Санкт-Петербургский государственный университет
Кафедра математического моделирования энергетических систем

**Калинина Анастасия Викторовна**

**Выпускная квалификационная работа бакалавра**

# Оптимизация порядка реагирования аварийно-спасательных служб на чрезвычайное увеличение снежного покрова и наледи на кровлях городских зданий

Направление 010302

Прикладная математика, фундаментальная информатика и основы программирования

Научный руководитель,
доктор физ.-мат. наук,
профессор
Крылатов А. Ю.

Санкт-Петербург

2020

# Содержание

# Введение

На сегодняшний день задачи маршрутизации транспорта используются повсеместно. От международных доставок товаров до доставок еды сервисами доставки. Нахождение оптимального пути дает компаниям возможность сэкономить, не используя невыгодные маршруты. Неизвестно, кто впервые рассмотрел задачу коммивояжера, но многие века она волновала множество ученых и интузиастов, а впоследствие нашла свое продолжение в множествах других работ и вылилась в задачу маршрутизации транспорта, которая в современном мире является крайне важной темой, поскольку ввиду глобализации и технологических прорывов, локальные и международные передвижения стали повсеместными для многих компаний.

Одной из угроз жизни и здоровья граждан является падение наледи с крыш зданий. Согласно данным по здравоохранению Санкт-Петербурга [1], с 1 января 2019 года по 13 марта 2019 года из-за падения наледи с крыш в Петербурге пострадал 91 человек, один случай оказался смертельным. По данным официального сайта Администрации Санкт-Петербурга [2] за 2019 год из 13,5 тысяч кровель города 4,2 тысячи являются потенциально опасными, то есть находятся на пешеходных маршрутах и подвержены образованию наледи.

Возможным решением данной проблемы является проведение нормализации температурно-влажностного режима на кровлях зданий, однако основной путь борьбы с падением наледи – очистка крыш домов аварийными службами от снега и сосулек. Для решения вышеописанной проблемы была поставлена задача разработать программный комплекс, который по полученным изображениям и адресам зданий, классифици-

рует фотографию на наличие необходимости очистки здания и строит оптимальный маршрут для аварийно-спасательных служб, тем самым снижая риск получения ущерба здоровью граждан из-за несвоевременной очистки крыш от наледи.

# Постановка задачи

Целью выпускной квалификационной работы является изучение некоторых видов задачи маршрутизации транспорта, построение собственной задачи для оптимизации порядка реагирования аварийно-спасательных служб на чрезвычайное увеличение снежного покрова и наледи на кровлях городских зданий. Также задачей является изучение некоторых видов нейронных сетей, построение на их основе задачи классификации изображений на наличие снежных покровов и наледи на кровлях зданий.

- Изучить задачи маршрутизации транспорта.
- Кратко рассмотреть способы их решения.
- Построить собственную задачу маршрутизации транспорта для оптимизации порядка реагирования аварийно-спасательных служб и решить ее программно.
- Рассмотреть некоторые виды нейронных сетей.
- Программно решить задачу классификации изображений на наличие снежных покровов и наледи на кровлях зданий и задачу маршрутизации транспорта для оптимизации порядка реагирования аварийно-спасательных служб.

# Обзор литературы

При написании данной работы были использованы научная, учебно-методическая литература, а также публикации из научных изданий.

Основным источником для определений и понятий задачи маршрутизации транспорта стала работа «A generalized formulation for vehicle routing problems»[5]. В своей статье Pedro Munari, Twan Dollevoet, Remy Spliet подробно рассмотрели некоторые виды транспортных задач и обобщили их. Это позволило составить необходимую в данной работе задачу. В работе C. Y. Liong, I. Wan, Khairuddin Omar «Vehicle routing problem: Models and solutions» были представлены другие формулировки популярных видов задач маршрутизации транспорта и описаны методы их решения. Работы [6-30] также позволили изучить большое количество методов и алгоритмов решения данной задачи.

Главным источником в изучении нейронных сетей стали работы их первооткрывателей и вдохновителей [31][33][35]. Работа «The Perceptron - a perceiving and recognizing automaton» [31] позволила рассмотреть понятия элементарного персептрона и понять идеи, на которых он основан, а работы «Parallel Distributed Processing: Explorations in the Microstructures of Cognition Cambridge»[33] и «Receptive fields and functional architecture of monkey striate cortex»[35] помогли углубиться и подробно изучить структуры многослойного персептрона и сверточной нейронной сети, что дало возможность сконструировать архитектуру нейронной сети для задачи данной работы.

# Глава 1. Математические модели маршрутизации

## 1.1. Классическая задача коммивояжёра

Задача Коммивояжера заключается в поиске самого выгодного пути посещения городов. В условиях этой задачи задаются критерий выгодности пути (минимальная стоимость, минимальная длина и другие) и матрица расстояний, полученных из критерия выгодности.

Сеть дорог, доступных для прохождения Коммивояжером, представим в виде полного конечного графа $G = (V, E)$, где $V = \{1, 2, \ldots, n\}$ - множество вершин графа (городов), $E = \{(i, j)|i, j \in V\}$ - множество ребер графа (путей между городами). Для каждого ребра $v_{ij} = (i, j) \in E$ задается вес $c_{ij} > 0$, являющийся эквивалентом выгодности пути (стоимости, расстояния, времени и т.д.) от вершины $i$ до $j$. Целью задачи является нахождение замкнутого пути, единожды проходящего через каждую вершину графа и имеющего минимальную суммарную стоимость рёбер среди всех замкнутых путей.

Переменные задачи:

$$x_{ij} = \begin{cases} 1 & \text{если от клиента } i \text{ едем к клиенту } j \\ 0 & \text{иначе} \end{cases}$$

$u_i \geq 0$ – номер шага, на котором посетили клиента $i$

Задача:

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{1}$$

Ограничения:

$$\sum_{i \in V} x_{ij} = 1, \quad \forall j \in V, \tag{2}$$

$$\sum_{j \in V} x_{ij} = 1, \quad \forall i \in V, \tag{3}$$

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \geq 1, \quad \forall S \subset V, \quad S \neq \varnothing \tag{4}$$

$$u_i - u_j + n x_{ij} \leq n - 1, \quad i, j \in V, \quad 2 \leq i \neq j \leq n. \tag{5}$$

Ограничения (2), (3) гарантируют, что каждый клиент посещён, причём только единожды. Ограничения (5) гарантируют, что все клиенты посещены одним маршрутом (обепечивают исключение подциклов полиномиальным числом ограничений).

## 1.2. Классическая задача машрутизации транспорта

Задача маршрутизации траспорта (ЗМТ) является расширением задачи Коммивояжера. Основными нововведениями в ЗМТ являются:

- наличие точки депо
- несколько транспортных средств

Существует множество видов ЗМТ. Некоторые из наиболее распространенных: [3]

- ЗМТ со вместимостью (Capacitated Vehicle Routing Problem)
- ЗМТ с временными окнами (Vehicle Routing Problem with Time Windows)
- ЗМТ с сбором и доставкой (Vehicle Routing Problem with Pick-Up and Delivery)

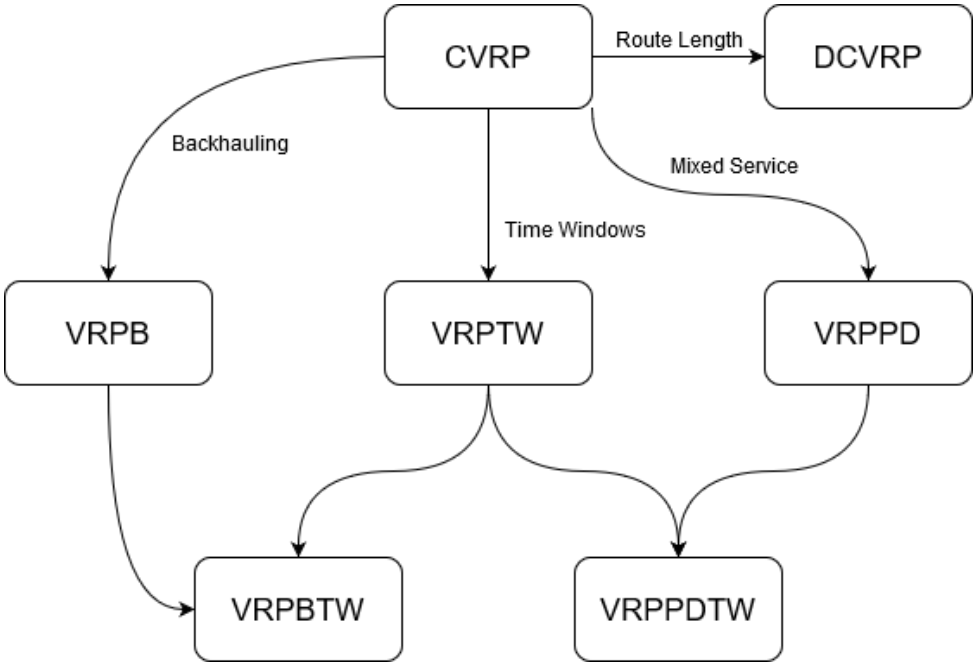


Рис. 1: Карта, показывающая соотношения между общими подзадачами ЗМТ [4].

Для начала рассмотрим классическую задачу. Под классической задачей часто понимают ЗМТ со вместимостью (CVRP), поскольку именно эта задача является основой для многих других видов ЗМТ. Здесь также будет рассмотрена эта задача в качестве классической. [5]

Имеется множество клиентов $N = \{1, 2, ..., n\}$ и точка депо $\{0\}$, $\{n+1\}$ (поскольку из депо нужно выехать и в депо нужно вернуться). Общее множество точек обозначим за $V = \{0\} \cup N \cup \{n+1\}$. Рассмотрим также $C = \{c_{i,j} | i, j \in V\}$ - матрицу стоимостей путей $(i, j)$. В дополнении, примем, что доступно $m$ транспортных средств, базирующихся в депо. Каждый автомобиль имеет максимальную вместимость $Q$. У каждого клиента есть спрос $q_i > 0 \; \forall i \in V \backslash \{0, n+1\}$. Кроме того, $y_j$ - это переменная непрерывного решения, соответствующая накопленной потребности в маршруте, который посещает клиента $j \in N$ до этого посещения. Классическая задача маршрутизации транспорта заключается в нахождении набора маршрутов транспортных средств с наименьшей стоимостью таким образом, чтобы:

- Каждый клиент из $V \backslash \{0, n+1\}$ посещался единожды и всего одним транспортным средством.
- Все маршруты начинались и заканчивались в депо.
- Все ограничения в задаче были выполнены.

Переменные задачи:

$$x_{i,j} = \begin{cases} 1 & \text{если от клиента } i \text{ едем к клиенту } j \\ 0 & \text{иначе} \end{cases}$$

$$y_i \quad \forall i \in V,$$

Задача:

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{6}$$

Ограничения:

$$\sum_{j \in V \setminus \{0,i\}} x_{ij} = 1, \quad \forall i \in V \setminus \{0, n+1\}, \tag{7}$$

$$\sum_{i \in V \setminus \{n+1,h\}} x_{ih} - \sum_{j \in V \setminus \{0,h\}} x_{hj} = 0, \quad \forall h \in V \setminus \{0, n+1\}, \tag{8}$$

$$\sum_{j \in V \setminus \{0,n+1\}} x_{0j} \leq m, \tag{9}$$

$$y_j \geq y_i + q_j x_{ij} - Q(1 - x_{ij}), \quad i, j \in V, \tag{10}$$

$$d_i \leq y_i \leq Q, \quad i \in V, \tag{11}$$

Ограничения (7) гарантируют, что все клиенты посещаются ровно один раз. Ограничения (8) гарантируют правильный поток транспортных средств через дороги, утверждая, что если транспортное средство прибывает к клиенту $h \in N$, то оно должно отходить от этого узла. Ограничение (9) ограничивает максимальное количество маршрутов до $m$ - количества транспортных средств. Ограничения (10) и (11) вместе гарантируют, что вместимость автомобиля не будет превышена. Целевая функция определяется формулой (6) и предполагает, что общая стоимость проезда по маршрутам минимизирована. Ограничения (10) также позволяют избежать обходных путей в решении, то есть задействовать маршруты, которые не проходят через депо.

## 1.3. Задача маршрутизации транспорта с временными окнами (VRPTW)

Задача маршрутизации транспорта с временными окнами является обобщением задачи маршрутизации транспорта. Она заключается в том, чтобы оптимизировать использование парка транспортных средств, которые должны сделать несколько остановок для обслуживания группы клиентов, и указать, какие клиенты должны обслуживаться каждым транспортным средством, и в каком порядке минимизировать стоимость с учетом вместимости транспортного средства и ограничения по времени обслуживания [6].

ЗМТ с временными окнами может быть определена следующим образом [5]:

VRPTW является расширением CVRP, в котором для посещений установлены временные рамки, в которые можно посетить клиентов. Временное окно соответствует временному интервалу $[w_i^a, w_i^b]$, который подразумевает, что посетить клиента $i \in N$ можно не раньше, чем в момент времени $w_i^a$, и не позже, чем $w_i^b$. Если транспортное средство прибывает до момента времени $w_i^a$, то оно должно ожидать, чтобы начать обслуживание клиента. Каждому возможному пути $(i, j)$ мы назначаем время прохождения $t_{ij}$, которое учитывает неравенство треугольника. Кроме того, каждый клиент $i$ имеет время обслуживания $s_i$, которое соответствует минимальному количеству времени, в течение которого транспортное средство должно оставаться в посещаемом узле.

Пусть $w_i$ - переменная непрерывного решения, представляющая момент времени, когда начинается обслуживание клиента $i \in N$.

$$x_{i,j} = \begin{cases} 1 & \text{если от клиента } i \text{ едем к клиенту } j \\ 0 & \text{иначе} \end{cases} \quad \forall i, j \in V,$$

$$y_i \quad \forall i \in V,$$

$$w_i \quad \forall i \in V$$

Задача:

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{12}$$

Ограничения:

$$\sum_{j \in V \setminus \{0,i\}} x_{ij} = 1, \quad \forall i \in V \setminus \{0, n+1\}, \tag{13}$$

$$\sum_{i \in V \setminus \{n+1,h\}} x_{ih} - \sum_{j \in V \setminus \{0,h\}} x_{hj} = 0, \quad \forall h \in V \setminus \{0, n+1\}, \tag{14}$$

$$\sum_{j \in V \setminus \{0,n+1\}} x_{0j} \leq m, \tag{15}$$

$$y_j \geq y_i + q_j x_{ij} - Q(1 - x_{ij}), \quad i, j \in V, \tag{16}$$

$$d_i \leq y_i \leq Q, \quad i \in V, \tag{17}$$

$$w_j > w_i + (s_i + t_{ij}) x_{ij} - M_{ij}(1 - x_{ij}), \quad i \in V \setminus \{n+1\}; j \in V \setminus 0, \tag{18}$$

$$w_i^a \leq w_i \leq w_i^b, \quad i \in V, \tag{19}$$

где $M_{ij} = max\{w_i^b - w_i^a, 0\}$.

## 1.4. Задача маршрутизации транспорта для оптимизации порядка реагирования аварийно-спасательных служб

На основе CVRP и VRPTW введем задачу маршрутизации транспорта для оптимизации порядка реагирования аварийно-спасательных служб.

В данной задаче имеется несколько особенностей:

- Имеется депо, из которого выезжает транспорт и в котором транспорт заканчивает свой путь
- Есть несколько одинаковых машин
- Есть ограничение по времени в виде времени рабочего дня
- На каждое здание тратится время для его очистки от снежного покрова и наледи

В случае этой задачи будет необходимо исключить ограничения вместимости транспорта, поскольку дорожные службы подобными параметрами не ограничены.

Имеется множество зданий, подлежащих очистке, $N = \{1, 2, ..., n\}$ и точка депо $\{0\}$, $\{n + 1\}$ (поскольку из депо нужно выехать и в депо нужно вернуться). Общее множество точек обозначим за $V = \{0\} \cup N \cup \{n + 1\}$. Рассмотрим также $T = \{t_{i,j} | i, j \in V\}$ - матрицу времени прохождения путей $(i, j)$. Множество путей обозначим за матрицу $A = \{(i, j) | i \in V, j \in \ V, j \neq i, t_{i,j} \neq 0\}$. Под $A^+$ понимается множество $A \cup \{(0, n + 1)\}$. В дополнении, примем, что доступно $k$ транспортных средств, базирующихся в депо, за множество транспортных средств обозначим $K = \{1, \ldots, k\}$. Обозначим за $\tau$ среднее время, необходимое для

очистки здания, а за $T_{max}$ - время рабочего дня.

Сформулируем задачу математически:

Переменные задачи:

$$x_{i,j}^r = \begin{cases} 1 & \text{если от здания } i \text{ r-ая машина едет к зданию } j \\ 0 & \text{иначе} \end{cases} \qquad (i,j) \in A^+$$

$$y_i^r = \begin{cases} 1 & \text{если здание } i \text{ есть на маршруте } r \\ 0 & \text{иначе} \end{cases} \qquad r \in K, i \in V$$

Задача:

$$\min \sum_{r \in K} \sum_{i \in V} \sum_{j \in V} t_{ij} x_{ij}^r \tag{20}$$

Ограничения:

$$\sum_{j \in V} x_{ij}^r = y_i^r, \quad i \in N, r \in K, \tag{21}$$

$$\sum_{r \in K} y_i^r = 1, \quad i \in N, \tag{22}$$

$$\sum_{i \in V} x_{ih}^r - \sum_{j \in V} x_{hj}^r = 0, \quad h \in N, r \in K, \tag{23}$$

$$\sum_{i \in V} x_{0i}^r = 1, \quad r \in K, \tag{24}$$

$$\sum_{i \in V} x_{i(n+1)}^r = 1, \quad r \in K, \tag{25}$$

$$\sum_{i \in V} \sum_{j \in V} t_{ij} x_{ij}^r + \sum_{i \in N} y_i^r \tau \le T_{max}, \quad r \in K, \tag{26}$$

$$\sum_{i \in V} y_i^r \ge \sum_{i \in V} \sum_{j \in V} x_{ij}^r + 1, \quad r \in K, \tag{27}$$

$$x_{ij}^r + x_{ji}^r \le 1, \quad i,j \in V, r \in K. \tag{28}$$

Ограничение (22) устанавливает, что каждое здание посещено единожды.

Ограничения (23), (24), (25) - ограничения сохранения потока, описыва-

ющие пути транспортных средств. Неравенство (26) устанавливает ограничение на общее время пути и очистки зданий. Ограничения (27), (28) гарантируют отсутствие подциклов.

# Глава 2. Способы решения задач маршрутизации транспорта

## 2.1. Обзор и классификация методов

Задачи маршрутизации транспорта являются задачами комбинаторной оптимизации, поэтому не существует аналитического метода их решения. Они также являются NP-трудными, это значит, что не существует алгоритма нахождения оптимального решения быстрее чем за полиномиальное время. Поскольку нет наиболее эффективного способа решения, существует огромное множество алгоритмов решения задач маршрутизации.

Алгоритмы решения можно разделить на две основные группы:

- Точные алгоритмы
- Эвристические алгоритмы

Точные алгоритмы гарантированно приходят к точному решению, но не всегда быстро, поскольку не ограничены по времени.

Наиболее популярные точные алгоритмы:

- Метод ветвей и границ [8]
- Метод ветвей и сечений [9]
- Метод релаксации [10]

Эвристические алгоритмы в свою очередь могут найти решение эффективно по времени, но не всегда такое решение является оптимальным.

**Algorithm 1** Алгоритм ветвей и границ

1: Начинаем с некоторой задачи $P_0$
2: $S = \{P_0\}$ - множество активных подзадач
3: Верхняя граница $L = \infty$
4: **while** $S$ не пусто **do** выбрать подзадачу $P$ из $S$ и удалить ее из $S$, разбить $P$ на меньшие подзадачи $P_1, P_2, ..., P_k$
5: **for** $i = 1$ **to** $k$ **do**
6: **if** $P_i$ - полное решение **then** обновить лучший результат
7: **if** нижняя граница $P_i <$ верхней границы $L$ **then** добавить $P_i$ в $S$
8: **end if**
9: **end if**
10: **end for**
11: **end while**
12: **return** $L$

Эвристические алгоритмы можно также разделить на несколько категорий. [11]

- Методы построения

  Постепенно создается допустимое решение, учитывая рост его стоимости. Но такие методы не содержат этап дальнейшего улучшения.

  - Алгоритм сбережений Кларка-Райта [12]
  - На основе соответствие (Matching based) [13]
  - Эвристика улучшения нескольких маршрутов
    - Алгоритм Томпсона-Псарафтиса [14]
    - Алгоритм Вана Бридама [15]
    - Алгоритм Киндуотер-Савельсберга [16]

- Двухфазные алгоритмы

Задача разбивает на две составные части: (1) кластеризация вершин в допустимые маршруты и (2) фактическое построение маршрутов с возможными петлями обратной связи между двумя этапами.

  - Алгоритмы кластер-маршрут (Cluster-First, Route-Second Algorithms)
    * Алгоритм Фишера-Джайкумара [17]
    * Алгоритм лепестков (Petal Algorithm) [18]
    * Алгоритм заметания (Sweep Algorithm) [19]
    * Алгоритм Тэйларда [20]
  - Алгоритмы маршрут-кластер (Route-First, Cluster-Second Algorithms)

- Метаэвристические алгоритмы
  - Муравьиные алгоритмы [21][22]
  - Программирование в ограничениях [23]
  - Алгоритм детерминированного отжига [24]
  - Генетические алгоритмы [25]
  - Алгоритм имитации отжига [26]
  - Поиск с запретами [27]
    * Поиск с гранулированными запретами [28]
    * Процедура адаптивной памяти [29]
    * Алгоритм Келли-Сюй [30]

**Algorithm 2** Муравьиный алгоритм в решении задачи коммивояжера

1: Задание матрицы расстояний $D$
2: Инициализация параметров алгоритма – $\alpha$, $\beta$, e, q
3: Инициализация рёбер – присвоение видимости и начальной концентрации феромона
4: Размещение муравьев в случайные города без совпадений
5: Выбор кратчайшего маршрута и определение $L^*$
6: **for** $t = 1$ **to** $max\ t$ **do**
7:  **for** $k = 1$ **to** $m$ **do**
8:  Построить маршрут $T(t)$ и рассчитать длину $L(t)$
9:  **if** $L(t) < L^*$ **then** $L^* = L(t)$, $T* = T(t)$
10:  **end if**
11:  **end for**
12:  **for** все ребра графа **do**
13:  Обновить следы феромона на ребре
14:  **end for**
15: **end for**
16: **return** кратчайший маршрут $T^*$ и его длину $L^*$

**Algorithm 3** Алгоритм имитации отжига

Задание начального состояния $S_0$, начальной температуры $T_{max}$ и конечной температуры $T_{min}$, на которой алгоритм закончит работу
$i := 0$
$T_0 := T_{max}$
$S_{best} := S_0$
**while** $T_i > T_{min}$ **do**
    Получаем новое состояние $S_new$
    $\Delta E := E(S_{new}) - E(S_{i-1}$ ▷ Вычисляем разность энергий состояний
    **if** $P(\Delta E, T_i \geq random(0,1)$ **then** ▷ Переходим ли в новое состояние. $P$ - функция вероятности перехода
        $S_i := S_{new}$
    **else**
        $S_i := S_{i-1}$
    **end if**
    **if** $E(S_i) < E(S_{best})$ **then** ▷ Запоминаем лучшее состояние
        $S_{best} := S_i$
    **end if**
    $i := i + 1$
    $T_{i+1} := D(i)$ ▷ Понижаем температуру
**end while**
**return** $S_{best}$

# Глава 3. Распознование наледи посредством искусственных нейронных сетей

## 3.1. Персептрон Розенблатта

Одним из первых представителей нейронных сетей, способных к восприятию (перцепции) и реакции на стимул, принято считать *персептрон Розенблатта*[31]. В своей работе автор рассматривал его как модель работы мозга.

Персептрон Розенблатта содержит подобные нейронам мозга элементы трех типов. S-элементы являются сенсорными или рецепторными элементами (сетчатка), принимающими двоичные сигналы (0–состояние покоя, 1–состояние активации). Они передают сигналы в слой A-элементов (ассоциативный слой) по S-A связям. Ассоциативные элементы представляют собой нейроны, выполняющие нелинейную обработку сигналов. S-A связи могут иметь веса, равные только -1, 0 или 1. Если количество сигналов, поступивши на A-элемент, превышает некоторый его порог $\theta$, то этот A-элемент активируется и выдает сигнал, равный 1. В ином случае, генерируется сигнал, равный 0. Дальше сигналы, созданные активированными A-элементами, идут к R-элементам (реагирующим элементам), формирующим реацию персептрона на входные сигналы, по A-R связям. R-элементы суммируют взвешенные сигналы от A-элементов и, если превышен определенный порог, генерируют выходной сигнал, равный 1. Математически, функцию, реализуемую $j$-ым R-элементом, можно записать так:

$$f_j(x) = \text{sign}(\sum_{i=1}^{n} w_{ij} - \theta_j)$$

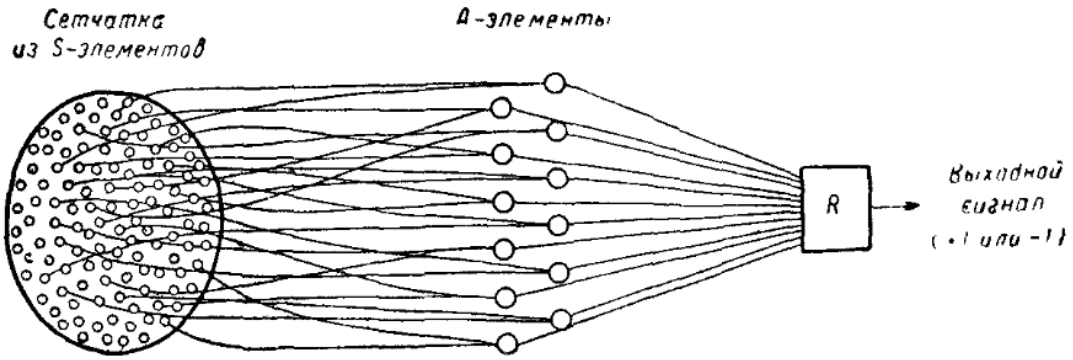


Рис. 2: Структурная схема элементарного персептрона [32].

Данная модель нейронной сети в современном понимании называется однослойной, поскольку имеет только один слой A-элементов. Однослойный персептрон характеризуется матрицей синаптических связей $W$, чьи элементы $w_{ij}$ являются *весом* A-R связи и отвечают за связи, идущие от $i$-ых A-элементов к $j$-ым R-элементам.

Суть обучения персептрона состоит в изменении весовых коэффициентов матрицы $W$ (веса S-A связей выбираются случайно) и порогов $\theta_j$ с целью уменьшения разности между желаемыми и получаемыми значениями на выходе.

Классическим методом обучения является *метод коррекции ошибок*. Его суть заключается в том, что веса связей изменяются только в том случае, если реакция персептрона была неверной. В случае неправильной реакции вес изменяется на 1 с противоположным знаком от знака ошибки.

## 3.2. Многослойный персептрон

*Многослойный персептрон* - это нейронная сеть прямого распространения, являющаяся обобщением персептрона Розенблатта. Элементы сети образуют послойную топологию с прямой передачей сигнала в том смысле, что входной сигнал в таких сетях передается от слоя к слою в прямом направлении.

Свойства многослойного персептрона:

- Наличие входного слоя, состоящего из входных элементов.
- Наличие одного и нескольких скрытых слоев вычислительных нейронов.
- Наличие выходного слоя нейронов, дающего реакцию персептрона на входные элементы.

Примерная структура многослойного персептрона демонстрируется на рисунке 3. Такая архитектура сети была предложена в работе (Rumelhart, McClelland) (1986)[33]. В отличие от персептрона Розенблатта, каждый узел сети считает свою взвешенную сумму входов, учитывая порог активации, и пропускает эту сумму через функцию активации, а не только R-элементы. Далее выходное значение отправляется в следующий слой. Веса связей и их пороговые значения являются параметрами сети, вычисляемыми в процессе обучения. Подобная структура может решать задачу практически любой сложности, благодаря выбору числа слоев и числа элементов в каждом слое.

Одним из самых популярных способов обучения многослойного персептрона является *метод обратного распространения ошибки.*

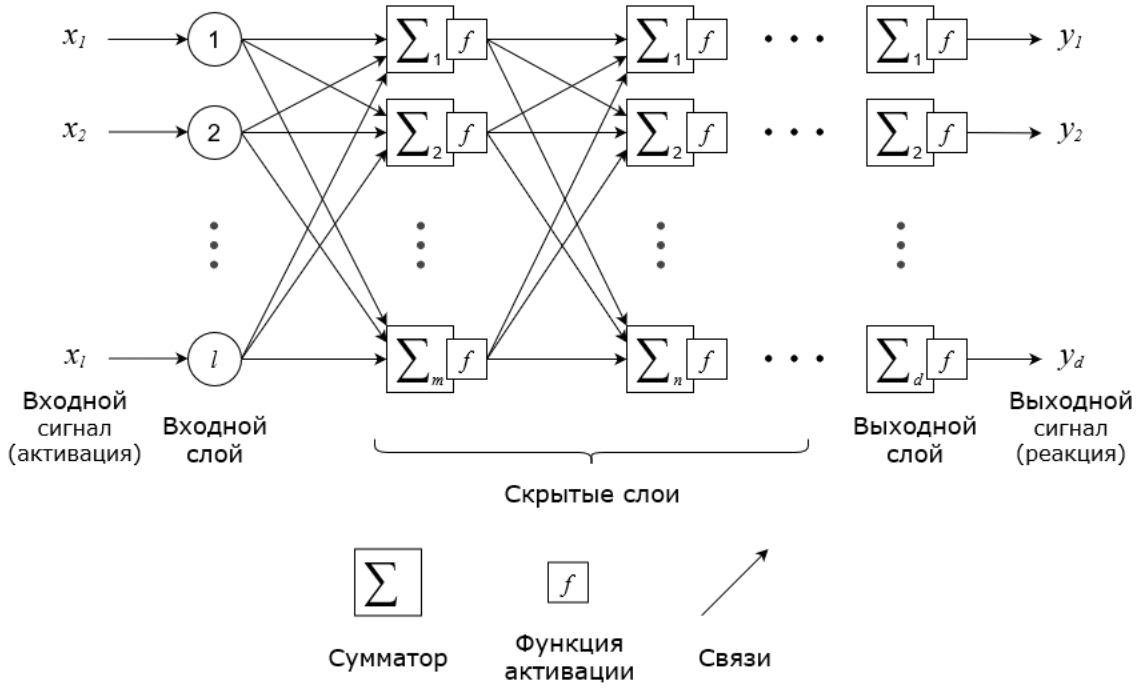


Рис. 3: Структурная схема многослойного персептрона.

Все веса в данном методе изначально задаются случайным образом. После обработки многослойным персептроном входных данных, то есть прохождения этих данных через входной слой, скрытые слои, а затем до входного уровня (прямого распространения). При достижении выхода на последнем слое вычисляется разница между предсказанным значением и необходимым (ошибка). Данная ошибка распространяется обратно, корректируя веса связей. Для этого используются частные производные функций активации. Определение изменений в весах определяется градиентным спуском и значением скорости обучения. Градиент позволяется показать величину наклона направления уменьшения или увеличения весов связей. Скорость обучения - скалярный параметр, указывающий на размер изменений. Меньшая скорость обучения часто способствует улучшению качества обучения. Обратное распространение ошибки применяется до тех пор, пока не будет достигнута требуемая точность, или не закончится время обучения нейронной сети.

Рассмотрим также несколько популярных видов функций активации нейронов.

*Функцией активации* в данной работе называется функция, вычисляющая выходное значение искусственного нейрона.

1. **Ступенчатая пороговая функция**

   Если выходное значение меньше значения порога, то значение функции равно минимально допустимому (обычно нулю), в ином случае – максимально допустимому (обычно единице).

$$f(x) = \begin{cases} 0 & x < \theta \\ 1 & x \geq \theta \end{cases}$$

   где $x$ - взвешенная сумма с учетом порогового значения.

2. **Линейная пороговая функция**

   Кусочно-линейная функция, имеющая два участка, где равна минимально допустимому (обычно нулю) и максимально допустимому значению (обычно единице) и также есть участок, на котором функция строго монотонно возрастает.

$$f(x) = \min\{\max\{\alpha x, 0\}, 1\},$$

   где $x$ - взвешенная сумма с учетом порогового значения, а $\alpha$ - коэффицент наклона линейной функции.

3. **Сигмоидальная пороговая функция**

   Гладкая монотонная возрастающая нелинейная функция. Сигмоидальная функия позволяет усиливать слабые сигналы и не насыщаться от сильных сигналов. Главным достоинством этой функции,

является то, что она не является бинарной, в отличие от ступенчатой функции, для нее характерен гладкий градиент. Данная функция хороша для задач классификации.

$$f(x) = \frac{1}{1 + e^{-\alpha x}},$$

где $x$ - взвешенная сумма с учетом порогового значения, а $\alpha$ - коэффицент наклона сигмоидальной функции.

4. **Гиперболический тангенс**

   Гиперболический тангенс является скорректированной сигмоидальной функцией.

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 = 2\text{sigmoid}(2x) - 1,$$

   где $x$ - взвешенная сумма с учетом порогового значения.

5. **ReLu** (Выпрямитель)

   Данная функция возвращает значение взвешенной суммы с учетом порога $x$, если $x$ положительно, и 0 в противном случае. Данная функция хороша для задач аппроксимации.

$$f(x) = \max\{x, 0\}$$

   Функция ReLu является менее требовательной для вычислений, чем гиперболический тангенс или сигмоидальная, поскольку упрощает математические операции.

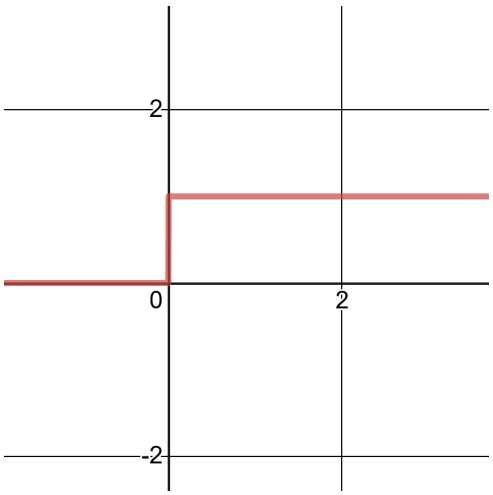


Рис. 4: Ступенчатая функция.

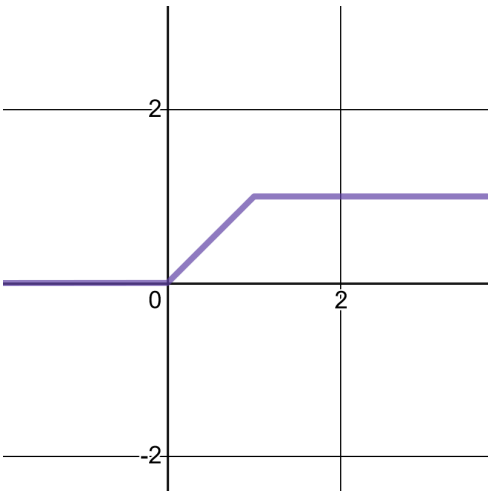


Рис. 5: Линейная функция.

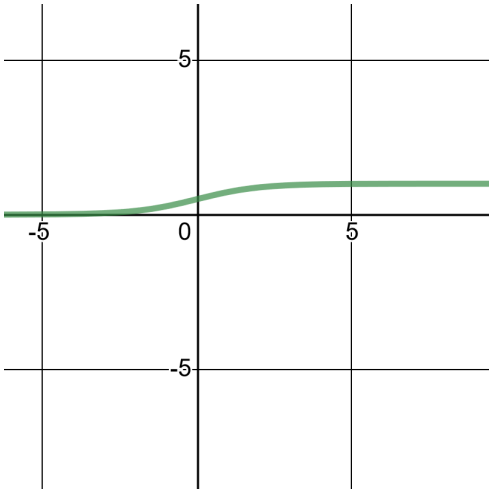


Рис. 6: Сигмоидальная функция.

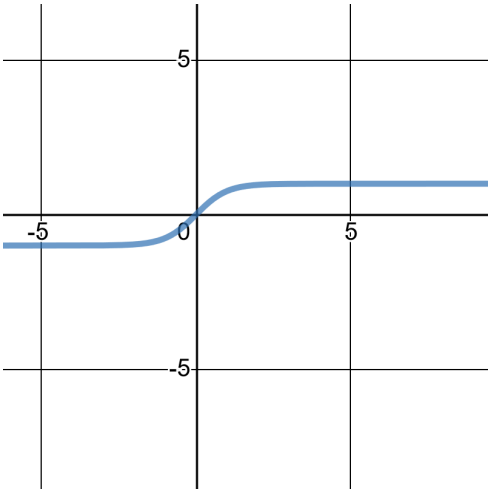


Рис. 7: Гиперболический тангенс.

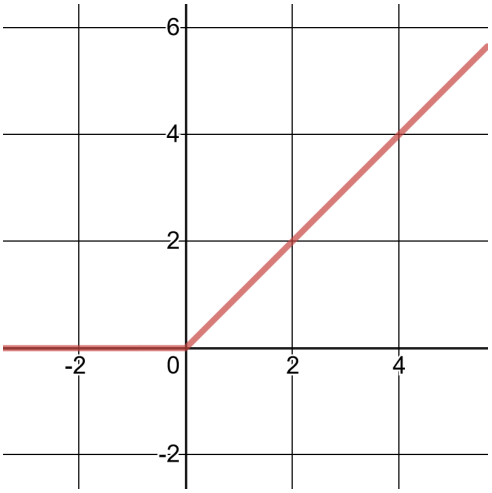


Рис. 8: ReLu.

### 3.3. Сверточная нейронная сеть

Одной из задач данной работы является классификация изображений на наличие снега и наледи. Отличным решением для данной стороны вопроса может послужить сверточная нейронная сеть.

Архитектура этой сети была предложена Яном Лекуном в 1988 году [34]. Из-за особенностей структуры сверточной сети ее главным применением является распознавание образов. Идея данной сети основана на особенностях зрительной коры. Из исследований Хьюбела 1968 года [35] стало ясно, что в зрительной коре обезьян и кошек существуют отдельные нейроны, реагирующие на раздражители только на небольших областях поля зрения, называемых рецептивным полем. Все такие нейроны покрывают всю область поля зрения.

Главным отличием сверточных нейронных сетей от многослойного персептрона, является то, что сверточные сети работают с матрицами образов. На примере распознавания изображений можно заметить, что многослойные персептроны работают с векторами пикселей, и поэтому не обращают внимание на соседние пиксели, все точки для них совершенно равнозначны. Изображения же обладают повсеместной локальной связностью.

Рассмотрим структуру сверточной нейронной сети подробнее. Матрица образов (изображение в нашем случае), проходит через серию свёрточных, слоев пулинга, слоев активации и полносвязных слоёв, и генерируется вывод. Выводом может быть класс или вероятность классов, которые лучше всего описывают изображение.

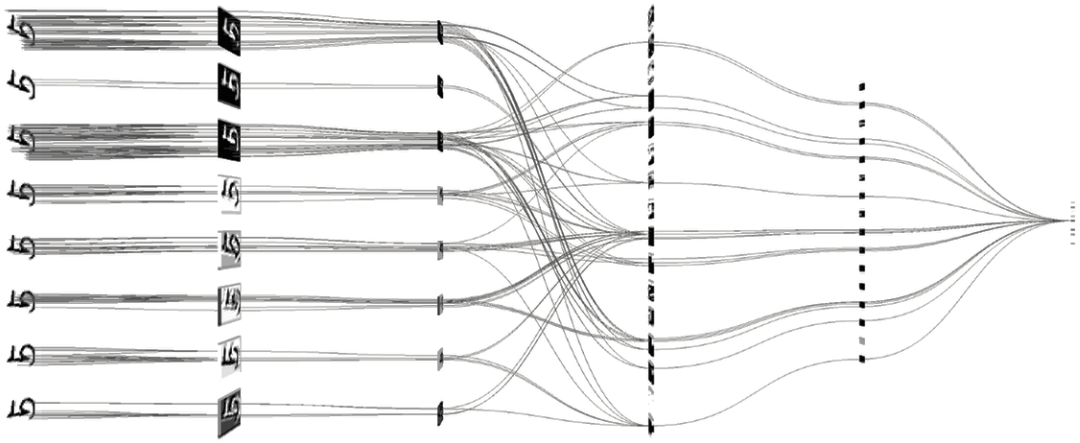

Рис. 9: Изображение работы сверточной нейронной сети в разпознавании рукописных цифр.

1. **Предобработка изображений**

   Поскольку изображения могут иметь разный и часто большой размер, изображения масштабируются и иногда кадрируются для приведения их к оптимальному виду. Огромный размер изображений влечет за собой огромное количество математических операций, общее время выполнения которых может быть слишком велико.

   Также могут выполнены операции снижения шума, например с помощью медианного или гауссовского фильтров. Изображения часто поворачивают, например для задачи распознавания лиц изображение можно повернуть, чтобы глаза оказались на одной горизонтальной прямой.

   Также возможны выделения контуров на изображениях, их сегментация, нормализация. В общем, делается все, чтобы облегчить задачу для нейронной сети.

2. **Сверточный слой** Основным блоком сверточной нейронной сети является слой свертки. Его основной целью является выделение при-

знаков на входном изображении и формирование карты признаков, то есть массива матриц, где каждая матрица отвечает за определенный выделенный признак. Слой свертки содержит для каждого канала входного изображения свой фильтр (ядро свертки).

Карта признаков в этом слое получается благодаря операции свертки, то есть операции вычисления значения выбранного пикселя, учитывая значения окружающих пикселей

Свертка происходит следующим образом. Начиная с верхней левой части для каждого канала изображения матрица фильтра поэлементно умножается на матрицу значений фрагмента и суммируется, получается значение пикселя карты признака, после чего фильтр перемещается дальше по изображению, пока все фрагменты не будут обработаны. Затем получившиеся для каждого канала матрицы суммируются. После этого к каждому элементу матрицы прибавляется одинаковое число – значение смещения данного фильтра. Полученная таким образом матрица составляет один канал выходной карты признаков.

В процессе обучения изменяются веса фильтров (элементы матриц фильтров) и значения смещений.

Неизменяемые параметры сверточного слоя:

- **Число фильтров**
- **Размер фильтров** - высота и ширина матриц фильтров
- **Шаг свертки** (Stride) - количество пикселей, на которое матрица фильтра перемещается по изображению за раз
- **Дополнение нулями** (Padding) - количество пикселей, кото-

рое добавляется по краям входного изображения. Эта параметр позволяет избежать уменьшение размерности изображения после фильтрации

3. **Слой пулинга** (подвыборки)

   Данный слой необходим для уменьшения размера каналов карты признаков и сохранения наиболее важных признаков. Данная операция помогает значительно уменьшить число вычислений, за счет уменьшения размерности, подавить шумы, оставить доминирующие признаки, там самым помогая избежать переобучения.

   Виды пулинга:

   - Максимальный пулинг (Max Pooling)

     Выдает максимальное значение из матрицы изображения, покрываемой ядром пулинга. Максимальный пулинг также полностью подавляет шум.

   - Средний пулинг (Average Pooling)

     Выдает среднее арифметическое значений из матрицы изображения, покрываемой ядром пулинга.

   - Пулинг суммы (Sum Pooling)

     Выдает сумму значений из матрицы изображения, покрываемой ядром пулинга.

   Наиболее часто используется максимальный пулинг.

   У данного слоя есть один неизменяемый параметр - шаг пулинга, то есть число раз во сколько нужно уменьшить размер входящего изображения.

Слой пулинга обычно находится после сверточного слоя или слоя активации перед слоем следующей свёртки.

4. **Слой активации**

   В данном слое к каждому значению входящего изображения применяется функция активации. Слой активации, как правило, вставляется после сверточного слоя.

5. **Полносвязный слой** (Многослойный персептрон)

   Данный слой переводит полученные после нескольких операций свертки, активации, пулинга высокоуровневые признаки в вектор классов, то есть классифицирует изображение.

   Полносвязные слои в такой сети теряют пространственную структуру писелей и обладают сравнительно небольшой размерностью. То есть полученные высокоуровневые признаки необходимо перевести от матричного вида к виду вектора для дальнейшей классификации полносвязной сетью.

В качестве обучающего метода часто используют метод обратного распространения ошибки. Также используются методы обучения без учителя.

Для повышения устойчивости работы сети и предотвращения переобучения часто используется метод тренировки подсети путем выбрасывания случайных одиночных нейронов (drop out).

# Глава 4. Оптимизация порядка реагирования аварийно-спасательных служб в зимнее время года

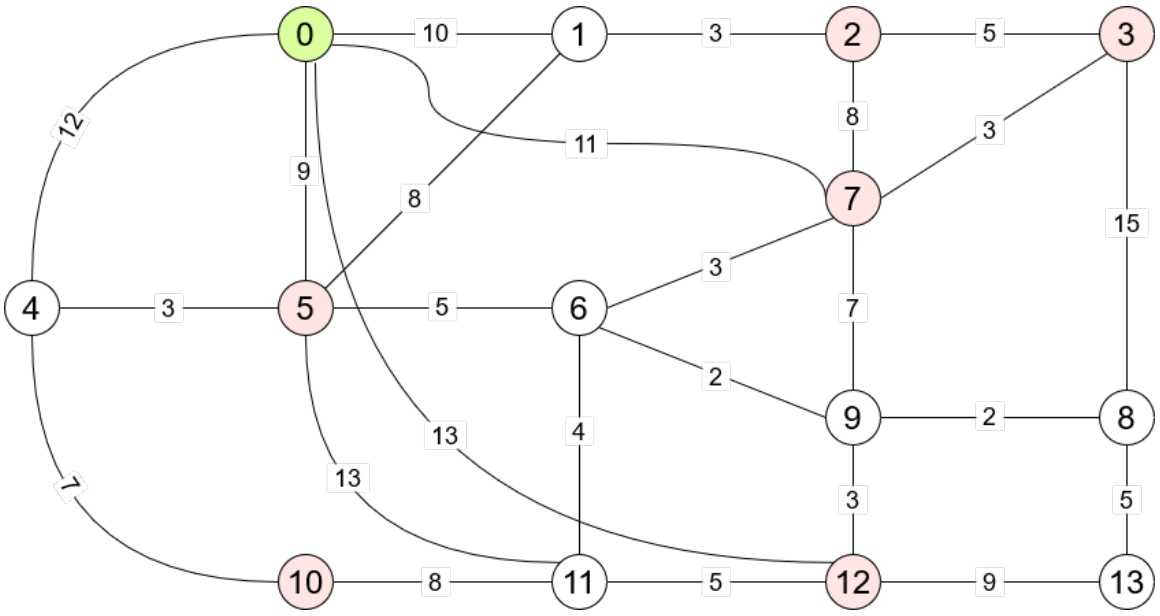


Рис. 10: Схема города, представленная в виде графа.

Рассмотрим задачу, представленную на (Рис. 10). Имеется некоторый город, состоящий из 13 зданий, соединенных различными дорогами, также имеется депо (на рисунке представлено в виде кружочка зеленого цвета). Граждане отправляют фотографии на специализированный ресурс или же используются камеры видеонаблюдения, либо иной способ получения фотографий крыш зданий. Допустим, на специализированный ресурс поступило 10 фотографий.

Допустим, у аварийно спасательной службы имеется две машины, рабочий день составляет 80 минут, а на уборку здания требуется в среднем 10 минут.

```
solve_problem(city, 2, 10, 80, [2, 3, 5, 7, 10, 12])
```

Сверточная нейронная сеть, построенная в рамках данной задачи, определила, что фотографии домов 2, 3, 5, 7, 10, 12 являются подходя-


Рис. 11: Результат классификации фотографий (изображения с номером классифицировались как подходящие).

щими, то есть эти дома требуют очистки от наледи или снега (Рис. 11).

Рассмотрим подробнее структуру сверточной нейронной сети:

Листинг 1: Код класса ConvNet

```
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.drop_out = nn.Dropout()
        self.fc1 = nn.Linear(25 * 25 * 64, 12000)
        self.fc2 = nn.Linear(12000, 2)
```

```
15
16      def forward(self, x):
17          out = self.layer1(x)
18          out = self.layer2(out)
19          out = out.reshape(out.size(0), -1)
20          out = self.drop_out(out)
21          out = self.fc1(out)
22          out = self.fc2(out)
23          return out
```

Как видно из кода, сверточная нейронная сеть состоит из:

1. Первая группа слоев
   (a) Входной слой с тремя каналами 100x100
   (b) Сверточный слой с 32-мя ядрами фильтров размером 5x5

   Полученное изображение дополняется 2-мя пикселями с каждой стороны. Свертка происходит с шагом в 1 пиксель.
   (c) Слой активации

   Используется функция ReLu.
   (d) Слой Максимального пулинга

   Уменьшает изображение в два раза.
2. Вторая группа слоев
   (a) Входной слой с 32-мя каналами 100x100
   (b) Сверточный слой с 64-мя ядрами фильтров размером 5x5

   Полученное изображение дополняется 2-мя пикселями с каждой стороны. Свертка происходит с шагом в 1 пиксель.
   (c) Слой активации

   Используется функция ReLu.

(d) Слой максимального пулинга

Уменьшает изображение в два раза.

3. Drop Out

4. Полносвязные слои

(a) Входной слой с 40000 нейронов

(b) Скрытый слой с 12000 нейронов

(c) Выходной слой с 2-мя нейронами

Классифицирует как подходящее (выход второго нейрона больше) или не подходящее (выход первого нейрона больше).

Перестроим карту города в соответствие с полученными данными: оставим только дома, которым требуется очистка, и проведем новые минимальные пути через другие здания с помощью следующей функции:

Листинг 2: Код функции create_routes_map

```
def create_routes_map(city: List[List[float]], selected_points: List[
    int]):
    selected_points = [0] + selected_points + [len(city) - 1]
    new_city = copy.deepcopy(city)
    for i in range(0, len(selected_points) - 1):
        for j in range(i + 1, len(selected_points)):
            points_for_removing = copy.copy(selected_points)
            points_for_removing.remove(selected_points[i])
            points_for_removing.remove(selected_points[j])
            city_without_points = remove_point_paths(city,
                points_for_removing)
            routes = get_all_routes(city_without_points,
                selected_points[i], selected_points[j])
            time_for_route = get_min_time(city, routes)
```

```
            new_city[selected_points[i]][selected_points[j]] =
                time_for_route
            new_city[selected_points[j]][selected_points[i]] =
                time_for_route
    new_city[0][-1] = 0
    new_city[-1][0] = 0
    new_city = remove_points(new_city, list(set(range(0, len(city)))
        ^ set(selected_points)))
    return new_city
```

Получим следующую карту города (Рис. 12):

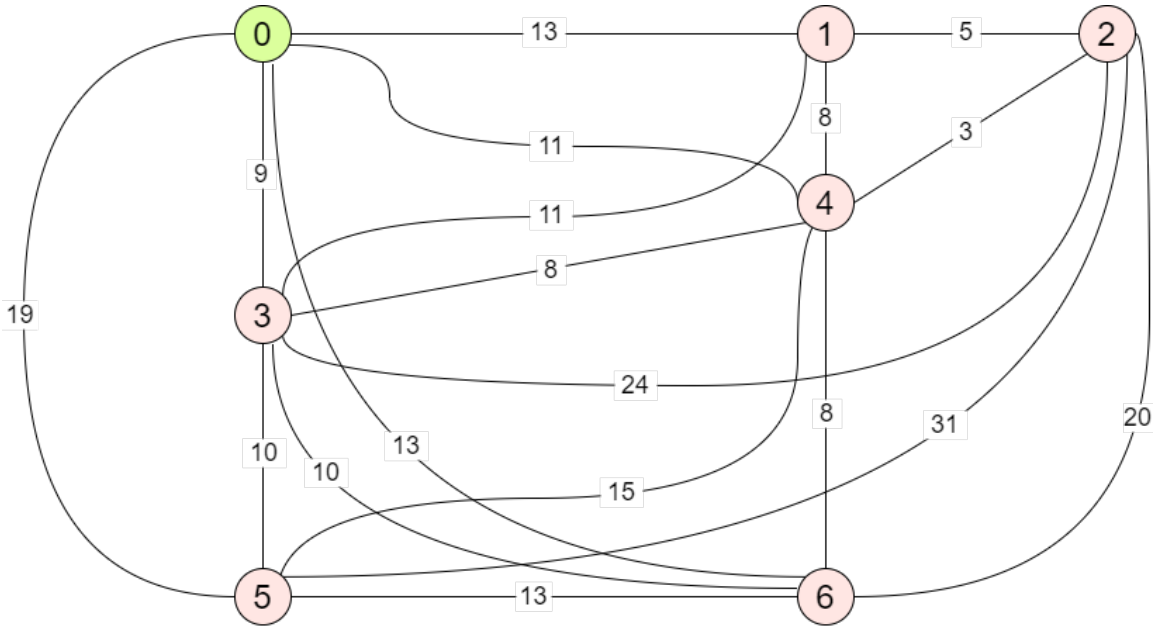


Рис. 12: Схема города только с домами, требующими очистку, представленная в виде графа.

Основой решения транспортной задачи является метод *ilp* модуля *glpk* библиотеки *cvxopt*[36] языка программирования Python, поскольку в процессе решения он использует как точные методы, так и эвристические, что позволяет добиться решения задачи за достаточно малое количество времени.

В результате решения задачи, получены следующие оптимальные

маршруты (Рис. 13):

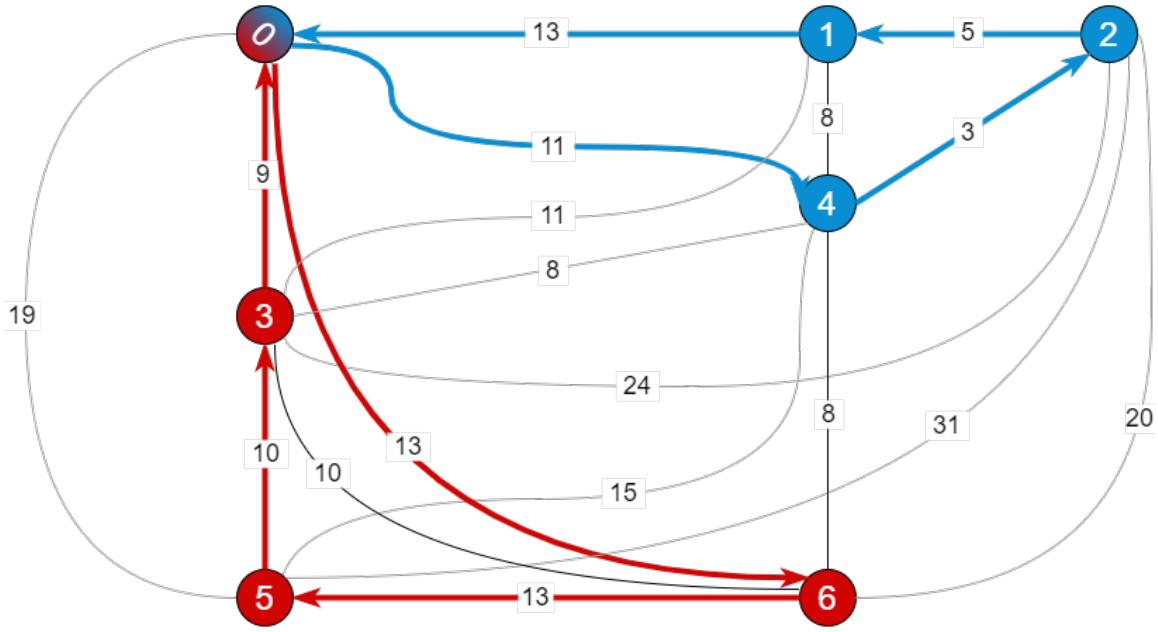


Рис. 13: Граф, демонстрирующий решение задачи.

В нашем случае, метод *ilp* нашел решение с помощью алгоритма релаксации и двойственного симплекс метода.

# Заключение

### Результаты

В работе были рассмотрены различные задачи маршрутизации транспорта и возможные методы их решения, а также принципы работы искусственных нейронных сетей. В результате проделанной работы была программно решена задача оптимизации порядка реагирования аварийно-спасательных служб на наличие необходимости проведения работ по очистки крыш зданий от наледи. На первом этапе программной реализации задачи была подробно изучена, описана и реализована сверточная нейронная сеть для классификации изображений зданий на наличие снежных покровов и наледи на крышах, на втором этапе – поставлена задача маршрутизации транспорта для оптимизации порядка реагирования аварийно-спасательных служб и ее ограничения, реализован программный метод ее решения.

### Перспективы развития

Реализованный программный комплекс решает задачи, рассматриваемые без привязки к дорогам реальной местности, что дает возможность развивать разработку для конкретного населенного пункта с его особенностями транспортного потока и располагаемыми ресурсами. А также есть возможность более точной классификации объектов по срочности выполнения работ аварийно-спасательными службами, получая больше фотографий зданий.

# Список литературы


1. Официальный сайт комитета по здравоохранению Санкт-Петербурга. http://zdrav.spb.ru/ru/

2. Официальный сайт Администрации Санкт-Петербурга. https://www.gov.spb.ru/gov/admin/

3. C. Y. Liong, I. Wan, Khairuddin Omar. Vehicle routing problem: Models and solutions // Journal of Quality Measurement and Analysis, 2008. C. 205-218.

4. Vehicle routing problem. https://en.wikipedia.org/wiki/Vehicle-routing-problem

5. Pedro Munari, Twan Dollevoet, Remy Spliet. A generalized formulation for vehicle routing problems, v2, 2017.

6. Ellabib I., Otman A. B., Calamai P. An Experimental Study of a Simple Ant Colony System for the Vehicle Routing Problem with Time Windows // Dorigo M., Di Caro G., Sampels M. (eds) Ant Algorithms, 2002.

7. Azi N., Gendreau M., Potvin J. Y. An exact algorithm for a single-vehicle routing problem with time windows and multiple routes // European Journal of Operational Research 178, 2007. C. 755–766.

8. M. L. Fisher, Optimal Solution of Vehicle Routing Problems Using Minimum K-trees // Operations Research 42, 1994. C. 626-642.

9. U. Blasum, W. Hochstattler. Application of the Branch and Cut Method to the Vehicle Routing Problem // Zentrum fur Angewandte Informatik Koln Technical Report, 2000. C. 386.

10. Malandraki C., Daskin M. S. Time dependent vehicle routing problems: formulations, properties and heuristic algorithms // Transportation science, 1992. C. 185-200.

11. Solution Methods for VRP,
http://neo.lcc.uma.es/vrp/solution-methods/

12. G. Clarke, J. Wright. Scheduling of vehicles from a central depot to a number of delivery points // Operations Research, 12 №4, 1964. C. 568-581.

13. M. Desrochers, T. W. Verhoog. A Matching Based Savings Algorithm for the Vehicle Routing Problem. Department of Production Engineering and Management, Decision Support Systems Laboratory, Technical University of Crete, Chania, Greece, 1989.

14. P. M. Thompson, H. N. Psaraftis. Cyclic Transfer Algorithms for the Multivehicle Routing and Scheduling Problems // Operations Research 41, 1993. C. 935-946.

15. A. Van Breedam. An Analysis of the Behavior of Heuristics for the Vehicle Routing Problem for a Selection of Problems with Vehicle-Related. Customer-Related, and Time-Related Constraints. Ph.D. dissertation, University of Antwerp, 1994.

16. G. A. P. Kinderwater, M. W. P. Savelsbergh. Vehicle Routing: Handling Edge Exchanges // E. H. L. Aarts, J. K. Lenstra, Local Search in Combinatorial Optimization Wiley, Chichester, 1997.

17. M. L. Fisher, R. Jaikumar. A Generalized Assignment Heuristic for Vehicle Routing // Annals of Operations Research, 1981. C. 109-124.

18. D. M. Ryan, C. Hjorring, F. Glover. Extensions of the Petal Method for Vehicle Routing // Journal of the Operational Research Society, 44, 1993. C. 289-296.

19. A. Wren. Computers in Transport Planning and Operation. Ian Allan, London, 1971.

20. Ã. D. Taillard. Parallel Iterative Search Methods for Vehicle Routing Problems // Networks 23, 1993. C. 661-673.

21. B. Bullnheimer, R. F. Hartl, C. Strauss. Applying the Ant System to the Vehicle Routing Problem // 2nd International Conference on Metaheuristics, Sophia-Antipolis, France, 1997.

22. L. M. Gambardella, E. Taillard, G. Agazzi, MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows // D. Corne, M. Dorigo and F. Glover, New Ideas in Optimization. McGraw-Hill, 1999.

23. P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems // Solving Vehicle Routing Problems Using Constraint Programming and Metaheuristics, 1998. C. 501-523.

24. G. Dueck, T. Scheurer. Threshold Accepting: A General Purpose Optimization Algorithm // Journal of Computational Physics, 90, 1990. C. 161-175.

25. J. H. Holland. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, 1975.

26. Kirkpatrick, S., Gelatt Jr, C. D., Vecchi, M. P. Optimization by Simulated Annealing // Science, 220, 1983. C. 671–680.

27. F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence // Computers and Operations Research, 13, 1986. C. 533-549.

28. P. Toth, D. Vigo. The Granular Tabu Search (and its Application to the Vehicle Routing Problem) // INFORMS Journal on Computing, 1998. C. 333-346.

29. Y. Rochat, A. D. Taillard. Probabilistic Diversification and Intensification in Local Search for Vehicle Routing // Journal of Heuristics, 1995. C. 147-167.

30. J. Kelly, J. P. Xu. A Network Flow-Based Tabu Search Heuristic for the Vehicle Routing Problem // Transportation Science, 30, 1996. C. 379-393.

31. Rosenblatt, Frank. The Perceptron - a perceiving and recognizing automaton. Report 85-460-1. Cornell Aeronautical Laboratory, 1957.

32. Розенблатт, Ф. Принципы нейродинамики: Перцептроны и теория механизмов мозга. М.: Мир, 1965. С. 480.

33. Rumelhart D. E., McClelland J. L. Parallel Distributed Processing: Explorations in the Microstructures of Cognition Cambridge. MA: MIT Press, 1986.

34. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition // Neural Computation, 1(4), 1989. C. 541-551.

35. Hubel, David H., Torsten N. Wiesel. Receptive fields and functional

architecture of monkey striate cortex // The Journal of physiology 195 1, 1968. C. 215-243.

36. Martin S. Andersen, Joachim Dahl, and Lieven Vandenberghe, CVXOPT Documentation. Release 1.2.5, https://buildmedia.readthedocs.org/media/pdf/cvxopt/latest/cvxopt.pdf

# Приложение

Листинг 3: Файл main.py (Основной)

```
import copy
from typing import List, Tuple

import cv2
import numpy as np
import torch
from cvxopt import matrix
from cvxopt.glpk import ilp

import conv_neural
from DIRECTORIES import MODEL_STORE_PATH

PIC_NUM = 2
model = conv_neural.ConvNet()
model.load_state_dict(torch.load(MODEL_STORE_PATH + "pic_conv_model.
    ckpt"))
model.eval()


def preprocessing(original_image):
    temp_image = original_image.copy()
    output = cv2.resize(temp_image, (100, 100, 3))
    output = output.astype("float32") / 255
    return output


def predict(model, image):
    tensor_input = torch.tensor(np.array([[image]]).transpose((0, 1,
        2, 3)))
    with torch.no_grad():
        result = model(tensor_input)
```

```
30          _, predicted = torch.max(result.data, 1)
31          predicted = predicted.numpy()[0]
32          return predicted
33
34
35  def remove_points(city: List[List[float]], points: List[int]) -> List
        [List[float]]:
36      new_city = copy.deepcopy(city)
37      points = sorted(points)
38      for i, point in enumerate(points):
39          del new_city[point - i]
40          for arr in new_city:
41              del arr[point - i]
42      return new_city
43
44
45  def remove_point_paths(city: List[List[float]], points: List[int]) ->
        List[List[float]]:
46      new_city = copy.deepcopy(city)
47      for point in points:
48          new_city[point] = [0] * len(city[point])
49          for arr in new_city:
50              arr[point] = 0
51      return new_city
52
53
54  def get_all_routes(city: List[List[float]], start_point: int,
        end_point: int, current_route: List[int] = []):
55      new_current_route = copy.copy(current_route)
56
57      if len(new_current_route) == 0:
58          new_current_route.append(start_point)
59      result = []
60      i = new_current_route[-1]
```

```
61
62      if i == end_point:
63          return [new_current_route]
64
65      for j in range(0, len(city[0])):
66          if city[i][j] > 0 and j not in new_current_route:
67              new_new_current_route = copy.copy(new_current_route)
68              new_new_current_route.append(j)
69              end_routes = get_all_routes(city, start_point, end_point,
                    new_new_current_route)
70
71              if len(end_routes) > 0:
72                  result = result + end_routes
73
74      return result
75
76
77  def get_time_for_route(city: List[List[float]], route: List[int]):
78      result = 0
79      for i in range(1, len(route)):
80          result += city[route[i - 1]][route[i]]
81      return result
82
83
84  def get_min_time(city: List[List[float]], routes: List[List[int]]):
85      routes_times = [get_time_for_route(city, route) for route in
           routes]
86      if len(routes_times) == 0:
87          return 0
88      return min(routes_times)
89
90
91  def create_routes_map(city: List[List[float]], selected_points: List[
       int]):
```

```
92      selected_points = [0] + selected_points + [len(city) - 1]
93      new_city = copy.deepcopy(city)
94      for i in range(0, len(selected_points) - 1):
95          for j in range(i + 1, len(selected_points)):
96              points_for_removing = copy.copy(selected_points)
97              points_for_removing.remove(selected_points[i])
98              points_for_removing.remove(selected_points[j])
99              city_without_points = remove_point_paths(city,
                    points_for_removing)
100             routes = get_all_routes(city_without_points,
                    selected_points[i], selected_points[j])
101             time_for_route = get_min_time(city, routes)
102             new_city[selected_points[i]][selected_points[j]] =
                    time_for_route
103             new_city[selected_points[j]][selected_points[i]] =
                    time_for_route
104     new_city[0][-1] = 0
105     new_city[-1][0] = 0
106     new_city = remove_points(new_city, list(set(range(0, len(city)))
            ^ set(selected_points)))
107     return new_city
108
109
110 def get_routes(city: List[List[float]]) -> List[Tuple[str, str, float
        ]]:
111     result = []
112     for i in range(0, len(city)):
113         for j in range(i + 1, len(city)):
114             if city[i][j] > 0:
115                 result.append((str(i), str(j), city[i][j]))
116     return result
117
118
119 def get_target_vector(city: List[List[float]], m: int):
```

```
120     inf_city = copy.deepcopy(city)
121     for i in range(0, len(inf_city)):
122         new_arr = []
123         for x in inf_city[i]:
124             if x != 0:
125                 new_arr.append(x)
126             else:
127                 new_arr.append(10000000000000000000000000)
128         inf_city[i] = new_arr
129     inf_city[-1][0] = 0
130     zero_city = copy.deepcopy(inf_city)
131     for i in range(len(zero_city)):
132         for j in range(len(zero_city[0])):
133             zero_city[i][j] = 0
134     return (flatten(inf_city) + [0] * len(inf_city)) * m, [[copy.
        deepcopy(zero_city)] + [[[0] * len(inf_city)]] for i in
135                                                         range(m)]
136
137
138 def get_answer_matrix(city: List[List[float]], x):
139     route_vehicle_matrix = []
140     new_x = flatten(x)
141     point_matrix = []
142     route_matrix = []
143     route_vector = []
144     for i in new_x:
145         route_vector.append(i)
146         if len(route_vector) == len(city):
147             route_matrix.append(copy.deepcopy(route_vector))
148             route_vector = []
149         if len(route_matrix) == len(city) + 1:
150             temp = copy.deepcopy(route_matrix)
151             route_vehicle_matrix.append(temp[:-1])
152             point_matrix.append(temp[-1][1:-1])
```

```
153             route_matrix = []
154
155     return route_vehicle_matrix, point_matrix
156
157
158 def set_value(zero_target: List[List[List]], x_values: List[Tuple[int
    , int, int, float]] = None,
159               y_values: List[Tuple[int, int, float]] = None):
160     new_vector = copy.deepcopy(zero_target)
161     if x_values is not None:
162         for r, i, j, value in x_values:
163             new_vector[r][0][i][j] = value
164     if y_values is not None:
165         for r, i, value in y_values:
166             new_vector[r][1][0][i] = value
167     return flatten(new_vector)
168
169
170 def flatten(x: List) -> List:
171     result = []
172     for el in x:
173         if hasattr(el, "__iter__") and not isinstance(el, str):
174             result.extend(flatten(el))
175         else:
176             result.append(el)
177     return result
178
179
180 def get_constraint_matrix(city: List[List[float]], m: int, tau: float
    , t_max: float):
181     c, zero_target = get_target_vector(city, m)
182     A_eq = []
183     A_ub = []
184     b_eq = []
```

```
185     b_ub = []
186     # 1 constraint
187     for i in range(1, len(city) - 1):
188         for r in range(0, m):
189             x_values = []
190             y_values = [(r, i, -1)]
191             for j in range(0, len(city)):
192                 x_values.append((r, i, j, 1))
193             A_eq.append(set_value(zero_target, x_values, y_values))
194             b_eq.append(0)
195 
196     # 2 constraint
197     for i in range(1, len(city) - 1):
198         y_values = []
199         for r in range(0, m):
200             y_values.append((r, i, 1))
201         A_eq.append(set_value(zero_target, y_values=y_values))
202         b_eq.append(1)
203 
204     # 3 constraint
205     for r in range(0, m):
206         for h in range(1, len(city) - 1):
207             x_values = []
208             for i in range(0, len(city)):
209                 x_values.append((r, i, h, 1))
210                 x_values.append((r, h, i, -1))
211             A_eq.append(set_value(zero_target, x_values))
212             b_eq.append(0)
213 
214     # 4,5 constraints
215     for r in range(0, m):
216         x_values1 = []
217         x_values2 = []
218         for i in range(0, len(city)):
```

```
219             x_values1.append((r, 0, i, 1))
220             x_values2.append((r, i, len(city) - 1, 1))
221         A_eq.append(set_value(zero_target, x_values1))
222         b_eq.append(1)
223         A_eq.append(set_value(zero_target, x_values2))
224         b_eq.append(1)
225 
226     # 6 constraint
227     for r in range(0, m):
228         x_values = []
229         y_values = []
230         for i in range(0, len(city)):
231             if i != 0 and i != (len(city) - 1):
232                 y_values.append((r, i, tau))
233             for j in range(0, len(city)):
234                 x_values.append((r, i, j, city[i][j]))
235         A_ub.append(set_value(zero_target, x_values, y_values))
236         b_ub.append(t_max)
237 
238     # 7 constraint
239     for r in range(0, m):
240         x_values = []
241         y_values = []
242         for i in range(0, len(city)):
243             y_values.append((r, i, -1))
244             for j in range(0, len(city)):
245                 x_values.append((r, i, j, 1))
246         A_ub.append(set_value(zero_target, x_values, y_values))
247         b_ub.append(-1)
248 
249     # 8 constraint
250     for r in range(0, m):
251         for i in range(0, len(city)):
252             for j in range(0, len(city)):
```

```
253                 x_values = []
254                 x_values.append((r, i, j, 1))
255                 x_values.append((r, j, i, 1))
256                 A_ub.append(set_value(zero_target, x_values))
257                 b_ub.append(1)
258 
259     return np.array(c, dtype=float), np.array(A_ub, dtype=float), np.
        array(b_ub, dtype=float), \
260             np.array(A_eq, dtype=float), np.array(b_eq, dtype=float)
261 
262 
263 def print_matrix(array):
264     for arr in array:
265         print(arr)
266     print()
267 
268 
269 def solve_problem(old_city: List[List[float]], m: int, tau: float,
    t_max: float, selected_points: List[int]):
270     city = create_routes_map(old_city, selected_points)
271     c, A_ub, b_ub, A_eq, b_eq = get_constraint_matrix(city, m, tau,
        t_max)
272     (status, x) = ilp(matrix(c), matrix(A_ub), matrix(b_ub), matrix(
        A_eq), matrix(b_eq), set(), set(range(len(c))))
273     print(status)
274     print(x)
275     route_vehicle_matrix, point_matrix = get_answer_matrix(city, x)
276     for i, vehicle in enumerate(route_vehicle_matrix):
277         print("\nVehicle ", i)
278         for arr in vehicle:
279             print(arr)
280     return route_vehicle_matrix, point_matrix
281 
282 
```

```
city = [
    [0, 10, 0, 0, 12, 9, 0, 11, 0, 0, 0, 0, 13, 0, 0],
    [10, 0, 3, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0, 10],
    [0, 3, 0, 5, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 5, 0, 0, 0, 0, 3, 15, 0, 0, 0, 0, 0, 0],
    [12, 0, 0, 0, 0, 3, 0, 0, 0, 0, 7, 0, 0, 0, 12],
    [9, 8, 0, 0, 3, 0, 5, 0, 0, 0, 0, 13, 0, 0, 9],
    [0, 0, 0, 0, 0, 5, 0, 3, 0, 2, 0, 4, 0, 0, 0],
    [11, 0, 8, 3, 0, 0, 3, 0, 0, 7, 0, 0, 0, 0, 0],
    [0, 0, 0, 15, 0, 0, 0, 0, 0, 2, 0, 0, 0, 5, 0],
    [0, 0, 0, 0, 0, 0, 2, 7, 2, 0, 0, 0, 3, 0, 0],
    [0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0],
    [0, 0, 0, 0, 0, 13, 4, 0, 0, 0, 8, 0, 5, 0, 0],
    [13, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 5, 0, 9, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 9, 0, 0],
    [0, 10, 0, 0, 12, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0]
]

route_vehicle_matrix, point_matrix = solve_problem(city, 2, 10, 80,
    [2, 3, 5, 7, 10, 12])
```

Листинг 4: Файл conv„eural.py

```
import torch
import torch.nn as nn

# Convolutional neural network (two convolutional layers)
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
```

```
13              nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
14              nn.ReLU(),
15              nn.MaxPool2d(kernel_size=2, stride=2))
16          self.drop_out = nn.Dropout()
17          self.fc1 = nn.Linear(25 * 25 * 64, 12000)
18          self.fc2 = nn.Linear(12000, 2)
19
20      def forward(self, x):
21          out = self.layer1(x)
22          out = self.layer2(out)
23          out = out.reshape(out.size(0), -1)
24          out = self.drop_out(out)
25          out = self.fc1(out)
26          out = self.fc2(out)
27          return out
28
29
30  def train(model, train_loader, num_epochs, optimizer, criterion):
31      total_step = len(train_loader)
32      loss_list = []
33      acc_list = []
34      for epoch in range(num_epochs):
35          for i, (images, labels) in enumerate(train_loader):
36              # Run the forward pass
37              outputs = model(images)
38              loss = criterion(outputs, labels)
39              loss_list.append(loss.item())
40
41              # Backprop and perform Adam optimisation
42              optimizer.zero_grad()
43              loss.backward()
44              optimizer.step()
45
46              # Track the accuracy
```

```
            total = labels.size(0)
            _, predicted = torch.max(outputs.data, 1)
            correct = (predicted == labels).sum().item()
            acc_list.append(correct / total)

            if (i + 1) % 10 == 0:
                print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f},
                    Accuracy: {:.2f}%'
                    .format(epoch + 1, num_epochs, i + 1,
                        total_step, loss.item(),
                            (correct / total) * 100))


def test(model, test_loader):
    model.eval()
    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in test_loader:
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        print('Test Accuracy of the model: {} %'.format((correct /
            total) * 100))


def save(model, file_path):
    torch.save(model.state_dict(), file_path + 'pic_conv_model.ckpt')
```

Листинг 5: Файл pictures_dataset.py

```
import csv

```

```
3  import cv2
4  from torch.utils.data import Dataset
5  from torchvision import transforms
6
7
8  def read_csv(file_path):
9      with open(file_path, "r", newline="") as file:
10         reader = csv.reader(file)
11         return list(reader)
12
13
14 class PicturesDataset(Dataset):
15     def __init__(self, csv_file, root_dir, train, transform=None):
16         if train:
17             self.dir = "train/"
18         else:
19             self.dir = "test/"
20         self.root_dir = root_dir + self.dir
21         self.data = read_csv(self.root_dir + csv_file)
22         self.to_tensor = transforms.ToTensor()
23         self.transform = transform
24
25     def __len__(self):
26         return len(self.data)
27
28     def __getitem__(self, idx):
29         img_name = self.data[idx][0]
30         image = cv2.imread(img_name)
31         image = image.astype("float32") / 255
32         tensor_image = self.to_tensor(image)
33         answer = self.data[idx][1]
34
35         sample = (tensor_image, int(answer))
36
```

```
        if self.transform:
            sample = self.transform(sample)

        return sample
```

Листинг 6: Файл neural_learning.py

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader

import conv_neural
import pictures_dataset
from DIRECTORIES import DATASET_PATH, MODEL_STORE_PATH

NUM_EPOCHS = 6
NUM_CLASSES = 2
BATCH_SIZE = 50
LEARNING_RATE = 0.001

model = conv_neural.ConvNet()
print("Model created.")

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)

train_dataset = pictures_dataset.PicturesDataset("data.csv",
    DATASET_PATH, True)
print("Train dataset loaded.")
test_dataset = pictures_dataset.PicturesDataset("data.csv",
    DATASET_PATH, False)
print("Test dataset loaded.")

train_loader = DataLoader(dataset=train_dataset, batch_size=
    BATCH_SIZE, shuffle=True)
```

```
print("Train dataset converted to DataLoader.")
test_loader = DataLoader(dataset=test_dataset, batch_size=BATCH_SIZE,
    shuffle=True)
print("Test dataset converted to DataLoader.")

print("Train starting.")
conv_neural.train(model, train_loader, NUM_EPOCHS, optimizer,
   criterion)
print("Train completed.")
print("Test starting.")
conv_neural.test(model, test_loader)
print("Test completed.")
conv_neural.save(model, MODEL_STORE_PATH)
print("Neural model saved.")
```

Листинг 7: Файл DIRECTORIES.py

```
ORIGINALS_PATH = 'data/original_images/'
DATASET_PATH = 'data/dataset/'
MODEL_STORE_PATH = 'models/'
CHECK_PATH = 'check/'
```