

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего
образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

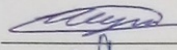
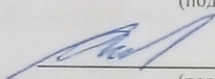
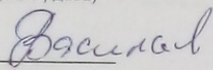
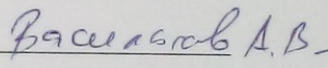
Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

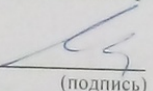
МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

ИСПОЛЬЗОВАНИЕ ИСКУССТВЕННЫХ НЕЙРОННЫХ
СЕТЕЙ ДЛЯ РАСПОЗНАВАНИЯ МАЛОГАБАРИТНЫХ БПЛА

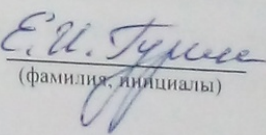
Студент	 (подпись, дата) 07.06.2024	Шумкин Иван Сергеевич (ФИО полностью)
Руководитель	 (подпись, дата) 07.06.2024	Козлов А.Ю. (фамилия, инициалы)
Нормоконтролёр	 (подпись, дата) 07.06.2024	Бычков А.С. (фамилия, инициалы)
Рецензент	2.1. специальность «Информатика и вычислительная техника» АО «ИПП» РЗБЧ  (должность, место работы) (подпись, дата)	 (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 г. № 11)

Заведующий кафедрой 
(подпись) Митрохин М.А.
(фамилия, инициалы)

Работа защищена с отметкой Отлично (протокол заседания ГЭК
от 14.06.2024 № 20)

Секретарь ГЭК 
(подпись)


(фамилия, инициалы)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин

« 13 » сентября 2022 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент

гр. 22ВВИм2, факультета ВТ, Шумкин Иван Сергеевич.

2. Тема работы

«Использование искусственных нейронных сетей для распознавания малогабаритных БПЛА».

Тема утверждена приказом ПГУ № 765/о от 13.09.2022 г.

3. Руководитель магистерской диссертации.

д.т.н., профессор, зав. кафедрой МОиПЭВМ А.Ю. Козлов.

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

Входные данные: изображения в формате JPG, в цветовой модели RGB, с разрешением не превышающем 2048 пикселей;

Выходные данные: в зависимости от задачи распознавания: классификация — изображения, идентичные входным с присвоением метки класса «БПЛА» или «Птица».

Назначение разработки: осуществление распознавания объектов воздушного пространства на изображениях путем классификации входных изображений, подающихся на вход обученной модели нейронной сети.

Разработать архитектуру модели нейронной сети для задачи распознавания объектов.

В качестве основы архитектуры модели нейронной сети использовать фреймворки LeNet5, RetinaNet или аналоги.

5. Перечень подлежащих разработке вопросов.

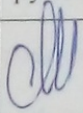
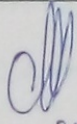
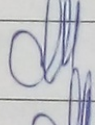
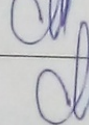
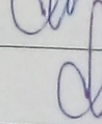
1. Анализ предметной области искусственных нейронных сетей и возможностей их использования для распознавания изображений.

2. Разработка собственной архитектуры нейронной сети, либо разработка на одном из фреймворков LeNet5, RetinaNet или аналогах для задачи распознавания малогабаритных беспилотных летательных аппаратов.

3. Обучение модели нейронной сети и получение метрик качества — точности и ошибки модели в результате проверки качества распознавания на валидационной выборке — изображений, не участвующих при обучении модели и составляющей не менее 25% от общего количества изображений. Минимальная точность распознавания — не менее 80%.

4. Тестирование качества распознавания изображений обученной моделью нейронной сети с помощью изображений, смазанных ядром Гаусса различной размерности.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области искусственных нейронных сетей	10%	13.09.2022-21.11.2022	
2.	Анализ применимости искусственных нейронных сетей для задачи распознавания изображений	10%	22.11.2022-22.01.2023	
3.	Разработка архитектуры нейронной сети	30%	23.01.2023-26.06.2023	
4.	Обучение модели нейронной сети	30%	26.06.2023-29.01.2024	
5.	Тестирование модели нейронной сети	10%	30.01.2024-30.03.2024	
6.	Оформление выпускной работы	10%	31.03.2024-31.05.2024	

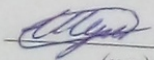
Дата выдачи задания "13" сентября 2022г.

Руководитель магистерской диссертации



(подпись, дата)

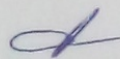
Задание к исполнению принял студент



13.09.2022г

(подпись, дата)

Работу к защите допустить
Декан факультета



А.Р. Рогов

(подпись, дата)

13.09.2022г

РЕФЕРАТ

Пояснительная записка содержит 77 л., 41 рис., 14 источ., 2 прил.

**НЕЙРОННЫЕ СЕТИ, КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ,
КОМПЬЮТЕРНОЕ ЗРЕНИЕ, ОБУЧЕНИЕ МОДЕЛИ, ТОЧНОСТЬ ОБУЧЕНИЯ**

Объект исследования - нейронные сети для задач распознавания изображений.

Предмет исследования - процесс предварительной обработки и классификации изображений, используемый в алгоритмах распознавания, а также влияние этого процесса на точность и скорость получения конечного результата распознавания (обучаемой модели).

Цель работы - исследование и разработка модели нейронной сети для классификации изображений.

Задачи, решаемые в работе:

- исследование и анализ видов нейронных сетей, применяемых для задач распознавания изображений,
- разработка архитектуры модели нейронной сети для задачи классификации изображений,
- обучение нейронной сети на данных и получение метрики точности модели; а также для уменьшения скорости обучения применить технологию вычислений на графическом процессоре,
- тестирование обученной модели нейронной сети.

<h3 style="margin: 0;">ПГУ1.09.04.01.48.001 ПЗ</h3>								
Изм.	Лист	№ докум.	Подпись	Дата	<p>Использование искусственных нейронных сетей для распознавания малогобаритных БПЛА</p> <p>Пояснительная записка</p>	Лит.	Лист	Листов
Разраб.		Шумкин И.С.		7.6.2014				
Пров.		Козлов А.Ю.		7.6.2014			2	77
Н. Контр.		Бычков А.С.		7.06.2014				
Утв.		Митрахин М.А.		7.06.2014				
						<p>ФВТ, каф. ВТ</p> <p>гр. 22ВВИм2</p>		

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ.....	5
ВВЕДЕНИЕ.....	6
1 Нейронные сети как средство для распознавания образов	8
1.1 Анализ принципов построения, классификации и функционирования отдельных видов искусственных нейронных сетей.....	8
1.1.1 Принципы структурирования и функционирования искусственного нейрона.....	9
1.2 Основные свойства нейронных сетей, выбор и обоснование архитектуры искусственной нейронной сети.....	11
1.2.1 Алгоритм обратного распространения ошибки.....	21
1.3 Обзор существующих нейронных сетей, используемых для распознавания изображений.....	21
1.3.1 Сверточные нейронные сети.....	22
1.3.2 Глубокие сверточные обратные графические нейронные сети.....	23
1.3.3 Деконволюционные нейронные сети.....	26
1.4 Выводы по разделу 1.....	28
2 Метод классификации для распознавания образов.....	29
2.1 Применение метода классификации для распознавания изображений.....	30
2.1.1 Типы классификации изображений.....	31
2.1.2 Применение сверточных нейронных сетей для классификации изображений.....	33
2.2 Постановка задачи.....	38
2.3. Экспериментальная часть.....	40
2.3.1 Обучение нейронной сети.....	42
2.3.2 Обучение сети с измененной архитектурой.....	42
2.3.3 Обучение сети с измененной активационной функцией.....	45

2.3.4 Обучение сети с измененной скоростью обучения.....	46
2.3.5 Увеличение выборки.....	47
2.3.6 Построение графиков точности и ошибки.....	49
2.4 Виды тестирования обученной модели нейронной сети.....	50
2.4.1 Тестирование нейронной сети на устойчивость.....	53
2.4.2 Тестирование нейронной сети на минимальное возмущение.....	54
2.4.3 Результаты тестирования.....	54
2.5 Выводы по разделу 2.....	58
ЗАКЛЮЧЕНИЕ.....	59
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	60
ПРИЛОЖЕНИЕ А Листинг программы.....	62
ПРИЛОЖЕНИЕ Б Презентация.....	74

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей работе применяют следующие сокращения и обозначения

МНСПР – Многослойная нейронная сеть прямого распространения

НС – Нейронная сеть

ОЗУ – Оперативное запоминающее устройство

СНС – Сверточная нейронная сеть

CNN (Convolution neural network) – Сверточная нейронная сеть

DCIGN (Deep convolution inverse graphics network) – Глубокая сверточная обратная графическая сеть

DNN (Deconvolution neural network) – Развертывающая нейронная сеть

IDE (Integrated development environment) – Интегрированная среда разработки

ML (Machine learning) – Машинное обучение

ВВЕДЕНИЕ

В современное время повышается необходимость в разработке систем, которые могут не только действовать в рамках однажды запрограммированного алгоритма, но и способны самостоятельно анализировать информацию, которая поступает непосредственно в какой-либо отрезок времени. А также находить в ней необходимые закономерности, прогнозировать дальнейшее поведение потока анализируемой информации, распознавать какие-либо данные, поступающие на вход данной системы. В этой области приложений одним из лучших вариантов исполнения зарекомендовали себя искусственные нейронные сети, создаваемые посредством кода, возможно на базе различных фреймворков (framework). Нейронные сети построены на имитации сети нейронов человеческого мозга [1], поэтому их относят к разработкам в области искусственного интеллекта. За последние несколько десятилетий в мире науки и техники сформировалась новая область математических приложений, основанных на искусственных нейронных сетях и искусственном интеллекте.

Актуальность исследований и разработок в данной области подтверждается большим количеством различных применений нейронных сетей. Можно уже с уверенностью заявить, что в современных условиях ведение боя давно выходит за рамки понимания этого процесса в классическом смысле. Все чаще можно заметить, что образцы вооруженной техники снабжаются элементами искусственного интеллекта, позволяющие отслеживать и контролировать некоторые процессы, происходящие в достигаемой видимости данных образцов. Все чаще производится продукция, снабженная устройствами и приборами, позволяющими оператору данной техники переложить необходимые действия искусственному интеллекту, автоматизирующему деятельность данного оператора при выполнении определенных задач. Элементы автоматизированного управления позволяют решать уже более интеллектуальные задачи, способствующие принятию верного решения,

причем существует масса исследований, доказывающих, что искусственный интеллект в ряде случаев допускает меньше ошибок при обработке данных, чем мозг обычного человека. Данные элементы способствуют дополнению деятельности человека, который управляет современной техникой. В частности, такой род деятельности представляет собой наблюдение за полем боя и обнаружение целей. Данная работа будет посвящена применению искусственных нейронных сетей при решении ею задачи распознавания.

Объект исследования - нейронные сети для задач распознавания изображений.

Предмет исследования – процесс предварительной обработки и классификации изображений, используемый в алгоритмах распознавания, а также влияние этого процесса на точность и скорость получения конечного результата распознавания (обучаемой модели).

Цель работы – исследование и разработка модели нейронной сети для классификации изображений.

Задачи, решаемые в работе:

- исследование и анализ видов нейронных сетей, применяемых для задач распознавания изображений,
- разработка архитектуры модели нейронной сети для задачи классификации изображений,
- обучение нейронной сети на данных и получение метрики точности модели - количественной метрики; а также для уменьшения скорости обучения применить технология вычислений на графическом процессоре,
- тестирование результатов распознавания обученной моделью.

Методология и методы исследования:

- методы искусственного интеллекта,
- методы цифровой обработки и распознавания изображений,
- метод классификации.

1 Нейронные сети как средство для распознавания образов

1.1 Анализ принципов построения, классификации и функционирования отдельных видов искусственных нейронных сетей

Рассуждая о нейронных сетях, можно начать с появления науки, которая задала начало развития изучения нейронов и способов применения сетей на их основе в технических системах. Эта наука называлась «нейрокибернетика» и первый период её развития был в конце 40-х годов прошлого столетия. В то время в мире науки появились работы Розенблатта [3], Эшби, Винера и многих других, которые предпринимали попытки на основе теории о строении мозга и входящих в него нейронов, а также различных живых организмов построить технические устройства, похожие на них по функционалу. Но так как в то далекое время еще не существовало необходимой технической и элементной базы для продвижения данной науки, трудности, которые были связаны с обучением и исследованием все более сложных нейронных структур и сетей, замедлили воплощение данных идей в реальность. В какой-то мере даже мнения ученых Пейперта и Минского, критиковавших идеи с применением искусственных перцептронов, также замедлили прогресс в данном направлении на многие годы.

Бум интереса к нейронным сетям был начат после публикации работы Хопфилда в 1982 году. Наиболее подробно обсуждаемая в технической литературе работа сети Хопфилда заключается в релаксации первоначального «спинового» портрета в матрице двоичных кодов к одному из стационарных состояний, которое определяется правилом обучения (правилом Хебба).

Далее, в 1986 году появилась работа Румельхарта, Хинтона и Вильямса, в которой представлялся ответ на проблему, которая долгое время не давала развитию нейроинформатики должного прогресса — как возможно обучение иерархических слоистых нейронных сетей, для которых пионерами

нейрокибернетики еще в 40-50-х годах прошлого столетия была доказана универсальность для большого класса задач. В последующие годы алгоритм обратного распространения ошибки, предложенный Хинтоном, приобрел огромное количество модификаций и вариаций.

Новые подходы, предложенные для решения поставленных задач позволили применять математическую физику для исследования структур, основанных на искусственных нейронах. Работы Кохоннена и Гроссенберга позволили прилично расширить взгляды на возможности искусственных нейронных сетей в области их применения.

Наличие мощной элементной базы вкупе с взаимодействием науки, в том числе математических наук продвинуло реализацию и применение на практике искусственных нейронных сетей в различных сферах деятельности.

1.1.1 Принципы структурирования и функционирования искусственного нейрона

Искусственные нейроны связываются друг с другом, образуя слои, которые передают и обрабатывают сигналы. Каждый нейрон принимает входные данные, вычисляет взвешенную сумму сигналов и применяет активационную функцию для определения своего выходного значения. В итоге, нейронные сети могут обучаться на основе больших объемов данных и выполнять сложные вычисления, эмулируя работу человеческого мозга.

Внешний вид нейрона можно представить схематически, с телом (сомой) 1 в центре, аксонами 2, аксонным холмиком 3, дендритами 4 и синаптической бляшкой 5. Схематическое изображение нейрона приведено на рисунке 1.1.

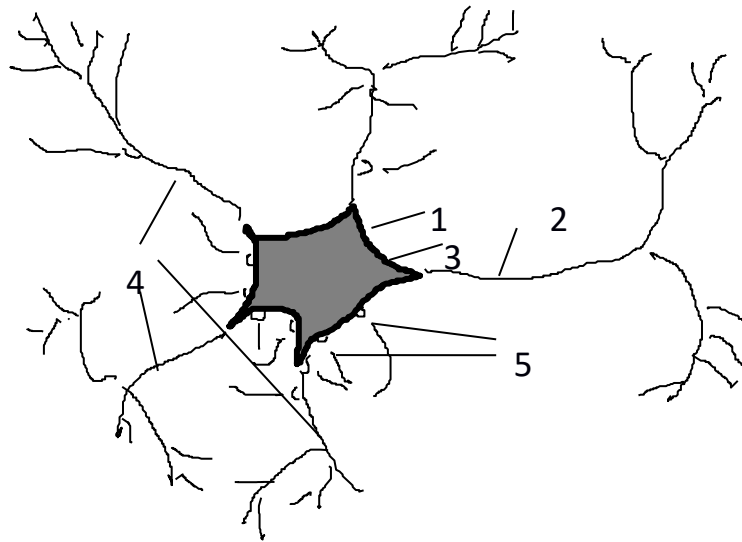


Рисунок 1.1 — Схема нейрона в человеческом мозге

Для моделирования нейронных сетей обычно используется искусственный нейрон, изображенный на рисунке 1.2. На этот искусственный нейрон поступает набор входных сигналов $x_1, x_2, \dots, x_m$ (или входной вектор X), которые содержат выходные сигналы других нейронов. Данная модель представляет входной вектор как функцию сигналов, поступающих в синапсы биологических нейронов. Каждый из этих сигналов умножается на соответствующий весовой коэффициент $w_1, w_2, \dots, w_m$.

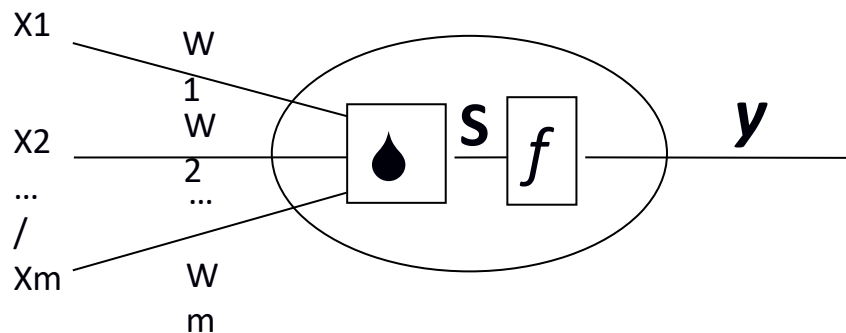


Рисунок 1.2 – Структура нейрона

Взвешенные сигналы, определенные весами связей, поступают в блок суммирования, соответствующий телу клетки, где алгебраически складываются и фиксируется уровень возбуждения нейрона S :

$$S = \sum_{i=1}^{M_{in}} w_i x_i, \quad (1.1)$$

где x_i - набор входных сигналов, w_i - весовые коэффициенты, M – общее количество сигналов.

Активационная функция f - это нелинейная функция, представленная различными вариантами, как показано на рисунке 1.3.

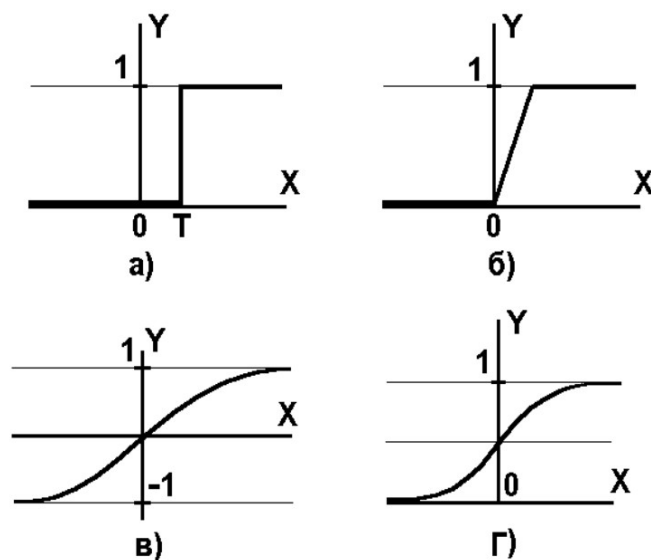


Рисунок 1.3 – Логистические функции

На рисунке показаны: а) функция единичного скачка; б) линейный порог; в) гиперболический тангенс; г) сигмоид.

1.2 Основные свойства нейронных сетей, выбор и обоснование архитектуры искусственной нейронной сети

Анализ возможностей функций биологических нейронных структур, которые наделены способностью за короткий промежуток времени обрабатывать поступающие сигналы при сравнительно медленном распространении сигналов в аксоне нейронов, а также сохранять в себе большой объем данных, ассоциативно восстанавливать их, преследует общую цель, которая состоит в поиске модели нейрона и структуры связей в нейросети, которые бы обеспечивали реализацию заданных свойств при решении необходимой поставленной задачи.

Если упростить классификацию нейросетей на основе физиологических и

биологических данных, структуры естественных нейросетей можно разделить на два типа:

- естественные нейросети с послойной организацией со структурой экранного типа. Распространение сигнала в них происходит от предшествующего слоя нейронов к последующему,

- естественные нейросети со структурой типа «ядро». Закономерности связей между нейронами внутри ядра состоят в том, что проекция ядерной структуры на структуру экранного типа в большинстве случаев обычно находится по проекции предыдущей экранной структуры на ядро.

При составлении модели одного отдельного нейрона известны попытки смоделировать все происходящие в нём процессы. При этом общего конкретного мнения о том, какие из свойств, принадлежащих реальным нейронам, должны найти реализацию в обрабатывающем элементе искусственной нейронной сети, в настоящее время нет. При моделировании естественных нейронных сетей, в которых применяются достаточно сложные элементы структуры, за рамками исследований обычно остаётся открытым также вопрос о том, в какой степени наблюдаемые эффекты являются наиболее важными при аналитическом проектировании нейросетей, и возможно ли достижение цели проектирования на иных моделях и без реализации конкретных свойств.

Анализ результатов последних научных разработок и практических реализаций в области искусственного интеллекта показывает, что исходя из требований простоты модели при проектировании модели нейронной сети для задачи распознавания, можно ограничиться структурой нейронной экранного типа с элементами, которые применяют статическое преобразование.

Получаемая таким образом структура относится к нейронным сетям коннекционистского типа (КНС). В такого рода нейронных сетях основные свойства модели определяются свойствами не отдельных нейронов, но за счёт

синаптических связей, которыми они соединены с огромным количеством соседних нейронов.

Искусственные нейронные сети проектировались на основе своих биологических прототипов, поэтому они обладают подобной структурой и имеют основные свойства естественных нейронных сетей. Они характеризуются тем, что в их структуре имеется большое количество простых элементов (нейронов), в которых происходит нелинейное преобразование сигнала, и большое число взаимосвязей между ними. Каждая взаимосвязь имеет свою функцию преобразования сигнала. Вес связи — это величина коэффициента усиления.

Структура искусственной нейронной сети имеет множество внутренних параметров, определяющих её необходимое функционирование. Обучение нейронной сети — это процесс изменения данных параметров, а закон, характеризующий величину их изменения, называется алгоритмом обучения.

В современное время существует множество различных моделей нейронных сетей, которые достаточно сильно пересекаются как по структурным, так и по функциональным особенностям. Приведённая в данной работе классификация существующих моделей НС основана на их разделении по структурным особенностям, которые являются наиболее конкретными и наглядными. Классификация нейронных сетей представлена по тексту ниже.

Нейронные сети можно классифицировать по следующим признакам:

– по структуре:

- 1) многослойные нейронные сети прямого распространения;
- 2) нейронные сети с обратными связями;
- 3) ансамблевые нейронные сети;

– по алгоритму обучения:

- 1) нейронные сети с учителем;
- 2) нейронные сети без учителя (самообучающиеся);

– по типу сигналов:

1) бинарные нейронные сети – значение сигнала в сети может принимать два значения;

2) цифровые нейронные сети – у которых значение сигнала в сети может принимать конечное число значений;

3) аналоговые нейронные сети – у которых значение сигнала в сети может принимать бесконечное число значений;

– по типу функционирования:

1) синхронные нейронные сети – в которых все нейроны срабатывают одновременно; т.е. в такой сети имеет место синхронизация работы нейронов;

2) асинхронные нейронные сети – не имеющие синхронизации работы нейронов.

Рассмотрим некоторые из видов нейронных сетей. Многослойные нейронные сети прямого распространения. Проектирование таких моделей нейронных сетей было основано на детальном исследовании зрительного тракта головного мозга, состоящего из нейронных экранов, расположенных в определенной последовательности. Каждый экран реализует предназначенную ему функцию. Поэтому, в многослойных нейронных сетях прямого распространения (МНСПР) нейронные связи обладают послойной структурой. Сигналы в подобных структурах проходят к последующему слою от предыдущего.

Очень часто МНСПР применяют для решения проблем, которые решают их биологические прототипы. А также для задач, связанных с классификацией и обработкой изображений или другой информации, имеющей большие объёмы. В общем случае они выполняют функцию нелинейного преобразователя, обучающегося на определенной выборке.

Существуют две структуры многослойных сетей прямого распространения: простой перцептрон и многослойный перцептрон.

Обычно число назначаемых определяющих элементов равно числу классов, на которые необходимо разделить изображения, поступающие на вход перцептрона. Наиболее часто МНСПР используется для задач, выполняющих свои естественные прототипы. А также для задач, связанных с обработкой изображений, классификацией или обработкой другой информации, которая подразумевает большие объёмы данных. В большинстве случаев МНСПР реализуют свойства нелинейного преобразователя, обучающийся на определенной выборке.

Особенность перцептрона заключается в том, что его логические свойства определяются двумя факторами :

- его структурой, которая определяет взаимосвязи между элементами, обрабатывающими сигналы,
- алгоритмом, управляющим обработкой и передачей сигналов.

В отличие от других моделей нейронных сетей и распознающих систем, перцептрон обладает большой свободой в установлении взаимосвязей и его поведение не определяется заранее заданным логическим алгоритмом, а изменяется постепенно в зависимости от изменения его характеристик.

На вектор входных сигналов модель реагирует формированием выходного сигнала от решающего элемента. В перспективе перцептрон формирует гиперплоскость, которая делит многомерное пространство $x_1, x_2, \dots, x_m$ на две части и определяет, к какой части принадлежит входное изображение, выполняя его классификацию. Возникает вопрос о том, какие значения весов следует задать, чтобы перцептрон мог решить поставленную задачу. Эта проблема решается в процессе обучения. Если для разделения двух классов изображений можно использовать гиперплоскость, то при обучении перцептронной модели, которая требует значительных усилий, изображения будут правильно классифицированы. Однако линейное разделение ограничивает круг задач, которые могут быть решены с помощью перцептрона, что упрощает анализ

модели. Эту проблему подробно изучили Минский и Паперт в своей работе [5], где было показано, какие задачи невозможно решить с помощью перцептрона, имеющего только один слой обучаемых связей. Одним из таких примеров является выполнение логической операции XOR. На рисунке 1.4 показан трехрядный перцептрон, который используется для обработки сигналов изображений. Этот перцептрон состоит из сенсорных элементов (S-элементов), которые получают входные сигналы изображения. Связи между элементами S и ассоциативными элементами (А-элементами) установлены случайным образом. А-элементы активируются только в том случае, если достаточное количество S-элементов активировано и связано с данным А-элементом. А-элементы связаны с элементами R, и коэффициенты усиления этих связей являются переменными (весаи).

Определение того, к какому образу относится входное изображение, осуществляется путем анализа взвешенных совокупностей выходов А-элементов, что в свою очередь определяет реакцию системы. В процессе классификации каждому образу обычно соответствует один выходной элемент (R-элемент), который представляет собой линейную комбинацию выходов А-элементов. Важно отметить, что коэффициенты связей между А-элементами и R-элементами являются переменными и могут изменяться в процессе адаптации или обучения.

Предположим, что исследователь хочет обучить модель перцептрона распознавать и отличать два образа V1 и V2. Поэтому из двух выходных элементов один R-элемент предназначен для образа V1, а другой – для образа V2. Модель перцептрона будет функционировать правильно, если значение R1 будет больше R2, когда в качестве входного изображения подается любое изображение из выборки V1, и, наоборот, значение R2 будет больше R1 для входного изображения из выборки V2.

Классификацию изображений на два образа можно сделать и при помощи

одного R-элемента. Тогда изображениям образа V1 должна присваиваться положительная реакция R-элемента, а изображениям образа V2 – отрицательная.

Модель перцептрон обучается при помощи подачи на вход этой модели некоторых последовательных изображений из обоих выборок V1 и V2 .

Исследователь, наблюдающий поведение модели перцептрона, должен вводить различные коррекции коэффициентов связей. В частности, если применяется алгоритм подкрепления с коррекцией ошибок, он, прежде всего, должен установить, является ли решение, принятое моделью перцептрона, верным. Если решение (реакция) модели перцептрона верная ($R1 > R2$ для изображений выборки V1 или $R2 > R1$ для изображений выборки V2), исследователь «поощряет» модель перцептрона, увеличивая веса связей от тех A-элементов, которые применялись при правильной классификации, не изменяя веса связей A-элементов, которые не применялись. Можно оставлять неизменными веса связей A-элементов при ошибочной классификации. В некоторых случаях комбинируют оба метода.

Продолжая процесс обучения, исследователь подает на вход модели перцептрона в определенной последовательности совокупность изображений как из выборки V1, так и из выборки V2 и путем подкрепления весов тех или иных элементов обучает модель перцептрона распознавать образы из выборки V1 и выборки V2 с возможно меньшей ошибкой. Далее модель перцептрона уже самостоятельно (т.е. обучение без учителя) определяет, то есть классифицирует новые изображения, не используемые в процессе обучения. Это выполняется сравнением выходов R1 и R2.

Принципиальная схема модели элементарного трехрядного статического перцептрона показана на рисунке 1.4.

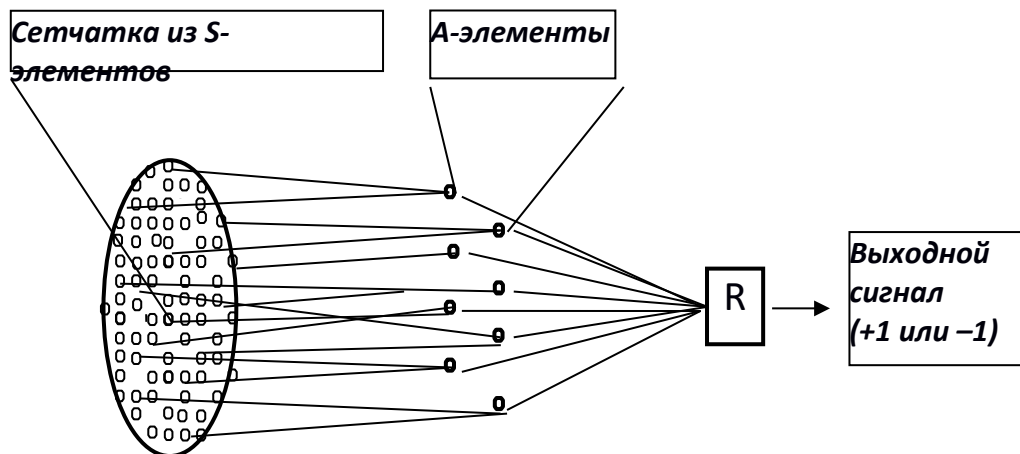


Рисунок 1.4 – Принципиальная схема модели элементарного трехрядного статического перцептрона

Мера приближения изображения к эталону (формула 1.2):

$$E_j = (ax) = a_1 x_1 + a_2 x_2 + \dots + a_n x_n, \quad (1.2)$$

где a_n – векторы изображения,

x_n – векторы эталона.

Исходные положения (эвристики) модели перцептрона такие:

- в качестве входных признаков применяются бинарные сигналы о черно-белых клетках сетчатки: черная клетка $x_i = 1$, белая $x_i = 0$,
- в качестве меры приближения изображения к эталону применяется величина скалярного произведения векторов изображения и вектора эталона,
- интегральные воздействия существуют в виде порогов по эвристическому критерию: скалярное произведение должно быть больше некоторой величины: $E_j \geq E$.

Особенности модели перцептрона:

- структура модели перцептрона многорядна, причем все его элементы (А-элементы) могут сами решить задачу, поставленную всем перцептронам

полностью, хотя и по меньшему числу аргументов,

- после каждого ряда А-элементов находится порог (или компаратор), который может убирать малоэффективные комбинации входных аргументов (признаков),

- сложность комбинаций (число учитываемых аргументов и нелинейность решающих функций) от ряда к ряду модели перцептрона возрастает. Поэтому пороговое значение в одном ряду не может быть заменено по эффекту пороговым значением в другом последующем ряду.

Отсюда можно сделать вывод, что отличием модели перцептрона от других систем распознавания и классификации является множество возможностей его архитектуры: вид решающего правила меняется от одного ряда к другому, причем сложность его постоянно возрастает. Это дает возможность устанавливать при синтезе перцептрона принцип самоорганизации и выбрать решающее правило необходимой сложности, т.е. такое, которое принадлежит минимуму заданного критерия в классификации. Метод синтеза решающего правила – опробование большого числа правил.

Существует две версии алгоритма обучения: стохастическую и детерминированную, известную как обратное распространение (backpropagation). Обратное распространение является обобщением правила Уидроу-Хоффа, используемого для обучения простого перцептрона, и требует наличия обучающей выборки. Эта выборка представляет собой набор изображений, которые нужно классифицировать и сопоставить, и может быть представлена в виде задачи векторного отображения вектор-функции, область действия которой - множество входных изображений, подлежащих классификации, а множество значений - множество выходов. Рассмотрим многослойную нейронную сеть с прямолинейными связями (рисунок 1.5).

Входные элементы (блоки) образуют нижний слой сети, выходные – верхний. Между ними может быть много скрытых слоев.

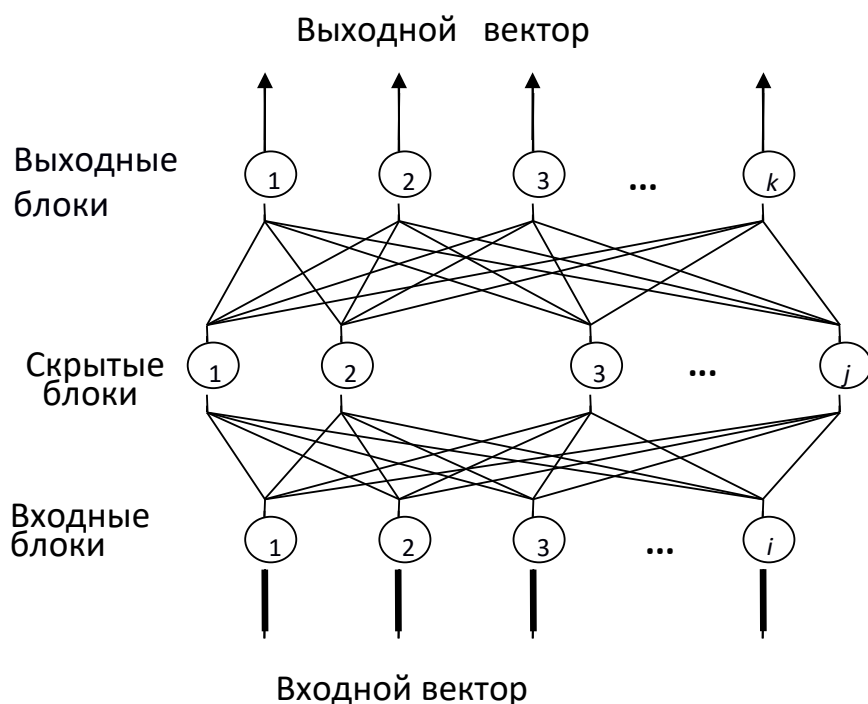


Рисунок 1.5 – Многослойная нейроподобная сеть с прямыми связями

Такие нейронные сети обладают приличными функциональными возможностями при том же числе нейронных взаимосвязей, что и в нейронных сетях, проектируемых в виде слоистых структур. Но также увеличивается и сложность их аналитического исследования. Одной из наиболее исследованной и распространённой разновидностью таких нейронных сетей являются полносвязные сети с симметричными связями. Эти нейронные сети применяются для распознавания образов и решения сложных задач.

Задача, решаемая данной нейронной сетью в контексте ассоциативной памяти, предназначена для решения задачи следующим образом: сначала ей предоставляется набор двоичных сигналов, которые являются образцами (такие сигналы могут быть изображениями, звуковыми сигналами или описаниями процессов и характеристик сигналов). Затем сеть должна иметь возможность извлечь соответствующий класс из случайного искаженного входного сигнала (частично информативного) или дать оценку о том, что входные сигналы или данные не принадлежат ни одному из классов.

1.2.1 Алгоритм обратного распространения ошибки

Алгоритм обратного распространения ошибки [6, 7] является расширением концепции перцептрона на многослойные нейронные сети. Алгоритм обратного распространения использует не менее трех слоев обрабатывающих элементов (нейронов). Архитектура типичной трехслойной сети для алгоритма обратного распространения показана на рисунке 1.6. Элементный слой слева - это входной слой, на который подаются входные данные. За ним следует скрытый слой, в котором обрабатывающие элементы назначаются на предыдущий и последующий слою. Слой справа - выходной слой. Элементы не имеют связи с другими элементами в том же слое. Заметим, что сети обратного распространения не обязательно являются полностью связанными, а количество скрытых слоев может быть произвольным. На рисунке 1.6 изображен многослойный перцептрон.

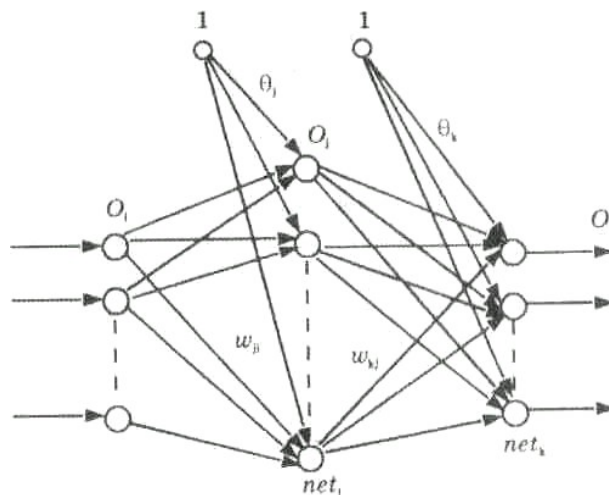


Рисунок 1.6 – Многослойный перцептрон

1.3 Обзор существующих нейронных сетей, используемых для распознавания изображений

Из всего многообразия нейронных сетей, целью которых является классификация, можно отдельно выделить те, основной задачей которых является обработка изображений. Наиболее часто используемые из них :

- свёрточные нейронные сети (convolutional neural networks, CNN) – направлены на задачи поиска и классификации образов на изображении,
- развёртывающие нейронные сети (deconvolutional networks, DN) – применяются при генерации изображений по исходным данным или изображениям с общим классификатором.
- глубокие свёрточные обратные графические сети (deep convolutional inverse graphics networks, DCIGN) – направлены на создание изображений, обладающих несколькими классификаторами, на базе тренировочной выборки графических данных для всех классификаторов в частности. Может работать как в прямом, так и обратном направлениях.

1.3.1 Сверточные нейронные сети

Предложенная Яном ЛеКуном в 1988 году, конволюционная (сверточная) нейронная сеть (CNN) [8] является особой архитектурой искусственных нейронных сетей, специализированной для эффективного распознавания образов. CNN является ключевым компонентом глубокого обучения и вдохновлена устройством зрительной коры, где обнаружены простые клетки, откликающиеся на прямые линии под разными углами, и сложные клетки, активируемые определенными наборами простых клеток. Идея конволюционных нейронных сетей заключается в чередовании конволюционных слоев и слоев субдискретизации (Pooling Layers, Subsampling Layers). Структура сети является однонаправленной (без обратной связи) и в принципе многоуровневой. Для обучения используются стандартные методы, наиболее распространенным из которых является метод распространения обратной связи по ошибке. Функция активации нейрона (передаточная функция) произвольна и остается на усмотрение исследователя.

1.3.2 Глубокие сверточные обратные графические нейронные сети

Глубокая сверточная обратная графическая сеть (DCIGN) - это модель, которая применяется для обработки интерпретируемых представлений изображений, которые связаны с различными изменениями перспективы, такими как структура 3D-изображения, поворот глубины и изменения освещения. Модель DCIGN состоит из многослойных операторов свертки и развертки и обучается по алгоритму Байеса с применением стохастического градиента (SGVB). Обучение заключается в том, чтобы заставить нейроны графического слоя кода выполнять определенные преобразования, связанные с изменением положения объекта. Имея на входе изображение, наша модель может генерировать другие изображения того же объекта с изменением положения или уменьшением освещенности.

Глубокое машинное обучение сделало серьезный прорыв в изучении иерархического представления на основе обработки изображений. Такие модели, как сверточные нейронные сети (CNN) и автокодеры, были успешно применены для создания все более и более абстрактных многослойных визуальных представлений. Но, несмотря на это, существует лишь небольшое количество работ, связанных с поиском наилучших методов обработки и распознавания изображений.

Парадигма "визуализация как обратное изображение" предполагает предоставление представление изображения с заданными характеристиками. Компьютерная графика включает в себя различные элементы, начиная от графического кода и заканчивая изображениями. Графический код обычно делится на некоторые части, обеспечивающие визуализацию сцены, текстур за счет точного управления преобразованиями, такими как положение объекта, положение, освещение и т. д. Целью этого кодирования является простое и понятное представление последовательности реальных данных, чтобы

программный код можно было использовать для компактного представления общих преобразований.

В настоящее время существует лишь небольшое количество методов, которые могут автоматически генерировать семантически интерпретируемые графические коды и изучать механизмы для воспроизведения изображений. На рисунке 1.7 представлена архитектура с кодером и декодером.

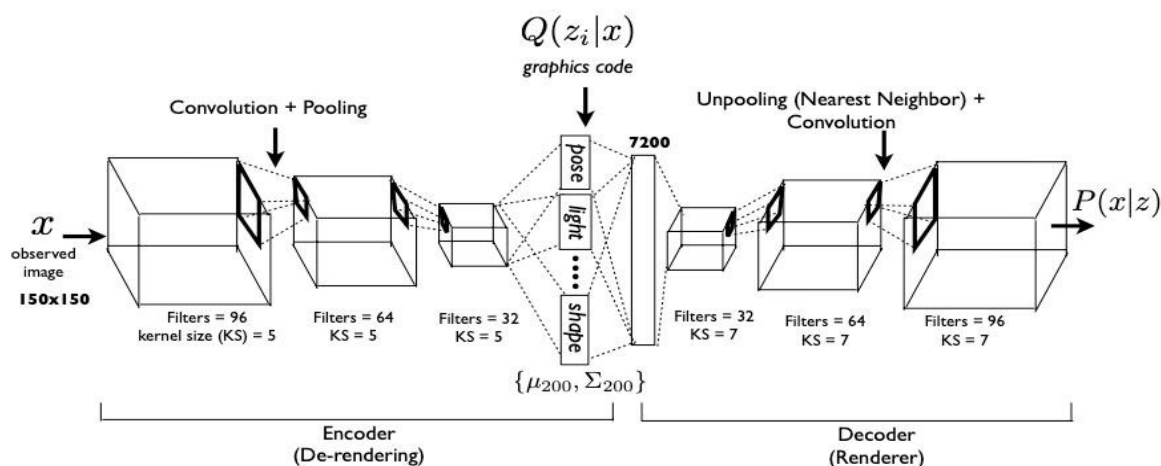


Рисунок 1.7 - Глубокая сверточная обратная графическая сеть (DCIGN)

Кодер состоит из нескольких уровней свертки, за которыми следует уровень максимальной комбинации, а декодер - из нескольких уровней декомпрессии (с использованием выборки ближайших соседей), за которыми следует свертка. В машинном обучении данные "x" проходят через кодер и генерируют апостериорную аппроксимацию $Q(z_i|x)$, где z_i состоит из скрытых переменных сцены (таких как поза, свет или текстура). Для определения параметров в DCIGN для каждой z_i необходимо использовать вариационную объектную функцию для распространения градиента назад с помощью стохастического градиентного спуска. DCIGN можно обучить независимым представлениям (например, повороту лица в заданном направлении). Во время тестирования данные x могут быть пропущены через кодер для получения скрытых значений z_i . Определив соответствующую группу графических (z_i)

кодов, можно повторно воспроизвести изображение при различных ракурсах, условиях освещения, изменении формы и т.д.

Как показано на рисунке 1.7, базовая структура глубокой сверточной обратной графической сети (DCIGN) состоит из двух частей: сети кодировщика (Encoder), которая фиксирует графический код Z для заданных данных x распределения; сети декодера (Decoder), изученное условное распределение заключается в получении приблизительного значения $\hat{X}$ при заданном Z . Z может быть несвязанным представлением, которое содержит набор скрытых переменных $z_i \in Z$, таких как поза, свет и форма.

Обозначим выходной сигнал кодера DCIGN как $y_e = \text{encoder}(x)$. Выходные данные кодера используются для аппроксимации $Q(z_i|y_e)$ параметризованными переменными, где Q выбирается как многомерное нормальное распределение. Есть две причины для использования этой параметризации: используя метод повторной параметризации, можно легко получить градиент выборки относительно параметра Q_θ , и каждое обучение на данных 3D-сканера статистической модели формы, например, потому, что лица имеют то же многомерное нормальное скрытое распределение.

Естественным выбором для целевого представления информации о сцене является разработка для использования в графических движках. Если мы сможем разделить изображение лица, разложив его на переменные позы, освещения и формы, мы можем просто представить те же преобразования, которые эти переменные используют в графических приложениях. На рисунке 1.8 изображено представление, которое мы попытаемся изучить.

$$Z = \begin{array}{|c|c|c|c|} \hline z_1 & z_2 & z_3 & z_{[4,n]} \\ \hline \end{array}$$

corresponds to $\phi \quad \alpha \quad \phi_L$ intrinsic properties (shape, texture, etc)

Рисунок 1.8 Векторная структура представления

где φ - азимутальный угол лица,

α - угол возвышения лица относительно камеры,

φ_L - азимутальный угол источника света.

1.3.3 Деконволюционные нейронные сети

Подход с использованием DNN основан на методе поиска глобального минимума, а также использовании фильтров, полученных при обучении CNN, и предназначен для сведения к минимуму плохо обусловленных затрат, которые возникают в свёрточном подходе.

Концепция деконволюции была впервые представлена Мэттью Зайлером [9] в 2010 году. Деконволюционные сети являются нейросетями, которые используются, например как адаптивные деконволюционные сети для обучения функциям среднего и высокого уровня. Благодаря успешному применению деконволюции в визуализации нейронных сетей, она находит применение во все большем количестве работ, таких как сегментация сцен, генерация изображений и так далее. Деконволюция также имеет много других названий, таких как транспонированная свертка, свертка с дробным шагом и т. д.

Чтобы объяснить, как работают деконволюционные сети, необходимо объяснить взаимосвязь между сверточным слоем и уровнем деконволюции и выяснить взаимосвязь между размером входного объекта деконволюции и размером выходного объекта.

Сверточный уровень выглядит следующим образом:

- двумерная дискретная свертка,
- ввод квадратных объектов,
- размер ядра квадратной свертки,
- одинаковый размер шага для каждого измерения,
- одинаковое заполнение для каждого измерения.

Исходя из приведенного примера мы можем наблюдать взаимосвязь между входными характеристиками сверточного слоя и параметрами ядра.

Для вышеупомянутой операции свертки мы расширяем ядро свертки 3×3 , в разреженную матрицу, где ненулевые элементы представляют первую строку и столбец ядра свертки.

$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

Рисунок 1.8 — ядро свертки 3×3 , расширенное в матрицу

Из приведенного выше анализа мы уже знаем, что прямая операция сверточного слоя может быть выражена как умножение на матрицу, тогда мы можем легко получить обратное распространение сверточного слоя путем умножения транспонирования суммы.

Деконволюцию также называют транспонированной сверткой. Мы можем видеть, что процесс прямого распространения сверточного слоя является обратным процессом распространения слоя деконволюции. Процесс представляет собой процесс прямого распространения уровня деконволюции.

Если мы не рассматриваем канал для вычисления операции деконволюции с помощью обратной операции свертки, мы также можем использовать метод дискретной свертки для нахождения деконволюции. Также определяются параметры операции деконволюции следующим образом:

- двумерная дискретная свертка,
- ввод квадратной функции,
- размер ядра квадратной свертки,
- одинаковый размер шага для каждого измерения,
- одинаковое заполнение для каждого измерения.

Деконволюцию иногда называют сверткой с фильтрацией (свертка с дробных шагом). Для свертки размера шага могут возникнуть ассоциации с соответствующем размере шага деконволюции. Как показано на рисунке выше,

операция деконволюции соответствует операции свертки с параметрами. Для дробного шага операции деконволюции мы можем понять, что необходимо: вставить 0 между его входными элементами объекта, посмотреть на него как на новый ввод объекта, тогда размер шага будет больше не десятичным числом, а 1.

1.4 Выводы по разделу 1

Искусственные нейронные сети — это математические и программные решения, моделирующие естественные связи между нейронами, например, как в мозге человека. Такие нейронные сети используются для решения различных задач: прогнозирования, для распознавания образов, в задачах управления и др.

Также, из материала первой главы можно сделать вывод, что сверточные нейронные отлично подходят для задачи распознавания изображений.

2 Метод классификации для распознавания образов

2.1 Применение метода классификации для распознавания изображений

Что такое классификация изображений? В данном разделе вы узнаете, что такое классификация изображений, как она работает, какие существуют типы классификации изображений и сколько существует практических применений классификации изображений.

Вот общие вопросы:

- что такое классификация изображений,
- как работает классификация изображений,
- виды классификации изображений,
- области применения классификации изображений.

Перейдем непосредственно к сути. Что такое классификация изображений? Классификация изображений - это задача машинного обучения под наблюдением, в которой модель искусственного интеллекта обучается распознавать объекты на изображении на основе помеченных примеров фотографий и затем делать предсказания. Это задача компьютерного зрения, в которой изображения анализируются пиксель за пикселем и относятся к различным заранее определенным классам или категориям. Модель классификации изображений к примеру, может правильно классифицировать изображение как кошку. Модели искусственного интеллекта можно обучать классификации изображений так же, как человек обучается и идентифицирует различных животных и объекты.

Как происходит классификация изображений. Процесс классификации изображений состоит из семи этапов, а именно: сбор данных, предварительная обработка данных, извлечение признаков, обучение, проверка и оценка модели, разработка и оптимизация модели, внедрение. Мы рассмотрим каждый этап в отдельности. Сбор данных: На первом этапе необходимо собрать набор маркированных изображений. Эти изображения должны представлять классы

или категории, которые вы хотите классифицировать. Набор данных должен быть разнообразным и достаточно большим для эффективного обучения и оценки модели классификации. Предварительная обработка данных: Предварительная обработка подготавливает изображения к задаче классификации. Этот этап может заключаться в масштабировании изображений до однородного разрешения, преобразовании их в стандартное цветовое пространство (например, RGB), нормализации значений пикселей и удалении нерелевантной или зашумленной информации. Предварительная обработка позволяет уменьшить вариации и несоответствия в наборе данных, что облегчает обучение и обобщение модели. Извлечение признаков: извлечение признаков предполагает преобразование исходных данных изображения в формат, который может быть понят и использован алгоритмами машинного обучения. Традиционные методы предполагают ручную разработку признаков, например, извлечение цветовых гистограмм, текстурных дескрипторов или граничных признаков. Однако модели глубокого обучения, в частности конволютивные нейронные сети (CNN), стали доминировать в извлечении признаков для классификации изображений. CNN учатся извлекать соответствующие признаки с помощью нескольких конволютивных и ассоциативных слоев, автоматически улавливая паттерны и представления изображений. Обучение: на этапе обучения в модель машинного обучения загружаются предварительно обработанные изображения и соответствующие им метки. Модель учится сопоставлять извлеченные объекты с правильными метками классов в процессе оптимизации. Для обучения могут использоваться различные алгоритмы, включая машины опорных векторов (SVM), деревья решений или архитектуры глубокого обучения, такие как CNN. В процессе обучения модель настраивает свои внутренние параметры таким образом, чтобы минимизировать разницу между прогнозируемыми и реальными тегами в обучаемых данных. Валидация и оценка модели: после обучения модели

необходимо оценить ее эффективность и обобщенность. Для этого используется отдельный набор данных, называемый валидационным или тестовым набором. Модель предсказывает метки изображений для проверки, и эти предсказания сравниваются с реальными метками для расчета таких показателей, как точность, точность и отзыв, а также балла F1. Эта оценка позволяет оценить точность классификации модели и выявить потенциальные проблемы, такие как чрезмерная или недостаточная подгонка. Адаптация и оптимизация модели: на основе результатов оценки модель может быть адаптирована и оптимизирована. Такие методы, как регуляризация, настройка гиперпараметров или изменение архитектуры модели, могут быть использованы для повышения производительности модели и ее обобщения. Развертывание: После обучения и настройки модели ее можно использовать для классификации новых, еще не опубликованных изображений. Модель получает изображение на вход, проходит необходимые этапы предварительной обработки, а затем использует изученные признаки и параметры для предсказания соответствующей метки класса изображения.

2.1.1 Типы классификации изображений

Существуют различные типы методов классификации изображений. Выбор метода зависит от сложности задачи, имеющихся данных и специфических требований приложения. Ниже приведены некоторые распространенные типы классификации изображений.

Бинарная классификация. При бинарной классификации изображения делятся на два класса. Например, для различения изображений кошек и собак или для определения того, содержит ли изображение определенный объект или нет.

Многоклассовая классификация. При многоклассовой классификации изображения делятся более чем на два класса. Каждому изображению присваивается метка из набора заранее определенных категорий. Например,

изображения делятся на такие категории, как животные, автомобили или здания.

Тонкая классификация. Тонкая классификация направлена на выделение подклассов внутри более крупной категории. Она включает в себя классификацию изображений с тонкими визуальными различиями, например, различение разных видов птиц или разных типов цветов.

Иерархическая классификация. При иерархической классификации классы располагаются в иерархической структуре. Сначала изображения классифицируются на более широком уровне, а затем делятся на более конкретные подкатегории. Такой подход позволяет получить более организованный и детальный процесс классификации.

Обнаружение и локализация объектов. Обнаружение объектов выходит за рамки простой классификации, позволяя не только выявить наличие объектов, но и определить их положение на изображении. Она заключается в классификации черт лица и сравнении их с базой данных известных лиц. Классификация изображений позволяет осуществлять контентный поиск изображений, при котором пользователи могут искать изображения по их визуальному содержанию. Она находит применение в системах поиска изображений, платформах электронной коммерции и системах управления цифровыми активами.

Геопространственный анализ. Классификация изображений используется при анализе спутниковых и аэрофотоснимков для классификации землепользования, городского планирования, мониторинга окружающей среды и оказания помощи при стихийных бедствиях. Она используется для идентификации и мониторинга землепользования, растительности, водных пространств и изменений в природной среде.

2.1.2 Применение сверточных нейронных сетей для классификации изображений

В данном подразделе мы рассмотрим теоретический обзор сверточных нейронных сетей (CNN) и их применение в задаче классификации изображений. Сверточные нейронные сети являются мощным инструментом в обработке изображений, где входными данными являются изображения, а выходными могут быть как изображения, так и характеристики, связанные с ними. В данном подразделе мы сосредоточимся на задаче классификации изображений, так как она является основой для многих других задач.

Общий подход к решению задачи классификации изображений.

Решение задачи классификации изображений включает два этапа: формирование значимых признаков изображения и классификация изображения на основе этих признаков. В прошлом, для формирования признаков, эксперты использовали различные методы, такие как многослойный перцептрон (MLP), машина опорных векторов (SVM), метод k ближайших соседей и логистическая регрессия. Признаки формировались с помощью различных преобразований и генерировались дескрипторы, такие как гистограмма направленных градиентов (HOG) или масштабно-инвариантное преобразование признаков (SIFT).

Однако, основным ограничением существующих методов является то, что набор и последовательность шагов для генерации признаков выбирает эксперт. Это может быть трудоемким и требовать большого количества времени и усилий.

В последние годы стало очевидно, что большинство методов генерации признаков можно обобщить с использованием сверточных нейронных сетей. Сверточные нейронные сети используют фильтры (ядра) для обработки изображений. Фильтры представляют собой небольшие матрицы, которые проходят через всё изображение, вычисляя скалярное произведение с текущим положением. Результатом является карта признаков, которая содержит важные

характеристики изображения. Сверточные нейронные сети являются мощным инструментом для обработки и классификации изображений. Они основаны на принципе свертки, который позволяет извлекать значимые признаки из входных данных. В CNN вся информация изображения преобразуется в числа, а затем подвергается свертке с использованием фильтров. Результатом свертки является карта признаков, которая представляет собой набор нейронов, каждый из которых вычисляет свою функцию. Карта признаков формируется с учетом двух основных принципов: локальной связности и общих параметров.

Локальная связность означает, что каждый нейрон связан только с небольшой частью входных данных. Общие параметры подразумевают использование одного и того же фильтра всеми нейронами. Таким образом, каждый нейрон вычисляет свою функцию для своей части входных данных, что позволяет обнаруживать различные признаки в изображении.

Карты признаков, полученные в результате свертки, могут объединяться и рассматриваться как новая трехмерная матрица. Это позволяет использовать несколько фильтров, которые работают независимо друг от друга, но с одними и теми же входными данными. Такое параллельное применение фильтров позволяет обогатить классификатор и расширить область восприимчивости сети.

Сверточные слои в CNN устанавливаются друг на друга, и их размеры и количество фильтров могут варьироваться. Эти слои позволяют обнаруживать значимые признаки в изображении, причем с увеличением глубины сети уровень абстракции признаков возрастает.

Помимо сверточных слоев, в CNN используются слои субдискретизации. Они выполняют понижающую дискретизацию, или даунсэмплинг, которая позволяет снизить пространственную размерность данных и увеличить рецептивную область. Это достигается путем агрегации информации в окрестности каждого пикселя с помощью функций, таких как максимум или

среднее значение.

Важным отличием понижающих слоев является то, что они работают только в пространственном измерении и не изменяют глубину данных. Эти слои позволяют сети быть инвариантной к малым пространственным сдвигам паттернов во входном сигнале.

Последний шаг в CNN - это плотные слои, которые выполняют классификацию входного изображения на основе полученных признаков. Плотные слои соединяют все признаки и принимают решение о классификации.

Таким образом, сверточные нейронные сети являются мощным инструментом для обработки изображений. Они позволяют извлекать значимые признаки и классифицировать изображения с высокой точностью. Эта технология находит применение в различных областях, включая компьютерное зрение, распознавание образов, медицинскую диагностику и автоматическую обработку изображений.

Основными преимуществами сверточных нейронных сетей являются:

- минимум предварительной настройки: в отличие от предыдущих методов, CNN не требуют заранее определенных признаков. Процесс обучения происходит от начала до конца, что упрощает и автоматизирует процесс,
- способность к обобщению: CNN обучаются на основе большого количества образцов, что позволяет им обобщать и распознавать образы в новых данных,
- иерархическая структура: CNN имеют иерархическую архитектуру, которая позволяет извлекать более абстрактные и сложные характеристики изображений на более высоких уровнях.

Типичная архитектура CNN состоит из нескольких слоев: сверточные слои, слои подвыборки и плотные слои. Сверточные слои выполняют свертку фильтров с изображением для извлечения признаков. Слои подвыборки уменьшают размерность карты признаков для уменьшения вычислительной

сложности. Плотные слои используются для классификации на основе извлеченных признаков.

Поэтому можно заключить, что сверточные нейронные сети являются мощным инструментом для решения задачи классификации изображений. Они позволяют автоматизировать процесс извлечения признаков и обучения модели, что упрощает и ускоряет работу. Кроме того, CNN обладают способностью к обобщению и могут распознавать образы в новых данных. Это делает их эффективным инструментом в области обработки изображений.

Свертка в сверточных нейросетях — это цифровой фильтр, который сам обучается. При этом этот фильтр можно трактовать как нейрон, выполняющий операцию взвешенного суммирования входных сигналов. Веса нейронов — это коэффициенты ядра свертки. При применении его ко всему изображению вся эта процедура представляет собой прохождение частей изображения через слой одинаковых нейронов. Результат записывается в новое изображение, называемое картой признаков. В сверточном слое используют нескольких сверток с различными ядрами. При этом формируется несколько карт признаков. Каждая карта признаков, полученная при применении одной свертки отражает одинаковые локальные признаки во всех частях изображения.

Карта признаков представляет собой набор нейронов, каждый из которых имеет тем большее значение, чем больше связанный с ним фрагмент изображения похож на ядро.

Каждый сверточный уровень преобразует предыдущую карту признаков в несколько карт последующих признаков, которые можно представить в виде 3D массива.

Пулинг основан на том, что изображения обладают свойством локальной скоррелированности пикселей — соседние пиксели, как правило, не сильно отличаются друг от друга. Таким образом, если из нескольких соседних получить какой-либо агрегат, то потери информации будут незначительными.

Слой пулинга. Есть кубик какого-то объема — 3 канала, 10, или 100 каналов, которые насчитали свертками. Он просто уменьшает его по ширине и по высоте, остальные размерности не трогает. Уменьшение по высоте и ширине как правило в два раза. Поэтому окно пулинга берут 2×2 со сдвигом 2.

Первая сверточная сеть Ле Куна – в ней представлены основные принципы сверточных сетей. Использовалась для распознавания рукописных почтовых индексов. Эта архитектура была стандартным шаблоном для построения сверточных сетей вплоть до недавнего времени: свертка чередуется с пулингом несколько раз, затем несколько полносвязных слоев.

Основные строительные блоки — свертки 5×5 со сдвигом 1 и пулинг 2×2 со сдвигом 2. Свертки играют роль детекторов признаков, а пулинг (или сабсемплинг) используют для уменьшения размерности.

СНС состоит из нескольких пар чередующихся сверточных и субдискретизирующих слоев, после прохождения которых исходное изображение вырождается в вектор либо скаляр и поступает на вход многослойному персептрону либо любой другой выбранной пользователем полносвязной сети. При последовательном прохождении пар чередующихся сверточных и субдискретизирующих слоев был получен бипирамидальный эффект, что означает увеличение количества карт признаков на каждой паре C-S слоев при одновременном уменьшении пространственного разрешения карт по сравнению с предыдущей C-S парой.

Функция активации нейронов как в сверточной, так и в выходной полносвязной сети может быть любой, удовлетворяющей условию гладкости, так как обучение структуры производится с помощью метода обратного распространения ошибки.

Также существуют полносвязные сверточные сети. В них полносвязный слой преобразуется в свертку. Для этого надо пересчитать веса полносвязного слоя в ядро свертки. При этом для эквивалентного сверточного слоя необходимо

установить размерность фильтра равной размерности карты признаков, а количество фильтров должно равняться количеству нейронов данного слоя.

2.2 Постановка задачи

Для решения задачи идентификации объектов поражения необходимо исследовать эффективность применения алгоритма воспроизведения ошибок для обучения НС. Исходя из теории НС, необходима разработка программы, имитирующей работу НС и способной обеспечить функционирование сети в соответствии с требованиями обучения, тестирования и обнаружения. Оценка должна проводиться по следующим параметрам:

- эффективность применения выбранного алгоритма обучения,
- эффективность обнаружения смоделированного объекта,
- зависимость времени обучения от количества изображений, подвергаемых обучению.

Исходными данными для такой задачи должны быть двумерные изображения образцов двух классов. Представленные изображения должны соответствовать стандарту по размеру и параметрам формата. Для решения этой задачи должны использоваться растровые изображения размером примерно 128x128 пикселей. Каждое из изображений должно было представлять собой изображение, близкое у реальному. Выходными данными должна быть информация о принадлежности представленного образца к тому или иному классу. Для наглядности следует использовать процентное выражение степени принадлежности. Разработанная программа должна моделировать полносвязную многослойную нейронную сеть с прямыми связями и, к примеру, сигмоидальной функцией активации нейронов. На вход сети должна подаваться информация об изображении, а информация о принадлежности изображения к тому или иному виду должна удаляться со входа.

Программа должна выполнять следующие функции:

- обучение сети,
- тестирование сети,
- применение сети для распознавания предъявленных ей изображений;

Кроме того, программа должна отвечать требованиям объективности и модульности. Программа должна была быть разработана на платформе ОС Windows. Программа должна уметь работать с изображениями любого размера в растровом формате.

Задание должно выполняться под управлением операционной системы MS Windows на компьютерной системе со минимальными характеристиками:

- процессор - 2000 МГц,
- оперативная память - 1024 Мб,
- объем жесткого диска - 120 Гб.

Необходимо создать выборку изображений для обучения сети и выборку для верификации, содержащую как ранее предъявленные изображения, так и не предъявленные, а также серию искаженных, т.е. зашумленных изображений. Информация о степени принадлежности изображения к тому или иному классу изображений может выражаться в процентах.

Модель нейронной сети должна обучаться на персональном компьютере, причем могут использоваться аппаратные средства, такие как центральный процессор или графический процессор. Применение графического процессора преимущественно лучше, так как известно, что их применяют для уменьшения скорости обучения нейронных сетей. Обученная нейронная сеть проверяется на тестовой выборке — то есть выборке изображений, не участвующих непосредственно при обучении данной нейронной сети. Также необходима валидационная выборка — это изображения, не применяющиеся для обучения, но проверяют точность модели в этом процессе. Разработанная

программа должна была использоваться для проведения серии экспериментов, направленных на обучение сети, ее тестирование и выполнение вышеуказанных операций с сетью. Обучение сети проводилось по заранее выбранному алгоритму на предоставленных образцах изображений. Результаты тестирования сети и график, показывающий поведение значения функции ошибки в процессе обучения, представлены в данной работе.

2.3 Экспериментальная часть

Для того, чтобы понять, какая архитектура сверточной нейронной сети лучше подходит для распознавания объектов двух классов (или задачи классификации), необходимо протестировать выбранную изначально модель машинного обучения методом функционального тестирования и посмотреть на метрики качества модели, то есть в основном часто используемую метрику — точность (ассурасу).

Изначально используем модель сверточной нейронной сети, похожую на модель LeNet-5 1998 года Яна Лекуна — первопроходца в области применения сверточных нейронных сетей. Данная сеть представлена на рисунке 2.1.

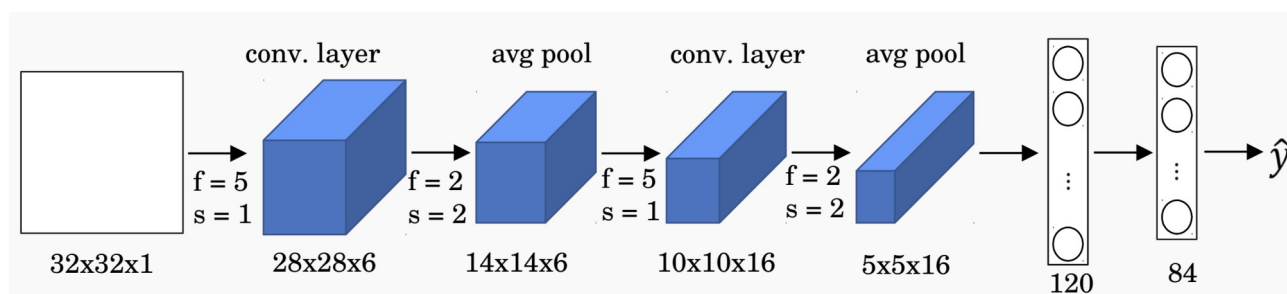


Рисунок 2.1 — Структурная схема сверточной нейронной сети LeNet-5

Данная модель сверточной нейронной сети была предназначена для распознавания рукописных цифр. Если быть точнее — решалась задача классификации объектов 10 классов — по количеству повторяемых цифр. Можно модифицировать данную модель, задав в full-connected слое не 10

нейронов — классов, а всего 2. К примеру, если посмотреть на рисунок выше — преобразование full-connected слоя 120-84-10-2. Свою модель я преобразовал к такому виду (рисунок 2.2):

```
class ConvNet(nn.Module):
    def __init__(self, num_classes=2):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(128 * 32 * 32, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.pool(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        return x
```

Рисунок 2.2 — Собственная модель нейронной сети

Как работает данная нейронная сеть? Нейронная сеть принимает в первом сверточном слое исходное изображение глубиной, равной 3, что означает расцветку RGB – по красному, зеленому и синему цвету. Слой свертки увеличивает глубину изображения до 64. Далее идет функция активации ReLU. Следующий слой пулинга MaxPool2d “обрезает” изображение, при этом сохраняя глубину, полученную с сверточном слое.

Следующий сверточный слой увеличивает глубину изображения с 64 до 128. Далее, после слоя пулинга, у нас остается изображение размером 32x32 и глубиной 128.

Теперь мы должны данное изображение подать на полносвязный full-connected слой. Преобразуем исходное изображения к векторному виду — вектору признаков. Данный вектор признаков через радиально-базисную функцию (RBF) определяет класс объекта.

2.3.1 Обучение нейронной сети

Обучим нейронную сеть на 10, 20, 30 и 50 эпохах. Выборка будет составлять 130 изображений обучаемой и 40 изображений в валидационной. Изображения птиц будем брать из датасета CUB-200, изображения БПЛА — из сети интернет. Из результатов обучения можно заметить, что точность модели (ассурагу) возрастает по мере увеличения количества эпох, но наступает предел на 85% точности. Это может быть решено устранением неравномерности обучающей и валидационной выборки (70% к 30%), подбором других активационных функций, а также изменением архитектурного построения модели нейронной сети. Обучение сети показано на рисунке 2.3.

```
Эпоха 66 из 80 Шаг 0 Ошибка: 0.07058005034923553
Эпоха 67 из 80 Шаг 0 Ошибка: 0.05760715901851654
Эпоха 68 из 80 Шаг 0 Ошибка: 0.10558926314115524
Эпоха 69 из 80 Шаг 0 Ошибка: 0.06537754833698273
Эпоха 70 из 80 Шаг 0 Ошибка: 0.11720863729715347
Эпоха 71 из 80 Шаг 0 Ошибка: 0.0652952492237091
Эпоха 72 из 80 Шаг 0 Ошибка: 0.7392018437385559
Эпоха 73 из 80 Шаг 0 Ошибка: 0.8220566511154175
Эпоха 74 из 80 Шаг 0 Ошибка: 0.09448367357254028
Эпоха 75 из 80 Шаг 0 Ошибка: 0.08514457941055298
Эпоха 76 из 80 Шаг 0 Ошибка: 0.11780187487602234
Эпоха 77 из 80 Шаг 0 Ошибка: 0.15071746706962585
Эпоха 78 из 80 Шаг 0 Ошибка: 0.08442924171686172
Эпоха 79 из 80 Шаг 0 Ошибка: 0.14345748722553253
614.7219569683075
Точность модели: 85.71428571428571%
```

Рисунок 2.3 — Обучение сети на 80 эпохах, предел точности — 85%

2.3.2 Обучение сети с измененной архитектурой

В качестве продолжения эксперимента попробуем добавить дополнительный сверточный слой (conv3) и слой пулинга для увеличения глубины преобразованного изображения. Увеличение количества сверточных слоев показано на рисунке 2.4. Обучение такой нейронной сети показано на рисунке 2.5.

```

class ConvNet(nn.Module):
    def __init__(self, num_classes=2):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(256 * 16 * 16, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.conv3(x)
        x = self.relu(x)
        x = self.pool(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        return x

```

Рисунок 2.4 — Изменение архитектуры модели нейронной сети

Эпоха 18 из 30 Шаг 0 Ошибка: 0.38400909304618835	Эпоха 37 из 50 Шаг 0 Ошибка: 0.31500402092933655
Эпоха 19 из 30 Шаг 0 Ошибка: 0.4628978967666626	Эпоха 38 из 50 Шаг 0 Ошибка: 0.37453392148017883
Эпоха 20 из 30 Шаг 0 Ошибка: 0.9302642345428467	Эпоха 39 из 50 Шаг 0 Ошибка: 0.26145094633102417
Эпоха 21 из 30 Шаг 0 Ошибка: 0.41576293110847473	Эпоха 40 из 50 Шаг 0 Ошибка: 0.42483407258987427
Эпоха 22 из 30 Шаг 0 Ошибка: 0.776932418346405	Эпоха 41 из 50 Шаг 0 Ошибка: 0.4003957211971283
Эпоха 23 из 30 Шаг 0 Ошибка: 0.5559103488922119	Эпоха 42 из 50 Шаг 0 Ошибка: 0.2842353284358978
Эпоха 24 из 30 Шаг 0 Ошибка: 0.40912872552871704	Эпоха 43 из 50 Шаг 0 Ошибка: 0.7026075124740601
Эпоха 25 из 30 Шаг 0 Ошибка: 0.36556488275527954	Эпоха 44 из 50 Шаг 0 Ошибка: 0.24533973634243011
Эпоха 26 из 30 Шаг 0 Ошибка: 0.3422284424304962	Эпоха 45 из 50 Шаг 0 Ошибка: 0.3089185953140259
Эпоха 27 из 30 Шаг 0 Ошибка: 0.3031746745109558	Эпоха 46 из 50 Шаг 0 Ошибка: 0.1450861543416977
Эпоха 28 из 30 Шаг 0 Ошибка: 0.7847779393196106	Эпоха 47 из 50 Шаг 0 Ошибка: 0.4564577341079712
Эпоха 29 из 30 Шаг 0 Ошибка: 0.3254508972167969	Эпоха 48 из 50 Шаг 0 Ошибка: 0.2966757118701935
271.7309818267822	Эпоха 49 из 50 Шаг 0 Ошибка: 0.3720376789569855
Точность модели: 78.57142857142857%	491.42224860191345
	Точность модели: 78.57142857142857%

Рисунок 2.5 — Обучение сети

Обучаем нейронную сеть на 10, 20, 30 и 50 эпохах. Из результатов обучения можно заметить, что наступает предел точности модели на 78%. Этот показатель меньше, чем при предыдущей архитектуре - увеличение количества сверточных слоев не помогло, но предел точности модели наступил гораздо раньше — к 20й эпохе, что немного затрудняет анализ.

Попробуем еще увеличить количество сверточных слоев, чтобы лучше понять, что происходит с обучением данной модели нейронной сети (рисунки 2.6, 2.7.)

```

class ConvNet(nn.Module):
    def __init__(self, num_classes=2):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
        self.conv4 = nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(512 * 8 * 8, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.conv3(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.conv4(x)
        x = self.relu(x)
        x = self.pool(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        return x

```

Рисунок 2.6 — Изменение архитектуры модели нейронной сети

Добавим еще один сверточный слой, а затем слой пулинга.

Эпоха 0 из 10 Шаг 0 Ошибка: 0.6925126910209656	Эпоха 8 из 20 Шаг 0 Ошибка: 0.6668879389762878
Эпоха 1 из 10 Шаг 0 Ошибка: 0.705711305141449	Эпоха 9 из 20 Шаг 0 Ошибка: 0.6381415724754333
Эпоха 2 из 10 Шаг 0 Ошибка: 0.669447660446167	Эпоха 10 из 20 Шаг 0 Ошибка: 0.5746086239814758
Эпоха 3 из 10 Шаг 0 Ошибка: 0.6742146611213684	Эпоха 11 из 20 Шаг 0 Ошибка: 0.61082923412323
Эпоха 4 из 10 Шаг 0 Ошибка: 0.6841295957565308	Эпоха 12 из 20 Шаг 0 Ошибка: 0.5785169005393982
Эпоха 5 из 10 Шаг 0 Ошибка: 0.6355614066123962	Эпоха 13 из 20 Шаг 0 Ошибка: 1.126082420349121
Эпоха 6 из 10 Шаг 0 Ошибка: 0.6149827241897583	Эпоха 14 из 20 Шаг 0 Ошибка: 0.6408122777938843
Эпоха 7 из 10 Шаг 0 Ошибка: 0.5993834137916565	Эпоха 15 из 20 Шаг 0 Ошибка: 0.5222402811050415
Эпоха 8 из 10 Шаг 0 Ошибка: 0.5940759778022766	Эпоха 16 из 20 Шаг 0 Ошибка: 0.4631728231906891
Эпоха 9 из 10 Шаг 0 Ошибка: 0.5584381818771362	Эпоха 17 из 20 Шаг 0 Ошибка: 0.4510592818260193
104.4121687412262	Эпоха 18 из 20 Шаг 0 Ошибка: 0.9392331838607788
Точность модели: 72.32142857142857%	Эпоха 19 из 20 Шаг 0 Ошибка: 0.5652002692222595
	215.25481462478638
	Точность модели: 57.142857142857146%

Рисунок 2.7 — Обучение сети

Обучим сеть. Теперь можно сделать более-менее конкретные выводы по поводу влияния изменения архитектуры модели нейронной сети, а конкретно — увеличения сверточных слоев на точность при обучении.

При увеличении сверточных слоев в модели наблюдается более быстрое переобучение, уже к 20й эпохе, причем предельная точность не возрастает. Это может быть связано с более «быстрым» выделением признаков классифицируемых классов при проведении операции свертки.

Изменение количества сверточных слоев не привело к более улучшенному результату, поэтому продолжаем эксперимент.

2.3.3 Обучение сети с измененной активационной функцией

Попробуем изменить функцию активации на сигмоидальную — Sigmoid. Обучим сеть. Из результатов обучения можно сделать вывод, что модель довольно быстро переобучилась, причем ошибка loss выдает более нестабильные значения. Результаты показаны на рисунке 2.8.

Эпоха 0 из 20 Шаг 0 Ошибка: 0.7243953347206116	Эпоха 10 из 30 Шаг 0 Ошибка: 0.937504231929779
Эпоха 1 из 20 Шаг 0 Ошибка: 3.8052237033843994	Эпоха 11 из 30 Шаг 0 Ошибка: 0.6573017835617065
Эпоха 2 из 20 Шаг 0 Ошибка: 3.864903211593628	Эпоха 12 из 30 Шаг 0 Ошибка: 0.9600433707237244
Эпоха 3 из 20 Шаг 0 Ошибка: 3.0970914363861084	Эпоха 13 из 30 Шаг 0 Ошибка: 0.7817642092704773
Эпоха 4 из 20 Шаг 0 Ошибка: 4.5857696533203125	Эпоха 14 из 30 Шаг 0 Ошибка: 0.7968395352363586
Эпоха 5 из 20 Шаг 0 Ошибка: 0.6801090240478516	Эпоха 15 из 30 Шаг 0 Ошибка: 1.9508795738220215
Эпоха 6 из 20 Шаг 0 Ошибка: 1.4214656352996826	Эпоха 16 из 30 Шаг 0 Ошибка: 1.0665686130523682
Эпоха 7 из 20 Шаг 0 Ошибка: 1.1409800052642822	Эпоха 17 из 30 Шаг 0 Ошибка: 0.9585657715797424
Эпоха 8 из 20 Шаг 0 Ошибка: 1.811667799949646	Эпоха 18 из 30 Шаг 0 Ошибка: 0.6872204542160034
Эпоха 9 из 20 Шаг 0 Ошибка: 1.8656322956085205	Эпоха 19 из 30 Шаг 0 Ошибка: 0.8587204217910767
Эпоха 10 из 20 Шаг 0 Ошибка: 1.7986153364181519	Эпоха 20 из 30 Шаг 0 Ошибка: 0.6980143189430237
Эпоха 11 из 20 Шаг 0 Ошибка: 1.115685224533081	Эпоха 21 из 30 Шаг 0 Ошибка: 1.263115644454956
Эпоха 12 из 20 Шаг 0 Ошибка: 1.1379894018173218	Эпоха 22 из 30 Шаг 0 Ошибка: 0.6934729814529419
Эпоха 13 из 20 Шаг 0 Ошибка: 0.6909804344177246	Эпоха 23 из 30 Шаг 0 Ошибка: 1.286480188369751
Эпоха 14 из 20 Шаг 0 Ошибка: 0.8901185989379883	Эпоха 24 из 30 Шаг 0 Ошибка: 1.3365468978881836
Эпоха 15 из 20 Шаг 0 Ошибка: 1.4960424900054932	Эпоха 25 из 30 Шаг 0 Ошибка: 0.6206021308898926
Эпоха 16 из 20 Шаг 0 Ошибка: 0.9732036590576172	Эпоха 26 из 30 Шаг 0 Ошибка: 0.6348741054534912
Эпоха 17 из 20 Шаг 0 Ошибка: 0.7135075330734253	Эпоха 27 из 30 Шаг 0 Ошибка: 0.6842182874679565
Эпоха 18 из 20 Шаг 0 Ошибка: 1.2578405141830444	Эпоха 28 из 30 Шаг 0 Ошибка: 0.6409754753112793
Эпоха 19 из 20 Шаг 0 Ошибка: 1.7030737400054932	Эпоха 29 из 30 Шаг 0 Ошибка: 0.6309857964515686
161.4392285346985	239.5217854976654
Точность модели: 62.5%	Точность модели: 38.392857142857146%

Рисунок 2.8 — Обучение сети с сигмоидальной функцией активации

Изменим функцию активации на LeakyReLU – модификацию ReLU, преимущество которой состоит в том, что не отсекает отрицательные градиенты в некоторых пределах при обучении модели. Архитектура такой нейронной сети показана на рисунке 2.9. Обучение нейронной сети показано на рисунке 2.10.

```

class ConvNet(nn.Module):
    def __init__(self, num_classes=2):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
        self.leaky_relu = nn.LeakyReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(128 * 32 * 32, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.leaky_relu(x)
        x = self.pool(x)
        x = self.conv2(x)
        x = self.leaky_relu(x)
        x = self.pool(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        return x

```

Рисунок 2.9 - Архитектура сети с функцией активации LeakyReLU

```

Эпоха 37 из 50 Шаг 0 Ошибка: 0.21295109391212463
Эпоха 38 из 50 Шаг 0 Ошибка: 0.28301486372947693
Эпоха 39 из 50 Шаг 0 Ошибка: 0.39961227774620056
Эпоха 40 из 50 Шаг 0 Ошибка: 0.2103075534105301
Эпоха 41 из 50 Шаг 0 Ошибка: 0.21104781329631805
Эпоха 42 из 50 Шаг 0 Ошибка: 0.2693674564361572
Эпоха 43 из 50 Шаг 0 Ошибка: 0.1797628104686737
Эпоха 44 из 50 Шаг 0 Ошибка: 0.2726663053035736
Эпоха 45 из 50 Шаг 0 Ошибка: 0.22650814056396484
Эпоха 46 из 50 Шаг 0 Ошибка: 0.1641518622636795
Эпоха 47 из 50 Шаг 0 Ошибка: 0.30420419573783875
Эпоха 48 из 50 Шаг 0 Ошибка: 0.20998483896255493
Эпоха 49 из 50 Шаг 0 Ошибка: 0.1476813405752182
391.5932648181915
Точность модели: 85.71428571428571%

```

Рисунок 2.10 - Обучение сети с функцией активации LeakyReLU

Сейчас впервые можно заключить, что точность модели улучшилась в плане стабильности. Уже на 10й эпохе она составляет 58%, а на 20й — 80%. Предел точности остается такой же — 85%.

2.3.4 Обучение сети с измененной скоростью обучения

Изменим скорость обучения с 0.01 на 0.005. Модель быстро переобучается. Результаты обучения показаны на рисунке 2.11.

Эпоха 0 из 10 Шаг 0 Ошибка: 0.6898909211158752	Эпоха 7 из 20 Шаг 0 Ошибка: 0.4705394208431244	Эпоха 18 из 30 Шаг 0 Ошибка: 0.41072638882263184
Эпоха 1 из 10 Шаг 0 Ошибка: 0.6933567523956299	Эпоха 8 из 20 Шаг 0 Ошибка: 0.8224369287490845	Эпоха 19 из 30 Шаг 0 Ошибка: 0.6015390157699585
Эпоха 2 из 10 Шаг 0 Ошибка: 0.5453768968582153	Эпоха 9 из 20 Шаг 0 Ошибка: 0.44142112135887146	Эпоха 20 из 30 Шаг 0 Ошибка: 0.5182175040245056
Эпоха 3 из 10 Шаг 0 Ошибка: 0.6074350476264954	Эпоха 10 из 20 Шаг 0 Ошибка: 0.46643543243408203	Эпоха 21 из 30 Шаг 0 Ошибка: 0.2682320773601532
Эпоха 4 из 10 Шаг 0 Ошибка: 0.6100789308547974	Эпоха 11 из 20 Шаг 0 Ошибка: 0.35851845145225525	Эпоха 22 из 30 Шаг 0 Ошибка: 0.29700779914855957
Эпоха 5 из 10 Шаг 0 Ошибка: 0.5427272915840149	Эпоха 12 из 20 Шаг 0 Ошибка: 0.45867183804512024	Эпоха 23 из 30 Шаг 0 Ошибка: 0.4292171895503998
Эпоха 6 из 10 Шаг 0 Ошибка: 0.4588909149169922	Эпоха 13 из 20 Шаг 0 Ошибка: 0.36553457379341125	Эпоха 24 из 30 Шаг 0 Ошибка: 0.2830888628959656
Эпоха 7 из 10 Шаг 0 Ошибка: 0.37979528307914734	Эпоха 14 из 20 Шаг 0 Ошибка: 0.4412585496902466	Эпоха 25 из 30 Шаг 0 Ошибка: 0.43126213550567627
Эпоха 8 из 10 Шаг 0 Ошибка: 0.6291037797927856	Эпоха 15 из 20 Шаг 0 Ошибка: 0.3069823384284973	Эпоха 26 из 30 Шаг 0 Ошибка: 0.4137589633464813
Эпоха 9 из 10 Шаг 0 Ошибка: 0.5538496971130371	Эпоха 16 из 20 Шаг 0 Ошибка: 0.3255559504032135	Эпоха 27 из 30 Шаг 0 Ошибка: 0.3997543752193451
75.07454466819763	Эпоха 17 из 20 Шаг 0 Ошибка: 0.5338847041130066	Эпоха 28 из 30 Шаг 0 Ошибка: 0.5340152978897095
Точность модели: 77.67857142857143%	Эпоха 18 из 20 Шаг 0 Ошибка: 0.21312393248081207	Эпоха 29 из 30 Шаг 0 Ошибка: 0.31461775302886963
	Эпоха 19 из 20 Шаг 0 Ошибка: 0.38424402475357056	233.66639304161072
	151.6058864593506	Точность модели: 37.5%
	Точность модели: 49.107142857142854%	

Рисунок 2.11 - Обучение сети с измененной скоростью обучения на 0.005

На данном этапе эксперимента можно заключить, что увеличение слоев свертки, уменьшение скорости обучения приводит к более быстрому переобучению модели, а применение функции активации LeakyReLU показывает более стабильные результаты по точности модели.

2.3.5 Увеличение выборки

На всех предыдущих этапах эксперимента сеть обучалась на достаточно небольших данных — 130 изображений в обучающей и 40 изображений в валидационной выборках. Понятно, что мы установили некоторые зависимости при обучении нейронной сети, но при увеличении выборки модель станет менее склонна к переобучению, а также значение точности модели возможно не увеличится, но станет более адекватным.

Увеличим количество изображений до 2000 (1470 в обучающей и 630 в валидационной). Всего же — 4000 для двух классов (рисунок 2.12).

```
In [3]: runfile('C:/Users/79023/Desktop/nir/untitled0.py', wdir='C:/
Jsers/79023/Desktop/nir')
Эпоха 0 из 5 Шаг 0 Ошибка: 0.6871551871299744
Эпоха 1 из 5 Шаг 0 Ошибка: 0.6126210689544678
Эпоха 2 из 5 Шаг 0 Ошибка: 0.5408242344856262
Эпоха 3 из 5 Шаг 0 Ошибка: 0.6257917284965515
Эпоха 4 из 5 Шаг 0 Ошибка: 0.4642602503299713
545.2198927402496
Точность модели: 69.46666666666667%

In [4]: runfile('C:/Users/79023/Desktop/nir/untitled0.py', wdir='C:/
Jsers/79023/Desktop/nir')
Эпоха 0 из 5 Шаг 0 Ошибка: 0.6884472966194153
Эпоха 1 из 5 Шаг 0 Ошибка: 0.49703338742256165
Эпоха 2 из 5 Шаг 0 Ошибка: 0.6373722553253174
Эпоха 3 из 5 Шаг 0 Ошибка: 0.6281555891036987
Эпоха 4 из 5 Шаг 0 Ошибка: 0.5128803849220276
329.38241147994995
Точность модели: 70.9090909090909%
```

Рисунок 2.12 — Обучение сети

1e — обучение на большой выборке 1470/630 изображений, 5 эпох.

2e — обучение на большой выборке 1100/440 изображений, 5 эпох.

Здесь можно сделать вывод, что при небольшом уменьшении выборки с 2000 до 1640 — точность модели даже немного улучшилась.

Увеличим количество эпох обучения при различных скоростях обучения (рисунок 2.13, 2.14, 2.15).

```
Эпоха 10 из 20 Шаг 0 Ошибка: 0.4767942726612091
Эпоха 11 из 20 Шаг 0 Ошибка: 0.5853844881057739
Эпоха 12 из 20 Шаг 0 Ошибка: 0.5771688222885132
Эпоха 13 из 20 Шаг 0 Ошибка: 0.5525827407836914
Эпоха 14 из 20 Шаг 0 Ошибка: 0.4498608112335205
Эпоха 15 из 20 Шаг 0 Ошибка: 0.5733330845832825
Эпоха 16 из 20 Шаг 0 Ошибка: 0.5403940081596375
Эпоха 17 из 20 Шаг 0 Ошибка: 0.577012836933136
Эпоха 18 из 20 Шаг 0 Ошибка: 0.38560569286346436
Эпоха 19 из 20 Шаг 0 Ошибка: 0.4742441177368164
1349.3224160671234
Точность модели: 65.56818181818181%
```

Рисунок 2.13 — Обучение на большой выборке 1100/440, 20 эпох, lr = 0.005

```
Эпоха 10 из 20 Шаг 0 Ошибка: 0.39632001519203186
Эпоха 11 из 20 Шаг 0 Ошибка: 0.41638612747192383
Эпоха 12 из 20 Шаг 0 Ошибка: 0.3638090193271637
Эпоха 13 из 20 Шаг 0 Ошибка: 0.38575994968414307
Эпоха 14 из 20 Шаг 0 Ошибка: 0.4500216245651245
Эпоха 15 из 20 Шаг 0 Ошибка: 0.37038058042526245
Эпоха 16 из 20 Шаг 0 Ошибка: 0.5176800489425659
Эпоха 17 из 20 Шаг 0 Ошибка: 0.3362700343132019
Эпоха 18 из 20 Шаг 0 Ошибка: 0.4391564726829529
Эпоха 19 из 20 Шаг 0 Ошибка: 0.32115882635116577
1372.0942640304565
Точность модели: 75.22727272727273%
```

Рисунок 2.14 - Обучение на большой выборке 1100/440, 20 эпох, lr = .01

```
Эпоха 10 из 20 Шаг 0 Ошибка: 0.45500797033309937
Эпоха 11 из 20 Шаг 0 Ошибка: 0.5111812949180603
Эпоха 12 из 20 Шаг 0 Ошибка: 0.47667205333709717
Эпоха 13 из 20 Шаг 0 Ошибка: 0.39888647198677063
Эпоха 14 из 20 Шаг 0 Ошибка: 0.2959679961204529
Эпоха 15 из 20 Шаг 0 Ошибка: 0.35444414615631104
Эпоха 16 из 20 Шаг 0 Ошибка: 0.34290358424186707
Эпоха 17 из 20 Шаг 0 Ошибка: 0.3625101149082184
Эпоха 18 из 20 Шаг 0 Ошибка: 0.2753766179084778
Эпоха 19 из 20 Шаг 0 Ошибка: 0.4035477340221405
1300.956948518753
Точность модели: 77.61363636363636%
```

Рисунок 2.15 - Обучение на большой выборке 1100/440, 20 эпох, lr = .02

Здесь напрашивается вывод, что для данной выборки возможно нужно брать скорость обучения немного с завышением (0.02). Продолжаем эксперимент.

2.3.6 Построение графиков точности и ошибки

Построим графики точности и ошибки (рисунки 2.16, 2.17, 2.18, 2.19).

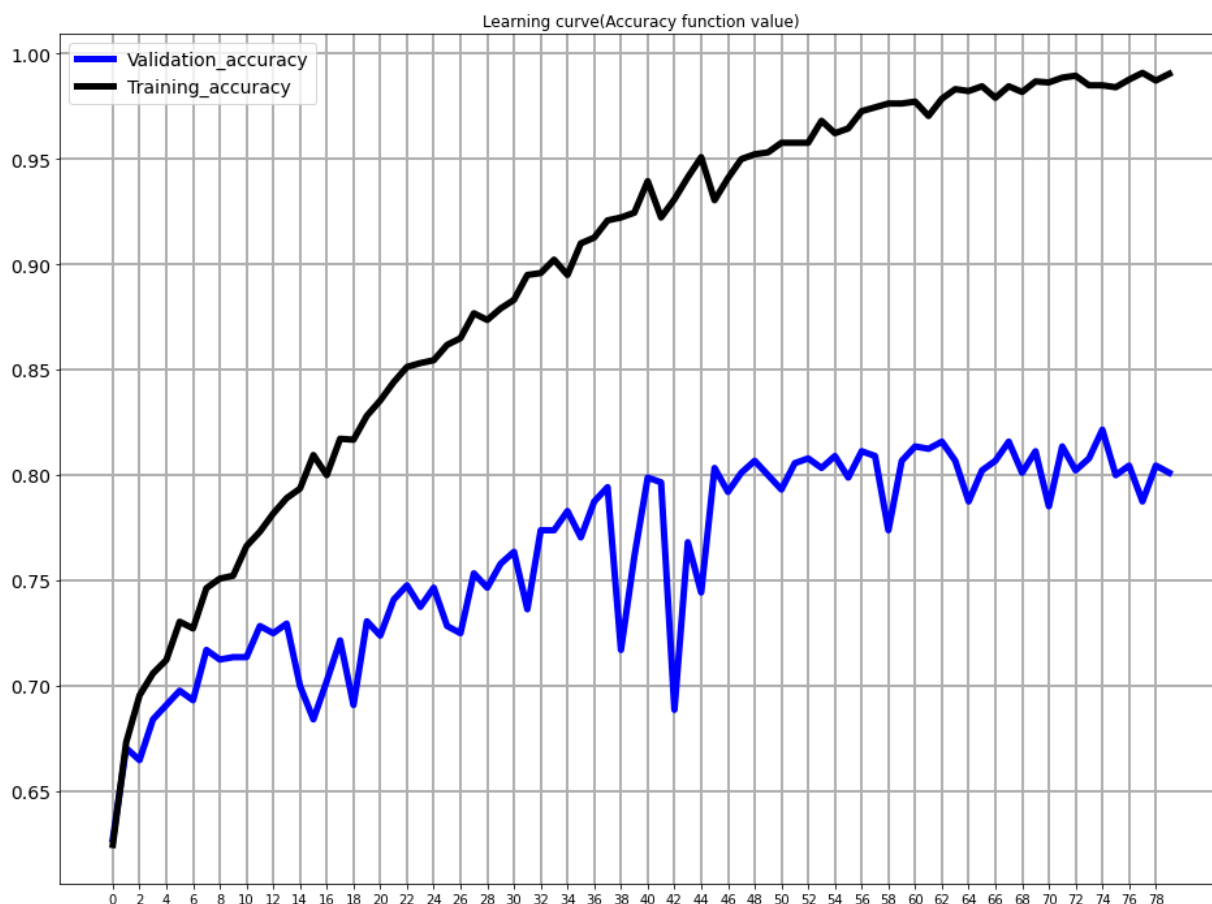


Рисунок 2.16 - График точности обучения, 80 эпох, $lr = .01$

По графику можно увидеть, что точность модели на валидационной выборке достигает уровня в 80-82 % на 45 эпохе обучения, а ближе к 80-ой эпохе обучения стабилизируется, потому что приближается предел точности на обучающей выборке, который приближается к 100%.

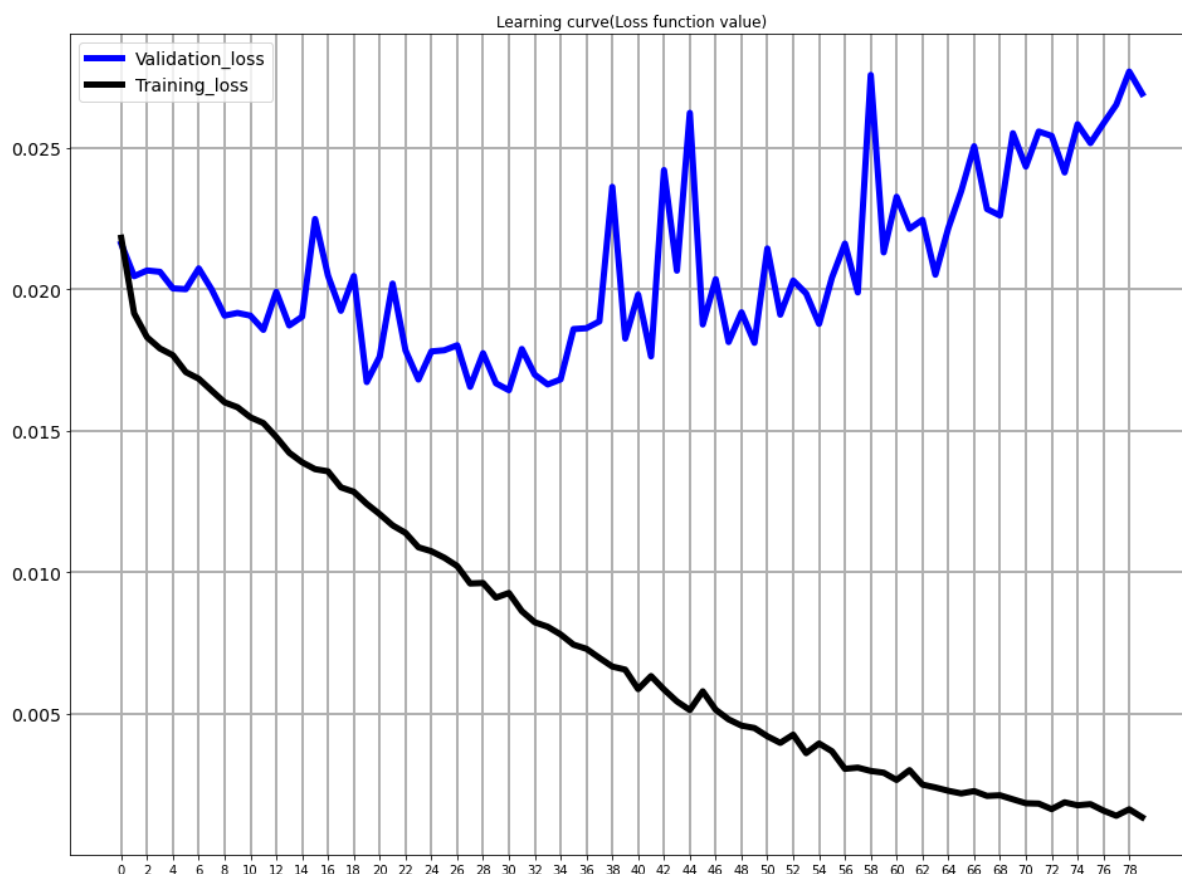


Рисунок 2.17 - График ошибки обучения, 80 эпох, $lr = .01$

Исходя из данных графиков, можно сделать вывод, что точность модели на валидационной выборке примерно на 45й эпохе составляет 80-82%, но уже на 40й эпохе начинается постепенно переобучение — это видно по графику ошибок.

Такое количество эпох возможно мало, чтобы увидеть полную картину обучения нейронной сети, поэтому попробуем увеличить количество эпох до 120.

Попробуем обучить модель с построением графиков по точности (accuracy) и ошибке (loss), со скоростью обучения 0.02.

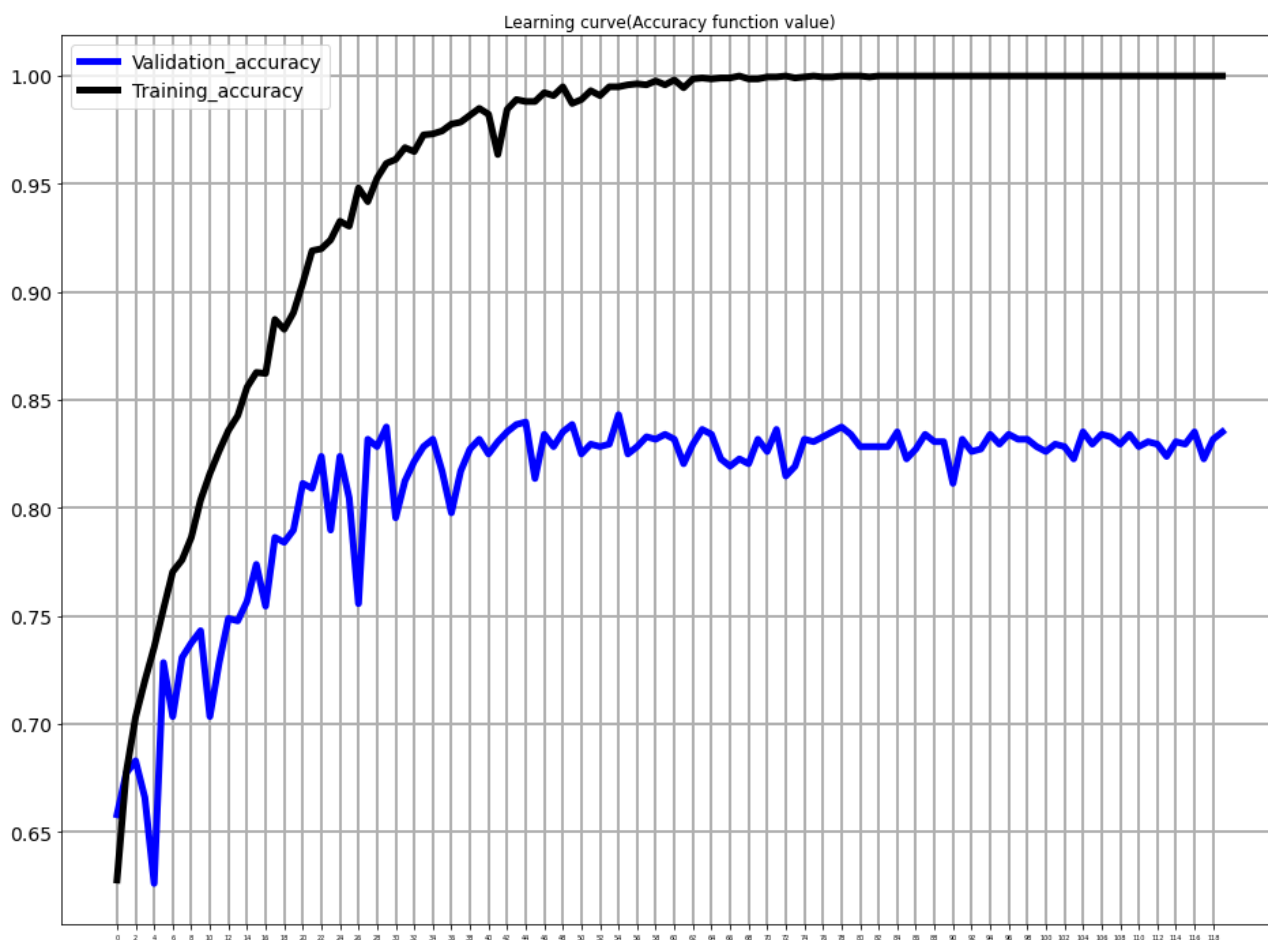


Рисунок 2.18 — График точности обучения, 120 эпох, $lr = .02$

По графику, приведенному выше, можно заметить, что к 65 эпохе точность модели на обучающей выборке составляет 100 % и далее не снижается. На валидационной — уже к 30 эпохе точность принимает значения 80-84% и сохраняет это значение до конца обучения.

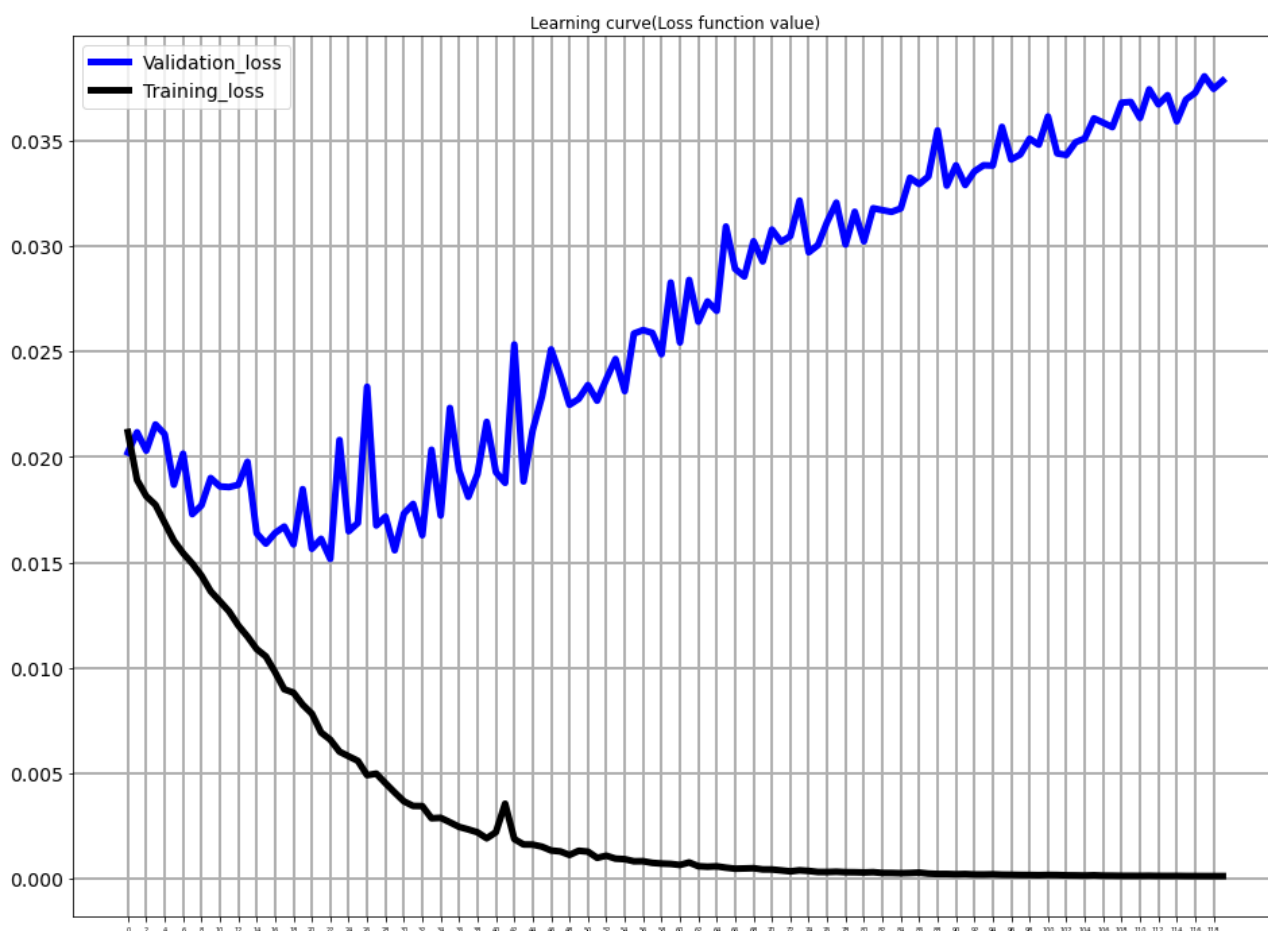


Рисунок 2.19 — График ошибки обучения, 120 эпох, $lr = .02$

По данному графику, приведенному выше, можно заметить, что значение ошибки на обучающей выборке начинает сильно снижаться где-то к 50й эпохе, и одновременно с этим начинает расти ошибка на валидационной выборке.

Поэтому, исходя из обоих графиков, можно сделать вывод, что самое оптимальное количество эпох в данном случае равно 50.

В какой-то момент модель начинает переобучаться на скорости 0.01, и на скорости 0.02. Но со скоростью обучения 0.02 значения точности и ошибки имеют намного меньше выбросов. Также значения точности на 1-2% выше, чем при обучении на скорости 0.01. Время обучения 80 эпох — 104 минуты, 120 эпох — 180 минут.

2.4 Виды тестирования обученной модели нейронной сети

После обучения сверточной нейронной сети, необходимо модель, полученную в процессе обучения, протестировать. Данный процесс можно разделить на два этапа — это тестирование нейронной сети на устойчивость к атакам и второй — нахождение такого порога размытости изображения, при котором обученная модель нейронной сети имеет наименьшую вероятность распознавания правильного класса, либо полностью неправильно классифицирует объект.

В первом случае, когда тестируется нейронная сеть на устойчивость, создаются состязательные примеры с помощью методов FGSM и PGD. Это необходимо, чтобы лучше понять, в чем уязвимость модели и увидеть границы принятия решения при осуществлении классификации.

2.4.1 Тестирование нейронной сети на устойчивость

FGSM — это метод быстрого определения знака градиента. Этот метод необходим для построения состязательного примера. FGSM использует знак градиента для искажения входных данных.

Для построения состязательного примера мы создаем объект FGSM, обеспечивающий желаемую нижнюю и верхнюю границы возмущенных входных данных. В этом случае, поскольку мы работаем с нормализованными входными данными, значения должны быть в диапазоне от -1 до 1. Затем мы вызываем `perturb` (возмущение) для объекта FGSM, предоставляя `epsilon` (величину возмущения), а также метку:

```
perturbed_image_fgsm = fgsm.perturb(normalize(image),epsilon=0.16, target=label).
```

PGD — это метод проекционного градиентного спуска (PGD). В простом объяснении - это итерированная версия FGSM, выполняющая несколько итераций на основе знака градиента, ограниченного фиксированной нормой L2.

2.4.2 Тестирование нейронной сети на минимальное возмущение

Чтобы найти минимальное возмущение, приводящее к неправильной классификации объекта, либо возмущение, которое приводит к минимальному уровню вероятности определения правильного класса, необходимо произвести размытие выборки изображений, предоставляемых к тестированию по Гауссу. В параметры функции, осуществляющей такое размытие, входит размер ядра, или радиус размытия, при изменении которого меняется степень размытости изображения. Протестировав обученную модель нейронной сети на минимальное возмущение, мы можем найти предел размытости изображения, приводящее к неправильной классификации.

2.4.3 Результаты тестирования обученной модели

В данном подразделе будут представлены результаты тестирования обученной модели нейронной сети с помощью определенной выборки смазанных изображений по Гауссу. Смазывание (blur) изображений осуществляется с помощью функции `GaussianBlur()` библиотеки `OpenCV`, реализованной на языке программирования `python`. Данная функция реализуется следующим образом:

```
blur = cv2.GaussianBlur(img, (25, 25), 0),
```

где `cv2` – библиотека `OpenCV`,

`GaussianBlur()` - функция из данной библиотеки,

Аргументы функции `GaussianBlur()`:

`img` – исходное, не смазанное изображение,

аргументы `(25, 25)` — это размер ядра свертки, по которому смазывается наше изображение,

`0` – это значение стандартного отклонения ядра Гаусса.

Установив данное значение в `0`, мы даем библиотеке `OpenCV` возможность автоматически вычислять его, в зависимости от размера ядра. Проведем тестирование модели на смазанных изображениях БПЛА с помощью ядра

свертки размером 25x25, 61x61 и 99x99 пикселей (рисунки 2.20, 2.21, 2.22, 2.23, 2.24).

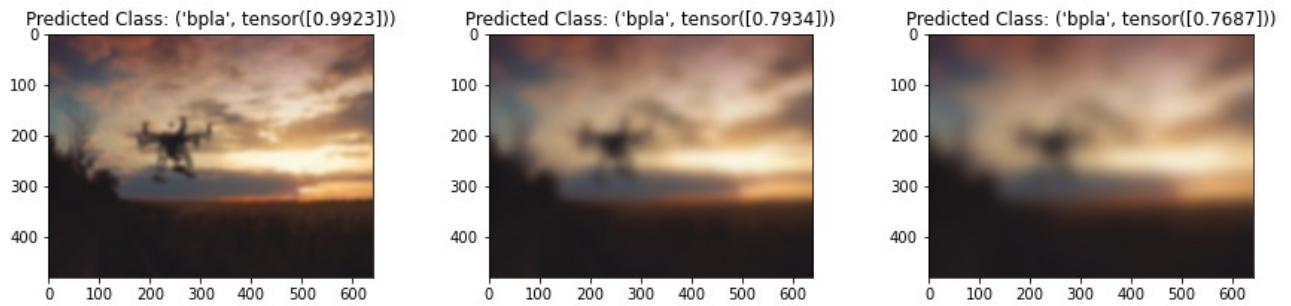


Рисунок 2.20 — Предсказание класса «БПЛА»

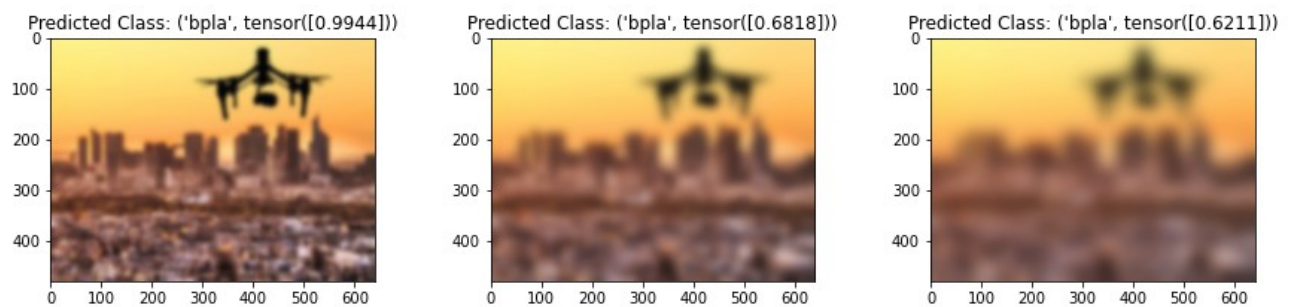


Рисунок 2.21 — Предсказание класса «БПЛА»

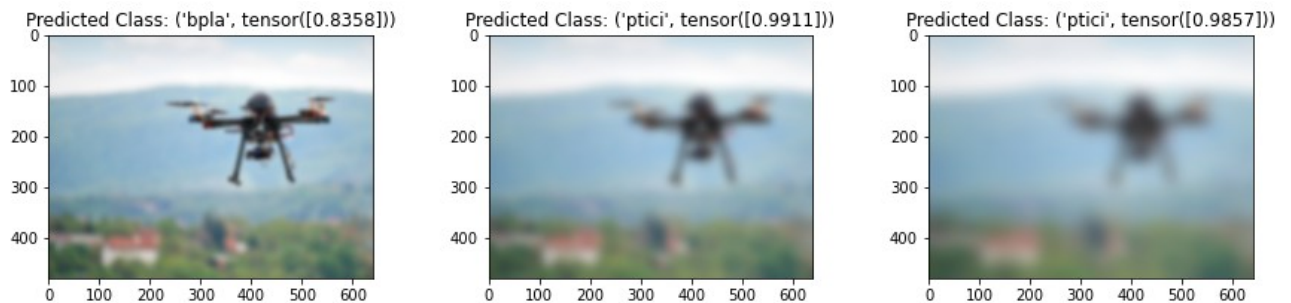


Рисунок 2.23 — Предсказание класса «БПЛА»

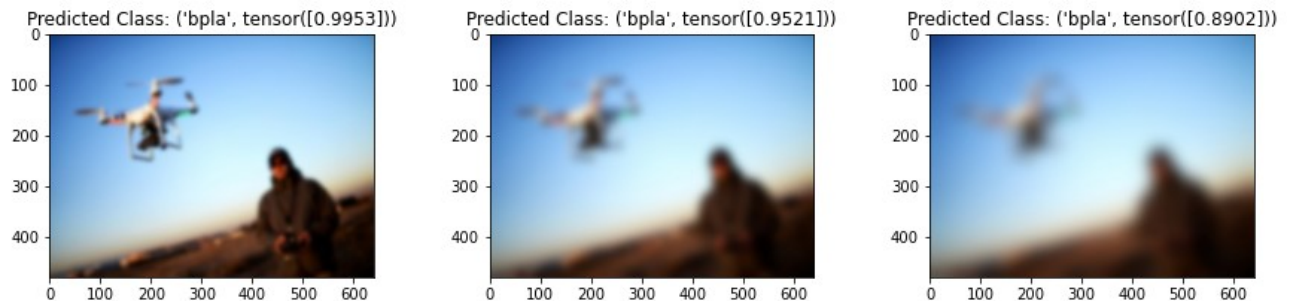


Рисунок 2.23 — Предсказание класса «БПЛА»

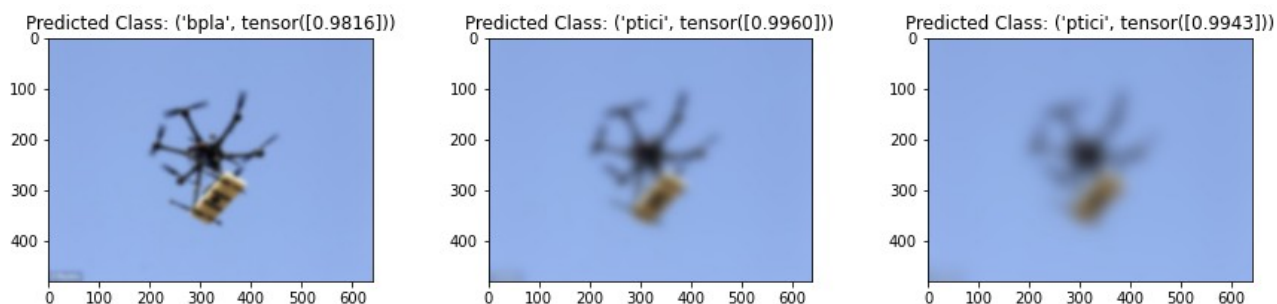


Рисунок 2.24 — Предсказание класса «БПЛА»

Из данных результатов предсказания можно сделать вывод, что обученная модель предсказывает класс объекта на изображении менее точно пропорционально увеличению размера ядра смазывания. Но также присутствуют результаты, которые дают неправильное предсказание уже при ядре смазывания 31x31. Это видно на рисунках 2.23 и 2.24.

Проведем тестирование модели на смазанных изображениях птиц с помощью ядра свертки размером 25x25, 61x61 и 99x99 пикселей (рисунки 2.25, 2.26, 2.27, 2.28, 2.29, 2.30).

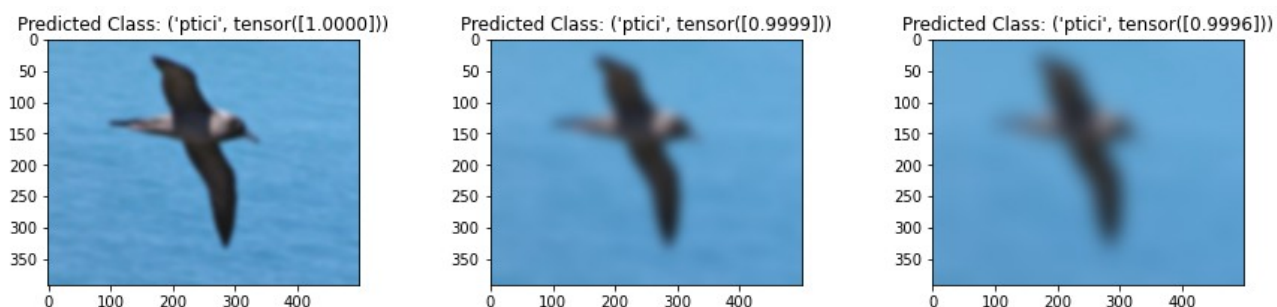


Рисунок 2.25 — Предсказание класса «Птица»

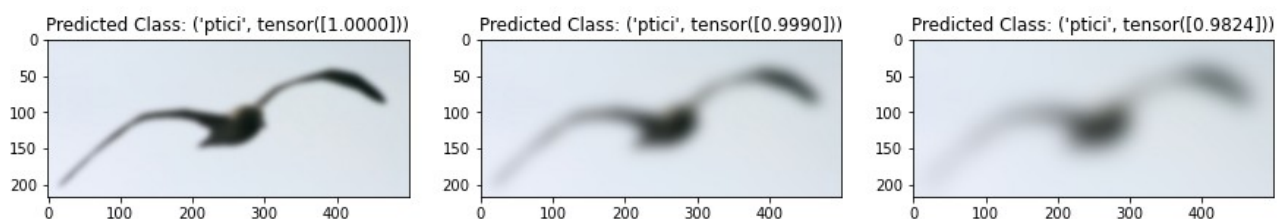


Рисунок 2.26 — Предсказание класса «Птица»

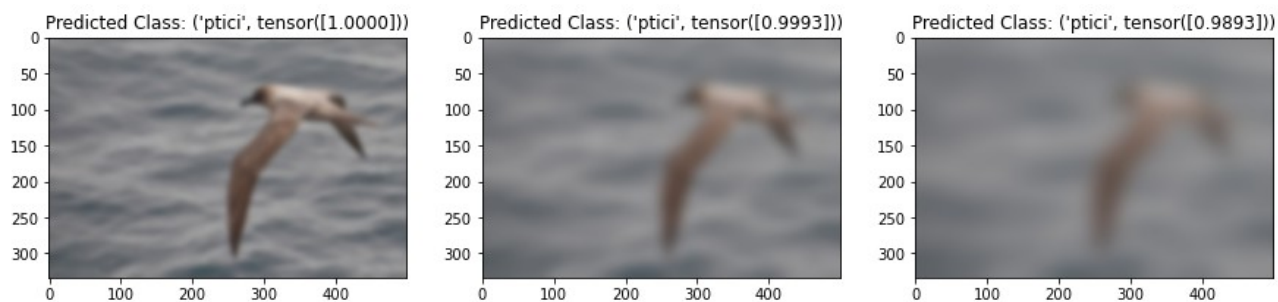


Рисунок 2.27 — Предсказание класса «Птица»

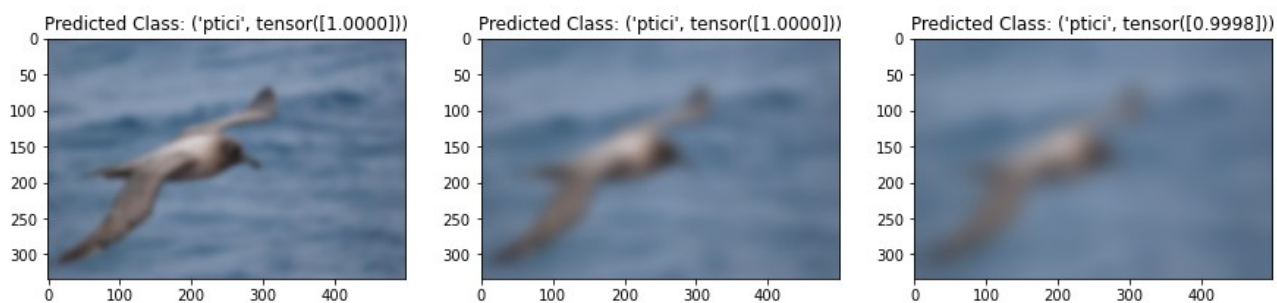


Рисунок 2.28 — Предсказание класса «Птица»

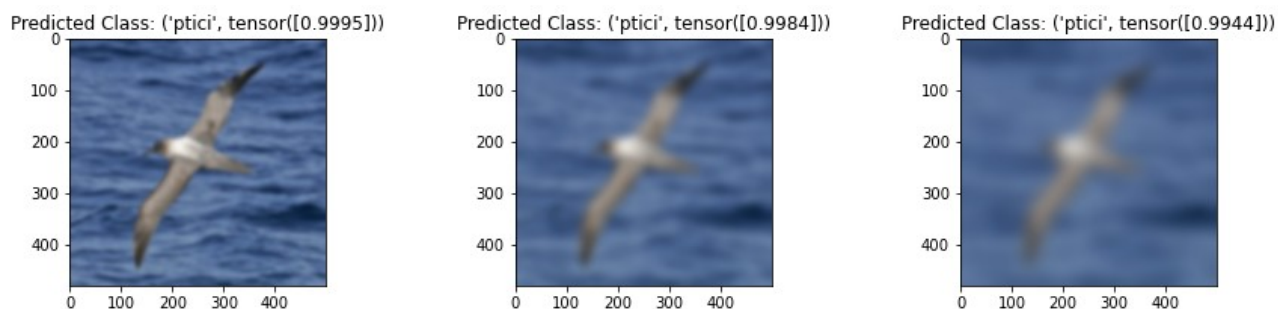


Рисунок 2.29 — Предсказание класса «Птица»

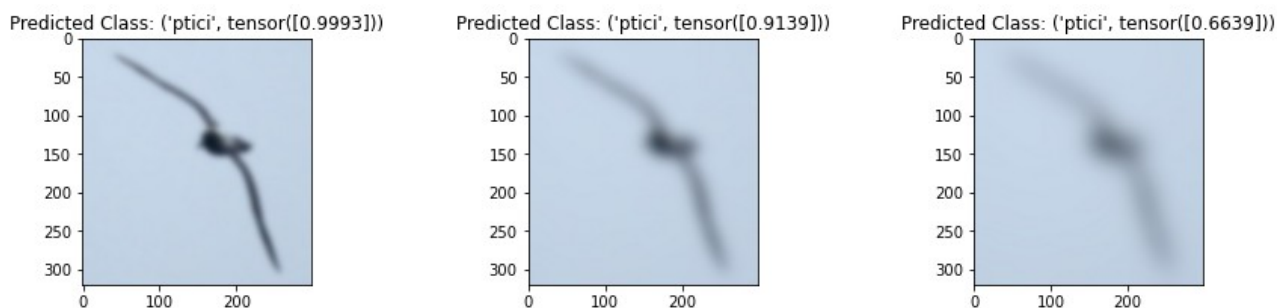


Рисунок 2.30 — Предсказание класса «Птица»

В целом предсказание класса «Птица» на смазанных изображениях с помощью ядра свертки размером 25x25, 61x61 и 99x99 пикселей показало более

лучшие результаты, чем предсказание класса «БПЛА». Обученная модель не ошибается в предсказаниях, но снижает вероятность при предсказании класса.

Попробуем увеличить размер ядра свертки и предскажем класс смазанных изображений с помощью ядра свертки размером 25x25, 61x61, 99x99 и 151x151 пикселей (рисунок 2.31, 2.32, 2.33).

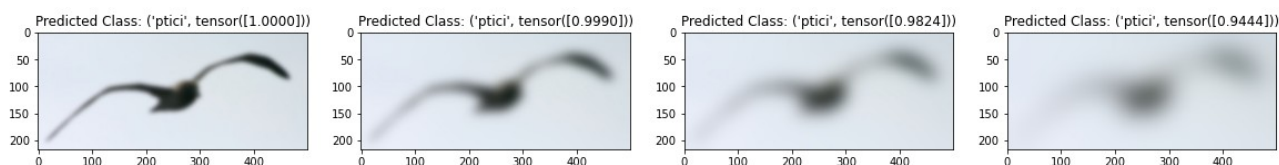


Рисунок 2.31 — Предсказание класса «Птица»

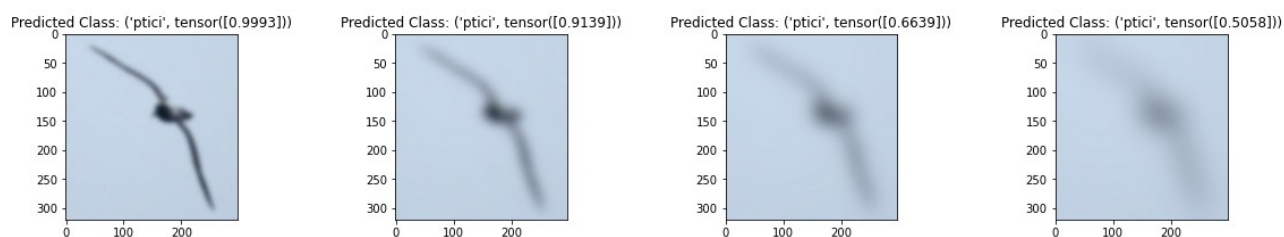


Рисунок 2.32 — Предсказание класса «Птица»

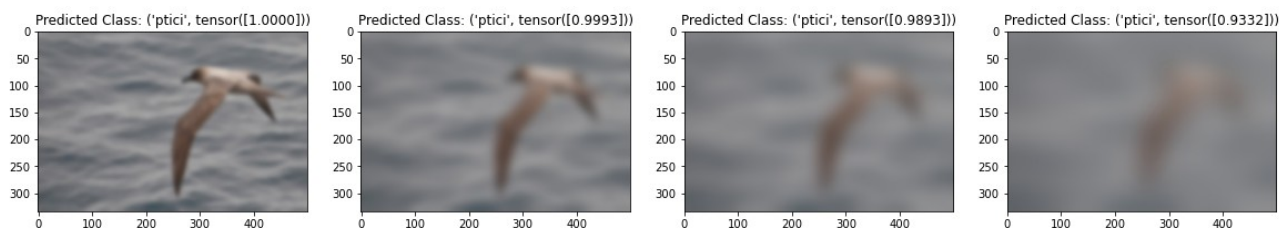


Рисунок 2.33 — Предсказание класса «Птица»

2.4 Выводы по разделу 2

Таким образом, можно сделать вывод, что обученная модель нейронной сети практически не ошибается в предсказаниях птиц, только снижает вероятность. Отсюда можно сделать вывод, что обученная модель отлично распознает птиц, но имеет небольшие сложности в распознавании БПЛА, при смазывании изображений.

ЗАКЛЮЧЕНИЕ

В процессе выполнения данной научно-исследовательской работы был проведен эксперимент по распознаванию малогабаритных беспилотных летательных аппаратов с изображений с помощью метода машинного обучения – классификации. В качестве исходных данных были выбраны изображения двух классов – птица и малогабаритные БПЛА. В каждой из выборок находилось по 1540 изображений, с разделением на обучающую и валидационную – 70 к 30 %. Максимальная точность модели составила 82-85%. Модель неоднократно тестировалась на выборках различного размера. Также в процессе эксперимента изменилась архитектура нейронной сети, то есть модели. Весь этот процесс может означать, что цель магистерской научно-исследовательской работы была достигнута в процессе функционального тестирования, и достигнута максимальная точность распознавания для конкретной модели нейронной сети.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гафаров Ф.М., Галимянов А.Ф . Искусственные нейронные сети и приложения: учеб. пособие / – Казань: Изд-во Казан. ун-та, 2018. – 121 с.
2. Вакуленко С.А., Жихарева А.А. Практический курс по нейронным сетям – СПб: Университет ИТМО, 2018. – 71 с.
3. Розенблатт, Ф.Г. Принципы нейродинамики / – М.: Мир, 1965. – 480 с.
4. Джон Д. Келлехер Глубокое обучение. Самый краткий и понятный курс / — Москва : Эксмо, 2022. — 160 с.
5. Минский, М. Перцептроны / М. Минский, П. Пейперт – М.: Мир, 1971. – 261 с.
6. Лакшманан В., Робинсон С., Мунн М. Машинное обучение. Паттерны проектирования: Пер. с англ./- СПб.: БХВ-Петербург, 2022. - 448 с.
7. Траск Э. Грокаем глубокое обучение. — СПб.: Питер, 2019 - 352 с.
8. Y. LeCun and Y. Bengio: Convolutional Networks for Images, Speech, and Time-Series, in Arbib, M. A. (Eds), The Handbook of Brain Theory and Neural Networks, MIT Press, 1995.
9. Zeiler M.D. Deconvolutional Networks / M.D. Zeiler, D. Krishnan, G.W.Taylor, R. Fergus // Proc. of IEEE Computer Vision and Pattern Recognition,CVPR'2010. - 2010. - P. 2528-2535.
10. Флах, П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных : учебник / П. Флах. - 2-е изд. - Москва.:ДМК Пресс, 2023. - 401 с. - Текст : электронный. - URL: <https://znanium.ru/catalog/product/2085038> (дата обращения: 01.05.2024).
11. Галушкин, А.И. Нейрокомпьютеры / А.И. Галушкин – М.: ИПРЖР, 2000. – С. 179.
12. Лимановская, О. В. Основы машинного обучения : учебное пособие / О. В. Лимановская, Т. И. Алферьева ; Мин-во науки и высш.

образования РФ. - Екатеринбург : Изд-во Уральского ун-та, 2020. - 88 с. -
Текст : электронный. - URL: <https://znanium.com/catalog/product/1960910>
(дата обращения: 09.05.2024).

13. Шалев-Шварц Ш., Бен-Давид Ш. Идеи машинного обучения: от теории к алгоритмам / пер. с англ. А. А. Слинкина.– М.: ДМК Пресс, 2019.– 436 с.: ил.

14. Шакла Н. Машинное обучение и TensorFlow.- СПб.: Питер, 2019.
- 336 с.: ил.

ПРИЛОЖЕНИЕ А

(обязательное)

Использование искусственных нейронных сетей для распознавания БПЛА

Листинг программы

```
import torch

import torch.nn as nn

import torchvision

import torchvision.transforms as transforms

import numpy as np

import matplotlib.pyplot as plt

from torchvision import datasets

from torch.utils.data import DataLoader, SubsetRandomSampler

import random

from PIL import ImageFile

ImageFile.LOAD_TRUNCATED_IMAGES = True

from sklearn.model_selection import LearningCurveDisplay,
learning_curve

from sklearn.metrics import accuracy_score, make_scorer

# Сначала определим на каком устройстве будем работать – GPU или
CPU
```

```
device = torch.device('cuda:0' if torch.cuda.is_available() else  
'cpu')
```

```
# Параметр train указывает какую часть набора будем загружать  
(True - обучающую, False - тестовую)
```

```
# Параметр transform указывает какие преобразования с данными  
необходимо проделать
```

```
# это могут быть обрезка, повороты, изменения яркости и др.
```

```
# загрузим отдельно обучающий набор
```

```
data_transform = transforms.Compose([  
    transforms.Resize((128, 128)),  
    transforms.ToTensor(),  
])
```

```
train_dataset = datasets.ImageFolder(root='viborki/train',  
transform=data_transform)
```

```
train_loader = DataLoader(train_dataset, batch_size=32,  
shuffle=True)
```

```
val_dataset = datasets.ImageFolder(root='viborki/val',  
transform=data_transform)
```

```
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
```

```
# посмотрим какие классы содержатся в наборе
```

```

train_dataset.classes

# сохраним названия этих классов

class_names = train_dataset.classes

# каждая итерация обращения к созданному DataLoader возвращает
batch данных и их классы

inputs, classes = next(iter(train_loader))

inputs.shape

classes

# Функция для отображения случайной выборки

def show_random_samples(dataset, num_samples=5):

    class_names = dataset.classes

    # Получаем случайные индексы из тренировочного датасета

    random_indices = random.sample(range(len(dataset)),
num_samples)

    # Создаем подзагрузчик данных для случайной выборки

    random_loader = DataLoader(dataset, batch_size=num_samples,
sampler=SubsetRandomSampler(random_indices))

    # Получаем один батч из случайной выборки

    images, labels = next(iter(random_loader))

    # Преобразование тензоров в изображения

    images = images.numpy()

    # Вывод изображений

```

```

fig, axes = plt.subplots(1, num_samples, figsize=(15, 3))

for i in range(num_samples):

    axes[i].imshow(np.transpose(images[i], (1, 2, 0)))

    axes[i].set_title(f'Class:
{class_names[labels[i].item()]}')

plt.show()

# Используйте функцию, передавая ваш тренировочный датасет
show_random_samples(train_dataset, num_samples=5)

class ConvNet(nn.Module):

    def __init__(self, num_classes=2):

        super(ConvNet, self).__init__()

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1,
padding=1)

        #self.sigmoid = nn.Sigmoid()

        self.leaky_relu = nn.LeakyReLU()

        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=1,
padding=1)

        self.fc1 = nn.Linear(128 * 32 * 32, num_classes)

```

```

def forward(self, x):

    x = self.conv1(x)

    x = self.leaky_relu(x)

    x = self.pool(x)

    x = self.conv2(x)

    x = self.leaky_relu(x)

    x = self.pool(x)

    x = x.view(x.size(0), -1)

    x = self.fc1(x)

    return x

```

Зададим количество эпох обучения (каждая эпоха прогоняет обучающий набор 1 раз)

```
num_epochs = 120
```

```
num_classes = 2
```

создаем экземпляр сети

```
net = ConvNet(num_classes).to(device)
```

Задаем функцию потерь и алгоритм оптимизации

```
lossFn = nn.CrossEntropyLoss()
```

```

optimizer = torch.optim.SGD(net.parameters(), lr=0.02)

# создаем цикл обучения и замеряем время его выполнения

train_loss_list = []

train_acc_list = []

val_loss_list = []

val_acc_list = []

import time

t = time.time()

for epoch in range(num_epochs):

    train_loss = 0

    train_acc = 0

    val_loss = 0

    val_acc = 0

    for i, (images, labels) in enumerate(train_loader):

        images = images.to(device)

        labels = labels.to(device)

        # прямой проход

        outputs = net(images)

        # вычисление значения функции потерь

        loss = lossFn(outputs, labels)

        train_loss += loss.item()

```

```

# Обратный проход (вычисляем градиенты)

optimizer.zero_grad()

loss.backward()

# делаем шаг оптимизации весов

optimizer.step()

predicted = outputs.max(1)[1]

train_acc += (predicted == labels).sum()

# выводим немного диагностической информации

if i%100==0:

    print('Эпоха ' + str(epoch) + ' из ' + str(num_epochs)
+ ' Шаг ' +

        str(i) + ' Ошибка: ', loss.item())

avg_train_loss = train_loss / len(train_loader.dataset)

avg_train_acc = train_acc / len(train_loader.dataset)

# посчитаем точность нашей модели: количество правильно
классифицированных изображений

# поделенное на общее количество тестовых примеров

correct_predictions = 0

num_test_samples = len(val_dataset)

```

```

net.eval()

with torch.no_grad(): # отключим вычисление градиентов, т.к.
будем делать только прямой проход

    for images, labels in val_loader:

        images = images.to(device)

        labels = labels.to(device)

        pred = net(images) # делаем предсказание по пакету

        _, pred_class = torch.max(pred.data, 1) # выбираем
класс с максимальной оценкой

        correct_predictions += (pred_class ==
labels).sum().item()

        loss = lossFn(pred, labels)

        val_loss += loss.item()

        predicted = pred.max(1)[1]

        val_acc += (predicted == labels).sum()

    avg_val_loss = val_loss / len(val_loader.dataset)

    avg_val_acc = val_acc / len(val_loader.dataset)

    print (f'Epoch [{epoch}/{num_epochs}], loss:
{avg_train_loss:.5f} acc: {avg_train_acc:.5f} val_loss:
{avg_val_loss:.5f}, , val_acc: {avg_val_acc:.5f}')

print('Точность модели: ' + str(100 * correct_predictions /
num_test_samples) + '%')

```

```

train_loss_list.append(avg_train_loss)

train_acc_list.append(avg_train_acc)

val_loss_list.append(avg_val_loss)

val_acc_list.append(avg_val_acc)


print(time.time() - t)


plt.figure(figsize=(16,12))

plt.plot(val_loss_list,label='Validation_loss', lw=5, c='b')

plt.plot(train_loss_list,label='Training_loss', lw=5, c='k')

plt.title('Learning curve(Loss function value)')

plt.xticks(size=10)

plt.yticks(size=14)

plt.grid(lw=2)

plt.legend(fontsize=14)

plt.xticks(np.arange(0, 120, 2))

plt.show()


plt.figure(figsize=(16,12))

plt.plot(val_acc_list,label='Validation_accuracy', lw=5, c='b')

```

```

plt.plot(train_acc_list, label='Training accuracy', lw=5, c='k')

plt.title('Learning curve(Accuracy function value)')

plt.xticks(size=10)

plt.yticks(size=14)

plt.grid(lw=2)

plt.legend(fontsize=14)

plt.xticks(np.arange(0, 120, 2))

plt.show()


# Нашу модель можно сохранить в файл для дальнейшего использования
torch.save(net.state_dict(), 'CnNet__.ckpt')


from PIL import Image

model = ConvNet().to(device)

model.load_state_dict(torch.load("CnNet__.ckpt"))

model.eval()

# Путь к вашему изображению
images = ['viborki_/val/bpla/1.jpg',

          'viborki_/val/ptici/00001038.jpg']


# Загрузка изображения

```

```

fig, axes = plt.subplots(1, len(images), figsize=(15, 3))

for image_path, i in zip(images, range(len(images))):

    image = Image.open(image_path)

    # Применение трансформаций к изображению

    transform = transforms.Compose([

        transforms.Resize((128, 128)),

        transforms.ToTensor(),

    ])

    # Преобразование изображения и добавление размерности пакета

    input_image = transform(image).unsqueeze(0)

    # Передача изображения через модель

    model.eval() # Установка модели в режим оценки (выключение
режима обучения)

    with torch.no_grad():

        input_image = input_image.to(device)

        output = model(input_image)

    # Получение предсказания

    _, predicted_class = torch.max(output, 1)

```

```
predicted_class = predicted_class.item()

# Вывод результата

class_names = train_dataset.classes # Замените на свои классы

predicted_class_name = class_names[predicted_class]

# Вывод изображения и предсказания

axes[i].imshow(np.asarray(image))

axes[i].set_title(f'Predicted Class: {predicted_class_name}')

plt.show()
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Использование искусственных нейронных сетей для распознавания БПЛА

Презентация

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ НА ТЕМУ «Использование искусственных нейронных сетей для распознавания малогабаритных БПЛА»

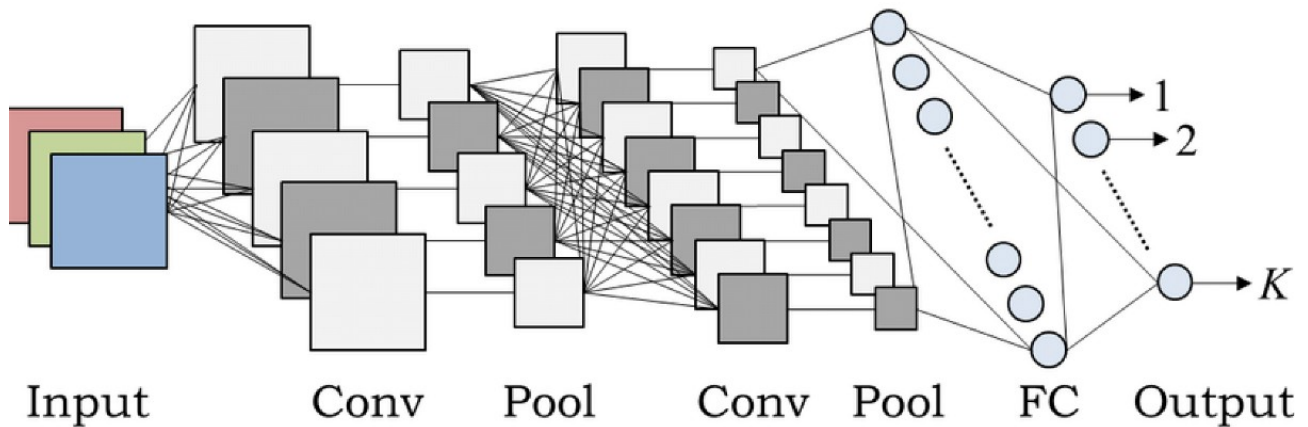
Сост. ст.гр_22ВВим2
Шумкин И.С.
Научный руководитель
д.т.н., профессор Козлов А.Ю.

Цель работы

Цель работы – исследование и разработка модели нейронной сети для классификации изображений, а также повышение точности модели посредством функционального тестирования.

Объект исследования - нейронные сети для задач распознавания изображений.

Сверточная нейронная сеть



3

GPGPU — вычисления общего назначения на GPU

GPGPU — это использование графического процессора (GPU), который обрабатывает вычисления только для компьютерной графики, для выполнения вычислений в приложениях, обрабатываемых центральным процессором (CPU).

Основные технологии GPGPU — CUDA(Nvidia) и OpenCL.



4

