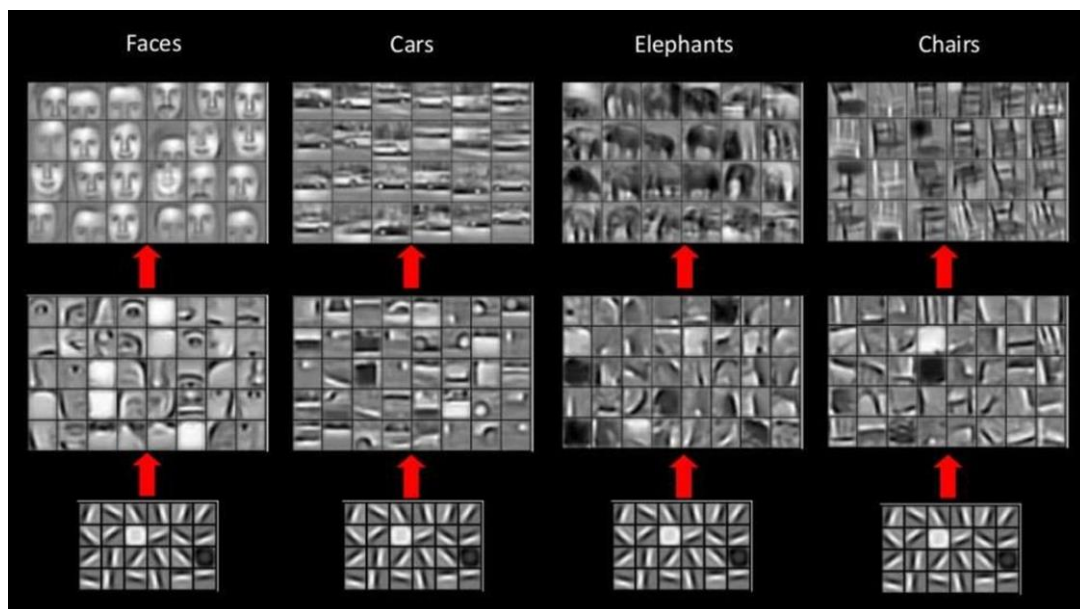




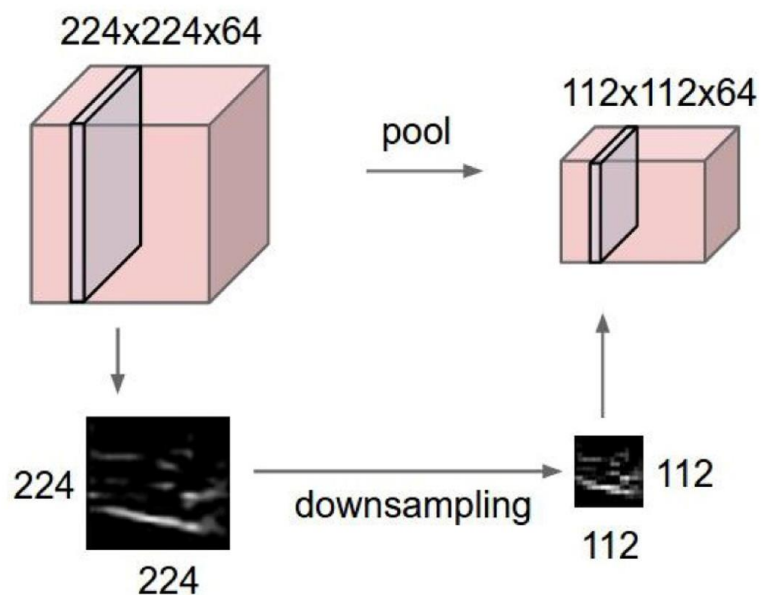
ILSVRC

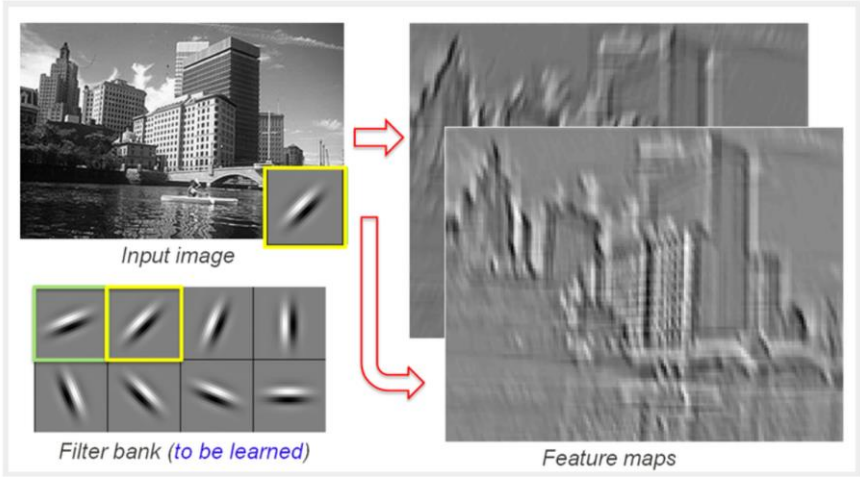
ImageNet Classification error throughout years and groups



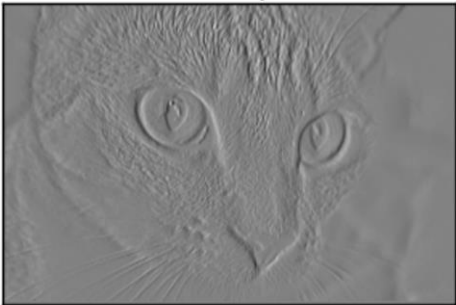


CNN: Pooling слой (downsampling)

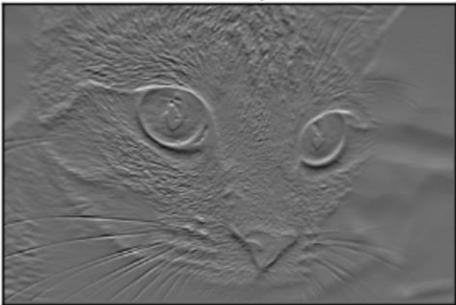


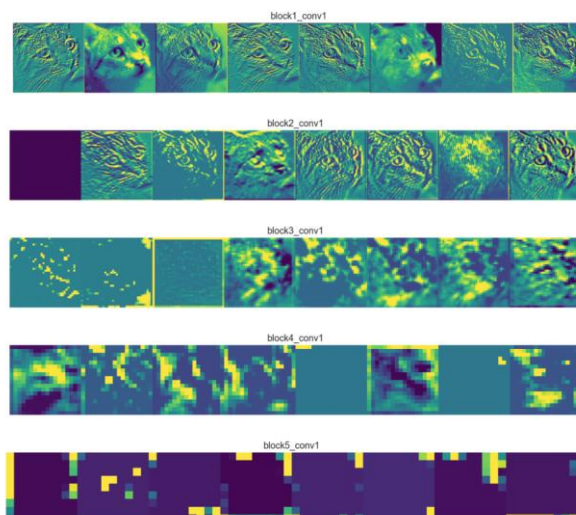
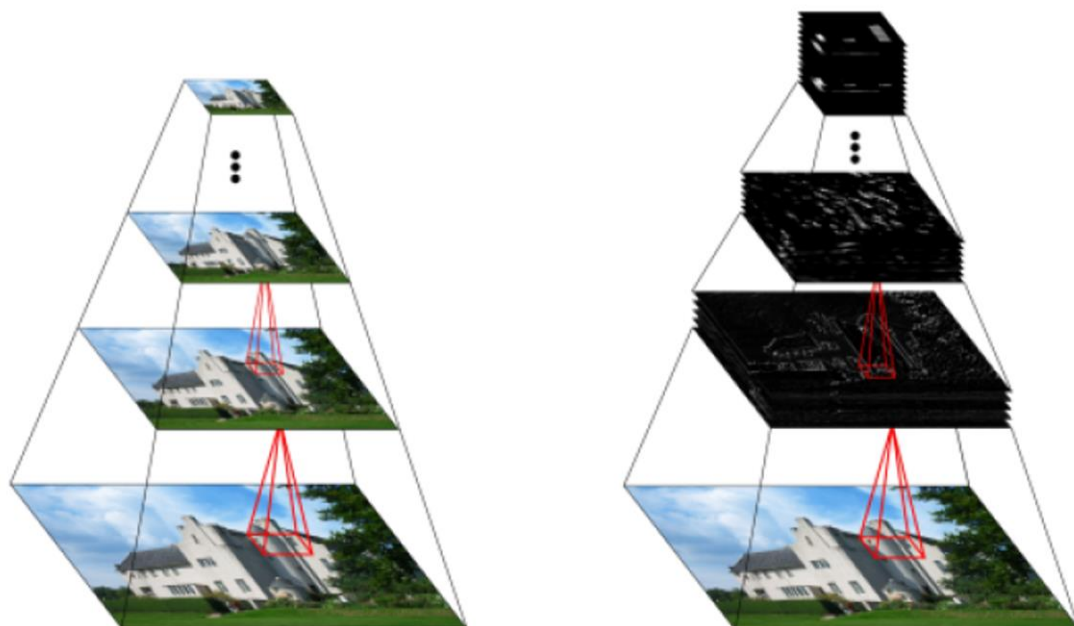


L1-Map1



L1-Map2





Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.10.001 ПЗ

Лист

108

Пример свертки с трехмерным ядром RGB

