Список литературы

1. Петров И.В. CODESYS – повседневный инструмент программиста ПЛК // Автоматизация в промышленности. – 2012. – № 8.
2. OPC-сервер для протокола МЭК-61850 [Электронный ресурс] // Сайт компании INSAT. – Режим доступа: https://opc-server.insat.ru/products/?category=1143 (дата обращения: 26.11.2019).
3. Библиотека IEC 61850 для интеграции устройств защиты в SIMATIC PCS 7 [Электронный ресурс] // Сайт компании SIMATIC-MARKET. – Режим доступа: http://simatic-market.ru/catalog/Siemens-CA01/10048453/info/ (дата обращения: 26.11.2019).

КЛАССИФИКАЦИЯ АВТОМОБИЛЕЙ В СИСТЕМЕ АВТОМАТИЧЕСКОГО ПОДСЧЕТА ИНТЕНСИВНОСТИ ТРАНСПОРТНЫХ ПОТОКОВ

Бондарь А.С., Головкин М.В., Боровской А.Е.

Белгородский Государственный Технологический Университет «БГТУ», Белгород, e-mail: aaiwprton@mail.ua

Интенсивность транспортных потоков является основополагающим параметром при организации дорожного движения. Подсчет интенсивности в ручном режиме сопряжен с высокими трудозатратами, поэтому данную операцию целесообразно возложить на специализированные технические средства. В разрабатываемой системе предполагается осуществлять подсчет по видеопотоку с камеры, установленной над участком улично-дорожной сети. Задача должна решаться в несколько этапов: детектирование автомобилей в кадре, классификация и отслеживание траекторий движения. В данной статье рассматривается проблема классификации, которую необходимо осуществлять по полученному изображению после локализации объекта в кадре. В статье описана разработка структуры и процесс обучения нейронной сети. За основу взята архитектура сверточной нейронной сети, как наиболее подходящая для задачи распознавания образов. Разработанная структура нейронной сети предусматривает классификацию по пяти классам – легковой автомобиль, грузовой автомобиль и автобус, а также выделяет пешеходов и тень на дороге. Последние два класса необходимы для устранения ошибочного детектирования. В статье описан процесс составления набора данных, обучения и использования сети. Приведены результаты обучения и тестирования на расширенном наборе данных.

Классифицировать автомобили предполагается по трем классам: легковой автомобиль, грузовой автомобиль и автобус. Кроме того, нужно иметь возможность отличать автомобили от ошибочно детектированных объектов, таких как движущиеся пешеходы, автомобильные тени и пр. В ходе работы будут применяться такие программные средства, как язык программирования *Python* [1], библиотека компьютерного зрения *OpenCV* [2], фреймворк для работы с низкоуровневыми библиотеками машинного обучения *Keras* [3].

Выбор архитектуры нейронной сети. Классификация объектов по их изображениям представляет собой задачу распознавания образов. Специально для этого в 1988 году Яном Лекуном была разработана архитектура сверточной нейронной сети (CNN – *convolutional neural network*) [4]. В отличие от обычной полносвязной, в сверточной сети слои локально связаны. Это позволяет не терять информацию о положении признаков на изображении. Операция свертки представляет собой последовательное прохождение локальных областей изображения через маленькую нейросеть, состоящую из одного слоя, называемую ядром свертки. Каждый сверточный слой может иметь множество таких ядер, каждое из которых занимается выделением определенного признака на всем изображении. Количество ядер свертки выбирает исследователь для каждого сверточного слоя эмпирическим путем. Минимальный размер ядра свертки, способного различать такие признаки, как вертикальные, горизонтальные и косые линии, составляет 3x3. Ядро свертки проходит по всему изображению с определенным шагом, который выбирает исследователь. При этом допускаются перекрытия одной и той же области. Обычной практикой является выбирать нечетный размер стороны ядра свертки. Для ускорения обучения ядра инициализируются особым образом. Помимо размера и шага нужно выбрать способ перемещения ядра по изображению (padding). В обычном случае на выходе сверточного слоя карта признаков уменьшается по формуле (1):

$$(x; y) = (a; b) - (m; n) + (1; 1), \qquad (1)$$

где x, y – искомый размер сторон; a, b – исходный размер сторон; m, n – размер сторон ядра свертки (*padding = valid*). Чтобы сохранись исходный размер карты признаков, можно перемещать ядро свертки с выходом за границы карты. При этом недостающие ячейки заполняются нулями (*padding = same*). В сверточных слоях может применяться любая функция активации. Из-за своей простоты чаще всего применяется функция ReLU (*rectified linear unit*), которая просто отсекает отрицательную часть. За сверточным слоем в архитектуре *CNN* может следовать слой субдискретизации или пулинга. Его задача заключается в уменьшении разрешения карт признаков с помощью функции наибольшего или среднего значения. Это позволяет упростить задачу без потери основных признаков. Размер окна пулинга и его шаг выбирает исследователь, при этом шаг выбирается таким образом, чтобы не допустить перекрытия. Что бы избежать переобучения сети, можно исключать случайным образом часть нейронов в специальном слое – *dropout*. Часть исключаемых нейронов выбирается исследователем эмпири-

ческим путем. При малом значении можно не достичь желаемого результата, а при слишком высоком сеть будет недообучаться. При обучении нейронной сети распределение входных данных каждого слоя изменяется в процессе обучения, что требует установки меньшей скорости обучения и тщательной инициализации входных параметров. Предложенный компанией *Google* в 2015 году метод пакетной нормализации (*Batch normalization*), позволяет решить эту проблему путем нормализации входных данных слоя [5]. На выходе сеть имеет полносвязные слои для классификации. Настраиваемыми параметрами здесь могут быть количество и размер этих слоев. На выходе должен быть слой, по размеру соответствующий количеству классов. В нем применяется функция активации *softmax* – обобщение логистической функции для многомерного случая. На основе выше описанного принципа действия сверточных сетей, была спроектирована нейронная сеть, представленная на рисунке 1.

Сеть принимает на вход одноканальное черно-белое изображение разрешением 64×64 пикселя. На выходе сети создается массив из пяти элементов в диапазоне от 0 до 1, каждый из которых соответствует вероятности того, что на изображении находится объект того или иного класса. Класс с наибольшей вероятностью считается ответом сети. В разрабатываемой системе нейронная сеть строится с помощью фреймворка *Keras*, процесс работы с которым подробно описан в статье [6]. Так как сеть имеет структуру прямого распределения, создается объект класса *Sequental*. Далее последовательно вызывается метод *add*, который добавляет в сеть новый слой. Таким образом добавляются слои *Activation*, *BatchNormalization*, *Conv2D*, *MaxPooling2D*, *Dropout*, *Flatten*, *Dense* в соответствии со структурой сети (рис. 1). Во входном слое дополнительно указывается параметр *input_shape*.

Подготовка набора данных. Было принять решение собирать данные с помощью детектора, интегрированного в систему автоматического подсчета интенсивности. Детектор работает на основе алгоритма выделения движения и метода Виолы-Джонса. Одним из недостатков этого детектора является детектирование практически любого движения в кадре, так при сборе данных были получены, помимо транспортных средств, движущиеся пешеходы и тени от автомобилей. Для решения этой проблемы принято решение обучить нейронную сеть различать пять классов объектов: легковой и грузовой автомобиль, автобус, пешеход и тень на дороге. Для сбора данных били взяты видео с камер наблюдения, установленных над участками улично-дорожной сети в г. Белгороде.

Обучение нейронной сети. Чтобы приступить к обучению необходимо разбить данные на обучающую и тестовую выборки. Это делается автоматически с помощью функции *train_test_split* из библиотеки *sklearn*. Так как метки представлены в виде строк, их нужно бинаризировать. Для этого создается объект класса *LabelBinarizer*. Для бинаризации тренировочных меток применяется метод *fit_transform*, а для тестовых – *transform*. *ImageDataGenerator* позволяет сгенерировать дополнительные данные за счет различных преобразований: сдвиг, поворот, масштабирование, отзеркаливание.

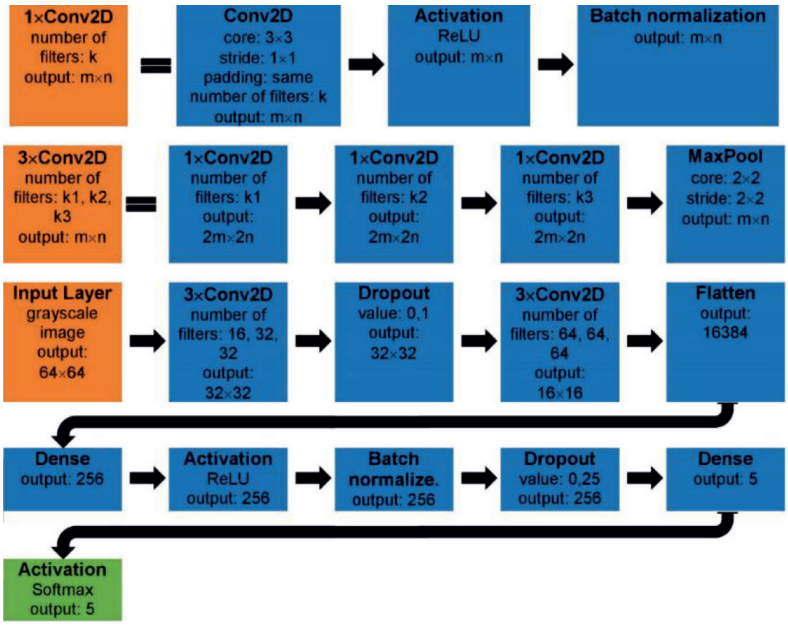

Рис. 1. Схема спроектированной сверточной нейронной сети

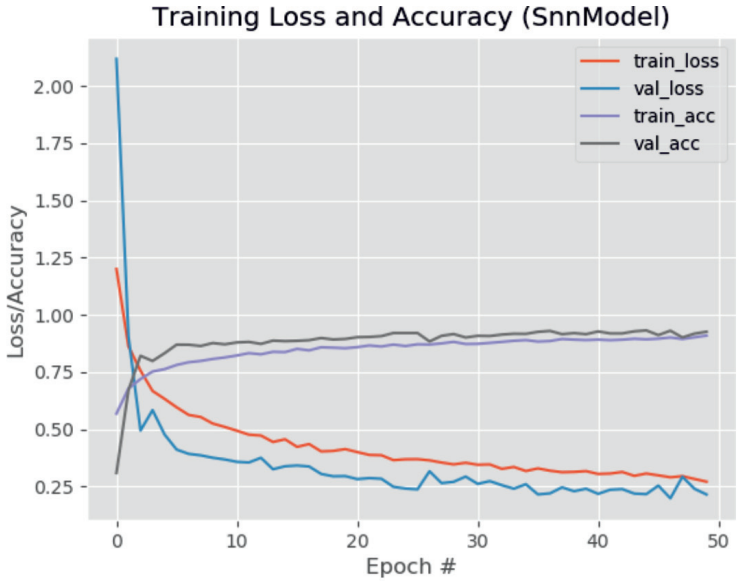

Рис. 2. История обучения нейронной сети – точность и потери

Результаты тестирования нейронной сети

Класс	Примеров классифицировано		Точность, %
	положительных	отрицательных	
Легковой автомобиль	23162	8472	73,22
Автобус	3446	437	88,75
Грузовой автомобиль	2473	144	94,50
Люди	3538	474	88,19
Тень	718	20	97,29
Всего	33337	9547	77,74

После создания модели, ее необходимо скомпилировать с помощью метода *compile*. Для компиляции сети необходимо указать такие параметры, как функция потерь, оптимизатор и параметры метрики. Для обучения применяется оптимизатор *SGD* – классический метод градиентного спуска [7]. В качестве функции потерь применяется категориальная перекрестная энтропия. Непосредственно обучение происходит в методе *fit_generator*. Метод возвращает историю обучения, которая сохраняется для последующего анализа. После обучения необходимо сохранить обученную модель с помощью метода *save*, и записать бинаризированные метки в файл для последующего использования сети. На основе полученной в ходе обучения истории был построен график (рис. 2). По оси абсцисс отложены итерации обучения, по оси ординат – четыре параметра: точность на учебной выборке, потери на учебной выборке, точность на тестовой выборке, потери на тестовой выборке. На последней итерации были достигнуты такие значения соответственно: 0,9088; 0,2704; 0,9259; 0,2144.

Применение и тест обученной сети. Распознавание происходит в методе *predict*, а возвращает массив вероятностей. Класс с наибольшей вероятностью считается предсказанием сети. В таблице приведены результаты тестов сети на 42884 примерах.

Вывод. Была разработана и обучена сверточная нейронная сеть для классификации ТС в транспортном потоке по трем классам: легковой автомобиль, грузовой автомобиль и автобус. Для устранения неточностей детектирования, нейронная сеть была обучена классифицировать пешеходов и тень от ТС на дороге. Итого нейронная сеть классифицирует объекты пяти классов. При обучении сети были получены высокие показатели точности – 92,59 %. Однако при проверке на большем количестве данных итоговая точность составила 77,74 %. Возможной причиной недостаточной точности может быть недостаточный размер обучающей выборки.

Список литературы

1. Python: официальный сайт. [Электронный ресурс]. – URL: https://www.python.org/ (дата обращения: 10.11.2019).

2. OpenCV: официальный сайт. [Электронный ресурс]. – URL: https://opencv.org/ (дата обращения: 10.11.2019).

3. Keras: официальный сайт. [Электронный ресурс]. – URL: https://keras.io/ (дата обращения: 21.12.2019).

4. 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. Winter 1989. vol. 1. no. 4. P. 541-551.

5. S. Ioffe, C. Szegedy Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift. – [Электронный ресурс] // proceedings IEEE Conf. on Computer Vision and Pattern Recognition. 2001. 9 p. URL: https://arxiv.org/pdf/1502.03167.pdf (date of the application 21.12.2019).

6. REG.RU: Как начать работу с Keras, Deep Learning и Python: сайт. – URL: https://www.reg.ru/blog/keras/ (дата обращения: 21.12.2019). – Текст: электронный.

7. Акулич И.Л. Математическое программирование в примерах и задачах. М.: Высшая школа, 1986. 310 с.

АНАЛИЗ И ПОСТРОЕНИЕ ФРАКТАЛОВ В РАЗЛИЧНЫХ ПРОГРАММНЫХ СРЕДАХ

Грищенко А.Ю., Фоменко Л.Н.

ФГБОУ ВО «Донской Государственный Технический Университет», Ростов-на-Дону, e-mail: reception@donstu.ru

Фрактальная геометрия одна из самых молодых областей математики. С помощью фрактальных алгоритмов возможно описание различных природных процессов, обладающих самоподобием (ветка дерева подобна всему дереву, при изменении масштаба береговая линия остается похожей на себя и др.) [1].

Фрактал (лат. *fractus* – дроблёный, сломанный, разбитый) – объект дробной размерности, обладающий свойством фрактального самоподобия (скейлинга) (например, кривая Коха, ковер Серпинского, траектория броуновской частицы и т.д.). В простых случаях небольшая часть фрактала содержит информацию о всем фрактале. Само понятие фрактал придумал ученый-математик Бенуа Мандельброт и ввел его в использование в 1975 году [2].

Впервые фрактал был построен в 1883 году Георгом Кантором. В то время он был определен как математический монстр. Кантор взял отрезок и разделил его на три части. Среднюю часть он вырезал, после чего оставались два крайних отрезка. Теперь из каждых оставшихся отрезков он вновь вырезал среднюю часть и так до бесконечности (рис. 1).

Отцом фрактальной геометрии и создателем понятия фрактал является американский, выросший во Франции математик – Бенуа Мандельброт. В 1967 году он опубликовал свою работу «Какова длина побережья Великобритании?», и именно она считается отправной точкой в цикле его работ по теории фракталов. В 1977 году вышла книга Мандельброта «Fractals: Form, Chance and Dimension» [3].

В нашей работе мы обратились к одному из самых известных фракталов – множеству Мандельброта [3]. Алгоритм построения такого множества основан на итеративном выражении (1):

$$Z_{n+1} = Z_n^2 + C. \qquad (1)$$

Здесь Z_{n+1} и Z_n, C – комплексные переменные. Длительность итерационного процесса зависит, сойдется ли Z_n к какой-либо точке окружности радиуса два и центром в начале координат, тогда эта точка C закрашивается в черный цвет. Точки, уходящие в бесконечность за конечное число итераций, образуют белый фон. Те точки, которые не уходят в бесконечность при конечном числе итераций, принадлежат границе множества, где и возникают сложные структуры. Выбирая в качестве фазового пространства двумерное, получим фазовый портрет итерационного процесса такой системы (рис. 2).

Фрактальный рисунок не имеет идентичных элементов, но обладает подобностью в любом масштабе. Построить такое изображение с высокой степенью детализации вручную ранее было просто невозможно, на это требовалось огромное количество вычислений.

На данный момент времени существует огромное количество программ для построения фракталов и генерации сложных неевклидовых объектов, например искусственных гор, облаков и др. [4]. Благодаря несложным формулам, фракталы можно легко запрограммировать.

В данной статье для создания и качественной визуализации данных мы будем применять инженерное математическое программное обеспечение Mathcad и высокоуровневый язык программирования Python с библиотекой Matplotlib. Проведем сравнительный анализ удобства работы с фракталами в этих средах.

Для построения фрактала в среде Mathcad будем использовать формулу множества Мандельброта (1) и встроенные функции Mathcad (рис. 3). Параметры width (ширина изображения в пикселях) и height (высота изображения) выбираются по желанию пользователя. Значение параметра iterations (количество итераций) так же выставляет пользователь (в нашем случае 29), но не более 100, т.к. при увеличении количества итераций изображение не изменяется ввиду ограниченных вычислительных способностей среды Mathcad. x1, x2, y1, y2 – координаты на плоскости, внутри которых мы будем строить множество Мандельброта, $C_{i,j}$ – вычисляемая матрица комплексных чисел.

Среда Mathcad проста в применении, не требует углубленных знаний информатики и языков программирования. Среди её минусов возможно отметить довольно запутанный и неудобный интерфейс программы.

Для построения и визуализации множества Мандельброта с использованием возможностей языка программирования Python на основании формулы (1) определим родительскую функцию (в нашем случае она названа mandel). Для размерности изображения необходима еще одна функция, определяемая пользователем (в нашем случае create_fractal). Для удобства визуализации графика подключим библиотеку Matplotlib. Применяя цикл с параметром, программа формирует график с поэтапным окрашиванием изображения с каждой новой итерацией.