СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Одной из основных задач, возникающих при обработке изображений, является распознавание различных объектов на снимке. В роли объектов может быть различная техника: вертолеты, самолеты, машины, корабли.

Обработка в реальном времени полезна для задач контроля движения судов, поиска объектов на местности в случае аварии и крушения, предотвращение таких ситуаций, картографии.

Распознавание объектов с аэрофотоснимков полезно во время непрерывно ведущегося наблюдения с воздуха. В этом случае можно вести наблюдение с воздуха и в автоматическом режиме выдавать информацию о происходящем на земле.

К примеру, в настоящее время один из способов разведки территорий – съемка с беспилотных летательных аппаратов. Человек в ручном режиме не всегда может обработать информацию, которая непрерывным потоком поступает с беспилотника, ведущего разведку, поэтому весь поток информации нужно обрабатывать автоматически.

В случае разведки летательной техники важным является определение моделей самолетов, стоящий в аэропорту. Такая классификации в случае военной техники поможет сделать вывод о готовящейся тактике и силах стороны, территория которой разведывается.

Во время выполнения выпускной квалификационной работы был разработан метод распознавания летательных аппаратов с аэрофотоснимков с использованием нейронных сетей.

1 Основная часть

1.1 Формализованная постановка задачи

Цель работы – разработать метод распознавания летательных аппаратов с аэрофотоснимков.

Для достижения поставленной цели требуется выполнить следующие задачи:

- провести анализ существующих программных подходов распознавания летательных аппаратов с аэрофотоснимков (проведено выше);
- разработать метод распознавания летальной техники на аэрофотоснимках;
- реализовать спроектированный метод;
- провести исследование точности распознавания модели на тестовой выборке при различных подходах к обучению.

На вход методу подается изображение со снимком аэропорта, где находятся летательные аппараты одного из 20 классов. Результатом работы метода является распознанные на изображении самолеты и их модели. Требуемая точность работы метода на тестовом наборе данных должна составлять не менее 75 процентов.

На входное изображение накладываются следующие ограничения:

- изображение сделано в дневное время суток;
- размер изображения 800 на 800 пикселей;
- размер самолетов на изображении более 70 пикселей.

На рисунке 1.1 приведена IDEF-0 диаграмма уровня A0 метода распознавания летательных аппаратов с аэрофотоснимков с использованием нейронных сетей.

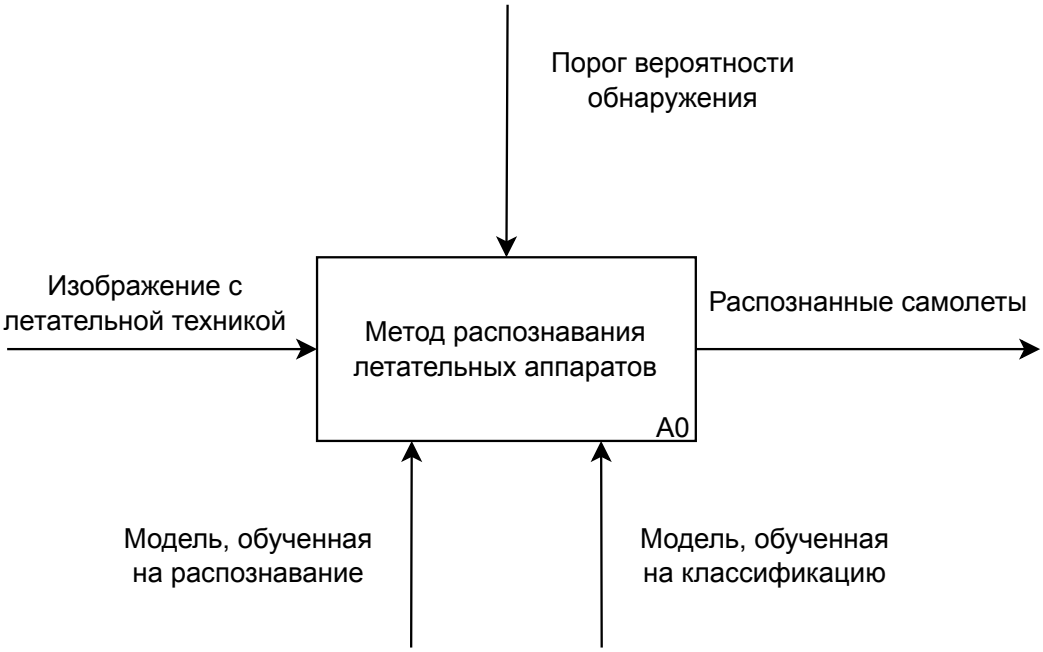

Рисунок 1.1 – IDEF-0 диаграмма метод распознавания летательных аппаратов с аэрофотоснимков

IDEF-0 диаграмма метода распознавания летательной техники с аэрофотоснимков уровня A1 приведена на рисунке 1.2.

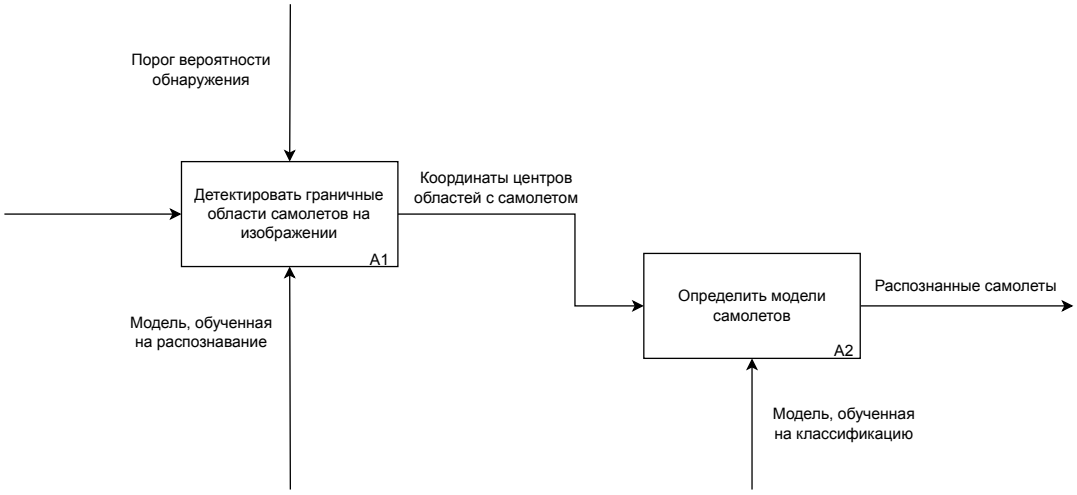

Рисунок 1.2 – IDEF-0 диаграмма уровня A1

Решение задачи распознавания самолетов на аэрофотоснимках было поделено на две подзадачи: детектирование объектов на изображении и их классификация. Результатом решения задачи детектирования являются ограничительные рамки, в которых предположительно находятся летательные объекты.

Перцептрон плохо подходит для задачи распознавания объектов на изображении в силу того, что для его работы требуется предварительная обработка изображений, он не устойчив к шумам и обучается дольше, чем сверточные и капсульные нейронные сети.

Капсульные нейронные сети в отличие от перцептрона не нуждаются в предварительной обработке изображений, устойчивы к сдвигу изображений и для их обучения нужен набор данных меньшего размера. Недостатком таких сетей является их неустойчивость к шумам на изображении.

Таким образом, лучше всего для решения задачи распознавания летательных аппаратов на изображении подходят сверточные нейронные сети, которые устойчивы к сдвигу и шумам на входном снимке и требуют меньший набор данных для обучения по сравнению с капсульными сетями.

1.2 Сравнение сверточных нейронных сетей

1.2.1 AlexNet

AlexNet по своему принципу не сильно отличается от LeNet. В AlexNet используется больше сверточных слоев, а в слоях субдисретизации используется Max Pooling. В качестве функции активации полносвязного слоя используется ReLU. Общая схема архитектуры сети приведена на рисунке 1.3.

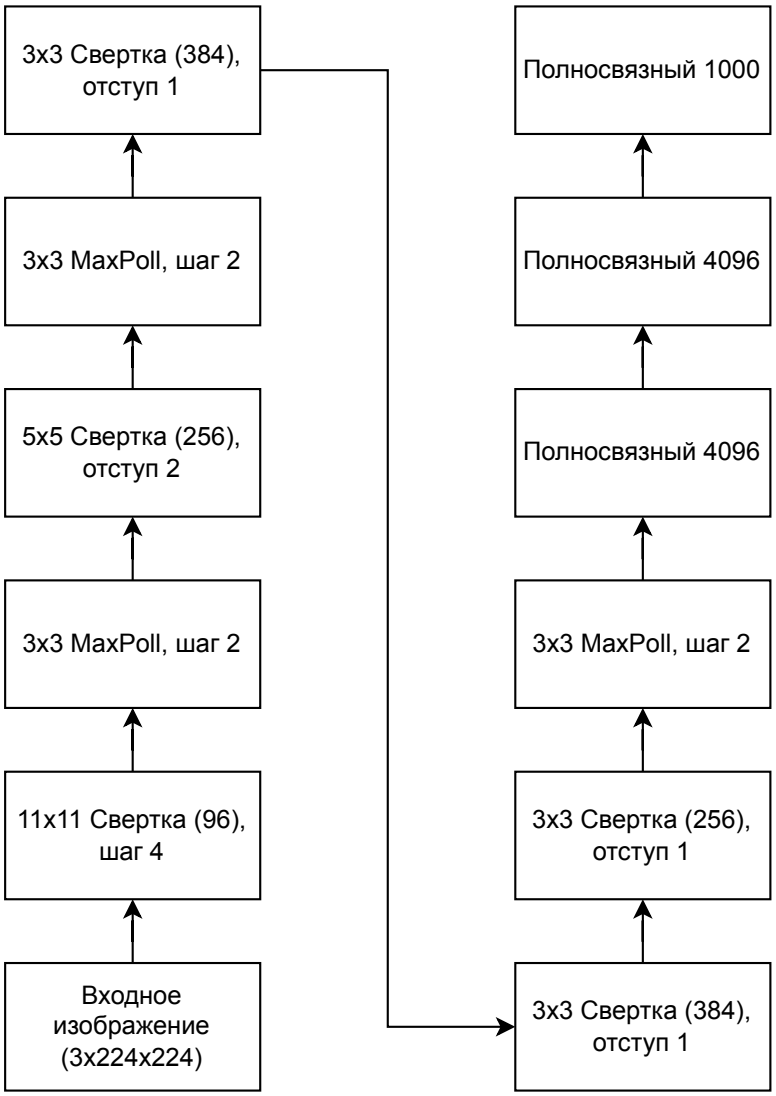

Рисунок 1.3 – Архитектура AlexNet

Преимуществом данной архитектуры является высокая точность распознавания – в 2012 AlexNet показала рекордный результат в точности распознания 1000 различных объектов в соревновании ImageNet.

Главным недостатком является большое число параметров, использующихся при обучении, – около 60 миллионов [18]. Такое количество параметров требует значительно большие вычислительные мощности и память в сравнении с LeNet. Также потребуется больше элементов в обучающей выборке, чтобы корректно настроить эти параметры.

1.2.2 GoogLeNet

В GoogLeNet было введено понятие Inception блока, формальное представление которого приведено на рисунке 1.4.

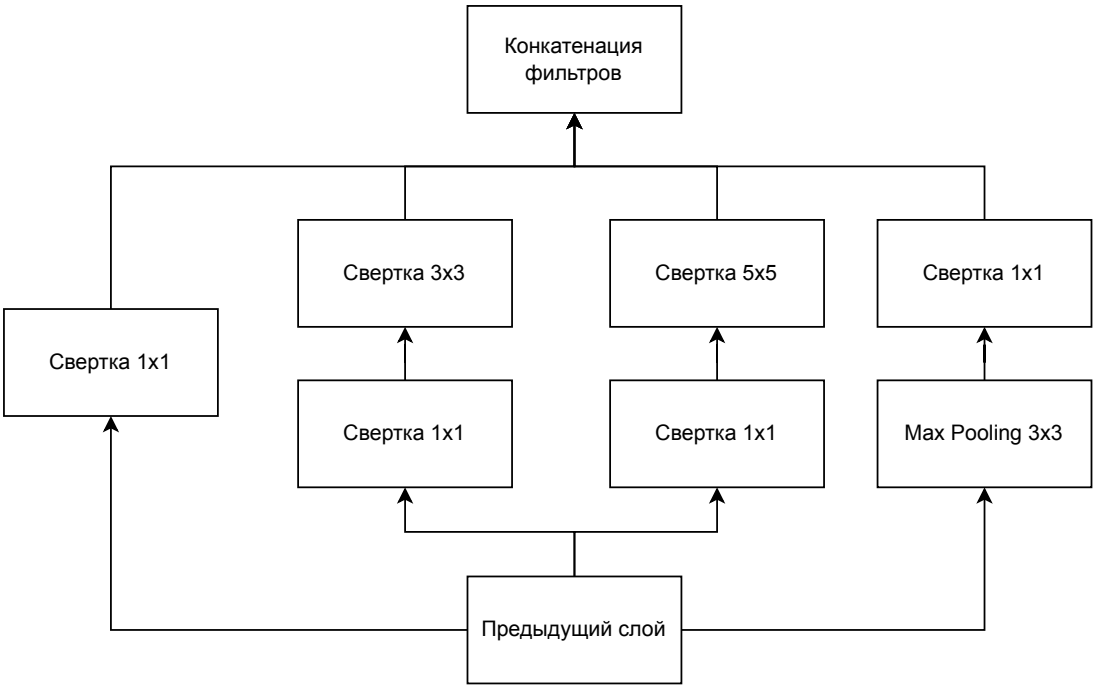

Рисунок 1.4 – Формальное представление Inception блока

В таком блоке выходы с предыдущего слоя параллельно обрабатываются сверткой 1 на 1, блоком из сверток 1 на 1 и 3 на 3, блоком из сверток 1 на 1 и 5 на 5, блоком из Max Pooling размером 3 на 3 и свертки 1 на 1. Далее выходы со всех этих блоков объединяются и передаются дальше.

Фильтр 1 на 1 используется для уменьшения количества параметров в сверточных слоях и ускорения вычислений. Он позволяет сократить количество каналов входного изображения до более низкого значения, что уменьшает количество вычислений при применении более крупных фильтров. Кроме того, фильтры 1 на 1 могут использоваться для комбинирования признаков разных каналов, что улучшает качество классификации [19].

Такой подход имеет два основных преимущества [19]:

- возможность увеличения количества блоков на каждом этапе без неконтролируемого роста вычислительной сложности;
- визуальная информация обрабатывается в различных масштабах, а затем агрегируется, чтобы на следующем этапе можно было абстрагировать признаки из разных масштабов одновременно.

Использование таких блоков в архитектуре GoogLeNet позволяет сократить число параметров примерно в 12 раз по сравнению с AlexNet, при этом точность классификации сети не падает [19].

Для борьбы с затуханием градиента в GoogLeNet используется следующий прием: помимо классификатора в конце нейронной сети добавляется еще один после третьего Inception блока и еще один после 6. Во время обучения функция потерь считается не только по значениям из последнего классификатора, но и по 2 добавленным, домжноженным на некоторый коэффициент. Итоговое выражение для подсчета функции потерь определяется следующей зависимостью 1.1

$$TL = l + k * (l_1 + l_2), \tag{1.1}$$

где TL – общее значение функции потерь, которое нужное уменьшать во время обучения, l – значение функции потерь, посчитанное по выходам из последнего классификатора, k – некоторый коэффициент значимости, на который домножаются значения функции потерь, посчитанные по выходам двух других классификаторов, в GoogLeNet этот коэффициент равен 0.3, l_1 и l_2 – это значения функции потерь посчитанные по выходам классификаторов, добавленный в начале и в середине соответственно.

1.2.3 SimpLeNet

Архитектура нейронной сети SimpLeNet состоит из сверточных слоев только размером 3 на 3 пикселя. Для уменьшения размеров изображения используются слои пуллинга с ядром размера 2 на 2 пикселя и шагом два пикселя. После каждого из сверточных слоев применяется слой нормализации и функция активации ReLU.

Благодаря использованию сверток только размером 3 на 3, в такой нейронной сети меньше обучаемых параметров по сравнению с сетями, где используются свертки больше размеров. Согласно исследованиям [21] в такой сети используется в два с половиной раза меньше вычислительных операций по сравнению с GoogLeNet и в 15 раз меньше обучаемых параметров по сравнению с AlexNet.

Использование меньшего числа параметров приводит к уменьшению скорости обучения нейронной сети, а также к меньшим ограничениям на вычислительные ресурсы при использовании модели, что важно, так как все изображения, которые поступают на вход модели, сделаны с беспилотных летательных аппаратов и могут на них же обрабатываться.

1.2.4 Yolo

Нейронная сеть Yolo предназначена для детектирования объектов на изображении. Результатом работы такой сети являются граничные области найденных объектов. В отличие от других сетей Yolo способна детектировать сразу несколько объектов на изображении.

Алгоритм работы yolo состоит из двух частей: сверточной нейронной сети и последующей обработки ее результатов. Сверточная нейронная сеть должна выделить ограничивающие рамки объектов на изображении. Задача последующей обработки заключается в том, чтобы провести пороговую фильтрацию полученных из сверточной сети ограничивающий рамок на основе вероятности нахождения в них объекта, а также исключить из рассмотрения схожие рамки.

Сверточная нейронная сеть разбивает входное изображения на ячейки некоторого размера и для каждой из них предсказывает следующий набор параметров:

- p – вероятность нахождения центра какого-нибудь объекта в этой ячейке, принимает значения в диапазоне от нуля до одного;
- x – центр объекта по оси абсцисс внутри этой ячейки, левый верхний угол имеет координаты 0 и 0, правый нижний – 1 и 1, соответственно значение x находится в диапазоне от нуля до одного;
- y – центр объекта по оси ординат;
- w – ширина ограничительной рамки, поделенная на ширину ячейки, принимает значения больше нуля;
- h – высота ограничительной рамки, поделенная на высоту ячейки, принимает значения больше нуля;
- c_i – вероятность того, что в рамке находится объект i-го класса.

Таким образом, выход из сверточной нейронной сети имеет размерность CxCx$(5 + \text{classes})$, где C – число размер сетки по высоте и ширине, $classes$ – число классов, которые распознает нейронная сеть.

Координаты центра по оси абсцисс и ординат ограничительной рамки получаются из формул 1.2 и 1.3 соответственно

$$x = \sigma(t_{\mathrm{x}}) + c_{\mathrm{x}}, \tag{1.2}$$

$$y = \sigma(t_{\mathrm{y}}) + c_{\mathrm{y}}, \tag{1.3}$$

где x и y – координаты центра рамка, c_{x} и c_{y} – координаты левого верхнего угла ячейки, в которой находится центр рамки, t_{x} и t_{y} – выходы из нейронной сети.

Высота и ширина ограничительной рамки определяются по формулам 1.4 и 1.5 соответственно

$$h = p_{\mathrm{h}} e^{t_{\mathrm{h}}}, \tag{1.4}$$

$$w = p_{\mathrm{w}} e^{t_{\mathrm{w}}}, \tag{1.5}$$

где t_{h} и t_{w} – выходы из нейронной сети, p_{h} и p_{w} – высота и ширина якоря, который используются для определения, какие ограничивающие рамки должны быть выданы алгоритмом для конкретных объектов на изображении.

В нейронной сети Yolo v3 входное изображение разделяется на сетка трех разных масштабов и для каждой ячейки используется по три якоря.

Для удаления схожих рамок на этапе обработки выходов сверточной сети используется алгоритм non maximum supperssion, суть работы которого заключается в следующем:

- ограничивающие рамки сортируются по вероятности нахождения в них объекта;
- выбирается рамка с самой высокой вероятностью и она сохраняется в окончательном списке ограничивающих рамок;
- после этого отбрасываются все рамки, которые нашли объект того же класса и имеют значение перекрытия с выбранной на предыдущем шаге рамкой больше порогового;
- процесс продолжается, пока не будут обработаны все оставшиеся рамки.

Перекрытие двух рамок высчитывается делением площади их пересечения на площадь их объединения.

1.2.5 Вывод

Архитектура CapsNet плохо подходит для решения задачи распознавания летательных аппаратов с аэрофотоснимков, так как не устойчива к шумам во входном изображении.

Архитектура LeNet менее глубокая по сравнению с другими архитектурами, что ограничивает ее способность к достижению большей точности классификации.

Нейронные сети AlexNet и GoogLeNet плохо подходят для решения поставленной задачи в силу большего числа обучающих параметров по сравнению с другими архитектурами, так как большее число параметров приводит к большим ограничениям на вычислительные ресурсы, увеличению времени обучения и требуемого размера обучающей выборки.

Таким образом, для решения задачи классификации лучше всего подходит архитектура SimpLeNet за счет использования меньшего числа параметров по сравнению с другими архитектурами, а также за счет использования слоев нормализации, которые ускоряют процесс обучения и повышают точность модели, а для решения задачи детектирования – Yolo.

1.3 Требования к предъявляемые к ПО

На вход метода поступает изображение, на которое накладываются следующие ограничения:

- изображение в формате PNG или JPG;
- размер изображения 800 на 800 пикселей;
- размер самолетов на изображении больше 70 пикселей;
- изображение сделано под углом 90 градусов к поверхности Земли.

Результатом работы метода являются распознанные самолеты, а именно их количество, расположение на входном изображении и их модели.

Программно обеспечение должно предоставлять интерфейс для разработанного метода со следующими функциями:

- возможность выбора и использования одной из заранее обученных моделей;
- возможность выбора и настройки метода обучения модели;
- возможность загрузки аэрофотоснимка с летательной техникой и получение информации о распознанных на нем летательных объектах, а именно их количество, местоположение и модель;
- возможность загрузки изображения с одним самолетом и получение результатов о классе летательной техники на нем.

Обучаемая модель имеет следующую структуру:

- слой свертки с 64 фильтрами, размером ядра 3 на 3 пикселя, шагом и отступом по одному пикселю;
- три аналогичных слоя с 128 фильтрами;
- слой max pooling, с размером ядра 2 на 2 пикселя и шагом 2, размер выходных матриц после этого слоя становится 48 на 48;
- два слоя с 128 фильтрами, размером ядра 3 на 3 пикселя, шагом и отступом по одному пикселю;

- аналогичный слой с 256 фильтрами;
- слой max pooling, с размером ядра 2 на 2 пикселя и шагом 2, размер выходных матриц после этого слоя становится 24 на 24;
- слой свертки с 256 фильтрами, размером ядра 3 на 3 пикселя, шагом и отступом по одному пикселю;
- слой max pooling, с размером ядра 2 на 2 пикселя и шагом 2, размер выходных матриц после этого слоя становится 12 на 12;
- слой свертки с 512 фильтрами, размером ядра 3 на 3 пикселя, шагом и отступом по одному пикселю;
- слой max pooling, с размером ядра 2 на 2 пикселя и шагом 2, размер выходных матриц после этого слоя становится 6 на 6;
- слой свертки с 2048 фильтрами, размером ядра 3 на 3 пикселя, шагом и отступом по одному пикселю;
- слой свертки с 256 фильтрами, размером ядра 3 на 3 пикселя, шагом и отступом по одному пикселю;
- слой max pooling, с размером ядра 2 на 2 пикселя и шагом 2, размер выходных матриц после этого слоя становится 3 на 3;
- слой свертки с 256 фильтрами, размером ядра 3 на 3 пикселя, шагом один пиксель;
- входной слой перцептрона, состоящий из 256 нейронов;
- выходной слой перцептрона, состоящий из 20 нейронов.

После каждого сверточного слоя применяется слой нормализации и функция активации ReLU. Такая архитектура нейронной сети имеет следующие преимущества:

- использование сверточных слоев позволяет выделять различные признаки, такие как границы, формы, текстуры, вне зависимости от их расположения в входном изображении;

- использование пуллинговых слоев позволяет уменьшить размерность изображения с сохранением отличительных признаков;

- несмотря на количество слоев, такая архитектура за счет использования пуллинговых слоев и сверток с ядром 3 на 3 вместо 5 на 5 и больших имеет относительно немного обучаемых параметров: 4910356 в сравнении с шестьюдесятью миллионами в архитектуре AlexNet, которая была описана в главе 1.2.1.

Использование меньшего числа параметров приводит к уменьшению скорости обучения нейронной сети, а также к меньшим ограничениям на вычислительные ресурсы при использовании модели, что важно, так как все изображения, которые поступают на вход модели, сделаны с беспилотных летательных аппаратов и могут на них же обрабатываться.

Для решения задачи детектирования объектов была выбрана нейронная сеть Yolo v3, состоящая из 106 сверточных слоев, среди которых сверточный слои с размером ядра 3 на 3 и 1 на 1 и шагом свертки 1 или 2 пикселя, а также блоки, в которых используется техника пропуска соединений.

Всего такая сеть выдает три матрицы предсказаний для размеров ячеек 25, 50 и 100 пикселей.

Схемы алгоритмов обучения нейронных сетей и прохождения одной эпохи приведены на рисунках 1.5 и 1.6 соответственно.

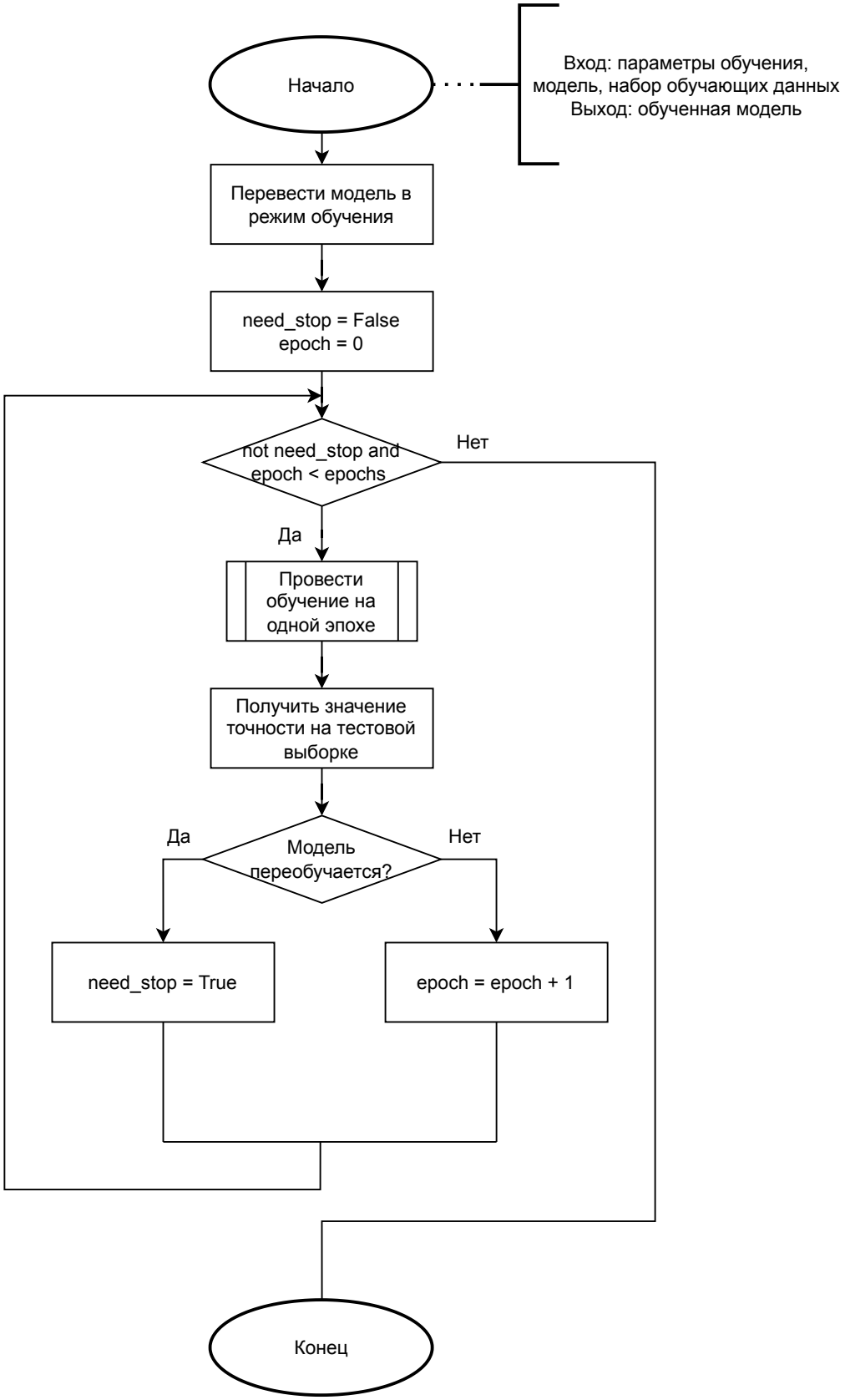

Рисунок 1.5 – Схема алгоритма обучения нейронной сети

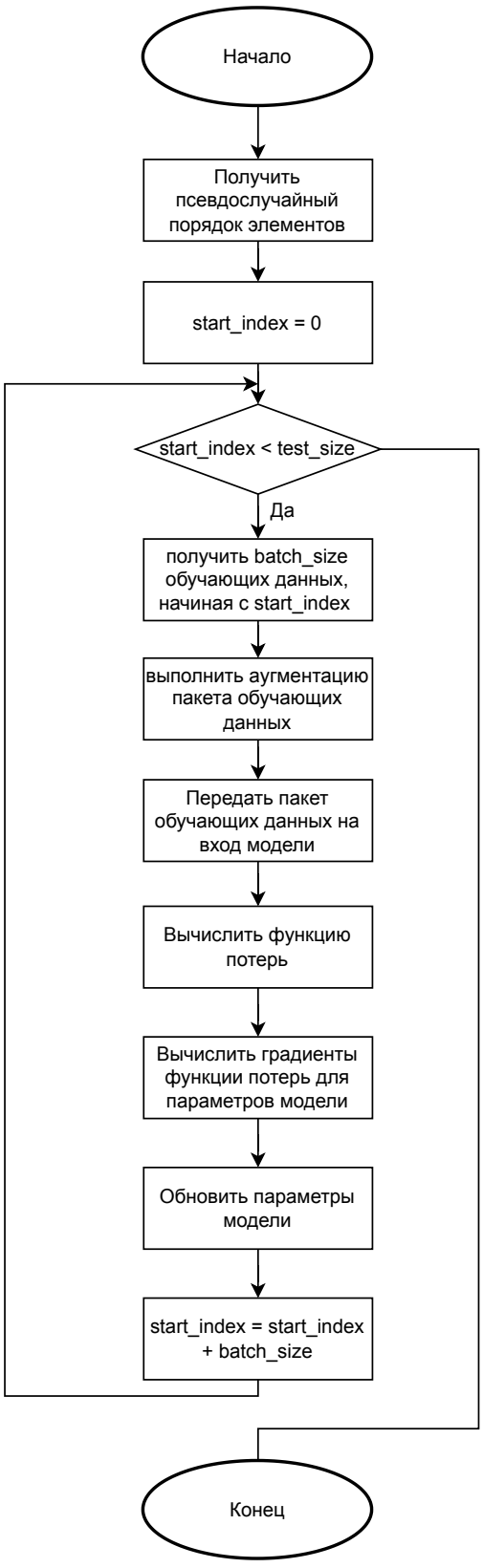

Рисунок 1.6 – Схема алгоритма прохождения одной эпохи

При подсчете точности модели на тестовой выборке важно учитывать ее размер и загружать в оперативную память по частям, каждый раз освобождая выделенные ресурсы.

1.4 Структура программного обеспечения

Программное обеспечение состоит из четырех модулей:

- модуль, реализующий модель нейронной сети распознавания летальной техники;
- модуль, реализующий модель нейронной сети классификации летальной техники;
- модуль, реализующий интерфейс взаимодействия с пользователем;
- модуль пользовательского интерфейса.

Структурная схема взаимодействия модулей разрабатываемого программного обеспечения представлена на рисунке 1.7.

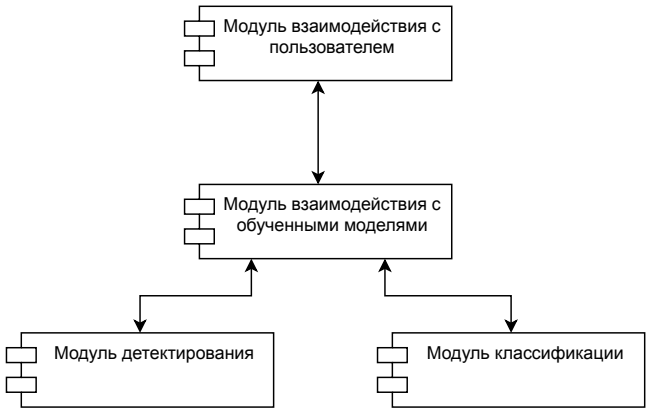

Рисунок 1.7 – Структурная схема взаимодействия модулей

Модули детектирования и классификации используются для обучения моделей и взаимодействия с уже обученными. После завершения обучения параметры модели должны быть сохранены в файл для обеспечения в дальнейшем загрузки модели без обучения.

Модуль взаимодействия с пользователем должен обеспечивать следующие возможности:

- загрузка одной из заранее обученных моделей;
- запуск обучения модели с выбранными параметрами и сохранение его результаты;

- загрузка изображения с летательной техникой и определение ее класса с помощью загруженной или обученной модели;
- загрузка аэрофотоснимка с различными классами летательной техники и определение их классов.

1.5 Набор обучающих данных

Для обучения нейронной сети был выбран набор данных из источника [22], состоящий из 3842 снимков аэропортов, сделанных с беспилотных летательных аппаратов. На этих снимках находится 22341 самолет 20 различных типов.

Пример элемента обучающей выборки приведен на рисунке 1.8.

Рисунок 1.8 – Пример элемента обучающей выборки

Вся выборка была поделена на обучающую и тестовую. Обучающая выборка содержит 20344 самолета, а тестовая – 1997.

1.6 Средства реализации программного обеспечения

В качестве языка программирования был выбран Python. Данный выбор обусловлен тем, что Python имеет множество библиотек, таких как TensorFlow, Keras, PyTorch и Theano, которые предоставляют множество инструментов для создания и обучения нейронных сетей.

В качестве библиотеки для создания нейронной сети был выбрана библиотека PyTorch версии 2.0.0, так как она имеет следующие возможности и инструменты:

- динамический граф вычислений, использование которого облегчает отладку моделей;
- возможность переноса вычислений на GPU;
- набор инструментов для создания различных слоев, из которых складывается архитектура нейронной сети;
- имеет API на языке C++, что позволяет обучить модель с использованием интерпретируемого языка Python, а использовать ее на компилируемом языке C++.

Для работы с большими данными была выбрана библиотека numpy версии 1.21.0, так как она использует оптимизированный код на C, что позволяет выполнять вычисления быстрее, чем с использованием чистого Python, а также потому что классы этой библиотеки интегрируются с библиотекой PyTorch, которая используется для создания нейронной сети.

Для работы с изображениями была выбрана библиотека Pillow версии 8.2.0, так как она позволяет загружать и трансформировать изображения разных форматов, таких как PNG, JPEG, BMP, а также переводить изображения в объекты, которые передаются на вход нейронной сети.

Для создания графического интерфейса использовалась библиотека PyQt версии 5.0, так как она является кроссплатформенной, имеет базовые виджеты, на которых может быть построен интерфейс, а также предоставляет возможность подписки на события, которые генерируются виджетами.

1.7 Реализация программного комплекса

В качестве модели классификации используется сверточная нейронная сеть с 12 слоями свертки и 5 слоями пуллинга, которая соединяется с двухслойным перцептроном. После каждого сверточного слоя за исключением последнего используется слой нормализации и функция активации ReLU.

Перед обучением модели выполняется предобработка всего обучающего набора данных, во время которой нужно разбить все входные изображения по классам, привести все изображения к размеру 96 на 96 пикселей, так как только такое изображение обрабатывается моделью, а также поделить весь набор данных на обучающую и тестовую выборки.

После выполнения предобработки нужно произвести аугментацию для расширения обучающей выборки. К каждому изображения в обучающем множестве применяются следующие преобразования: поворот на 30 и 330 градусов, увеличение яркости в полтора раза, а также гауссово размытие, которое создает фильтр Гаусса с заданным размером и силой размытия, и применяют его к изображению, используя операцию свертки. Коэффициенты в фильтре Гаусса вычисляются по формуле 1.6

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{1.6}$$

где x и y – расстояния от центра ядра по горизонтали и вертикали соответственно, σ – сила размытия.

Обучение модели проводилось на машине с процессором Intel Core i9-10900, 64 гигабайтами оперативной памяти и графической картой NVIDIA GeForce RTX 3080 с 16 гигабайтами памяти типа GDDR6. Время прохождения одной эпохи в среднем занимает 2 минуты 41 секунду.

В качестве оптимизатора функции потерь был выбран Adam, так как он автоматически адаптирует скорость обучения для каждого параметра в зависимости от его градиента, что позволяет более эффективно использовать скорость обучения и ускоряет сходимость.

Обучение модели останавливается, когда точность распознавания на тестовой выборке после прохождения очередной эпохи становится меньше, чем на двух предыдущих, так как это свидетельствует о том, что сеть начала переобучаться.

Код обработки и загрузки обучающего набора данных представлен в листинге 1. Реализация алгоритмов обучения и прохождения одной эпохи представлены на листингах 2 и 3 соответственно. Все листинги находятся в приложении А.

Графики зависимостей точности распознавания на тестовой и обучающей выборках от номера эпохи обучения приведены на рисунке 1.9.

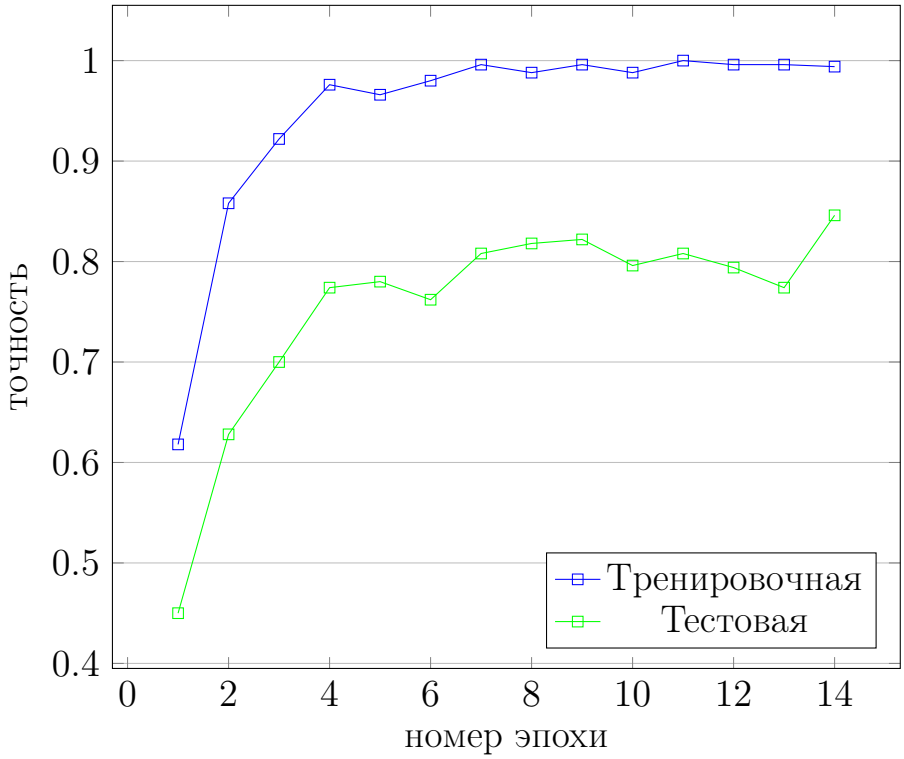

Рисунок 1.9 – Обучение модели

Из графиков видно, что после 7 эпохи модель начинает переобучаться и требуется остановить обучение. Итоговая точность полученной модели на тестовой выборке составила 84 процента.

В качестве модели детектирования была реализована сверточная нейронная сеть, состоящая из 106 слоев. В качестве функции активации была выбрана функция ReLU. Разработанная модель имеет три выходных слоя, каждый из которых имеет размерность (3xSxSx5), где 3 – число якорей, S – размер ячейки, для 3 выходов размеры ячеей будут 25, 50 и 100 соответственно, вектор из пяти элементов состоит из вероятности нахождения центра объекта внутри ячейки, а также координат его центра, высоты и ширины рамки.

1.8 Взаимодействие с разработанным ПО

Взаимодействие с разработанным программным обеспечение осуществляется через графический пользовательский интерфейс. Интерфейс приложения представлен на рисунке 1.10.

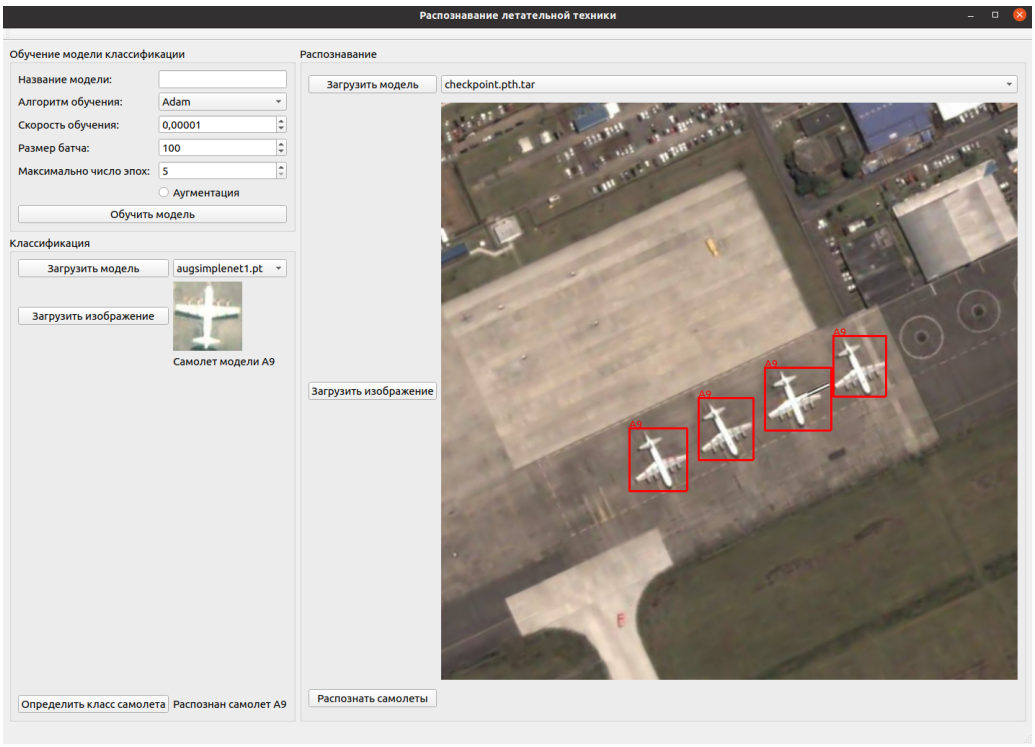

Рисунок 1.10 – Интерфейс приложения

Графический интерфейс поделен на две части. Первая часть позволяет выбрать и загрузить модель обученную на классификацию и проверить ее работу. Вторая позволяет выбрать и загрузить модель, обученную на детектирование, а также проверить ее работу вместе с моделью классификации.

ЗАКЛЮЧЕНИЕ

В рамках производственной практики было разработано программное обеспечение для распознавания летательных аппаратов с использованием нейронных сетей.

Были выбраны и реализованы средства распознавания летательных аппаратов. Для этого были выбраны методы аугментации обучающих данных, направленные на увеличение количества и разнообразия данных для обучения модели.

Были созданы модели для детектирования и классификации летательных аппаратов и реализован алгоритм их обучения. После этого был проведен ряд тестов, чтобы оценить качество модели. Итоговая точность модели классификации на тестовой выборке составила 84 процента. Точность детектирования на тестовой выборке составила 90 процентов.

Кроме того, был создан графический интерфейс, который позволяет пользователям использовать обученные модели для распознавания типов летательных аппаратов. После этого было проведено ручное тестирование обученных моделей, чтобы убедиться в правильности их работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Самойлин Е.* АЛГОРИТМ РАСПОЗНАВАНИЯ ИЗОБРАЖЕНИЙ НА ОСНОВЕ ГРАДИЕНТНОГО СОВМЕЩЕНИЯ ОБЪЕКТА С ЭТАЛОНОМ. — Сборник трудов XXV Международной научно-технической конференции, 2019. — с. 150.
2. *Мифтахова А. А.* ПРИМЕНЕНИЕ МЕТОДА ДЕРЕВА РЕШЕНИЙ В ЗАДАЧАХ КЛАССИФИКАЦИИ И ПРОГНОЗИРОВАНИЯ. — Поволжский государственный университет телекоммуникаций и информатики, 2016. — с. 7.
3. *Бабушкина Н. Е.* ВЫБОР ФУНКЦИИ АКТИВАЦИИ НЕЙРОННОЙ СЕТИ В ЗАВИСИМОСТИ ОТ УСЛОВИЙ ЗАДАЧИ. — Донской государственный технический университет, 2022. — с. 4.
4. *Антонов Г. В.* ПРОСТАЯ НЕЙРОННАЯ СЕТЬ И ЕЕ ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ. — ФГБОУ ВО Великолукская государственная сельскохозяйственная академия, 2021. — с. 11.
5. *Левченко К. М.* Нейронные сети. — Белорусский государственный университет информатики и радиоэлектроники, 2022. — с. 5.
6. *Барвинский Д. А.* Применение метода градиентного спуска в решении задач оптимизации. — Тенденции развития науки и образования, 2021.
7. *Апарнев А. Н.* Анализ функций потерь при обучении сверточных нейронных сетей с оптимизатором Adam для классификации изображений. — ВЕСТНИК МОСКОВСКОГО ЭНЕРГЕТИЧЕСКОГО ИНСТИТУТА. ВЕСТНИК МЭИ, 2020.
8. *Митина О.* ПЕРЦЕПТРОН В ЗАДАЧАХ БИНАРНОЙ КЛАССИФИКАЦИИ. — Национальная ассоциация ученых, 2021. — с. 6.
9. *Сикорский О. С.* Обзор свёрточных нейронных сетей для задачи классификации изображений. — Новые информационные технологии в автоматизированных системах, 2017. — с. 8.
10. *Алексеев И. П.* ПЕРСПЕКТИВЫ ПРИМЕНЕНИЯ КАПСУЛЬНЫХ НЕЙРОННЫХ СЕТЕЙ В РАСПОЗНАВАНИИ ОБЪЕКТОВ НА ИЗОБРАЖЕНИЯХ. — ТИНЧУРИНСКИЕ ЧТЕНИЯ - 2021 «ЭНЕРГЕТИКА И ЦИФРОВАЯ ТРАНСФОРМАЦИЯ», 2021. — с. 4.

11. *J. X.* CapsNet, CNN, FCN: Comparative Performance Evaluation for Image Classification // International Journal of Machine Learning and Computing. — 2019. — т. 9, № 6. — с. 9.

12. MNIST dataset. — Дата обращения: 28.04.2023. Режим доступа: `https://www.tensorflow.org/datasets/catalog/mnist`.

13. CIFAR-10 dataset. — Дата обращения: 28.04.2023. Режим доступа: `https://www.tensorflow.org/datasets/catalog/cifar10`.

14. *Паршин С. Е.* Исследование параметров алгоритмов распознавания лиц. — Сборник научных трудов Новосибирского государственного технического университета, 2019. — с. 6.

15. *Э. Я. Р.* Разработка программного средства для идентификации номерных знаков транспортных средств на основе методов компьютерного зрения // Journal of new century innovations. — 2022. — т. 15, № 1. — с. 81—93.

16. *Береснев Д. В.* КЛАССИФИКАЦИЯ ГАЛАКТИК С ПОМОЩЬЮ КАПСУЛЬНЫХ НЕЙРОННЫХ СЕТЕЙ. — БГУИР, 2019. — с. 4.

17. *Y. LeCun L. Bottou Y. B.*, *Haffner P.* Gradient-Based Learning Applied to Document Recognition. — Proceedings of the IEEE, 1998. — с. 46.

18. Different types of CNN Architectures. — Дата обращения: 28.04.2023. Режим доступа: `https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/`.

19. *C. Szegedy W. Liu Y. J.* Going Deeper With Convolutions. — Proceedings of the IEEE Conference on Computer Vision, Pattern Recognition (CVPR), 2015. — с. 12.

20. Understanding Capsule Networks. — Дата обращения: 28.04.2023. Режим доступа: `https://www.freecodecamp.org/news/understanding-capsule-networks-ais-allurin-new-architecture-bdb228173ddc`.

21. *Nitin R. Y. A.* MobileNets for flower classification using TensorFlow // 2017 international conference on big data, IoT and data science (BID). — IEEE. 2017. — с. 154—158.

22. Military Aircraft Recognition dataset. — Дата обращения: 28.02.2023. Режим доступа: https://www.kaggle.com/datasets/khlaifiabilel/military-aircraft-recognition-dataset.

Модель для детектирования самолетов

Листинг 1 – Класс для взаимодействия с обучаемым данными

```
from os import walk, path, listdir
import fnmatch
from torchvision import transforms
from torchvision.utils import save_image
from PIL import Image
import torch
import numpy as np
import matplotlib.pyplot as plt

DATASET_PATH: str = "../planes_dataset"
# TODO add base path
TRAIN_TENSORS_PATH: str = "./train_tensors"
TEST_TENSORS_PATH: str = "./test_tensors"

class DataSetHandler:
    _AUG_ROTATE_ANGLE = 30
    def TrainSize(self):
        files = listdir(TRAIN_TENSORS_PATH)
        return len(files) - 1 #file with y results

    def TestSize(self):
        files = listdir(TEST_TENSORS_PATH)
        return len(files) - 1 #file with y results

    def GetTrainingBatch(self, batchIndexes, needAug=False):
        xBatch, yBatch = self._GetBatch(batchIndexes,
           f"{TRAIN_TENSORS_PATH}/train")
        if needAug:
            xBatch, yBatch = self._AugmentateBatches(xBatch,
               yBatch)

        return xBatch, yBatch


    def GetTestBatch(self, batchIndexes):
        return self._GetBatch(batchIndexes,
           f"{TEST_TENSORS_PATH}/test")

    def UpdateData(self):
```

```
        self._UpdateTrainData()
        self._UpdateTestData()

    def _GetBatch(self, batchIndexes, pathPrefix):
        y = []
        with open(f"{pathPrefix}_results.txt") as classesFile:
            classes = classesFile.read().split('\n')
            for i in batchIndexes:
                y.append(int(classes[i]))

        x = []
        for i in batchIndexes:
            x.append(torch.load(f"{pathPrefix}_{i}.pt"))

        return torch.stack(x), torch.Tensor(y).to(torch.long)

    def _UpdateTrainData(self):
        xTrain = []
        yTrain = []
        with open(f"{DATASET_PATH}/ImageSets/Main/train.txt") as
           trainImagesFile:
            for imageNumber in trainImagesFile:
                images, classes =
                   self._FindAllImages(int(imageNumber))
                xTrain = xTrain + images
                yTrain = yTrain + classes

        for i in range(len(xTrain)):
            tensor: torch.Tensor =
               self._ConverToTensor(xTrain[i])
            torch.save(tensor,
               f"{TRAIN_TENSORS_PATH}/train_{i}.pt")

        with open(f"{TRAIN_TENSORS_PATH}/train_results.txt",
           "w") as f:
            for i in range(len(yTrain)):
                f.write(f"{yTrain[i]}\n")

    def _UpdateTestData(self):
        xTest = []
        yTest = []
```

```
        with open(f"{DATASET_PATH}/ImageSets/Main/test.txt") as
           testImagesFile:
            for imageNumber in testImagesFile:
                images, classes =
                   self._FindAllImages(int(imageNumber))
                xTest = xTest + images
                yTest = yTest + classes

        for i in range(len(xTest)):
            tensor: torch.Tensor = self._ConverToTensor(xTest[i])
            torch.save(tensor,
               f"{TEST_TENSORS_PATH}/test_{i}.pt")

        with open(f"{TEST_TENSORS_PATH}/test_results.txt", "w")
           as f:
            for i in range(len(yTest)):
                f.write(f"{yTest[i]}\n")


        # batchSize = 50
        # xTrainTensors = torch.stack(xTrain[:batchSize])
        # del(xTrain[:batchSize])
        # while len(xTrain) > 0:
        #     ram1 = psutil.virtual_memory().percent
        #     xTrainBatch = torch.stack(xTrain[:batchSize])
        #     xTrainTensors = torch.cat([xTrainTensors,
           xTrainBatch], dim=0)
        #     ram2 = psutil.virtual_memory().percent
        #     del(xTrain[:batchSize])
        #     # gc.collect()
        #     ram3 = psutil.virtual_memory().percent
        #     print(ram1, ram2, ram3)

        # return xTrainTensors, yTrain

    def _FindAllImages(self, imageNumber: int):
        images = []
        classes = []
        for root, dirnames, filenames in
           walk(f"{DATASET_PATH}/Parsed/"):
            for filename in fnmatch.filter(filenames,
```

```
            f"{imageNumber}_*.png"):
              planeImage = path.join(root, filename)
              images.append(planeImage)
              classNameIndex =
                 path.dirname(planeImage).rfind("A")
              classNumber =
                 int(path.dirname(planeImage)[classNameIndex+1:])
                 - 1
              classes.append(classNumber)

    return (images, classes)

def _ConverToTensor(self, imagePath: str) -> torch.Tensor:
    image: Image.Image = Image.open(imagePath)
    transform = transforms.ToTensor()
    tensor: torch.Tensor = transform(image)
    # remove alpha channel
    if (tensor.size(0) == 4):
        tensor = tensor[:-1]

    return tensor

def _AugmentateBatches(self, xBatch: torch.Tensor, yBatch:
   torch.Tensor):
    xBatchAug = []
    yBatchAug = []

    for i, tensor in enumerate(xBatch):
        augTensor1 = transforms.functional.rotate(tensor,
           self._AUG_ROTATE_ANGLE)
        augTensor2 = transforms.functional.rotate(tensor,
           -self._AUG_ROTATE_ANGLE)
        augTensor3 =
           transforms.functional.adjust_brightness(tensor,
           1.5)
        augTensor4 =
           transforms.functional.gaussian_blur(tensor,
           kernel_size=(5,9), sigma=3)
        # save_image(tensor, "tensor.png")
        # save_image(augTensor1, "augTensor1.png")
        # save_image(augTensor2, "augTensor2.png")
```

```
            # save_image(augTensor3, "augTensor3.png")
            # save_image(augTensor4, "augTensor4.png")
            # exit()

            xBatchAug += [tensor, augTensor1, augTensor2,
               augTensor3, augTensor4]
            yBatchAug += [yBatch[i].item()] * 5

        order = np.random.permutation(len(xBatchAug))
        xBatchAug = np.array(xBatchAug)[order]
        yBatchAug = np.array(yBatchAug)[order]

        return torch.stack(list(xBatchAug)),
           torch.Tensor(list(yBatchAug)).to(torch.long)



if __name__ == "__main__":
    DataSetHandler().UpdateData()
    print(DataSetHandler().TrainSize())
    print(DataSetHandler().TestSize())
    xTrain, yTrain =
       DataSetHandler().GetTrainingBatch(range(100))
    print(xTrain[0], yTrain[0])
    print(xTrain.size(0), xTrain.size(1), xTrain.size(2),
       xTrain.size(3))

    im2display = np.transpose(xTrain[0], (1,2,0))
    plt.imshow(im2display)
    plt.show()
```

Листинг 2 – Алгоритм прохода одной эпохи

```
from abc import ABC, abstractmethod
import numpy as np
from dataset_handler.dataset_handler import DataSetHandler
import gc

class INetworkController(ABC):
    @abstractmethod
    def TrainPrepare(self):
        pass
```

```
    @abstractmethod
    def TrainEpoch(self):
        pass

    @abstractmethod
    def GetResults(self, xBatch):
        pass

    @abstractmethod
    def GetResult(self, imagePath):
        pass

    @abstractmethod
    def SaveModel(self, modelPath):
        pass

    @abstractmethod
    def LoadModel(self, modelPath):
        pass

    @abstractmethod
    def GetAllModels(self):
        pass

    def TrainNetwork(self, epochs: int):
        self.TrainPrepare()

        trainAccs, testAccs = [], []
        for epoch in range(epochs):
            self.TrainEpoch()
            print(f"================================== {epoch}
               ==================================")
            trainAcc, testAcc = self.LogTraining()
            trainAccs.append(trainAcc)
            testAccs.append(testAcc)
            if len(testAccs) >= 3:
                # testAccsVar = np.var(testAccs[-3:])
                # print(f"testAccsVar == {testAccsVar}")
                # if testAccsVar < 0.05:
                if testAccs[-1] < testAccs[-2] and testAccs[-1]
                   < testAccs[-3]:
```

```
                break

    def LogTraining(self):
        datasetHandler = DataSetHandler()

        predsTest = []
        yBatchTest = []
        for startTestBatch in range(0,
           datasetHandler.TestSize(), 500):
            # testOrder =
               np.random.permutation(datasetHandler.TestSize())[:500]
            xBatch, yBatch =
               datasetHandler.GetTestBatch(list(range(startTestBatch,
               min(startTestBatch + 500,
               datasetHandler.TestSize()))))
            preds = self.GetResults(xBatch)
            predsTest = predsTest + preds.tolist()
            yBatchTest = yBatchTest + yBatch.tolist()
            gc.collect()

        predsTrain = []
        yBatchTrain = []
        for startTrainBatch in range(0,
           datasetHandler.TrainSize(), 500):
            # xBatchTrain, yBatchTrain =
               datasetHandler.GetTrainingBatch(np.random.permutation(
            xBatch, yBatch =
               datasetHandler.GetTrainingBatch(list(range(startTrainBa
               min(startTrainBatch + 500,
               datasetHandler.TrainSize()))))
            preds = self.GetResults(xBatch)
            predsTrain = predsTrain + preds.tolist()
            yBatchTrain = yBatchTrain + yBatch.tolist()
            gc.collect()

        print("====================================")
        testMisses = [0] * 20
        testClasses = [0] * 20
        recognizedTest = 0
        for i in range(len(predsTest)):
            testClasses[yBatchTest[i]] += 1
```

```
            if predsTest[i] == yBatchTest[i]:
                recognizedTest += 1
            else:
                testMisses[yBatchTest[i]] += 1


        trainMisses = [0] * 20
        trainClasses = [0] * 20
        recognizedTrain = 0
        for i in range(len(predsTrain)):
            trainClasses[yBatchTrain[i]] += 1
            if predsTrain[i] == yBatchTrain[i]:
                recognizedTrain += 1
            else:
                trainMisses[yBatchTrain[i]] += 1


        print("".join(map(lambda x: "{:6}".format(x),
           list(range(1,21)))))
        print("".join(map(lambda x: "{:6}".format(x),
           testClasses)))
        print("".join(map(lambda x: "{:6}".format(x),
           testMisses)))
        print("".join(map(lambda x: "{:6}".format(x),
           trainClasses)))
        print("".join(map(lambda x: "{:6}".format(x),
           trainMisses)))

        testAcc = float(recognizedTest) / len(predsTest)
        trainAcc = float(recognizedTrain) / len(predsTrain)
        print(float(recognizedTest) / len(predsTest))
        print(float(recognizedTrain) / len(predsTrain))
        print("====================================")

        return trainAcc, testAcc
```

Листинг 3 – Класс для взаимодействия с моделью

```
import torch
from PIL import Image
from torchvision import transforms
import numpy as np
import matplotlib.pyplot as plt
```

```
import os

from models.conv_network import PlanesNetwork
from models.simple_net import simplenet
from dataset_handler.dataset_handler import DataSetHandler

from network_controllers import INetworkController

class NetworkController(INetworkController):
    _TRAINED_MODELS_PATH = "trained_models/"

    def __init__(self, batchSize, learningRate, needAug=True):
        # self.m_planesNetwork = PlanesNetwork(20)
        # self.m_device = torch.device("cuda" if
           torch.cuda.is_available() else "cpu")
        # TODO
        self.m_device = "cpu"
        print(self.m_device)
        self.m_planesNetwork = simplenet(20).to(self.m_device)
        # print(sum(p.numel() for p in
           self.m_planesNetwork.parameters() if p.requires_grad))
        self.m_datasetHandler = DataSetHandler()

        self.m_batchSize = batchSize
        self.m_learningRate = learningRate
        self.m_needAug = needAug

        self.m_loss = torch.nn.CrossEntropyLoss()
        # TODO change on ADAM
        self.m_optimizer =
           torch.optim.Adam(self.m_planesNetwork.parameters(),
           lr=self.m_learningRate)
        self.m_datasetLen = self.m_datasetHandler.TrainSize()

    def TrainPrepare(self):
        pass

    def TrainEpoch(self):
        self.m_planesNetwork.train()
        order = np.random.permutation(self.m_datasetLen)
        for startIndex in range(0, self.m_datasetLen,
```

```
            self.m_batchSize):
            self.m_optimizer.zero_grad()

            xBatch, yBatch =
                self.m_datasetHandler.GetTrainingBatch(order[startIndex
                needAug=self.m_needAug)
            xBatch = xBatch.to(self.m_device)
            yBatch = yBatch.to(self.m_device)

            preds = self.m_planesNetwork.forward(xBatch)
            lossValue = self.m_loss(preds, yBatch)

            # print(preds.argmax(dim=1))
            # print(lossValue)
            lossValue.backward()

            self.m_optimizer.step()

    def GetResults(self, xBatch):
        self.m_planesNetwork.eval()
        results =
            self.m_planesNetwork.forward(xBatch).argmax(dim=1)
        self.m_planesNetwork.train()

        return results

    def GetResult(self, imagePath):
        image: Image.Image = Image.open(imagePath)
        transform = transforms.ToTensor()
        tensor: torch.Tensor = transform(image)
        # remove alpha channel
        if (tensor.size(0) == 4):
            tensor = tensor[:-1]

        # # TODO constant
        # stride = 5
        # i = 0
        # while i < len(tensor[0]) + 96:
        #     j = 0
        #     while j < len(tensor[0][0]) + 96:
        #         sector = tensor[:, i:i+96, j:j+96]
```

```
        #          im2display = np.transpose(sector, (1,2,0))
        #          plt.imshow(im2display)
        #          plt.show()

        #          t = torch.stack([sector])
        #          self.GetResults(t)


        #          j += stride
        #      i += stride

        t = torch.stack([tensor])
        return self.GetResults(t)[0]


    def SaveModel(self, modelPath):
        torch.save(self.m_planesNetwork,
           f"{self._TRAINED_MODELS_PATH}{modelPath}")

    def LoadModel(self, modelPath):
        self.m_planesNetwork =
           torch.load(f"{self._TRAINED_MODELS_PATH}{modelPath}")
        self.m_planesNetwork.to(self.m_device)

    def GetAllModels(self):
        models = []
        for _, _, filenames in
           os.walk(self._TRAINED_MODELS_PATH):
            for model in filenames:
                models.append(model)

        return models



if __name__ == "__main__":
    NetworkController().TrainNetwork(15, 100, 1e-3)
```