МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

WEB-дизайн и мультимедиа
(направленность (профиль)/ специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

на тему «Применение глубоких нейронных сетей для распознавания дорожных знаков»

Студент А.В. Бобков
(И.О. Фамилия) (личная подпись)

Руководитель С.В. Митин
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Аннотация

Тема бакалаврской работы: «Применение глубоких нейронных сетей для распознавания дорожных знаков».

В бакалаврской работе исследуется вопрос применения нейронных сетей для решения задачи распознавания дорожных знаков.

Цель работы – подбор архитектуры и тестирование сверточной нейронной сети, обеспечивающей распознавание дорожных знаков с точностью 95% при наименьшем количестве слоев и не требующей преобразования признакового пространства.

В результате выполнения бакалаврской работы была предложена и оптимизирована архитектура сверточной нейронной сети, на языке Python разработано программное обеспечение, реализующее исследовательский анализ исходных данных и работу нейронной сети при анализе изображений дорожных знаков.

Бакалаврская работа состоит из введения, трёх глав, заключения и списка литературы.

Во введении описывается актуальность проводимого исследования, дается краткая характеристика проделанной работы.

В первой главе проводится обзор существующих подходов к решению задачи распознавания дорожных знаков.

Во второй главе описывается выбор архитектуры нейронной сети для распознавания дорожных знаков.

В третьей главе описывается программная реализация технологии нейросетевого распознавания дорожных знаков.

В заключении представлены выводы по проделанной работе.

В работе присутствуют 1 таблица, 51 рисунок, 4 формулы. Список литературы состоит из 23 литературных источников. Общий объем выпускной квалификационной работы составляет 51 страницу.

Содержание

Введение

Распознавание дорожных знаков – сложная реальная задача, имеющая значение при разработке систем беспилотного управления транспортными средствами и систем обеспечения безопасности дорожного движения.

Распознавание знаков - это проблема классификации нескольких категорий с несбалансированными частотами классов. Дорожные знаки имеют широкий диапазон различий между классами по цвету, форме, наличию текста или специальных символов.

При этом существуют подмножества классов (например, знаки ограничения скорости), которые очень похожи друг на друга. Классификатор должен справляться с большими вариациями в визуальном восприятии из-за изменения освещенности, частичной окклюзии, поворота, погодных условий, масштабирования и т.д.

Существуют научные исследования по использования алгоритмов машинного обучения для построения классификаторов, способных с высокой точности производить распознавания знаков. Подробное описание аналогов представлено в первой главе.

В большинстве случаев используется подход, основанный на трансформации признакового пространства изображений (HOG features, Haar-like features, Color histogram) и последующем анализе признаков с помощью нейронных сетей. Такой подход обладает универсальностью и подходит для широкого ряда задач анализа изображений, несвязанных с дорожными знаками.

Недостатком такого универсального подхода является избыточность преобразований и вычислений в получаемых классификаторах. Такая избыточность влечет за собой повышение затрат на аппаратную реализацию системы распознавания.

В настоящем исследовании выдвигается следующее предположение. Эргономика знаков разрабатывалась так, чтобы их легко обнаруживали и распознавали водители-люди, поэтому с задачей их распознавания с высокой точностью (около 95%) должна справиться сверточная нейронная сеть с простой архитектурой (малым количеством слоев). При этом нет необходимости производить конвертирование признакового пространства до подачи изображение на вход нейронной сети.

Таким образом, цель исследования – подбор архитектуры и тестирование сверточной нейронной сети, обеспечивающей распознавание дорожных знаков с точностью 95% при наименьшем количестве слоев и не требующей преобразования признакового пространства.

Поставленная цель решается путем решения следующих задач:

1. Анализ существующих методов распознавания дорожных знаков на изображениях.
2. Выбор архитектуры нейронной сети для решения задачи дорожных знаков.
3. Разработка программного обеспечения для тестирования работы нейронной сети.

1 Анализ существующих технологий распознавания образов

1.1 Обзор существующих подходов к решению задачи распознавания дорожных знаков

В научной статье «Traffic sign recognition with multi-scale Convolutional Networks» [20] описываются различные существующие подходы для решения задачи распознавания дорожных знаков. В статье говорится, что несколько команд разработчиков (IDSIA, VISICS, INI-RTCV и др.) из разных стран тестировали разнообразные сочетания технологий машинного обучения и компьютерного зрения применительно к задаче распознавания изображений с дорожными знаками.

Для сопоставления предложенных ими подходов применялся набор данных, содержащих в себе тренировочные и тестовые изображения с дорожными знаками, предоставленный Институтом нейроинформатики (Германия) – https://benchmark.ini.rub.de/. Подробный анализ данного набора данных представлен во втором разделе.

Таблица 1 – Результаты работы различных методов в задаче распознавании дорожных знаков на наборе данных «German Traffic Sign Recognition Benchmark»

Точность (%)	Разработчик	Метод
98,98	IDSIA	HOG3 + CNN
98,97	IDSIA	HOG3 + HAAR + CNN
97,88	VISICS	HOG1 + HOG2 + LDA
97,35	VISICS	HOG1/HOG2 +LDA
96,32	INI-RTCV	HOG2 + LDA
94,73	INI-RTCV	HOG3 + LDA
94,51	INI-RTCV	HOG1 + LDA

Дальнейшему анализу подвергались решения обладающие точностью более 94,5% на тестовой выборке данных (таблица 1).

В данных решениях, при построении классификатора изображений использовалось предварительное преобразование признакового пространства к одному из следующих типов:

- Признаки HOG (HOG features). Три набора (HOG1, HOG2, HOG3) различных конфигураций признаков HOG (гистограммы ориентированных градиентов). Для их расчета изображения масштабируются до размера 40×40 пикселей и преобразуются в оттенки серого. Наборы содержат векторы признаков длиной 1568, 1568 и 2916 соответственно. Пример преобразования изображений в признаки HOG показан на рисунке 1.

Рисунок 1– Пример преобразования изображений в признаки HOG [20]

- Хаар-подобные признаки (Haar-like features). Как и при использовании признаков HOG, исходные изображения были изменены до размера 40×40 и преобразованы в градации серого. В представленных методах используется 5 различных типов, отличающихся по размеру, в результате чего на одно изображение пришлось 11 584 признака.

Рисунок 2 – Визуализация использования Хаар-подобных признаков при анализе изображений дорожных знаков [20]

- Цветовые гистограммы (Color histograms). Этот набор признаков рассчитан на основе градиента цветовой информации. Он содержит глобальную гистограмму значений оттенка в цветовом пространстве HSV, что дает 256 признаков на изображение.

В описываемых подходах, в качестве методов построения классификаторов в большинстве случаев используется или линейный дискриминантный анализ (LDA) и сверточные нейронные сети (CNN).

Недостатком описанных подходов, используемых при анализе изображений дорожных знаков, является необходимость преобразования признакового пространства исходных изображений. Это приводит к необходимости проведения дополнительных вычислений связанных с трансформацией изображений. Это приводит к снижению быстродействия при использовании таких подходов системах компьютерного зрения

Для преодоления этого необходимо подобрать минимально возможную архитектуру нейронной сети, обеспечивающей точность распознавания изображений около 95% без необходимости выполнения предварительного преобразования признакового пространства имеющихся изображений [3]. При этом для обеспечения корректности полученных результатов при обучении нейронной сети будет использован тот же набор данных («German Traffic Sign Recognition Benchmark»), который был использован в описанных выше исследованиях.

Выдвинутое предположение о возможности отказа от трансформации признакового пространства без значимой потери точности распознавания (точность около 95%) основано на следующих рассуждениях:

1. Сверточная нейронная сеть в силу наличия в ней слоев свертки способна самостоятельно определять значимые признаки анализируемых изображений.

2. Дорожные знаки имеют стандартизированный внешний вид. И если сверточная нейронная сеть решает задачу распознования рукописного текста без предварительной трансформации признакового пространства, то она и не требуется при решении задачи распознавания стандартизированных изображений.

1.2 Выводы по главе

Наиболее точные подходы по распознаванию изображений дорожных знаков с использованием набора данных «German Traffic Sign Recognition Benchmark» основаны на трансформации признакового пространства изображений (HOG features, Haar-like features, Color histogram) и последующем анализе признаков с помощью сверточных нейронных сетей.

Выдвинуто предположение, что с задачей распознавания дорожных знаков с высокой точностью (около 95%) должна справиться сверточная нейронная сеть с простой архитектурой (малым количеством слоев) без предварительного преобразования изображений к другому признаковому пространству.

2 Разработка алгоритма распознавания дорожных знаков с использованием глубокой нейронной сети

2.1 Алгоритм нейросетевого распознавания дорожных знаков

Распознавание дорожных знаков по изображению относится к задаче классификации. Предполагается, что заранее известно множество классов, каждый класс – это отдельный тип знака дорожного движения. Для распознавания дорожных знаков необходимо построить классификатор, способный классифицировать входное изображение, подаваемое на его вход. Номер класса определяет тип распознанного дорожного знака. В качестве классификатора будем использовать сверточную нейронную сеть.

При подборе архитектуры нейронной сети было принято решение следовать следующей стратегии:

- использовать минимальное количество необходимых слоев, обеспечивающих успешную работу нейронной сети;
- добавлять дополнительные слои только, если не удалось достигнуть требуемой точности работы;
- для обеспечения высокой скорости обучения и работы нейронной сети использовать наиболее простые функции (функция max для слоев субдискретизации и функция ReLU для всех остальных промежуточных слоев);
- предпочтение отдается предобработке тренировочных изображений, нежели увеличению сложности нейронной сети.

С учетом выбранной стратегии определим начальные параметры архитектуры нейронной сети.

Входной слой нейронной сети служит для получения входного изображения в виде числового массива. Есть два варианта подачи изображений на вход нейронной сети – без учета цвета, работая только с

яркостью (градиент серого) и с учетом цветовой составляющей, подавая на вход все три компонента цветовой модели RGB. Так как на дорожных знаках цвет имеет значение, то будем пользоваться вторым вариантом.

Теперь необходимо определить размер входного изображения. Так как все знаки легко вписываются в квадратную область, то настроим входной слой на получение квадратных изображений. Чем больше размерность входного слоя, тем больше потребуется слоев свертки. Поэтому выберем небольшую размерность слоя равную 30×30 пикселей. Как показывает практика такой размерности более чем достаточно, для распознавания стандартизированных изображений (символов, указателей и т.д.) [1].

Основными слоями сверточной нейронной сети являются: слой свертки и слой субдискретизации. Первый обеспечивает выделение признаков на числовом наборе данных, а второй обеспечивает снижение размерности признакового пространства. Минимальное количество подряд идущих блоков состоящих из сочетания этих слоев равно двум. Поэтому включим их в нашу структуру [2].

Обязательным элементом сверхточной нейронной сети является слой снижения размерности, который преобразует все данные предыдущего слоя в вектор. В конце сверточной нейронной сети используется обычный слой нейронов производящий классификацию данных, выделенных из изображения с помощью предыдущих слоев сети.

Таким образом, первоначальная структура нейронной сети будет состоять из следующих слоев.

- входной слой для получения изображения в виде числового массива 30×30×3;
- сверточный слой (convolution) с ядром 5×5, количеством плоскостей (каналов) 32 и функцией активации ReLU;
- слой субдискретизации (max-pooling) с группой уплотнения 2×2;

- сверточный слой (convolution) с ядром 3×3, количеством плоскостей (каналов) 64 и функцией активации ReLU;
- слой субдискретизации (max-pooling) с группой уплотнения 2×2;
- слой снижения размерности (flatten) данных до одномерного вектора;
- слой (core layer ReLU) из 500 нейронов прямого распространения с функцией активацией ReLU;
- выходной (core layer SoftMax) слой с функцией активацией SoftMax.

Графически архитектура нейронной сети показана на рисунке 3.

Рассмотрим математический аппарат работы слоев нейронной сети.

Размер ядра сверточного слоя определяется с использованием равенств (1) и (2):

$$\begin{cases} w_c = w_u - K + 1 \\ h_c = h_u - K + 1 \end{cases} \quad (2.1)$$

где w_c, h_c – ширина и высота сверточной плоскости, w_u, h_u – ширина и высота матрицы сигналов предыдущего слоя, K – ширина ядра свертки [10].

В качестве функции активации используется Rectified linear unit (ReLU):

$$f(a) = \begin{cases} 0 & a < 0 \\ a & a \geq 0 \end{cases}, \quad (2.2)$$

где $f(a)$ – искомое значение элемента, a – взвешенная сумма сигналов предыдущего слоя.

Данная функция является монотонной, с монотонной производной и не аппроксимирует тождественную функцию около начала координат [13].

Преимуществами данной функции ReLU по сравнению с другими вариантами функции активации (гиперболический тангенс, логистическая функция) являются:

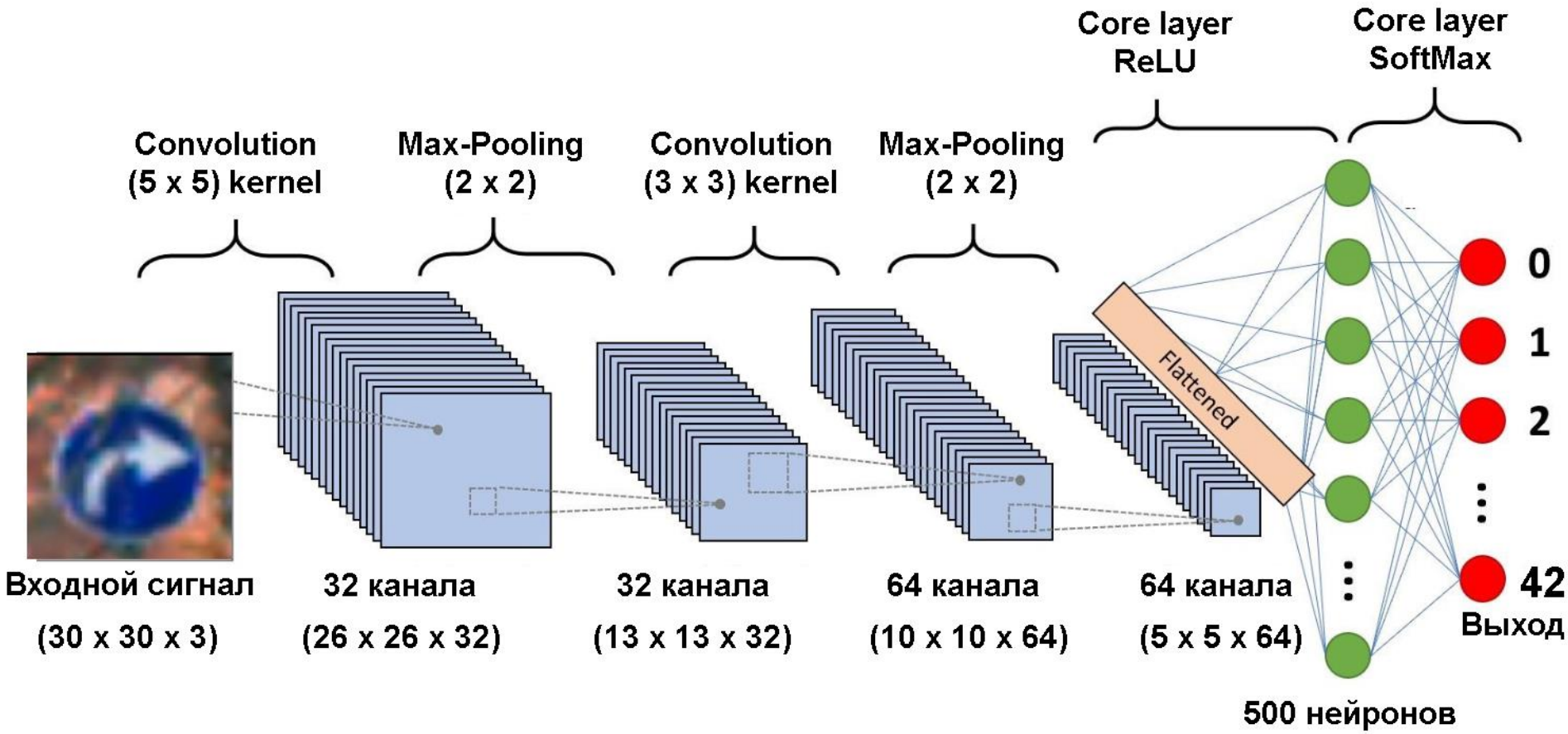


Рисунок 3 – Выбранная архитектура нейронной сети

- данная функция является наиболее простой с точки зрения математики;
- функция имеет простую производную, что дает экономию вычислений при обучении и работе нейронной сети [6].

Формула функционирования нейрона сверточного слоя:

$$y_k^{(i,j)} = b_k + \sum_{s=1}^{K}\sum_{t=1}^{K} w_{k,s,t} x^{((i-1)+s,(j+t))}, \quad (2.3)$$

где $y_k^{(i,j)}$ – нейрон k-й плоскости (k-ого канала) сверточного слоя, b_k – нейронное смещение k-й плоскости, K – размер рецептивной области нейрона (размер ядра свертки), $w_{k,s,t}$ – матрица синаптических коэффициентов, x – выходы нейронов предыдущего слоя [9].

Схема работы сверточного слоя показана на рисунке 4.

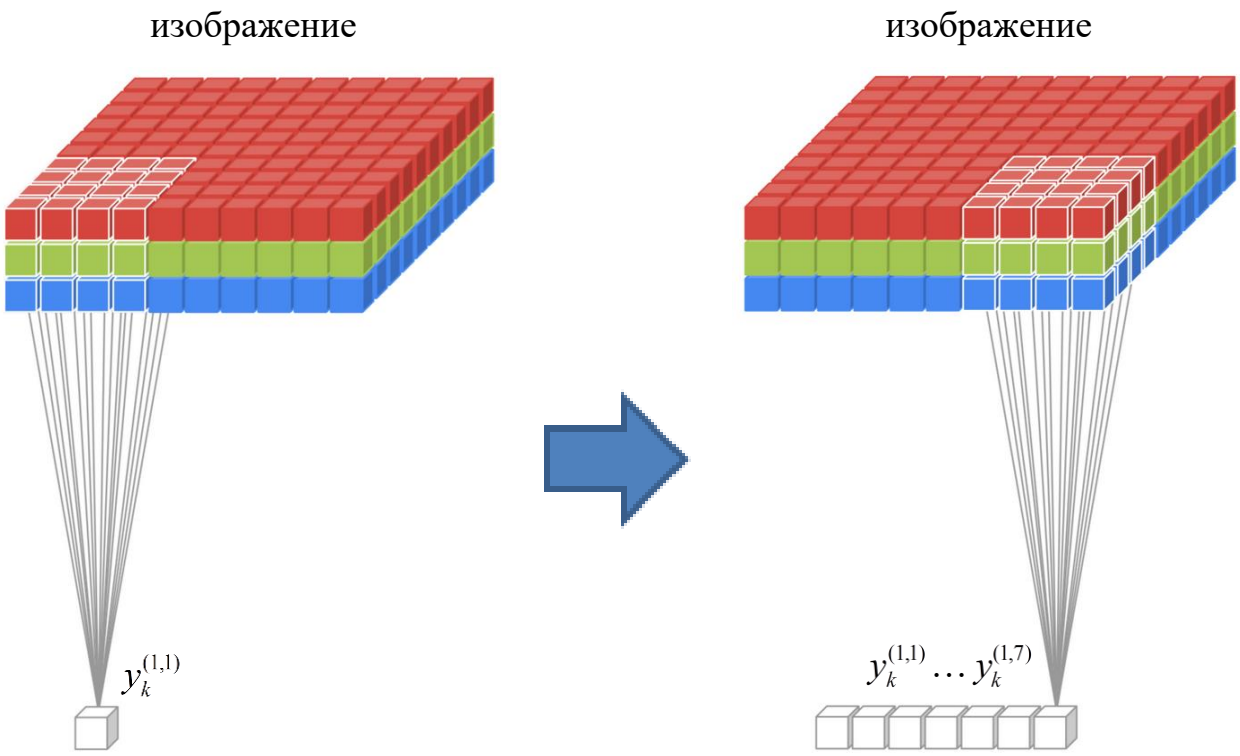


Рисунок 4 – Схема работы сверточного слоя k-й плоскости

Значения компонентов ядра определяется в ходе обучения сети. Начальные веса изначально имеют рандомные значения, и корректируются в процессе обучения [4]. Перемножение и суммирование повторяются для каждой локации, по которой проходит ядро.

Формула функционирования слоя субдискретизации (подвыборочныого слоя) с группой уплотнения 2×2 показана на 2.4:

$$y_k^{(i,j)} = b_k + w_k \cdot \max(x^{((2i,2j),(2i,2j))}, x^{((2i,2j),(2i,2j)+1)}, x^{((2i,2j)+1,(2i,2j))}, x^{((2i,2j)+1,(2i,2j)+1)}) \quad (2.4)$$

[7].

Пример выполнения субдискретизации с использованием функции max() с группой уплотнения 2×2 по казан на рисунке 5.

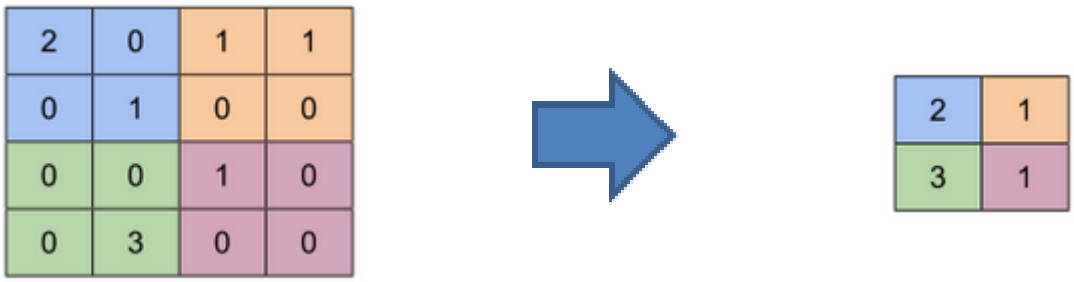


Рисунок 5 – Пример выполнения субдискретизации с использованием функции max() с группой уплотнения 2×2 (группы выделены цветом)

Аналогичным образом, на основе формул (2.1-2.4) работают второй сверточный слой и второй слой субдискретизации.

Слой снижения размерности (flatten) выполняет единственное действие – преобразует данные предыдущего слоя в одномерный вектор [8].

После слоя снижения размерности фактически идет стандартная нейронная сеть прямого распространения с одним скрытым слоем, состоящего из 500 нейронов. В выходном слой состоит из 43 нейронов (43 класса изображений) с функцией активации SoftMax. Таким образом, на

выходе каждого нейрона выходного слоя будет число выражающее степень уверенности сети в каждом из классов для рассматриваемого изображения[5].

2.2 Источник данных для обучения нейронной сети

Для обучения нейронной сети будет использован набор изображений дорожных знаков, размещенный на сайте Института нейроинформатики (https://benchmark.ini.rub.de/). Архив с набором данных занимает около 320 МБ. Внутренняя файловая структура представлена на рисунке 6.

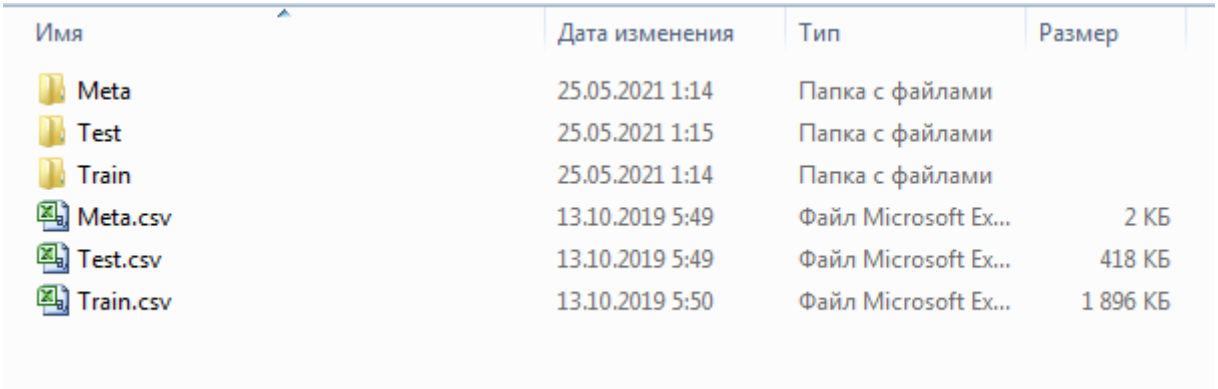


Рисунок 6 – Содержимое архива с набором данных

Архив содержит в себе следующие элементы:

- Каталог «Meta» в котором хранится 43 изображения с дорожными знаками (по одному изображению на каждый класс) (рисунок 7);
- Каталог «Test» в котором хранится 12631 тестовых изображений для проверки точности работы системы распознавания дорожных знаков (рисунок 8);
- Каталог «Train» в котором хранится 43 подкаталога (по одному на каждый класс) с различным количеством изображений дорожных знаков (рисунок 9);
- Файлы «Meta.csv», «Test.csv» и «Train.csv», содержащие дополнительную информацию об имеющихся изображениях.

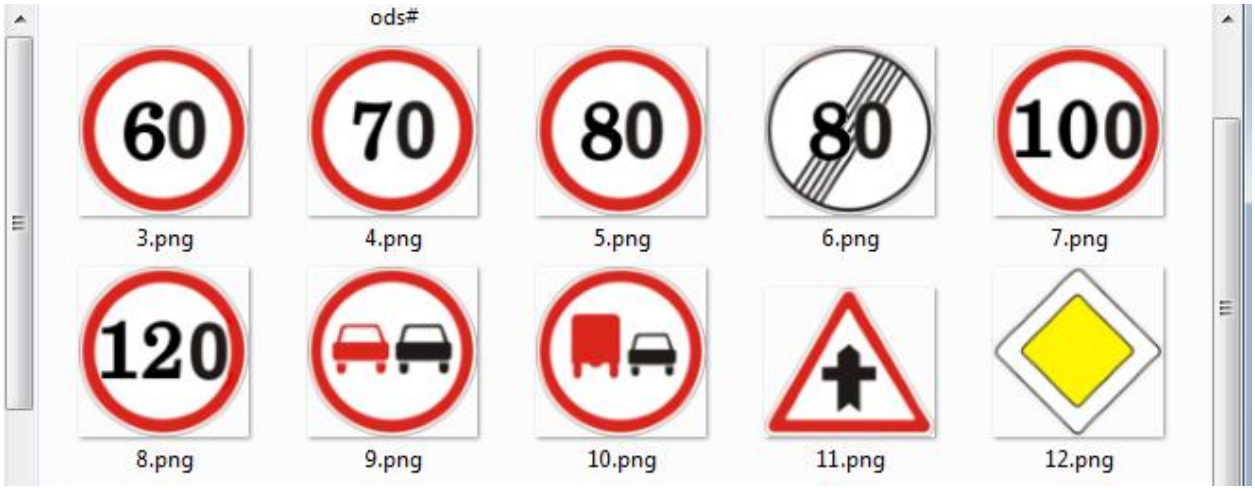


Рисунок 7 – Содержимое каталога «Meta»

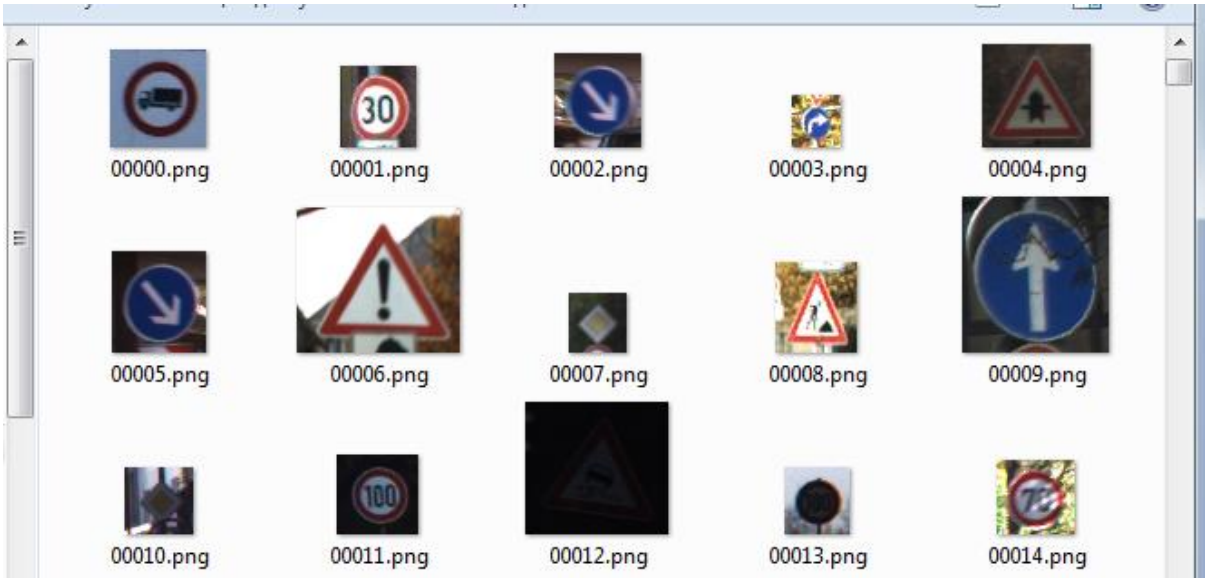


Рисунок 8 – Содержимое каталога «Test»

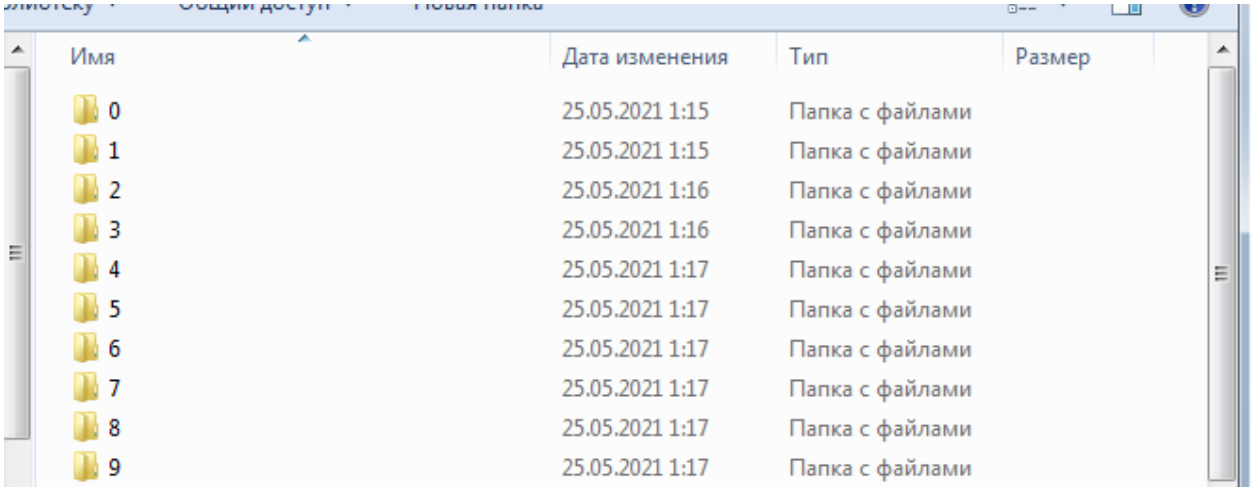


Рисунок 9 – Содержимое каталога «Train»

К дополнительной информации об имеющихся изображениях относится:

- Filename – путь до файла с изображений в каталоге набора данных;
- Width – высота изображения в пикселях;
- Height – ширина изображений в пикселях;
- ROI.x1 – первая X-координата прямоугольной области, описывающий фрагмент изображения со знаком;
- ROI.y1 – первая Y-координата прямоугольной области, описывающий фрагмент изображения со знаком;
- ROI.x2 – вторая X-координата;
- ROI.y2 – вторая Y-координата (рисунок 10).

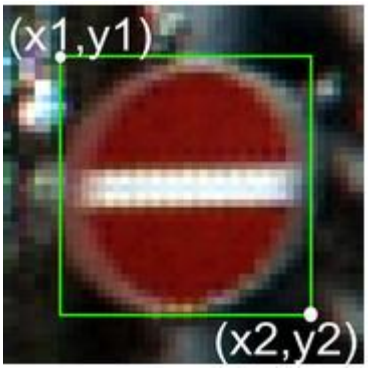


Рисунок 10 – Демонстрация координат прямоугольной области, описывающий фрагмент изображения со знаком

Дополнительная информация при обучении нейронной сети использоваться не будет.

2.3 Оптимизация структуры нейронной сети и данных для обучения

Тестирование выбранной в разделе 2.1 архитектуры нейронной сети на наборе «German Traffic Sign Recognition Benchmark» не позволило обеспечить требуемую точность распознавания изображений (точность распознавания составила порядка 70%). Анализ матрицы ошибок (confusion matrix) показал, что чаще всего нейронная сеть делает ошибки при распознавании изображений 0, 6, 19, 24, 27, 29, 32, 37, 41 и 42 классов.

При поиске причин возникновения большого количества ошибок при распознавании изображений этих классов была построена столбчатая диаграмма, отображающая количество изображений в тренировочной выборке данных.

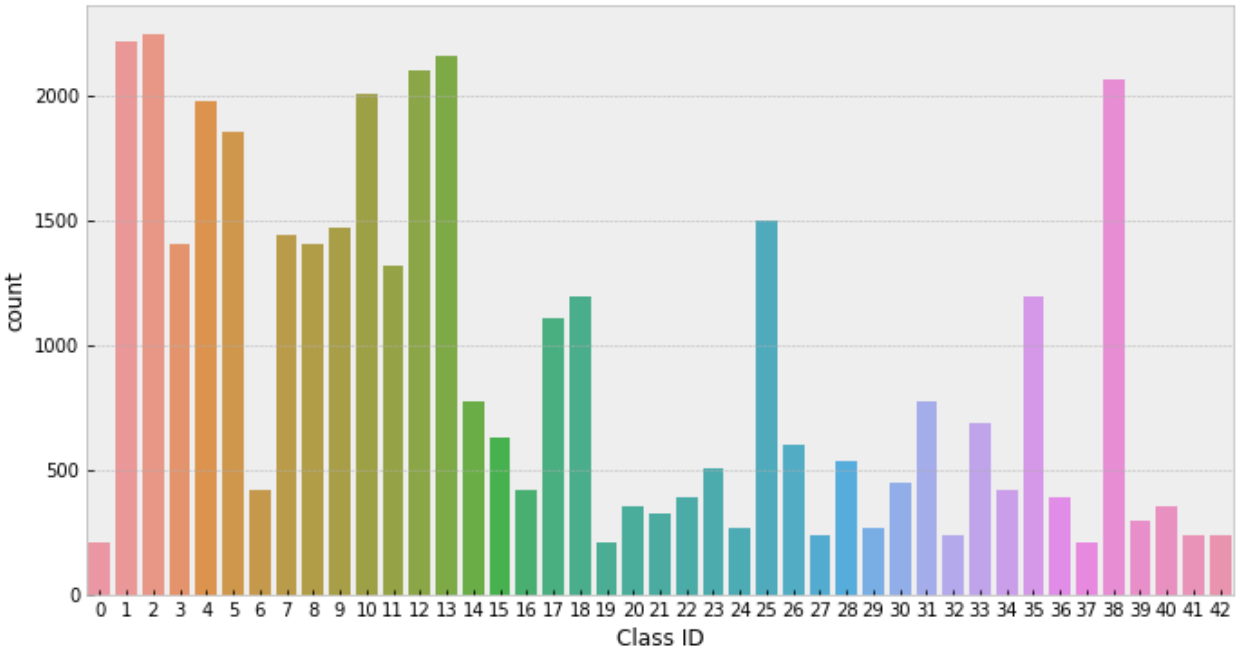


Рисунок 11 – Количество изображений различных классов в тренировочном наборе данных

Из построенной диаграммы видно, что больше всего ошибок возникает у классов с малым количеством объектов в тренировочном наборе данных.

Таким образом, на классах с малым набором данных возникает эффект переобучения (overfitting) нейронной сети.

Для снижения влияния эффекта переобучения было предложено применить 2 решения:

1. Добавить в структуру нейронной сети специальные слои под названием «dropout». Эти слои функционируют только в процессе обучения нейронной сети, при работе настроенной нейронной сети эти слои никак не учитываются (и поэтому не влияют на быстродействие в процессе распознавания). Слой dropout в процессе обучения сети блокирует передачу сигналов от небольшого количества случайно выбранных предыдущего слоя нейронов следующему слою. Это позволяет предотвратить акцентирование внимания нейронной сети на отдельных признаках изображения. Длительность обучения нейронной сети при использовании слоев dropout увеличивает в несколько раз [14]-[17].

Таким образом, оптимизированная структура нейронной сети включает в себя следующие слои (слои dropout работают только в процессе обучения сети):

- входной слой для получения изображения в виде числового массива 30×30×3;
- сверточный слой (convolution) с ядром 5×5, количеством плоскостей (каналов) 32 и функцией активации ReLU;
- слой субдискретизации (max-pooling) с группой уплотнения 2×2;
- слой dropout, блокирующий сигналы 25% нейронов предыдущего слоя
- сверточный слой (convolution) с ядром 3×3, количеством плоскостей (каналов) 64 и функцией активации ReLU;
- слой субдискретизации (max-pooling) с группой уплотнения 2×2;
- слой dropout, блокирующий сигналы 25% нейронов предыдущего слоя

- слой снижения размерности (flatten) данных до одномерного вектора;
- слой (core layer ReLU) из 500 нейронов прямого распространения с функцией активацией ReLU;
- слой dropout, блокирующий сигналы 50% нейронов предыдущего слоя
- выходной (core layer SoftMax) слой с функцией активацией SoftMax.

2. Дополнить классы с небольшим количеством тренировочных данных дополнительно синтезированными изображениями. Синтез новых изображений будет осуществляться на основе имеющихся изображений путем выполнения операций масштабирования, смещения, вращения со случайно выбранными параметрами в пределах заданного диапазона. Примеры синтезированных изображений показаны на рисунке 12 [11].

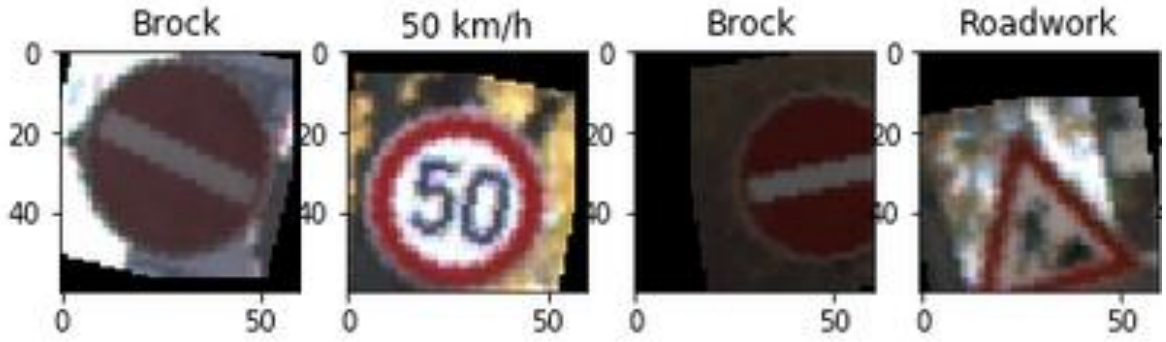


Рисунок 12 – Примеры синтезированных изображений

2.4 Выводы по главе

Обоснован выбор архитектуры нейронной сети, включающей в себя (в порядке следования) следующие слои: входной слой для получения изображения в виде числового массива 30×30×3; сверточный слой (convolution) с ядром 5×5, количеством плоскостей (каналов) 32 и функцией

активации ReLU; слой субдискретизации (max-pooling) с группой уплотнения 2×2; слой dropout, блокирующий сигналы 25% нейронов предыдущего слоя; сверточный слой (convolution) с ядром 3×3, количеством плоскостей (каналов) 64 и функцией активации ReLU; слой субдискретизации (max-pooling) с группой уплотнения 2×2; слой dropout, блокирующий сигналы 25% нейронов предыдущего слоя; слой снижения размерности (flatten) данных до одномерного вектора; слой (core layer ReLU) из 500 нейронов прямого распространения с функцией активацией ReLU; слой dropout, блокирующий сигналы 50% нейронов предыдущего слоя; выходной (core layer SoftMax) слой с функцией активацией SoftMax.

Разработаны решения для предотвращения переобучения нейронной сети. Первое решение заключает в добавлении в структуру нейронной сети нескольких блокирующих (dropout) слоев. Второе решение заключается в генерировании дополнительных изображений в тренировочном наборе данных для малочисленных классов.

3 Разработка программного обеспечения для реализации технологии распознования

3.1 Особенности разработанного программного обеспечения

Для тестирования работы сверточной нейронной сети с предложенной архитектурой на языке программирования Python было разработано программное обеспечение, обладающее следующими возможностями:

- подсчет количества элементов каждого класса для тренировочных и тестовых данных и визуализация результата в виде столбчатой диаграммы;
- построение диаграммы уровней, показывающей распределение размеров изображений по высоте и ширине для тренировочной и тестовой выборки данных;
- построение таблицы с примерами изображений для каждого класса;
- генерирование дополнительных изображений для малочисленных классов за счет выполнения операций масштабирования, смещения, вращения изображений;
- обучение нейронной сети с заданной архитектурой и визуализация процесса обучения в виде графика изменения точности и потерь в зависимости от эпохи обучения;
- расчёт точности работы нейронной сети на тестовой выборке изображений;
- распознавание дорожного знака на выбранном изображении с использованием обученной нейронной и визуализация результата распознавания.

3.2 Программные решения

В разработанном программном обеспечении используются следующие библиотеки:

- os – библиотека с реализациями методов работы с каталогами и получения списков имен хранящихся в них файлов;
- matplotlib – библиотека для визуализации аналитических данных и построения графиков различных видов;
- numpy – библиотека, содержащая в себе математические реализации математических функций и добавляющая поддержку массивов;
- PIL – библиотека для загрузки файлов с изображениями и отображение их на экране;
- keras – библиотека, работающая в связке с tensorflow и содержащая в себе настроенные пресеты различных типов слоев нейронных сетей;
- sklearn – библиотека, реализующая различные методы обработки данных (разбиение данных на подмножества, подсет точности работы классификатора и т.д.).

Программный код для подключения используемых библиотек показан на рисунке 13.

```
import os
import matplotlib
import numpy as np
from PIL import Image
from tensorflow.keras.preprocessing.image import img_to_array
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
from tensorflow.keras import backend as K
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
```

Рисунок 13 – Код для подключения необходимых программных библиотек

Набор данных «German Traffic Sign Recognition Benchmark» был размещен в облачном хранилище сервиса Google drive. Для подключения хранящихся в облаке файлов к проекту используется метод drive.mount(). После подключения облака с помощью команды «%cd» указывается рабочая папка проекта. Код представлен на рисунке 14.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
%cd drive/MyDrive/Colab\ Notebooks/recognition/
```

```
/content/drive/MyDrive/Colab Notebooks/recognition
```

Рисунок 14 – Код для подключения к облачному хранилищу данных, в котором хранятся изображения дорожных знаков

Так как программный код библиотек периодически обновляется, то разработчики сообщают о грядущих изменениях в будущем с помощью предупреждений типа «FutureWarning». Эти предупреждения можно отфильтровать, воспользовавшись методом simplefilter().

Для загрузки данных из csv файлов хранящихся в исходной выборке данных воспользуемся методом read_csv() и сохраним результат загрузки данных в переменные trainDf, testDf и metaDf.

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import pandas as pd

trainDf = pd.read_csv('Train.csv')
testDf = pd.read_csv('Test.csv')
metaDf = pd.read_csv('Meta.csv')
```

Рисунок 15 – Код для считывания файлов с разметкой изображений

Простроим распределение изображений по классам отдельно для тренировочной выборки данных (рисунок 17) и тестовой выборки данных (рисунок 18) [12].

Для этого воспользуемся методами библиотеки matplotlib. Также нам потребуется библиотека seaborn, которые содержит в себе различные стили оформления научных графиков.

Для подсчета количества изображений воспользуемся информацией из переменной trainDf (для тренировочного набора данных) и из переменной testDf (для тестового набора данных). Необходимая информация храниться в столбце ClassId. Для задания названия диаграммы используется метод set_title, для подписей осей используются методы set_xlabel и set_ylabel.

Программный код представлен на рисунке 16.

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

fig, axs = plt.subplots(1, 2, sharex=True, sharey=True, figsize=(25, 6))
axs[0].set_title('Train classes distribution')
axs[0].set_xlabel('Class')
axs[0].set_ylabel('Count')
axs[1].set_title('Test classes distribution')
axs[1].set_xlabel('Class')
axs[1].set_ylabel('Count')

sns.countplot(trainDf.ClassId, ax=axs[0])
sns.countplot(testDf.ClassId, ax=axs[1])
axs[0].set_xlabel('Class ID');
axs[1].set_xlabel('Class ID');
```

Рисунок 16 – Код для построения столбчатых распределения изображений дорожных знаков по классам для тренировочной и тестовой выборок

Рисунок 17 – Распределение изображений по классам для тренировочной выборки данных

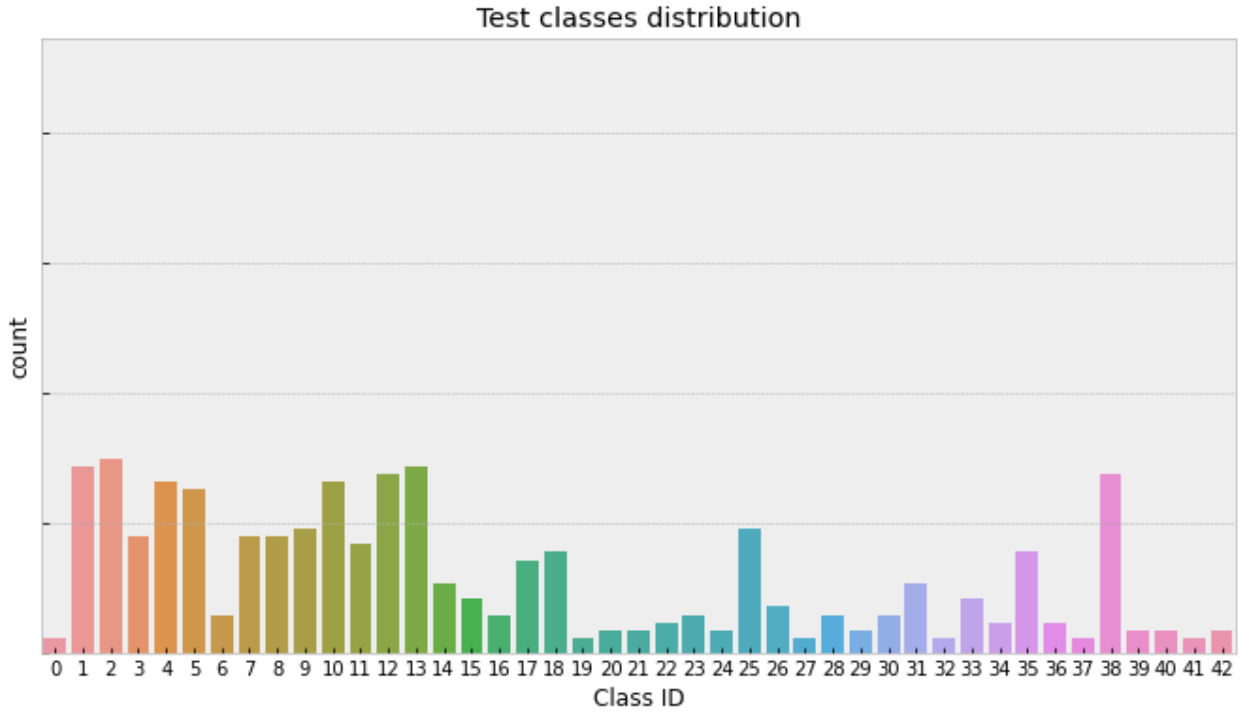


Рисунок 18 – Распределение изображений по классам для тестовой выборки данных

Построенные столбчатые диаграммы наглядно демонстрируют несбалансированность классов по количеству тренировочных изображений.

Так как входной слой нейронной сети имеет фиксированную размерность необходимо выяснить распределения размерности изображений для тренировочного и тестового набора данных [23].

Для оценки распределения размеров изображения в исходных данных была построена диаграмма уровней. По оси X расположен параметр Width – ширина изображения в пикселях, по оси Y расположен параметр Height – высота изображений в пикселях. Синим цветом показано распределение размеров изображений для тестового набора данных, а красным цветом – для тренировочного набора данных (рисунок 20).

Программный код для построения диаграммы уровней показан на рисунке 19.

Информация о размере изображений исходного набора данных храниться в переменных trainDf и testDf в столбцах Width и Height.

```
trainDfDpiSubset = trainDf[(trainDf.Width < 80) & (trainDf.Height < 80)];
testDfDpiSubset = testDf[(testDf.Width < 80) & (testDf.Height < 80)];

g = sns.JointGrid(x="Width", y="Height", data=trainDfDpiSubset)
sns.kdeplot(trainDfDpiSubset.Width, trainDfDpiSubset.Height, cmap="Reds",
        shade=False, shade_lowest=False, ax=g.ax_joint)
sns.kdeplot(testDfDpiSubset.Width, testDfDpiSubset.Height, cmap="Blues",
        shade=False, shade_lowest=False, ax=g.ax_joint)
sns.distplot(trainDfDpiSubset.Width, kde=True, hist=False, color="r",
             ax=g.ax_marg_x, label='Train distribution')
sns.distplot(testDfDpiSubset.Width, kde=True, hist=False, color="b",
             ax=g.ax_marg_x, label='Test distribution')
sns.distplot(trainDfDpiSubset.Width, kde=True, hist=False, color="r",
             ax=g.ax_marg_y, vertical=True)
sns.distplot(testDfDpiSubset.Height, kde=True, hist=False, color="b",
             ax=g.ax_marg_y, vertical=True)
g.fig.set_figwidth(8)
g.fig.set_figheight(8)
plt.show();
```

Рисунок 19 – Код для построения диаграммы уровней, показывающей распределение размеров изображений по высоте и ширине

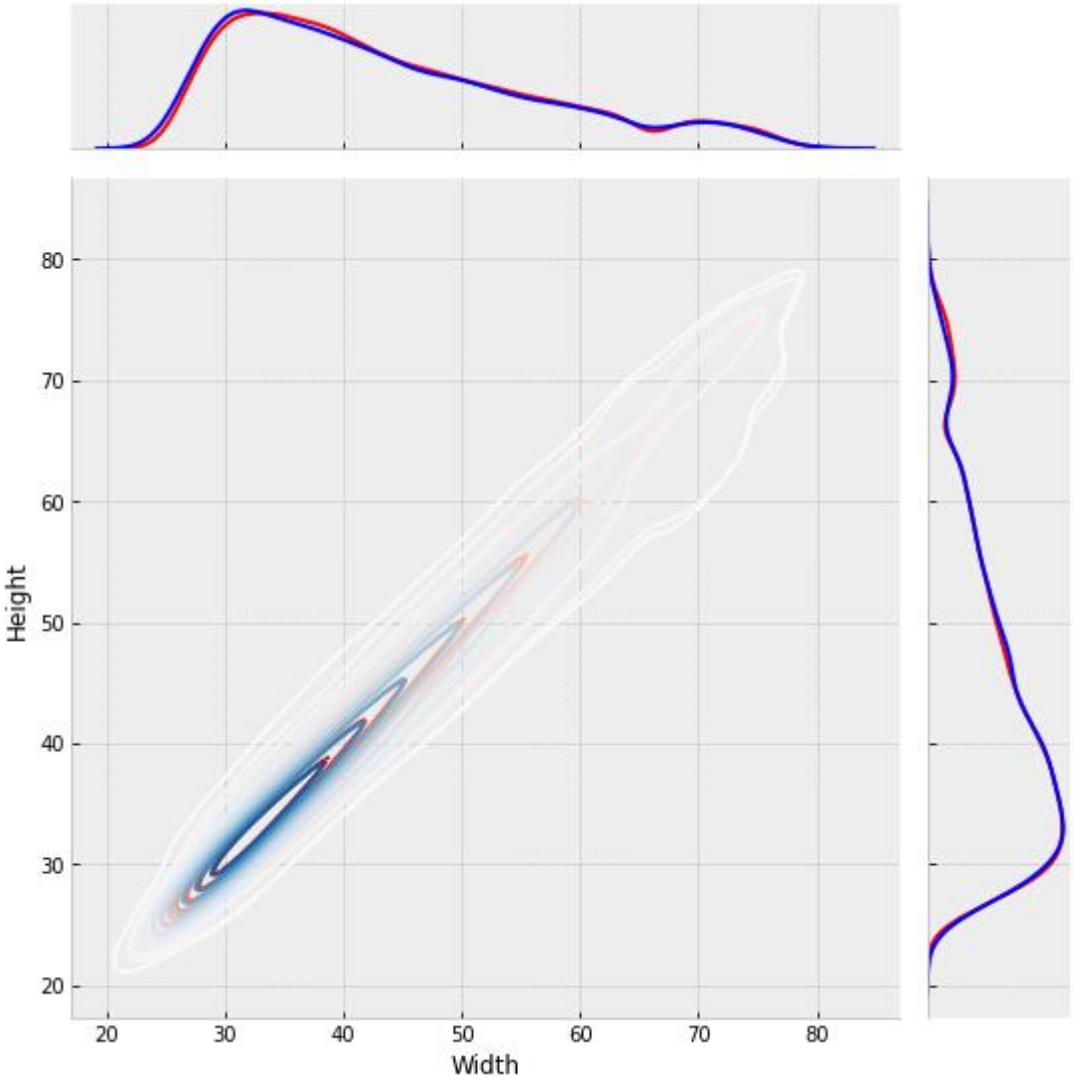


Рисунок 20 – Диаграммы уровней, показывающей распределение размеров изображений по высоте и ширине (красная линия – тренировочный набор данных, синяя линия – тестовый набор данных)

Из построенной диаграммы уровней можно сделать вывод, что большинство исходных изображений имеют размер от 30×30 px до 40×40 px. Поэтому базовым размером изображений принято - 30×30 px. И перед обучением нейронной сети все исходные изображения будут приведены к этому размеру.

Для того чтобы визуально оценить классовый состав исходной выборки данных воспользуемся информацией из переменной metaDf. Там содержаться ссылки на стандартизированные изображения дорожных знаков с привязкой к номеру класса. Для удобства было добавлено текстовое описание классов (переменная label). Затем стандартизированные изображения посредством методов библиотеки cv2 были объединены в таблицу, и выведены на экран (рисунок 21). Результат показан на рисунке 22.

```
import cv2
labels = ['20 km/h', '30 km/h', '50 km/h', '60 km/h', '70 km/h', '80 km/h', '80
              'No overtaking for tracks', 'Crossroad with secondary way', 'Mai
              'Other dangerous', 'Turn left', 'Turn right', 'Winding road', 'H
              'Pedestrian', 'Children', 'Bike', 'Snow', 'Deer', 'End of the li
              'Only straight and left', 'Take right', 'Take left', 'Circle cro

sns.set_style()
rows = 6
cols = 8
fig, axs = plt.subplots(rows, cols, sharex=True, sharey=True, figsize=(25, 12))
plt.subplots_adjust(left=None, bottom=None, right=None, top=0.9, wspace=None,
                    hspace=None)
metaDf = metaDf.sort_values(by=['ClassId'])

idx = 0
for i in range(rows):
    for j in range(cols):
        if idx > 42:
            break

        img = cv2.imread(metaDf["Path"].tolist()[idx], cv2.IMREAD_UNCHANGED)
        img[np.where(img[:,:,3]==0)] = [255,255,255,255]
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (60,60))

        axs[i,j].imshow(img)
        axs[i,j].set_facecolor('xkcd:salmon')
        axs[i,j].set_facecolor((1.0, 0.47, 0.42))
        axs[i,j].set_title(labels[int(metaDf["ClassId"].tolist()[idx])])
        axs[i,j].get_xaxis().set_visible(False)
        axs[i,j].get_yaxis().set_visible(False)
        idx += 1
```

Рисунок 21 – Код для построения таблицы с изображениями, показывающей обозначение классов

Рисунок 22 - Таблица с изображениями, показывающая обозначение классов (всего 43 класса)

Для обеспечения наглядности исходных данных осуществлён вывод случайных изображений из тренировочного набора данных. Каждое изображение перед выводом на экран масштабируется до размера 60×60 px. Метка класса берется из переменной trainDf (столбец ClassId). Для подписи изображений на основе значения ClassId из переменной labels берется текстовое описание класса.

С помощью библиотеки cv2 производится объединение изображений в таблицу с заданными размерами. Размеры таблицы задается через переменные rows и cols.

Программный код построения таблицы с примерами тренировочных изображений показан на рисунке 23. Результат построения таблицы с примерами показан на рисунке 24.

```
rows = 6
cols = 8+4
fig, axs = plt.subplots(rows, cols, sharex=True, sharey=True, figsize=(25, 12))
plt.subplots_adjust(left=None, bottom=None, right=None, top=0.9, wspace=None,
                    hspace=None)
visualize = trainDf.sample(rows*cols)

idx = 0
for i in range(rows):
    for j in range(cols):
        img = cv2.imread(visualize["Path"].tolist()[idx])
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (60,60))
        axs[i,j].imshow(img)
        axs[i,j].set_title(labels[int(visualize["ClassId"].tolist()[idx])])
        axs[i,j].get_xaxis().set_visible(False)
        axs[i,j].get_yaxis().set_visible(False)
        idx += 1
```

Рисунок 23 – Код для построения таблицы с примерами тренировочных изображений

Рисунок 24 – Таблица с примерами тренировочных изображений

Перед обучением нейронной сети необходимо перевести тренировочные изображения из формата png в вид числовой матрицы. Для этого объявляется переменная data. В переменной labels будут храниться метки классов (рисунок 25).

```
data = []
labels = []
classes = 43
```

Рисунок 25 – Код для объявления переменных, в которых будет храниться информация об изображениях в числовом виде

Теперь с помощью цикла for производится обход по всем классам. В исходных данных изображения распределены по директориям, у каждого класса своя директория. Цикл нужен для того чтобы перемещаться по этим папкам. При этом изображение масштабируется до размера 30×30 px и с помощью метода img_to_array переводиться в числовой массив (рисунок 26).

```
for num in range(0, 20):
    print('Reading Train/',num)
    path = os.path.join('Train',str(num))
    imagePaths = os.listdir(path)
    for img in imagePaths:
      image = Image.open(path + '/'+ img)
      image = image.resize((30,30))
      image = img_to_array(image)
      data.append(image)
      labels.append(num)
```

```
Reading Train/ 0
Reading Train/ 1
Reading Train/ 2
Reading Train/ 3
Reading Train/ 4
```

Рисунок 26 – Код для сбора данных об изображениях для классов от 0 до 19

Так как шаг перевода изображений в числовой вид является длительной процедурой, занимающей несколько часов, поэтому цикл for был поделен на две части (рисунок 27).

```
for num in range(20, 43):
    print('Reading Train/',num)
    path = os.path.join('Train',str(num))
    imagePaths = os.listdir(path)
    for img in imagePaths:
      #print(path + '/'+ img)
      image = Image.open(path + '/'+ img)
      image = image.resize((30,30))
      image = img_to_array(image)
      data.append(image)
      labels.append(num)
```

```
Reading Train/ 20
Reading Train/ 21
Reading Train/ 22
Reading Train/ 23
Reading Train/ 24
Reading Train/ 25
```

Рисунок 27 – Код для сбора данных об изображениях для классов от 20 до 42

Для проверки успешности перевода изображений в числовой формат воспользуемся выводом поля shape для переменных data и labels (рисунок 28). Как видно, были собраны данные 39209 изображений. Каждое изображение описывается массивом размерность 30×30 px и тремя цветовыми компонентами R, G, B.

```
print(data.shape, labels.shape)
#(39209, 30, 30, 3) (39209,)
```

```
(39209, 30, 30, 3) (39209,)
```

Рисунок 28 – Код для вывода размерности собранных данных (переменные data и labels)

При обучении нейронной сети тренировочные данные делятся на две группы: данные для валидации (X_test, y_test) и данные для настройки параметров (X_train, y_ train). Данные для валидации нужны для того, чтобы контролировать процесс настройки сети в рамках одного цикла – эпохи.

Для разделения обучающих данных на эти две группы применяется стандартный метод train_test_split из библиотеки sklearn (рисунок 29).

```
X_train, X_test, y_train, y_test = train_test_split(data, labels,
                                                    test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
#(31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)
```

```
(31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)
```

Рисунок 29 – Код для разделения данных на две категории обучающая выборка данных и данные для валидации

Для оценки несбалансированности классов по числу изображений в данных для настройки нейронной сети необходимо посчитать количество классов в наборе y_train. Для этого реализована функция cnt_img_in_classes(). Код функции представлен на рисунке 30.

```
def cnt_img_in_classes(labels):
    count = {}
    for i in labels:
        if i in count:
            count[i] += 1
        else:
            count[i] = 1
    return count

samples_distribution = cnt_img_in_classes (y_train)
```

Рисунок 30 – Подготовка данных для построения графика для демонстрации неоднородности распределения изображений по классам

Теперь, построив столбчатую диаграмму распределения изображений по классам можно увидеть, что у половины классов количество изображений меньше чем 500 штук. Код для построения диаграммы показан на рисунке 31. Результат выполнения кода показан на рисунке 32.

Поэтому необходимо дополнить синтетическими изображениями малочисленные классы.

```
def diagram(count_classes):
    plt.bar(range(len(dct)), sorted(list(count_classes.values())),
            align='center')
    plt.xticks(range(len(dct)), sorted(list(count_classes.keys())),
               rotation=90, fontsize=7)
    plt.show()
diagram(samples_distribution)
```

Рисунок 31 – Построения графика для демонстрации неоднородности распределения изображений по классам

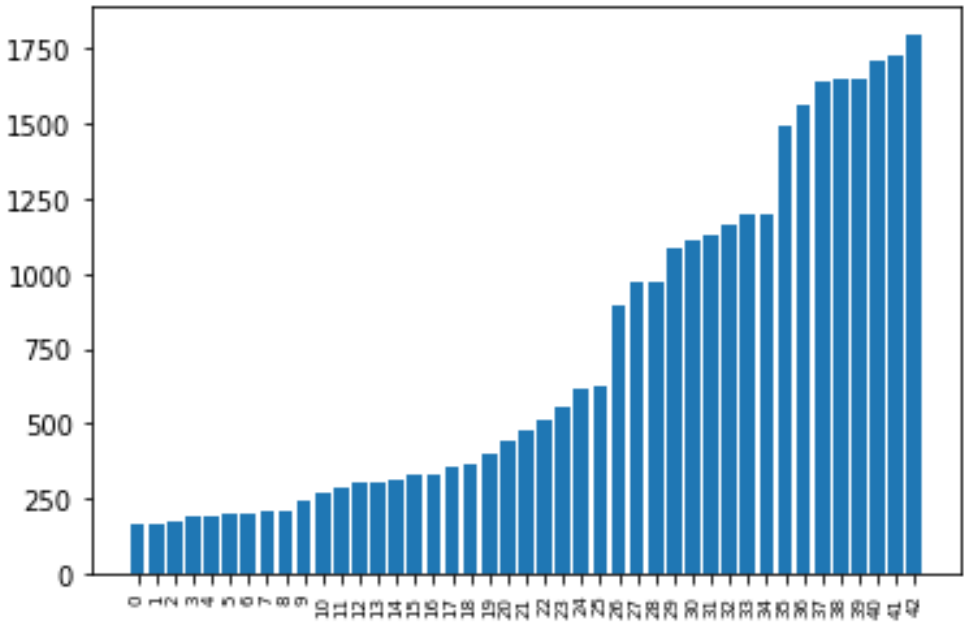


Рисунок 32 – График распределения объектов по классам (по оси X – ранг класса, присвоенный в зависимости от количества в нем изображений, по оси Y – количество изображений в классе)

Синтез новых изображений будет осуществляться на основе имеющихся изображений путем выполнения операций масштабирования, смещения, вращения со случайно выбранными параметрами в пределах заданного диапазона. Для выбора случайных параметров будет использоваться методы библиотеки random. Изменение изображений осуществляется с помощью методов библиотеки imgaug.

Допустимый диапазон значений методов масштабирования, смещения, вращения указан в качестве входного массива метода SomeOf().

Разработана функция aug_images(), которая генерирует одно новое изображение на основе имеющегося (рисунок 33).

```
import random

def aug_images(images, p):
    from imgaug import augmenters as iaa
    augs =  iaa.SomeOf((2, 4),
          [
              iaa.Crop(px=(0, 4)),
              iaa.Affine(scale={"x": (0.8, 1.2), "y": (0.8, 1.2)}),
              iaa.Affine(translate_percent={"x": (-0.2, 0.2), "y": (-0.2, 0.2)}),
              iaa.Affine(rotate=(-45, 45)),
              iaa.Affine(shear=(-10, 10))
])

    seq = iaa.Sequential([iaa.Sometimes(p, augs)])
    res = seq.augment_images(images)
    return res
```

Рисунок 33 – Функция для случайного генерирования изображения на основе существующего

Теперь с использованием функции aug_images() необходимо заполнить малочисленные классы сгенерированными изображениями. Для этого разработана функция augmentation(), которая вызывают функцию aug_images() для малочисленных классов до тех пор, пока количество изображений в них не достигнет заданного уровня. Уровень задается с помощью переменной min_imgs. В нашем случае min_imgs = 500.

Код функции augmentation() представлен на рисунке 34.

```
def augmentation(images, labels):
    min_imgs = 500
    classes = cnt_img_in_classes(labels)
    for i in range(len(classes)):
        if (classes[i] < min_imgs):
            add_num = min_imgs - classes[i]
            imgs_for_augm = []
            lbls_for_augm = []
            for j in range(add_num):
                im_index = random.choice(np.where(labels == i)[0])
                imgs_for_augm.append(images[im_index])
                lbls_for_augm.append(labels[im_index])
            augmented_class = aug_images(imgs_for_augm, 1)
            augmented_class_np = np.array(augmented_class)
            augmented_lbls_np = np.array(lbls_for_augm)
            images = np.concatenate((images, augmented_class_np), axis=0)
            labels = np.concatenate((labels, augmented_lbls_np), axis=0)
    return (images, labels)
X_train, y_train = augmentation(X_train, y_train)
```

Рисунок 34 – Функция для добавления в классы сгенерированных изображений

Для наглядности несколько примеров синтезированных изображений показаны на рисунке 35.

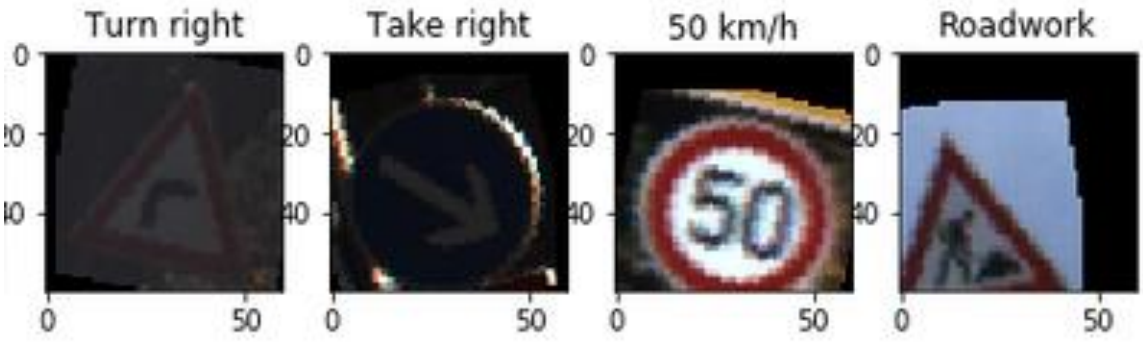


Рисунок 35 – Примеры генерирования дополнительных изображений на основе существующих

Для проверки изменения размера воспользуемся выводом поля shape для переменных содержащих информацию о тренировочных данных (рисунок 36). Как видно из первого числа вывода, было сгенерировано около 5000 дополнительных изображений [18].

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
#(36256, 30, 30, 3) (7842, 30, 30, 3) (36256,) (7842,)
```

```
(36256, 30, 30, 3) (7842, 30, 30, 3) (36256,) (7842,)
```

Рисунок 36 – Оценка количества объектов в двух категориях (тестовые данные и данные для валидации) после генерирования новых изображений

Оценим, как изменилось распределение объектов по класса, еще раз построив столбчатую диаграмму (рисунок 37).

```
diagram(augmented_samples_distribution)
```

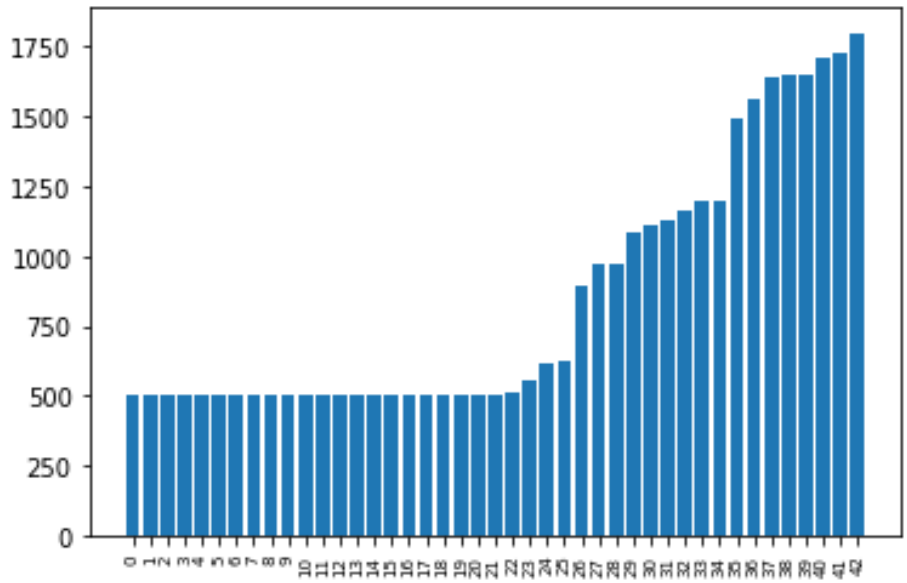


Рисунок 37 – График распределения объектов по класса после генерирования новых изображений (по оси X – ранг класса присвоенный в зависимости от количества в нем изображений, по оси Y – количество изображений в классе)

Как видно из рисунка 37, теперь в тренировочном наборе данных малочисленные классы отсутствуют. Минимальное число изображений в классе равно 500.

Теперь конвертируем номера классов в текстовый формат (рисунок 38). Это необходимо для того, что методы библиотеки keras правильно определили, что решается задача классификации данных [19].

```
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)
```

Рисунок 38 – Код для конвертирования числового целевого значения в категориальный признак

Зададим, выбранную во второй главе, структуру нейронной сети (рисунок 39).

```
class Net:
  @staticmethod
  def build(width, height, depth, classes):
    model = Sequential()
    inputShape = (height, width, depth)
    if K.image_data_format() == 'channels_first':
      inputShape = (depth, heigth, width)
    model = Sequential()
    model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu',
                     input_shape=inputShape))
    model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(Dropout(rate=0.25))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(Dropout(rate=0.25))
    model.add(Flatten())
    model.add(Dense(500, activation='relu'))
    model.add(Dropout(rate=0.5))
    model.add(Dense(classes, activation='softmax'))
    return model
```

Рисунок 39 – Код для послойного формирования нейронной сети

И запустим обучение с выбранными параметрами. Алгоритм обучения – adam, количество эпох обучения – 25. Соответствующий код показан на рисунке 40 [21].

```
epochs = 25
model = Net.build(width=30, height=30, depth=3, classes=43)
model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, y_train, batch_size=64,
                    validation_data=(X_test, y_test), epochs=epochs)
```

Рисунок 40 – Код для задания параметров нейронной сети и запуск процесса обучения

В процессе обучения нейронной сети будет выводиться статистическая информация с номером эпохи и скоростью обучения (рисунок 41).

```
Epoch 1/25
567/567 [==============================] - 137s 240ms/step
Epoch 2/25
567/567 [==============================] - 136s 240ms/step
Epoch 3/25
567/567 [==============================] - 135s 238ms/step
Epoch 4/25
567/567 [==============================] - 136s 240ms/step
Epoch 5/25
567/567 [==============================] - 136s 240ms/step
Epoch 6/25
567/567 [==============================] - 137s 241ms/step
Epoch 7/25
567/567 [==============================] - 141s 249ms/step
Epoch 8/25
567/567 [==============================] - 144s 254ms/step
Epoch 9/25
567/567 [==============================] - 142s 251ms/step
Epoch 10/25
567/567 [==============================] - 142s 250ms/step
Epoch 11/25
567/567 [==============================] - 143s 252ms/step
Epoch 12/25
567/567 [==============================] - 142s 251ms/step
```

Рисунок 41 – Вывод статистической информации в процессе обучения нейронной сети

Процесс обучения сверточной нейронной сети занял 3550 секунд. Теперь необходимо провести исследования точности работы нейронной сети.

3.3 Оценка точности работы нейронной сети

Для начала, для накопленных в процессе обучения сети данных, построим графики. График потерь показывает, как изменялось количество ошибок совершаемых нейронной сети по мере получения опыта (с течением эпох). Потери нейронной сети для каждой эпохи определяются на данных для валидации и данных для настройки нейронной сети [22].

Программный код для построения графика изменения потерь при работе нейронной сети на тренировочных данных и данных для валидации в зависимости от номера эпохи обучения показан на рисунке 42. А полученный график показан на рисунке 43.

```
plt.style.use("bmh")
plt.figure()
N = epochs
plt.plot(np.arange(0, N), history.history["loss"],
         label="train loss")
plt.plot(np.arange(0, N), history.history["val_loss"],
         label="validation loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(loc="upper right")
plt.show()
```

Рисунок 42 – Код построения графиков изменения потерь при обучении нейронной сети

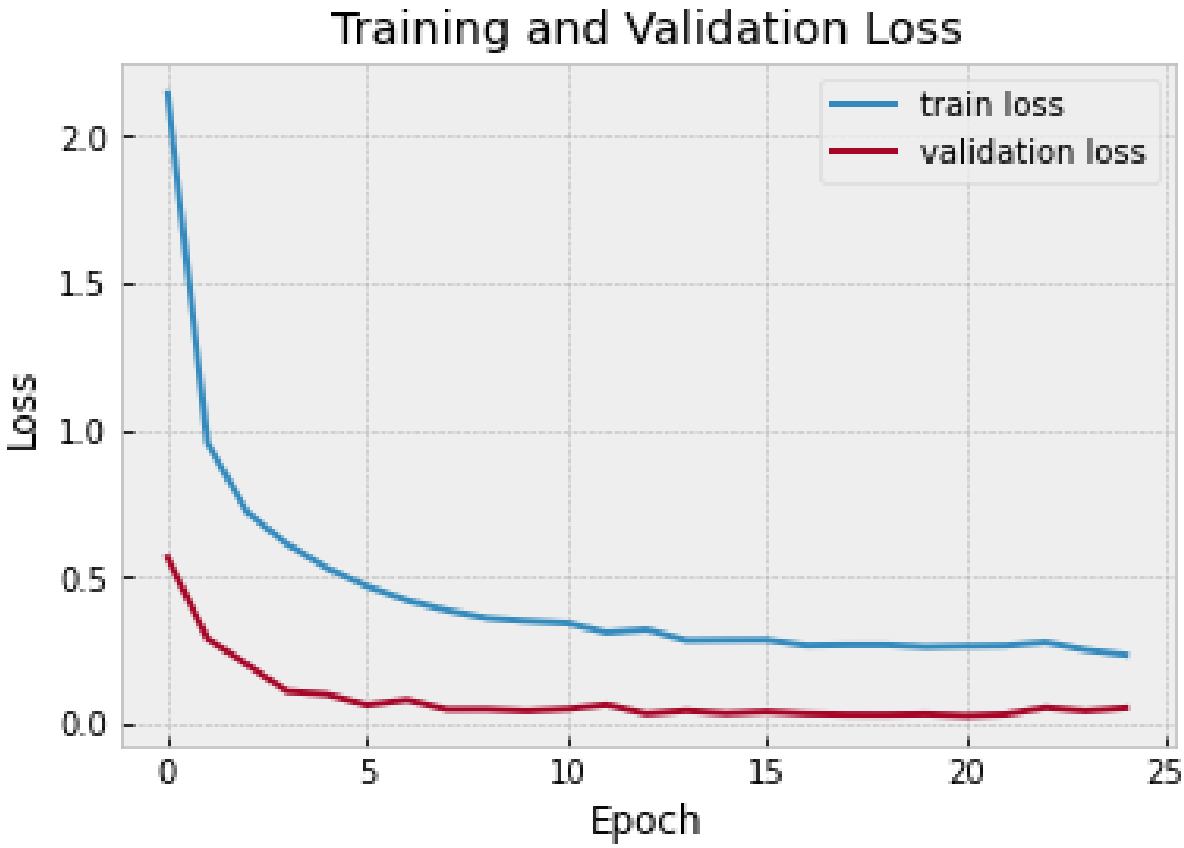


Рисунок 43 – График изменения потерь (Loss) при работе нейронной сети на тренировочных данных (train loss) и данных для валидации (validation loss) в зависимости от номера эпохи обучения (Epoch)

Также построим график изменения точности работы нейронной сети на тренировочных данных и данных для валидации. В отличие от потерь, чем выше значение точности, тем лучше.

Программный код для построения графика изменения точности при работе нейронной сети на тренировочных данных и данных для валидации в зависимости от номера эпохи обучения показан на рисунке 44. А полученный график показан на рисунке 45.

Оба графика показывают, что процесс обучения нейронной сети прошел удачно.

```
plt.style.use("bmh")
plt.figure()
N = epochs
plt.plot(np.arange(0, N), history.history["accuracy"],
         label="train accuracy")
plt.plot(np.arange(0, N), history.history["val_accuracy"],
         label="validation accuracy")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend(loc="lower right")
plt.show()
```

Рисунок 44 – Код построения графиков изменения точности при обучении нейронной сети

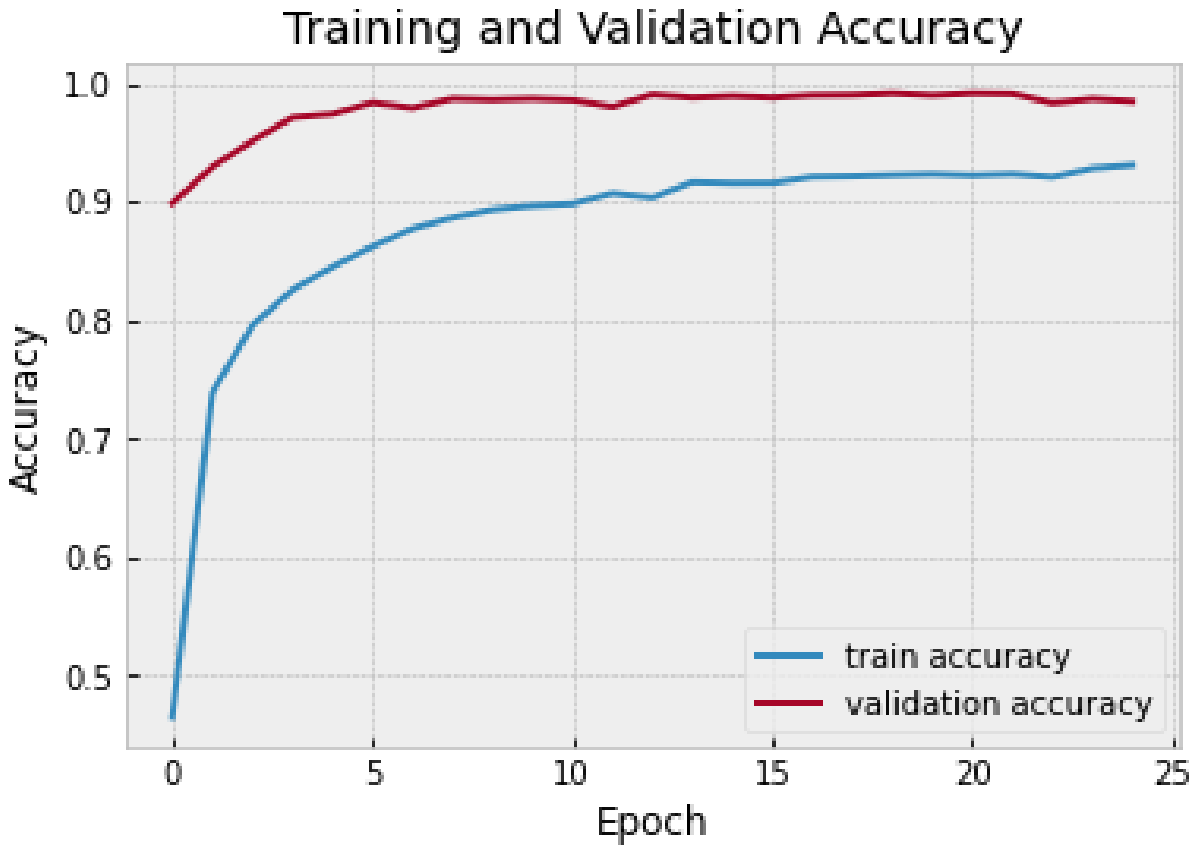


Рисунок 45 – График изменения точности (Accuracy) работы нейронной сети на тренировочных данных (train accuracy) и данных для валидации (validation accuracy) в зависимости от номера эпохи обучения (Epoch)

Теперь воспользуемся тестовыми изображениями из набора данных «German Traffic Sign Recognition Benchmark» для того чтобы определить конечную точность работы нейронной сети.

Для этого загрузим информацию о классах из файла Test.csv, масштабируем до размера 30×30 px и переведем каждое тестовое изображения в формат числовой матрицы.

Теперь, воспользовавшись функцией predict_classes() сохраним результат распознавания в переменную pred. С помощью метода accuracy_score() произведем сравнение результатов работы нейронной сети pred с эталонными значениями классов labels и выведем на экран процент случаев правильного распознавания дорожных знаков. Код представлен на рисунке 46. Точность работы нейронной сети на тестовых изображениях составила 94,8%.

```
y_test = pd.read_csv('Test.csv')
labels = y_test["ClassId"].values
imgs = y_test["Path"].values

images=[]

for img in imgs:
    image = Image.open(img)
    image = image.resize((30,30))
    images.append(img_to_array(image))

X_test=np.array(images)
pred = model.predict_classes(X_test)
print(accuracy_score(labels, pred))
#0.9480601741884402
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/
  warnings.warn('`model.predict_classes()` is deprecated and '
0.9480601741884402
```

Рисунок 46 – Программный код для определения точности нейронной сети на тестовых данных

Для демонстрации работы нейронной сети на выбранном пользователем изображении разработана функция test_image. Функция выводит на экран исходное анализируемое изображение, в виде числа – распознанный класс изображения и стандартизированное изображение дорожного знака, связанного с номером распознанного класса. Программный код функции представлен на рисунке 47.

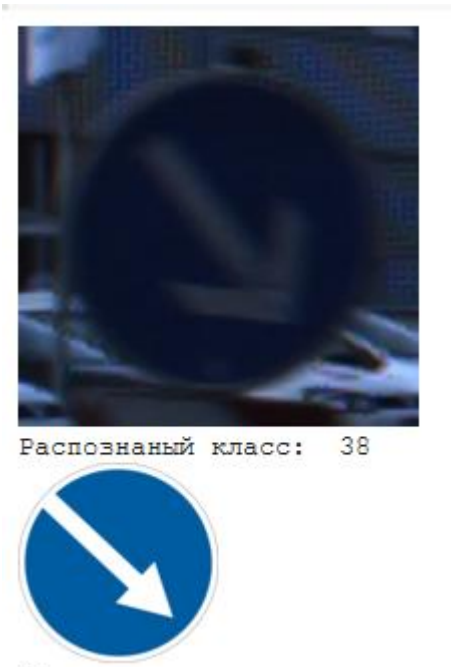
```
def test_image(test_image_id = 0):
  images=[]
  one_image = Image.open(y_test["Path"].values[test_image_id])
  one_image_view = one_image.resize((200,200))
  one_image = one_image.resize((30,30))
  warnings.simplefilter(action='ignore', category=UserWarning)
  images.append(img_to_array(one_image))
  one_X_test = np.array(images)
  one_pred = model.predict_classes(one_X_test)

  display(one_image_view)
  print('Распознаный класс: ', one_pred[0])
  one_image_class = Image.open('Meta/'+str(one_pred[0])+'.png')
  display(one_image_class)
  return one_pred[0]
```

Рисунок 47 – Программный код для тестирования работы нейронной сети на отдельных изображениях и визуализации результатов распознавания

Воспользуемся функцией test_image() для тестирования изображений в различных условиях. Примеры распознавания изображения в условиях сложного фона, неоднородной освященности, наличия эффекта «смазывания» в результате движения камеры, низкой детализации изображения, и повороте дорожного знака показаны на рисунках 48-51.

а.



б.

Рисунок 48 – Примеры распознавания дорожного знака: а – в условиях сложного неоднородного фона, б – в условиях низкого контраста (темное время суток)

а.

б.

Рисунок 49 – Примеры распознавания дорожного знака: а – в условиях неоднородной освещенности, б – при наличии эффекта «смазывания» в результате движения

а.

б.

Рисунок 50 – Примеры распознавания дорожного знака: а – при наличии фрагмента другого знака, б – при низкой детализации изображения

а.

б.

Рисунок 51 – Примеры распознавания дорожного знака: а – при наличии наклона знака, б – при расположении знака под углом

Таким образом, нейронная сеть показывает высокую точность распознавания дорожных знаков (точность около 95%) и способна распознавать изображения, полученные с ложных условиях.

3.4 Выводы по главе

На языке программирования Python разработано приложение для моделирования обучения и работы сверточной нейронной сети предложенной архитектуры. Кроме того, разработанное приложение обеспечивает предварительный анализ исходных данных с изображениями дорожных знаков, генерирования дополнительных изображений для малочисленных классов обучающей выборки, построение графиков изменения потерь и точности нейронной сети в зависимости от номера эпохи обучения и демонстрацию распознавания на выбранном изображении.

Точность работы нейронной сети с предложенной архитектурой на тестовых данных составляет 95%.

Заключение

Сформулируем полученные в ходе выполнения бакалаврской работы результаты:

- Наиболее точные подходы по распознаванию изображений дорожных знаков с использованием набора данных «German Traffic Sign Recognition Benchmark» основаны на трансформации признакового пространства изображений (HOG features, Haar-like features, Color histogram) и последующем анализе признаков с помощью сверточных нейронных сетей.
- Выдвинуто предположение, что с задачей распознавания дорожных знаков с высокой точностью (около 95%) должна справиться сверточная нейронная сеть с простой архитектурой (малым количеством слоев) без предварительного преобразования изображений к другому признаковому пространству.
- Обоснован выбор архитектуры нейронной сети, включающей в себя (в порядке следования) следующие слои: входной слой для получения изображения в виде числового массива 30×30×3; сверточный слой (convolution) с ядром 5×5, количеством плоскостей (каналов) 32 и функцией активации ReLU; слой субдискретизации (max-pooling) с группой уплотнения 2×2; слой dropout, блокирующий сигналы 25% нейронов предыдущего слоя; сверточный слой (convolution) с ядром 3×3, количеством плоскостей (каналов) 64 и функцией активации ReLU; слой субдискретизации (max-pooling) с группой уплотнения 2×2; слой dropout, блокирующий сигналы 25% нейронов предыдущего слоя; слой снижения размерности (flatten) данных до одномерного вектора; слой (core layer ReLU) из 500 нейронов прямого распространения с функцией активацией ReLU; слой dropout, блокирующий сигналы 50% нейронов предыдущего слоя; выходной (core layer SoftMax) слой с функцией активацией SoftMax.

- Разработаны решения для предотвращения переобучения нейронной сети. Первое решение заключает в добавлении в структуру нейронной сети нескольких блокирующих (dropout) слоев. Второе решение заключается в генерировании дополнительных изображений в тренировочном наборе данных для малочисленных классов.
- На языке программирования Python разработано приложение для моделирования обучения и работы сверточной нейронной сети предложенной архитектуры. Кроме того, разработанное приложение обеспечивает предварительный анализ исходных данных с изображениями дорожных знаков, генерирования дополнительных изображений для малочисленных классов обучающей выборки, построение графиков изменения потерь и точности нейронной сети в зависимости от номера эпохи обучения и демонстрацию распознавания на выбранном изображении.
- Точность работы нейронной сети с предложенной архитектурой на тестовых данных составляет 95%.

Таким образом, поставленная цель работы достигнута и все сформулированные задачи решены.

Список используемой литературы

1. Андриенко М.П. Понимание повторяющихся нейронных сетей: предпочтительная нейронная сеть для данных временных рядов / М.П. Андриенко, П.А. Юдин, Е.Ю. Вишневецкая // Актуальные вопросы современной науки и образования: сборник статей Международной научно-практической конференции : в 2 ч.. 2020. – Издательство: Наука и Просвещение (Пенза), 2020. – с. 96-98. – Текст : непосредственный.

2. Близко, М.В. Нейронные сети. Реализация нейронных сетей в MATLAB / М.В. Близко // Искусственный интеллект. Теоретические аспекты, практическое применение (ИИ-2020). Донецк, 27 мая 2020 года. – Издательство: Государственное учреждение Институт проблем искусственного интеллекта (Донецк), 2020. – с. 23-26. – Текст : непосредственный.

3. Божко, А.А. Применение сверточных нейронных сетей в задаче классификации изображений / А.А. Божко, П.В. Васильев // Концепция развития и эффективного использования научного потенциала общества: Сборник статей Международной научно-практической конференции. В 2-х частях. 2020. – Издательство: Общество с ограниченной ответственностью "ОМЕГА САЙНС" (Уфа), 2020. – с. 13-16. – Текст : непосредственный.

4. Голоскоков, К.П. Применение нейронных сетей в задачах прогнозирования и проблемы идентификации моделей прогнозирования на нейронных сетях / К.П. Голоскоков // Современные проблемы прикладной информатики: сборник научных трудов. – Издательство: Санкт-Петербургский государственный инженерно-экономический университет, 2006. – с. 116-120. – Текст : непосредственный.

5. Грушевская, А.Л. Сравнительный анализ решения одной задачи классификации четырьмя типами нейронных сетей / А.Л. Грушевская, А.Н. Покровский // Современные методы прикладной математики, теории

управления и компьютерных технологий (ПМТУКТ-2013) : сборник трудов VI международной конференции, 10–16 сентября 2013 года. – Издательство: Воронежский государственный университет (Воронеж), 2013. – с. 83-84. – Текст : непосредственный.

6. Дорогов, А.Ю. Нейронные сети глубокого обучения с управляемой коммутацией нейронных плоскостей / А.Ю. Дорогов // Дистанционные образовательные технологии: Материалы IV Всероссийской научно-практической конференции (с международным участием), 2019. – Издательство: Общество с ограниченной ответственностью «Издательство Типография «Ариал» (Симферополь), 2019. – с. 284-296. – Текст : непосредственный.

7. Жуковская, А.Н. Повышение разрешения изображения с использованием сверточной нейронной сети / А.Н. Жуковская, Л.Е. Свиридова, Д.Ю. Манылов // Технологические исследования: информационное обеспечение, алгоритм проведения, интерпретация результатов: сборник статей по итогам Международной научно-практической конференции. Стерлитамак, 2020. – Издательство: Общество с ограниченной ответственностью "Агентство международных исследований" (Уфа), 2020. – с. 40-44. – Текст : непосредственный.

8. Ибрагимов Р.М. Влияние функций активации нейронных сетей на скорость обучения на примере нейронной сети с обратным распространением ошибки / Р.М. Ибрагимов // Актуальные проблемы физической и функциональной электроники: материалы 21-й Всероссийской молодежной научной школы-семинара. 2018. – Издательство: Ульяновский государственный технический университет (Ульяновск), 2018. – с. 125-126. – Текст : непосредственный.

9. Кирюхин, А.В. Применение сверточных нейронных сетей для распознавания изображений / А.В. Кирюхин, С.А. Булычева, А.И. Пиляй // Системотехника строительства. Киберфизические строительные системы:

Сборник материалов Всероссийской научно-практической конференции. 2019. – Издательство: Национальный исследовательский Московский государственный строительный университет (Москва), 2019. – с. 223-228. – Текст : непосредственный.

10. Клячин В.Н. Использование агрегированных классификаторов при технической диагностике на базе машинного обучения / В.Н. Клячин, Ю.Е. Кувайскова, Д.А. Жуков // Информационные технологии и нанотехнологии (ИТНТ-2017) – сборник трудов III международной конференции и молодежной школы. Самарский национальный исследовательский университет имени академика С.П. Королева. 2017. – Предприятие "Новая техника" (Самара), 2017. – с. 1770-1773. – Текст : непосредственный.

11. Кононова, Н.В. Исследование подсистемы контентной фильтрации с использованием методов машинного обучения / Н.В. Кононова, Ю.А. Андрусенко, Т.А. Самокаева // Студенческая наука для развития информационного общества – сборник материалов VI Всероссийской научно-технической конференции. 22–26 мая 2017. – Северо-Кавказский федеральный университет (Ставрополь), 2017. – с. 268-270. – Текст : непосредственный.

12. Кузьмина, М.Г. О возможностях моделей глубоких сверточных нейронных сетей и многоагентных систем в задачах обработки гиперспектральных спутниковых изображений / М.Г. Кузьмина, Л.П. Басс, О.В. Николаева // XXI Международная научно-техническая конференция "НЕЙРОИНФОРМАТИКА-2019": Сборник научных трудов. – Издательство: Московский физико-технический институт (национальный исследовательский университет) (Долгопрудный), 2019. – с. 91-99. – Текст : непосредственный.

13. Лебедянцев, В.В. Влияние архитектуры нейронной сети и исходных данных на работу нейронной сети для задач классификации / В.В.

Лебедянцев, М.И. Озерова // Информационные технологии в науке и производстве : материалы VII Всероссийской молодежной научно-технической конференции, 2020. – Издательство: Омский государственный технический университет (Омск), 2020. – с. 145-152. – Текст : непосредственный.

14. Малинин, П.В. Иерархический подход в задаче идентификации личности по голосу с помощью проекционных методов классификации многомерных данных / П.В. Малинин, В.В. Поляков // Доклады томского государственного университета систем управления и радиоэлектроники. №21, 2010. – Томский государственный университет систем управления и радиоэлектроники (Томск), 2010. – с. 128-130. – Текст : непосредственный.

15. Седов А.Г. Обучение сверточной нейронной сети для сегментации спутниковых четырехканальных изображений / А.Г. Седов, В.А. Павлов, Р.В. Ларионов, Н.Д. Сидоров // Перспективные технологии в средствах передачи информации - ПТСПИ-2019: Материалы XIII международной научно-технической конференции. В 2-х томах. Редколлегия: А.Г. Самойлов [и др.]. 2019. – Издательство: Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых (Владимир), 2019. – с. 165-168. – Текст : непосредственный.

16. Юрко, В.Н. Оценивание состояния оператора по изображению лица на основе глубоких сверточных нейронных сетей / В.Н. Юрко, О.Н. Корсун // Научные чтения по авиации, посвящённые памяти Н. Е. Жуковского: материалы XVI Всероссийской научно-технической конференции. 2019. – Издательство: Военно-воздушная инженерная академия им. Н.Е. Жуковского (г. Москва) (Москва), 2019. – с. 266-270. – Текст : непосредственный.

17. Chandre, P.R. Intrusion Detection and Prevention Using Artificial Neural Network in Wireless Sensor Networks / Pankaj R. Chandre, Parikshit N. Mahalle, Gitanjali R. Shinde // Proceeding of First Doctoral Symposium on

Natural Computing Research (RSNCR 2020). – Springer Nature Singapore Pte Ltd. 2021. – pp. 113-121. – Текст : непосредственный.

18. Liu, Zh. Human Activities Recognition from Videos Based on Compound Deep Neural Network / Zhijian Liu, Yi Han, Zhiwei Chen, Yuelong Fang, Huimin Qian, Jun Zhou // Advances in Intelligent Systems and Computing : The 10th International Conference on Computer Engineering and Networks (CENet 2020). – Springer Nature Singapore Pte Ltd. 2021. – pp. 314-326. – Текст : непосредственный.

19. Ma, L. Fruit Detection Using Faster R-CNN Based on Deep Network / Linjuan Ma, Fuquan Zhang, Lin Xu // Advances in Smart Vehicular Technology, Transportation, Communication and Applications: Proceeding of the Second International Conference on Smart Vehicular Technology, Transportation, Communication and Applications, October 25-28, 2018 Mount Emei, China, Part 2. – Springer Nature Singapore Pte Ltd. 2019. – pp. 193-199. – Текст : непосредственный.

20. Sermanet, P. Traffic sign recognition with multi-scale Convolutional Networks / Pierre Sermanet, Yann LeCun // The 2011 International Conference on Neural Networks: IJCNN 2011 Conference Proceedings. – Institute of Electrical and Electronics Engineers, 2011. – pp. 25-30. – Текст : непосредственный.

21. Tambi, P. Person-Dependent Face Recognition Using Histogram of Oriented Gradients (HOG) and Convolution Neural Network (CNN) / Priya Tambi, Sarika Jain, Durgesh Kumar Mishra // International Conference on Advanced Computing Networking and Informatics: ICANI-2018. – Springer Nature Switzerland AG, 2019. – pp. 35-40. – Текст : непосредственный.

22. Xian-yan, M. Multilingual Short Text Classification Based on LDA and BiLSTM-CNN Neural Network / Meng Xian-yan, Cui Rong-yi, Zhao Ya-hui, Zhang Zhenguo // Web Information Systems and Applications: 16th International Conference, WISA 2019, Qingdao, China, September 20-22, 2019, Proceedings. –

Springer Nature Switzerland AG, 2019. – pp 319-323. – Текст : непосредственный.

23. Zhao, C. Recommendation Based Heterogeneous Information Network and Neural Network Model / Cong Zhao, Yan Wen, Ming Chen, Geng Chen // International Conference on Wireless and Satellite Systems: 11th EAI International Conference, WiSATS 2020, Nanjing, China, September 17-18, 2020, Proceedings, Part II. – Springer Nature Switzerland AG, 2021. – pp. 588-598. – Текст : непосредственный.