ПРОГРАММИРОВАНИЕ, *2024*, *№ 1*, *с. 40–52*

АНАЛИЗ ДАННЫХ

УДК 004.932

АЛГОРИТМ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АВТОМАТИЧЕСКОГО РАСЧЕТА ДЛИН И ПЛОЩАДЕЙ ТРЕЩИН НА БОРТАХ И ОТВАЛАХ УГОЛЬНЫХ РАЗРЕЗОВ

С. Е. Попов[a], *(ORCID: 0000-0001-9495-6561),
В. П. Потапов[a], **(ORCID: 0000-0002-1530-5902),
Р. Ю. Замараев[a], *(ORCID: 0000-0003-4822-4794)**

[a]*ФГБУН "Федеральный исследовательский центр информационных и вычислительных технологий" (ФИЦ ИВТ), 630090 Новосибирск, пр. Академика Лаврентьева, 6, Россия*

**E-mail: popov@ict.nsc.ru*
***E-mail: vadimptpv@gmail.com*
****E-mail: zamaraev@ict.nsc.ru*

Поступила в редакцию: 21.02.2023
После доработки: 24.05.2023
Принята к публикации: 26.05.2023

В работе представлен алгоритм и описание его программной реализации для обнаружения линеаментов (трещин) на изображениях аэрофотосъемки угольных разрезов. В основе предложенного подхода лежит аппарат сверточных нейронных сетей для семантической классификации бинаризованных изображений объектов, а также теория графов для определения геометрического расположения объектов с последующим вычислением их длин и площадей. В качестве исходных данных использовались трехканальные RGB-изображения аэрофотосъемки высокого разрешения (пиксел 10×10 см). Модель программного модуля логически разделена на три уровня: предобработка, детектирование и постобработка. Первый уровень включает в себя предобработку входных данных для формирования обучающей выборки на базе последовательных трансформаций RGB-изображения в бинарное с применением библиотеки OpenCV. Второй уровень информационной модели представлен нейронной сетью типа U-Net, включающей блоки сверточной (Encoder) и развертоной частей (Decoder). На данном уровне реализовано автоматическое детектирование объектов. Третий уровень модели отвечает за расчет площадей и длин. На вход ему передается результат работы сверточной нейронной сети. Площадь трещин вычисляется путем суммирования общего числа точек с умножением на размер пиксела. Длина рассчитывается путем линеаризации площадного объекта в сегментированный объект с узловыми пикселами и последующим расчетом длин между ними с учетом разрешения исходного изображения. Программный модуль может работать с фрагментами исходного изображения путем их объединения. Модуль реализован на языке программирования Python. Код доступен по адресу (https://gitlab.ict.sbras.ru/popov/lineaments/-/tree/master/lineaments-cnn).

Ключевые слова: обнаружение линеаментов, трещин и эрозии грунта; сверточные нейронные сети; семантическая сегментация; обнаружение и накопление; аэрофотосъемка

DOI: 10.31857/S0132347424010044 **EDN:** HLFTJY

1. ВВЕДЕНИЕ

Современный этап развития технологии добычи полезных ископаемых открытым способом характеризуется значительными глубинами извлечения, которые могут составлять несколько сотен метров. В этих условиях возрастают требования к обеспечению устойчивости бортов карьеров. Оползневые деформации уступов приводят к большим финансовым затратам на устранение последствий аварийных ситуаций. Количественная характеристика неоднородностей (промывов, трещин и пр.) имеет основополагающее значение для определения механического поведения не сплошных массивов горных пород [1]. Следовательно, растет интерес к автоматизированному обнаружению неоднородностей на основе компьютерного зрения, чтобы заменить традиционные процедуры проверки человеком.

Методы обнаружения неоднородностей на основе машинного зрения широко применяются на протяжении последнего десятилетия из-за их преимуществ относительно хорошей точности и производительности алгоритмов в реальном времени [2] с учетом дистанционного наблюдения за объектами.

В основе этих алгоритмов лежат методы обработки цифровых данных (значений пикселов) RGB-изображения, включая методы обнаружения краев

[3], преобразование Хафа [4], сегментацию изображения [5], идентификацию и детектирование характерных точек [6, 7], метод корреляции для цифровых изображений (Digital Image Correlation) [8, 9] и фотограмметрии [10] и прочее.

Однако общее ограничение этих подходов заключается в обнаружении линеаментов (трещин) путем поиска по всей области изображения. В результате возникают трудности в дифференциации истинных объектов от подобных им шумов, таких как границы структур (например, стрела экскаватора), крупные наземные коммуникации, провода и трубы темного цвета, тени от объектов линейной формы [11]. В то же время точное обнаружение формы линеамента, ответвлений, начального и конечного пикселов является ключевым фактором для измерения геометрических свойств (длины или площади) и остается сложной, нетривиальной задачей.

В последние годы машинное обучение и, в частности, сверточные нейронные сети продемонстрировали широкие возможности в области семантического обнаружения объектов и их признаковых классов в задачах дистанционного мониторинга состояния различных объектов. Так, в работе [12] рассматривается модель CNN для точной идентификации микросейсмических событий и взрывов. В статье [13] предложен метод, основанный на DCNN, для локализации повреждений строительных конструкций с высокой точностью на необработанных, зашумленных сигналах. Также встречается большое количество работ касательно процессов детектирования трещин на снимках, образованных на различных поверхностях. Авторы в [14] использовали обученную CNN и методы скользящего окна для обнаружения трещин на бетонных поверхностях. В исследовании [15] используется сверточная нейронная сеть типа R-CNN для обнаружения трещин на асфальтированных дорогах. В [16] продемонстрирована модель обнаружения повреждений на базе сквозного метода (end-to-end), основанного на глубоком обучении с использованием широкого набора данных о повреждениях дорог, их местоположении и типе повреждения. Также в работе [17] на основе сквозного метода предложена сверточная сеть для обнаружения трещин мостовых опор. В работе используется свертка с разделением, уменьшающая количество параметров и модуль объединения пространственных пирамид (ASPP). Представленная модель достигает точности обнаружения 96,37%.

Среди работ по обнаружению линеаментов выделяют исследования по практическому применению результатов детектирования, т.е. так называемый пост-процессинг. В частности, в работе [18] рассматривается задача определения длин трещин на шлифах бетонных срезов различных конструкций. Использована комбинация сверточной сети и методов определения вершин трещин на основе морфологических трансформаций графического объекта и функции пороговой сегментации.

В плане коммерческого программного обеспечения в основном на рынке присутствуют разработки для определения физических характеристик (например, определение величины кинетического сдвига и границ дилатации) трещин по их изображениям в динамике на основе алгоритмов машинного зрения. В работе [19] представлена полностью автоматизированная процедура обнаружения трещин и измерения их кинематики в лабораторных экспериментах с использованием цифровой корреляции изображений (DIC), которая позволяет извлекать гораздо более мелкие трещины с их местоположением в образце. Ширина трещины и подвижки измеряются с использованием поля смещения с учетом локальных поворотов образца.

Анализ представленной литературы показал, что подавляющее большинство работ, посвященных в той или иной степени детектированию линеаментов, используют в качестве исходных данных изображения с явно выделяющимися объектами на относительно равномерном по цветовой гамме фоне (асфальтированная дорога, бетонная стена, каменистая порода и т.п.). Авторам не удалось найти обучающих выборок с изображениями реальных бортов и отвалов на карьерах, где, например, на поверхности могут присутствовать камни, результаты рекультивации (молодая поросль), водные отстойники и прочее. Косвенно это подтверждается и отсутствием таких данных на ресурсе Kaggle (https://www.kaggle.com/).

Хотя методы машинного зрения, в том числе нейронные сети глубокого обучения, и обладают наибольшей точностью детектирования линеаментов, однако, остаются вопросы постпроцессинга результатов и имплементации предметного программного обеспечения. Например, расчета геометрических величин с использованием RGB-изображений аэрофотосъемки.

В этом исследовании представлен алгоритм обнаружения трещин и определения их длин и площадей на бортах и отвалах угольных разрезов. Приводится сравнение нейронных сетей для семантической классификации объектов, алгоритм линеаризации пиксельных данных на бинаризованных

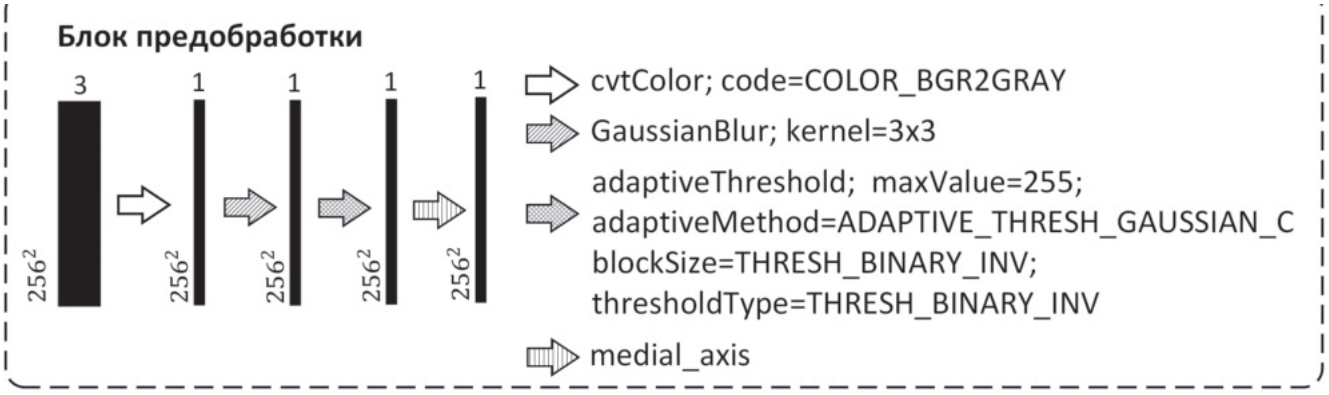


Рис. 1. Схема преобразования RGB-изображения в бинарное изображение. Названия компонентов соответствуют программной библиотеке OpenCV API (https://docs.opencv.org/4.x/) и scikit-image (https://scikit-image.org/). Число по вертикали – размер изображения, по горизонтали – количество каналов.

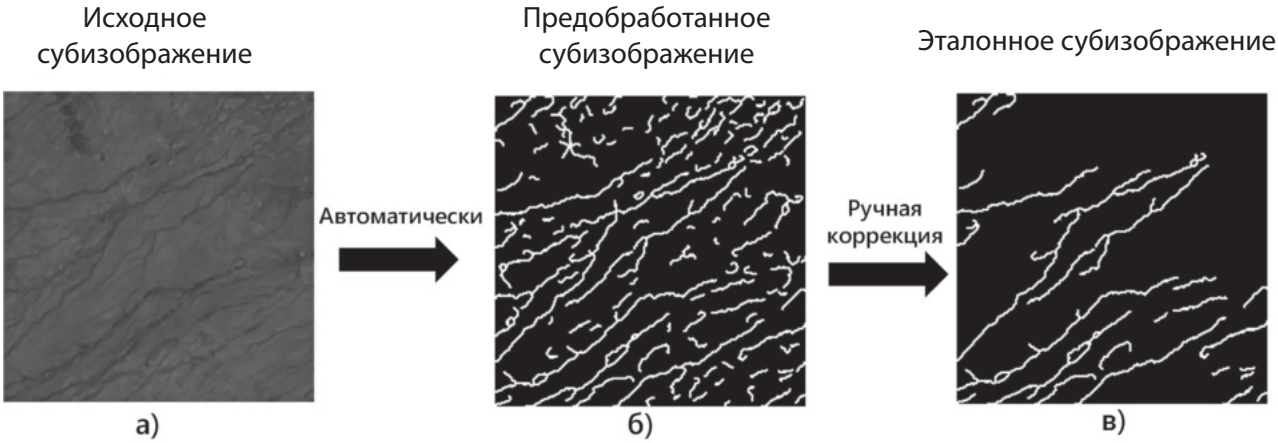


Рис. 2. Схема подготовки обучающей выборки на основе схемы преобразования (рис. 1).

изображениях трещин. Решается задача группировки объектов, подсчета пикселов и расчета на их основе длины и площади с учетом разрешения исходного изображения. Полный алгоритм обработки реализован в виде программного модуля на языке Python (https://gitlab.ict.sbras.ru/popov/lineaments/-/tree/master/lineaments-cnn).

2. ПОДГОТОВКА ДАННЫХ

В качестве исходных данных использовались трехканальные RGB-изображения аэрофотосъемки высокого разрешения (10 × 10 см). Высота съемки 300 м. Снимки получены с беспилотного летательного аппарата с камерой SONY DSC-RX1R. Период съемки 07.2022–08.2022 гг. Полный размер входного изображения 6000 × 4000 пикселов.

Входное изображение разбивалось на базисные субизображения (в текущей версии 256 × 256 пикселов), которые брались за основу для предварительной обработки и процесса ручной разметки данных. Предобработка строилась на базе схемы последовательных преобразований RGB-изображения в бинарное (рис. 1).

В схеме используется преобразование цветного изображения к схеме “градации серого” (cvtColor) с последующим применением процедуры размытия по Гауссу (GaussianBlur) для сглаживания резких переходов (значений) соседних пикселов.

Бинаризация выполняется на основе адаптивного выделения контуров с дилатацией (adaptiveThreshold) из библиотеки OpenCV и выделением медианных пикселов (medial_axis) для линеаризации трещин из библиотеки scikit-image (рис. 3)

На выходе получаются нормализованные бинарные изображения, пригодные для последующей ручной корректировки в рамках формирования обучающей выборки (рис. 2в).

Исходное и эталонное субизображения подвергались вертикальному, горизонтальному, симметричному отражению и случайному вращению (библиотека albumentation, https://albumentations.ai/).

```
...
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor (img, cv2.COLOR_BGR2GRAY)
img_blur = cv2.GaussianBlur (img_gray, (3, 3), 0)
img_bw = cv2.adaptiveThreshold (img_blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, \
                                cv2 .THRESH_BINARY_INV,
                                template.config['ADAPTIVE_THRESHOLD_BLOCK_SIZE']
                                template.config['ADAPTIVE_THRESHOLD_C'])
img_skel_bool, _ = medial_axis(img_bw, return_distance=True)
img__skel = img_skel_bool.astype (np.uint8)
...
transform = A. Compose ([A.OneOf ([A.HorizontalFlip (p=l), A.VerticalFlip(p=l),
                          A.RandomRotateSO(p=l)], p=0.75)])
image_t, mask_t = list(transform(image=imgs[key], mask=masks[key]).values())
...
```

Рис. 3. Фрагмент кода предобработки субизображений обучающей выборки.

Таким образом была сформирована обучающая выборка из более чем 2000 субизображений (с учетом аугментации), являющихся частями исходных изображений, содержащих как трещины, так и изображения с полным отсутствием таковых (фон). Принималось, что на субизображении трещины обозначены белым цветом (значение 1, класс “трещина”), а остальная часть фона - черным (0, класс “фон”).

Для процесса обучения, валидации и детектирования трещин датасет был поделен на части в пропорции 85% / 10% / 5%. Датасет доступен по ссылке https://www.kaggle.com/datasets/semionpopov/open-pit-cracks.

3. ОБНАРУЖЕНИЕ ТРЕЩИН

3.1 Конфигурация нейронной сети

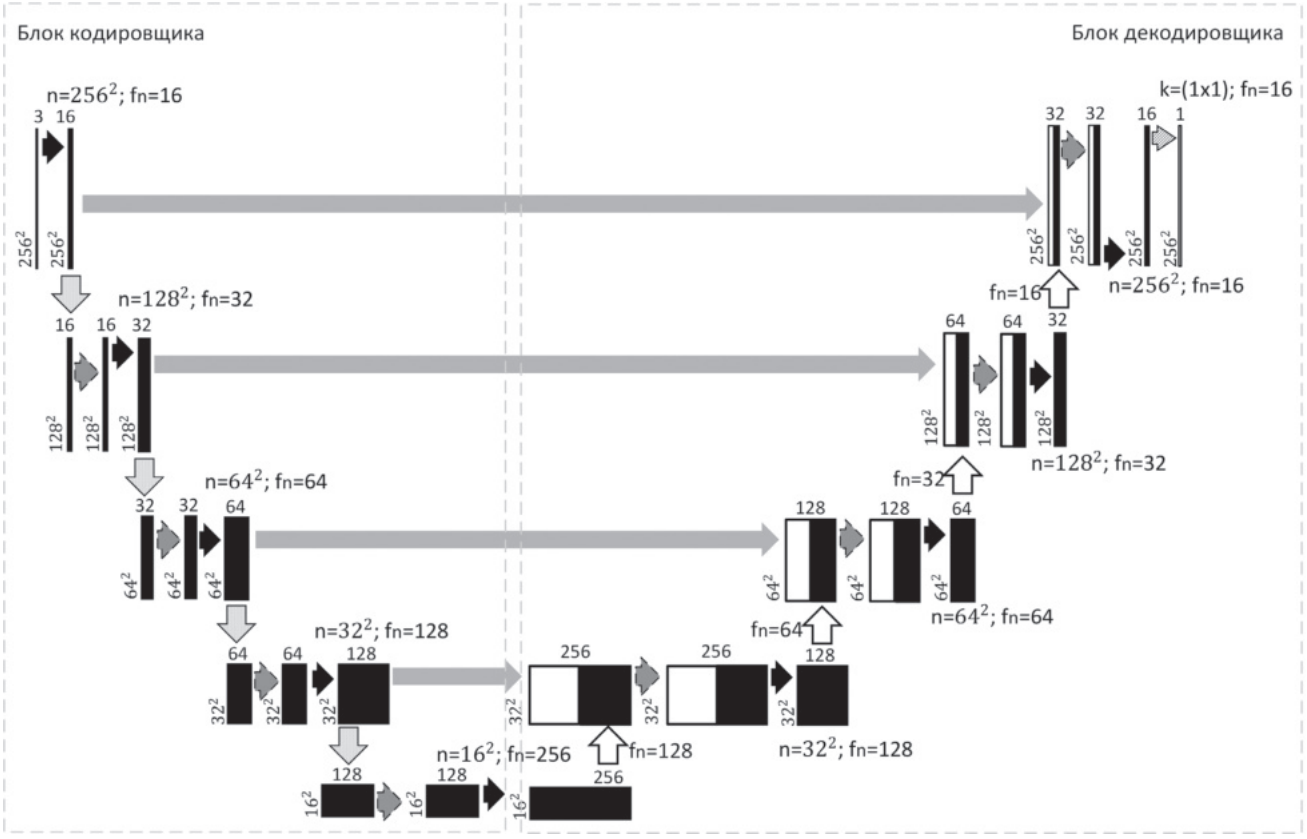
Процесс обнаружения трещин на изображениях строился на базе сверточной нейронной сети типа U-Net. Архитектура сети представляет собой полносвязную сверточную сеть [20], работающую на меньшем количестве примеров (обучающих образов) с более точной сегментацией. Сеть содержит блок кодировщика (Encoder) и блок де-кодировщика (Decoder) (рис. 4а, б). Программная имплементация сети реализована на основе пакета TensorFlow v2.11 с модулем Keras (https://www.tensorflow.org/api_docs/python/tf/keras).

Блок Encoder состоит из модулей свертки, включающих два сверточных слоя (Conv2D) с параметром kernel_size 3×3 (класс tf.keras.layers.Conv2D), слой активации ReLU (класс tf.keras.layers.Activation) и слой нормализации входных данных BatchNorm (класс tf.keras.layers.BatchNormalization). Также используется дополнительный слой пулинга с функцией максимума (параметр pool_size = 2×2) с шагом 2 и слой регуляризации (Dropout) для уменьшения переобучения с 50% исключением (параметр rate). На каждом шаге количество каналов признаков удваивается (параметр filters).

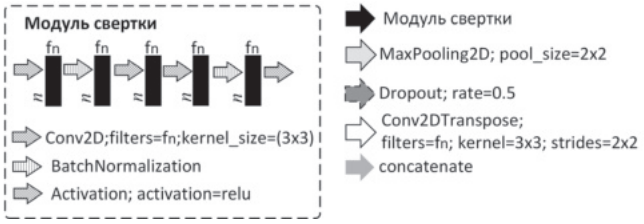
Параметр “фильтр” является ключевым. Фильтры – это массивы заданного размера, которые инициализируются случайными значениями с использованием метода, указанного в аргументе kernel_initializer (класс tf.keras.initializers). Во время обучения сети фильтры обновляются таким образом, чтобы минимизировать потери. В ходе обучения фильтры учатся обнаруживать определенные особенности объектов (например, края и текстуры трещин). Блок Encoder действует как экстрактор признаков и изучает абстрактное представление входного изображения через последовательность фильтров.

Блок Decoder содержит слой обратной свертки (класс tf.keras.layers.Conv2DTranspose), который расширяет карту признаков. После идет конкатенация (слой concatenate) с соответствующим образом обрезанной картой признаков и последовательность слоев свертки, как в части Encoder. Обрезка проводится из-за потери пограничных пикселов в каждой свертке. Завершающий слой (Conv2D) – свертка 1×1 (kernel_size) – используется для приведения каждого 16-компонентного вектора признаков до требуемого количества классов (2 класса: трещина, фон).

Декодирование необходимо для повышения дискретизации нейронной сети за счет объединения карты признаков объектов с более высокими разрешениями из блока кодировщика (апсэмплинг) с целью улучшения априорной оценки более ранних этапов свертки. Это позволяет лучше представить локализацию искомых объектов (трещин). На



а) Блоки кодировщика и де-кодировщика



б) Модуль свертки

Рис. 4. Конфигурация нейронной сети

рис. 5, 6 приведены фрагменты программного кода инициализации блоков Encoder и Decoder, компиляции и старта процесса обучения сети (класс tf.keras.Model).

3.2. Обучение нейронной сети

Сеть обучалась на датасете размером 2015 изображений. В качестве функции потерь использовалась бинарная кросс-энтропия (класс tf.keras.losses.BinaryCrossentropy). Для оценки качества работы моделей использовалась метрика Accuracy (класс tf.keras.metrics.Accuracy), показывающая процент пикселов в изображении, которые были правильно классифицированы. Количество эпох от 30 до 45. На выходе получалась полностью обученная сеть с оптимизированной матрицей весов. Сеть сохранялась в файл формата HDF (.h5) для последующей загрузки на этапе детектирования объектов.

Для оценки релевантности выбора конфигурации представленной сети проведено сравнение результатов обучения с похожей архитектурой нейронной сети DeepLabV3+ [21]. Обучение DeepLabV3+ проводилось на такой же обучающей выборке с аналогичной функцией потерь и метрикой. Ниже представлены сводные таблицы метрик (табл. 1.), матриц ошибок (confusion matrix, табл. 2) и скорости обучения (табл. 3.).

Метрики сети U-Net оказались значительно выше. Это объясняется тем, что сеть DeepLabV3+ использует в слое кодировщика так называемые расширенные свертки. Модель расширенных сверток (Atrous Convolutions) представляет способ ком-

```
def getUnet (inputImage, numFilters=16,
             droupouts=0.1, doBatchNorm=True):

    # Encoder
    c1 = Conv2dBlock (inputImage, numFilters * 1,
                      kernelSize=3, doBatchNorm=doBatchNorm)
    p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)
    p1 = tf.keras.layers.Dropout (droupouts)(p1)
    ...

    c4 = Conv2dBlock (p3, numFilters * 8,
                      kernelSize=3, doBatchNorm=doBatchNorm)
    p4 = tf.keras.layers.MaxPooling2D((2, 2))(c4)
    p4 = tf.keras.layers.Dropout (droupouts)(p4)
    c5 = Conv2dBlock (p4, numFilters * 16,
                      kernelSize=3, doBatchNorm=doBatchNorm)

    # Decoder
    u6 = tf.keras.layers.Conv2DTranspose (numFilters * 8, (3, 3),
                          strides=(2, 2), padding='same')(c5)
    u6 = tf.keras.layers.concatenate([u6, c4])
    u6 = tf.keras.layers.Dropout(droupouts)(u6)
    c6 = Conv2dBlock (u6, numFilters * 8,
                           kernelSize=3, doBatchNorm=doBatchNorm)
    ...

    u9 = tf.keras.layers.Conv2DTranspose (numFilters * 1, (3, 3),
                          strides=(2, 2), padding='same')(c8)
    u9 = tf.keras.layers.concatenate([u9, c1])
    u9 = tf.keras.layers.Dropout (droupouts)(u9)
    c9 = Conv2dBlock(u9, numFilters * 1,
                          kernelSize=3, doBatchNorm=doBatchNorm)

    output = tf.keras.layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)
    model = tf.keras.Model(inputs=[inputimage], outputs=[output])

    return model
```

Рис. 5. Программная реализация блоков Encoder и Decoder. Переменными “c1-c9” обозначены модули свертки.

бинировать признаки с нескольких масштабов без значительного увеличения количества параметров.

Регулируя показатель расширения, один и тот же фильтр распределяет значение веса дальше в пространстве [21]. Это позволяет изучать более общий контекст. То есть данный метод более подходит для распознавания площадных объектов, за счет увеличения шага пространственных пирамид ASPP на каждом этапе свертки.

Таблица 1. Сравнение нейронных сетей U-Net и DeepLabV3+

Нейронная сеть	Метрика (Accuracy)
U-Net	0,9801
DeepLabV3+	0,9100

Таблица 2. Сравнение матриц ошибок

Реальные значения	Сеть U-Net		Сеть DeepLabV3+	
Трещины	0,9544	0,0456	0,8103	0,1897
Фон	0,0413	0,9587	0,0533	0,9467
	Трещины	Фон	Трещины	Фон
	Предсказанные значения			

Таблица 3. Скорость обучения сети

Нейронная сеть	Скорость обучения на эпоху* (batch_size = 4)
U-Net	52 мс
DeepLabV3+	103 мс

* На GPU Nvidia RTX 3060.

```
...
unet = getUnet (inputs, droupouts=0.07)
unet.compile (optimizer='Adam', loss='binary_crossentropy', metrics=['accuracy'])

fit_result = unet.fit (imgs_array, masks_array, epochs=37, batch_size=1)
unet.save (template.config['MODEL'])
...
```

Рис. 6. Фрагмент кода компиляции и старта процесса обучения сети.

```
def predict(model_file, imgs_array):
    unet_model = tf.keras.models.load_model(model_file)
    predictions = unet_model.predict(imgs_array, batch_size=8)
    predictions = [predictions[i][:, 0] for i in range(len(predictions))]
    return predictions
```

Рис. 7. Фрагмент кода для запуска процедуры обнаружения трещин.

3.3. Тестирование нейронной сети

Для обнаружения трещин на тестовых субизображениях используется метод predict (класс tf.keras. Model) (рис. 7).

На вход метода подаются исходные субизображения (параметр imgs_array) тестовой выборки (рис. 2а). На выходе получается массив значений, каждый элемент которого соответствует положению пиксела на исходном субизображении, а значение – вероятности того, что соответствующий пиксел принадлежит классу “трещина”.

Задается порог значений (например, 0.95), выше которого пиксел классифицируется, как принадлежащий трещине. Таким образом формируется карта вероятностей классификаций пикселов (рис. 8).

Далее всем пикселам со значениями выше порога присваиваются значения, равные 255, а ниже или равными порогу – 0. Формируется бинарное (черно-белое одноканальное изображение) трещин.

После прохождения процедурой predict по всем субизображениям полученные бинарные субизображения объединяются в единый массив данных согласно ширине и длине исходного изображения. Формируется полная бинарная карта. Именно такое изображение используется в постпроцессинге для определения длин и площадей трещин.

4. ПОСТПРОЦЕССИНГ

4.1 Определение длин трещин

Будем рассматривать бинарное субизображение как двухмерный массив (img_bw). Аналогично этапу предобработки применяется метод medial_axis для выделения медианных пикселов. Данный метод уменьшает количество смежных пикселов для текущего медианного до минимального количества (по возможности до 2 пикселов), объекты подвергаются скелетизации (переменная, двухмерного массива img_skel) (рис. 3).

На следующем шаге запускается итерационная процедура формирования наборов точек для каждого объекта-линии. На каждой итерации используется метод label (scipy.ndimage), позволяющий назначить числовую метку (переменная key) пикселам каждой из обособленных трещин, где принадлежность пиксела однозначно определяется его меткой (переменные labeled_array, num_of_labels, рис. 9).

Формируется массив key_points, содержащий массивы координат пикселов (y, x – по вертикали, по горизонтали) для текущей метки объекта-трещины.

Измерение трещины по длине определяется по количеству последовательно-смежных (парных) пикселов между двумя краевыми. Пиксел называется краевым, если у него ровно один смежный пиксел. Если у трещины больше двух краевых пикселов (есть ответвления), то выбирается набор по максимальному количеству.

Таким образом, можно сформулировать задачу нахождения длины трещины, как поиск наибольшего пути в ациклическом графе между двумя заданными вершинами.

Для решения поставленной задачи использовались библиотеки SciPy (spatial.KDTree, https://docs.scipy.org/doc/scipy/) и NetworkX (объект Graph, https://networkx.org/documentation/stable/ reference/index.html). Поиск смежных пикселов осуществлялся при помощи метода query_ball_point с параметрами x, y – координаты текущего пиксела и длина радиуса (переменная r), в пределах которого ищутся все пикселы. Если длина получившегося массива (переменная neighbor_counter) равна 2, то текущий пиксел считается краевым (переменная edge_points).

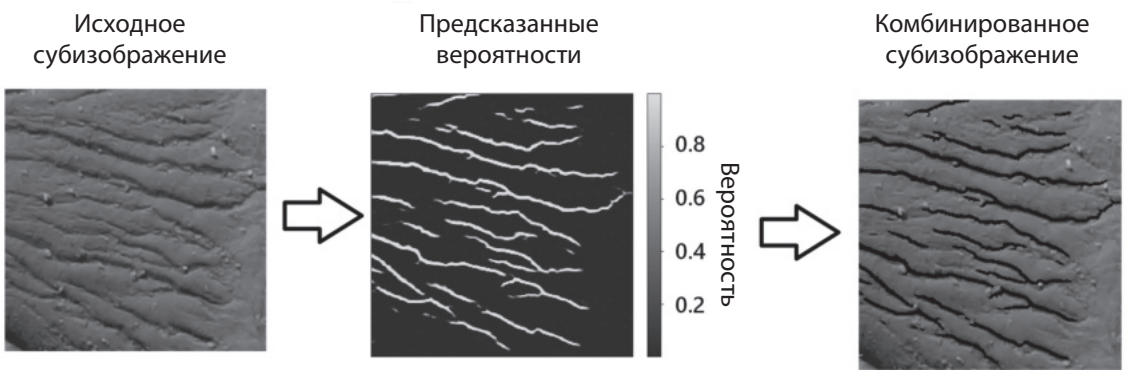


Рис. 8. Схема процесса детектирования трещин для тестового субизображения.

```
labeled_array, num_of_labels = label(img_skel, structure=s)
for key in range(num_of_labels):
    key_points = np.where(labeled_array == key + 1)
    key_points = np.column_stack((key_points[0], key_points[1]))

    labeled_points_tree = KDTree(key_points)
    ...
    for point in key_points:
        neighbor counter = labeled points tree
            .query_ball_point ([point[0], point[1]], r=np.sqrt(2))

        if neighbor_counter == 2: edge_points.append(point)

    labeled_point_pairs_index = np.array(labeled_points_tree
        .query_pairs(r=np.sqrt(2), output_type='ndarray'))

    G = nx.Graph()
    G.add_edges_from(labeled_point_pairs_index, weight=1)
    ...
    longest_path = get_longest_path(edge_points, labeled_points_tree, G)
    ...
    if len(longest_path) > SEGMENT_MIN_LENGTH:
        point_sets.append(key_points[longest_path])
        ...


def get_longest_path(edge_points, labeled_points_tree, G):
    longest_path = []
    for i in range(len(edge_points)):
        for j in range(i + 1, len(edge_points)):
            source = labeled_points_tree.query(edge_points[i])[1]
            target = labeled_points_tree.query(edge_points[j])[1]
                path = nx.shortest_path(G, source=source, target=target)
            if len(path) > len(longest_path):
                longest_path = path
    return longest_path
```

Рис. 9. Фрагмент программного кода получения набора координат пикселов для каждого объекта-трещины.

Для поиска всех возможных пар пикселов для текущей метки применялся метод query_pairs с параметрами r и типом выходного массива output_type, равном “ndarray”. Метод возвращает массив пар (переменная labeled_point_pairs_index), где в паре указаны два индекса из массива key_points, являющихся смежными (рис. 9).

Далее для составления объекта-графа (рис. 10) используется конструктор объекта Graph() (переменная G) и G.add_edges_from, на вход которому передаются переменные labeled_point_pairs_index и weight равная 1. Здесь длина ребра берется равной единице. Вне зависимости от того, как расположены точки в паре по вертикали, горизонтали или по диа-

гонали, важен факт количества точек в пути от одной краевой вершины графа до другой.

Для всех возможных попарных комбинаций пикселов из edge_points в цикле формируются пути прохождения графа между ними, используя метод shortest_path. На вход ему подаются переменная G и переменные source (начальный краевой пиксел) и target (конечный краевой пиксел), на выходе получаем массив индексов элементов (переменная longest_path, рис. 9) из key_points. При этом в цикле ищется максимально возможный по длине массив path, а также удовлетворяющий проверке на пороговое значение длины (переменная SEGMENT_MIN_LENGTH = 5).

Пикселы, соответствующие массиву longest_path, сохраняются в переменной (point_sets), а в массиве img_skel им присваиваются значения 0. Итерация переходит к следующему значению переменной key.

Данный алгоритм также учитывает случай, когда некоторые пикселы образуют три и более пар (трещины имеют ответвления). Тогда после завершения итераций по всем значениям переменной key алгоритм строит новую серию меток и повторяет серию вышеуказанных процедур снова, затем останавливается пока на какой-то итерации, длина массива longest_path будет всегда меньше, чем пороговое значение SEGMENT_MIN_LENGTH.

В переменной point_sets сохраняются координаты пикселов для каждой трещины на бинаризованном изображении. Общая длина трещины складывается из элементарных длин между двумя попарно смежными пикселами, то есть между вертикально- и горизонтально-смежными расстояние равно 20 см, а по диагонали ~28 см (при разрешении 10×10 см).

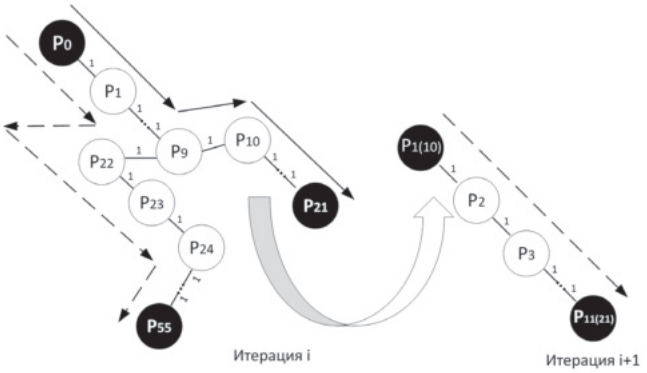


Рис. 10. Пример объекта-графа. Черным обозначены вершины, соответствующие краевым пикселам, стрелками – пути обхода графа между двумя краевыми точками. Штрихованные стрелки – наибольший путь по количеству пикселов на каждой итерации. Ребра графа имеют веса, равные 1.

4.2 Определение площадей трещин

Аналогично для вычисления площади трещины используется метод label. Далее для каждого ключа key подсчитывается общее количество пикселов в текущем объекте-трещине и умножается на 100 (при разрешении изображения 10×10 см).

На рис. 11а, б представлен результат расчета длин и площадей трещин на произвольном изображении аэрофотосъемки (на примере угольного карьера Восточный, Кемеровская область, Россия).

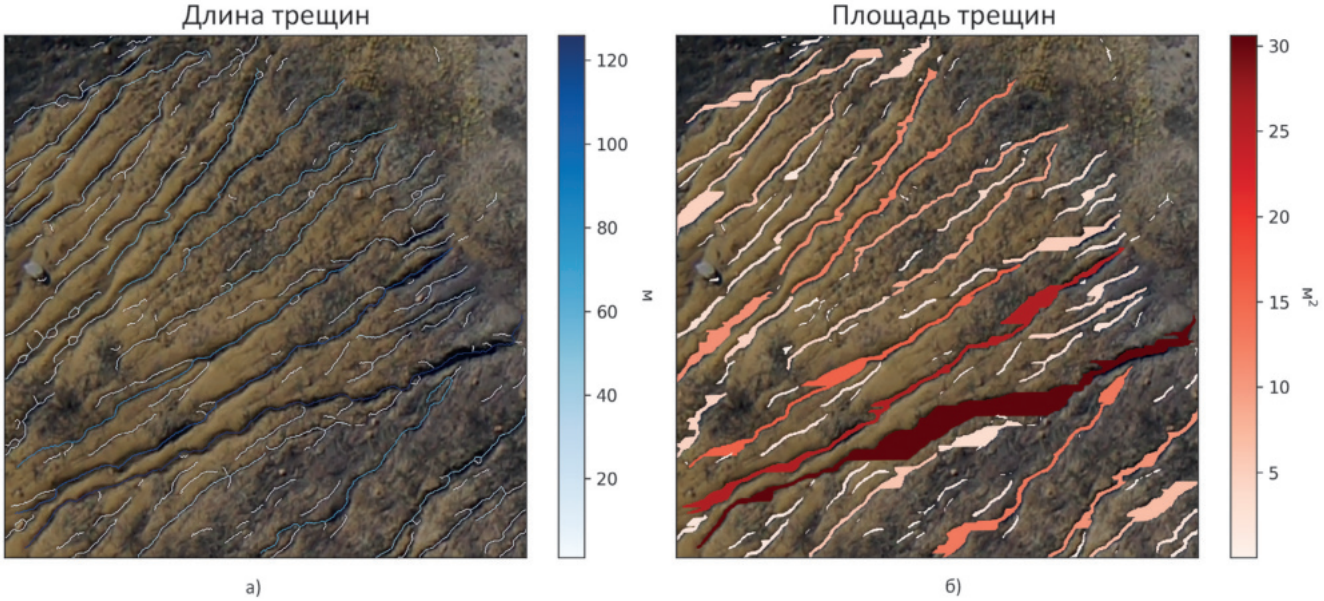


Рис. 11. Результаты расчета длин (слева) и площадей (справа) трещин.

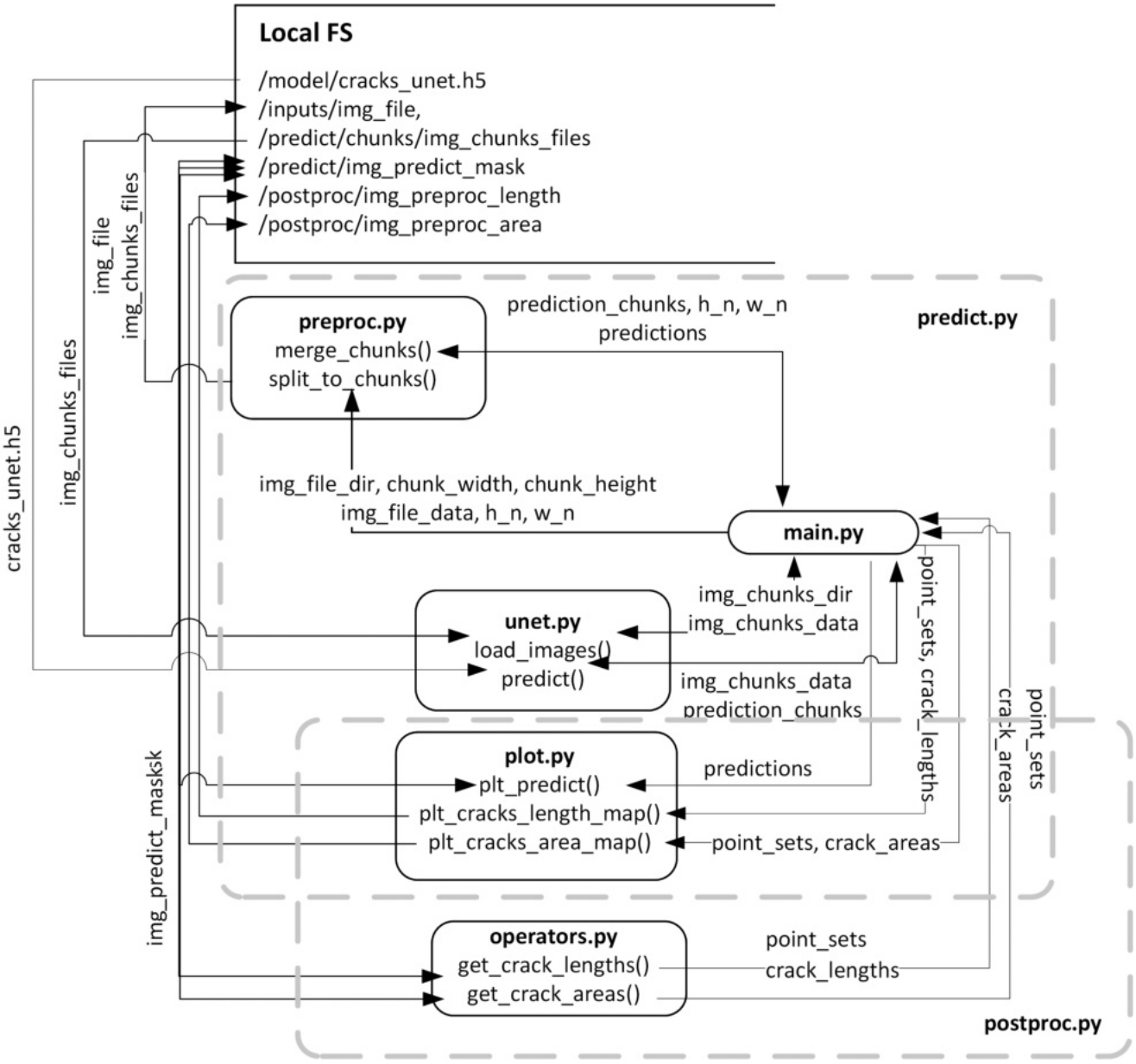


Рис. 12. DFS-диаграмма потоков данных процесса обнаружения трещин и расчета их длин и площадей.

Диаграмма потоков данных для полного алгоритма расчета длин и площадей трещин представлена на рис. 12.

5. ЗАКЛЮЧЕНИЕ

Разработаны алгоритм и его программная реализация, позволяющие автоматизировать процесс детектирования трещин на изображениях, полученных с летательных аппаратов с фото/видео- оборудованием мониторинга геодинамического состояния объектов угледобычи.

Предложенный подход для распознавания трещин на базе аппарата сверточных нейронных сетей позволяет использовать в качестве входных датасетов различные изображения с соответствующей бинаризованной разметкой объектов семантической сегментации. Нейронная сеть поддерживает процесс дообучения. Гибкие параметрические настройки алгоритма дают возможность пропорционально (с шагом разбивки) обрабатывать большие входные изображения нейронной сетью. Сравнение с другими аналогичными нейронными сетями показали относительное превосходство предложенного выбора сети по скорости и точности метрик.

Постпроцессинг, расчет длин и площадей, строится на количественной оценки или подсчете пикселов объектов с пороговой вероятностью больше или равной предсказанной, использует только

величину разрешения изображения и не зависит от RGB-значений. Это позволяет использовать алгоритм на других монохромных (черно-белых) изображениях.

Предложенный подход позволяет в процессе постобработки результатов вычислять количественные характеристики поверхностных неоднородностей (трещин), что имеет практическое значение для определения механического поведения не сплошных массивов горных пород.

Новизна предложенного подхода заключается в применении аппарата нейронных сверточных сетей для автоматизированного поиска характерных риск-объектов на открытых участках (отвалах) угольных разрезов, используя изображения аэрофотосъемки высокого разрешения. Результаты исследования могут быть применены в других предметных задачах многоуровневой адаптивной семантической классификации.

СПИСОК ЛИТЕРАТУРЫ

1. *Potapov V.P., Oparin V.N., Mikov L.S., Popov S.E.* Information Technologies in Problems of Nonlinear Geomechanics. Part I: Earth Remote Sensing Data and Lineament Analysis of Deformation Wave Processes. Journal of Mining Science. 2022. T. 58. P. 486–50.
2. *Hao X., Du W., Zhao Y., Sun Z., Zhang Q., Wang S., Qiao H.* Dynamic tensile behaviour and crack propagation of coal under coupled static-dynamic loading. Int. J. Min. Sci. Technol. 2020. T. 30. P. 659–668.
3. *Krull B., Patrick J., Har, K., White S., Sottos N.* Automatic optical crack tracking for double cantilever beam specimens. Exp. Tech. 2016. T. 40. P. 937–945.
4. *Sun H., Liu Q., Fang L.* Research on fatigue crack growth detection of M (T) specimen based on image processing technology. J. Fail. Anal. Prev. 2018. T. 18. P. 1010–1016.
5. *Zhang W., Zhang Z., Qi D., Liu Y.* Automatic crack detection and classification method for subway tunnel safety monitoring. Sensors. 2014. T. 14. P. 19307–19328.
6. *Kong X., Li J.* Vision-based fatigue crack detection of steel structures using video feature tracking. Comput.-Aided Civ. Inf. 2018. T. 33. P. 783–799.
7. *Kong X., Li J.* Non-contact fatigue crack detection in civil infrastructure through image overlapping and crack breathing sensing. Automat. Constr. 2019. T. 99. P. 125–139.
8. *Li D., Huang P., Chen Z., Yao G., Guo X., Zheng X., Yang Y.* Experimental study on fracture and fatigue crack propagation processes in concrete based on DIC technology. Eng. Fract. Mech. 2020. T. 235. P. 107–166.
9. *Vanlanduit S., Vanherzeele, J., Longo, R. Guillaume P.* A digital image correlation method for fatigue test experiments. Opt. Laser. Eng. 2009. T. 47. P. 371–378.
10. *Valença J., Dias-da-Costa D., Júlio E., Araújo H., Costa H.* Automatic crack monitoring using photogrammetry and image processing. Measurement. 2013. T. 46. P. 433–441.
11. *Yeum C.M., Dyke S.J.* Vision-based automated crack detection for bridge inspection. Comput.-Aided Civ. Inf. 2015. T. 30. P. 759–770.
12. *Dong L., Tang Z., Li X., Chen Y., Xue J.* Discrimination of mining microseismic events and blasts using convolutional neural networks and original waveform. J. Cent. S. Univ. 2020. T. 27. P. 3078–3089.
13. *Yu Y., Wang C., Gu X., Li J.* A novel deep learning-based method for damage identification of smart building structures. Struct. Health Monit. 2019. T. 18. P. 143–163.
14. *Su C., Wang W.* Concrete Cracks Detection Using Convolutional Neural Network Based on Transfer Learning. Mathematical Problems in Engineering. 2020. Article ID 7240129. 10 p. DOI: 10.1155/2020/7240129
15. *Pauly L., Hogg D., Fuentes R., Peel H.* Deeper networks for pavement crack detection. In Proc. 34th ISARC. 2017. P. 479–485.
16. *Maeda H., Sekimoto Y., Seto T., Kashiyama T., Omata Y.* Road damage detection using deep neural networks with images captured through a smartphone. arXiv preprint. 2018. P. 1-14. URL: https://arxiv.org/abs/1801.09454
17. *Xu H., Su X., Wang Y., Cai H., Cui K., Chen X.* Automatic bridge crack detection using a convolutional neural network. Applied Sciences. 2019. V. 9(14). P. 2867. DOI: 10.3390/app9142867
18. *Yuan Y., Ge Z., Su X., Guo X., Suo T., Liu Y., Yu Q.* Crack Length Measurement Using Convolutional Neural Networks and Image Processing. Sensors. 2021. T. 21. P. 5894.
19. *Gehri N., Mata-Falcón J., Kaufmann W.* Automated crack detection and measurement based on digital image correlation. Construction and Building Materials. 2020. T. 256. P. 119383. DOI: 10.1016/j.conbuildmat.2020.119383
20. *Shelhamer E., Long J., Darrell T.* Fully Convolutional Networks for Semantic Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2017. T. 39. № 4. P. 640-651. DOI: 10.1109/TPAMI.2016.2572683
21. *Chen L., Zhu Y., Papandreou G., Schroff F., Adam H.* Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. Proc. European conference on computer vision (ECCV). 2018. P. 801–818. URL: https://arxiv.org/abs/1802.02611

SOFTWARE IMPLEMENTATION OF THE ALGORITHM FOR AUTOMATIC DETECTION OF LINEAMENTS AND THEIR PROPERTIES ON OPEN-PIT DUMPS

S. E. Popov[a], V. P. Potapov[a], R. Y. Zamaraev[a]

[a] *Federal Research Center for Information and Computational Technologies, 6, Academician M.A. Lavrentiev av., Novosibirsk, 630090, Russia*

The paper presents an algorithm and a description of its software implementation for detecting lineaments (ground erosions or cracks) in aerial photography images of open-pits. The proposed approach is based on the apparatus of convolutional neural networks based on the semantic classification of binarized images of objects (lineaments), as well as graph theory for determining the geometric location of linearized objects, followed by determining their lengths and areas. Three-channel RGB images of high-resolution aerial photography (pixel 10x10 cm) were used as initial data. The software unit of the model is logically divided into three layers: pre-processing, detection and post-processing. The first level includes preprocessing of input data to form a training sample based on successive transformations of an RGB image into a binary one using the OpenCV library. A neural network of the U-Net type, which includes blocks of the convolutional (Encoder) and scanning parts (Decoder), represents the second level of the information model. At this level, automatic lineament detection (washouts) is implemented. The third level of the model is responsible for calculating the areas and lengths of lineaments. The result of the work of the convolutional neural network is transferred to the input. Lineament area is calculated by summing the total number of points multiplied by the pixel size. The length of the lineaments is computed by linearizing a plane object into a line segmental object with nodal points and then calculating the lengths between them, also relying on the resolution of the original image. The software module can work with input images, with their subsequent resulting merging to the size of the original image.

Keywords: detection of lineaments, cracks and ground erosion, convolutional neural networks, semantic segmentation, detection and accumulation, aerial photography

REFERENCES

1. *Potapov V.P., Oparin V.N., Mikov L.S., Popov S.E.* Information Technologies in Problems of Nonlinear Geomechanics. Part I: Earth Remote Sensing Data and Lineament Analysis of Deformation Wave Processes. Journal of Mining Science, 2022, vol. 58, pp. 486–50.
2. *Hao X., Du W., Zhao Y., Sun Z., Zhang Q., Wang S., Qiao H.* Dynamic tensile behaviour and crack propagation of coal under coupled static-dynamic loading. Int. J. Min. Sci. Technol, 2020, vol. 30, pp. 659–668.
3. *Krull B., Patrick J., Har, K., White S., Sottos N.* Automatic optical crack tracking for double cantilever beam specimens. Exp. Tech., 2016, vol. 40, pp. 937–945.
4. *Sun H., Liu Q., Fang L.* Research on fatigue crack growth detection of M (T) specimen based on image processing technology. J. Fail. Anal. Prev., 2018, vol. 18, pp. 1010–1016.
5. *Zhang W., Zhang Z., Qi D., Liu Y.* Automatic crack detection and classification method for subway tunnel safety monitoring. Sensors, 2014, vol. 14, pp. 19307–19328.
6. *Kong X., Li J.* Vision-based fatigue crack detection of steel structures using video feature tracking. Comput.-Aided Civ. Inf., 2018, vol. 33, pp. 783–799.
7. *Kong X., Li J.* Non-contact fatigue crack detection in civil infrastructure through image overlapping and crack breathing sensing. Automat. Constr., 2019, vol. 99, pp. 125–139.
8. *Li D., Huang P., Chen Z., Yao G., Guo X., Zheng X., Yang Y.* Experimental study on fracture and fatigue crack propagation processes in concrete based on DIC technology. Eng. Fract. Mech., 2020, vol. 235, pp. 107–166.
9. *Vanlanduit S., Vanherzeele, J., Longo, R. Guillaume P.* A digital image correlation method for fatigue test experiments. Opt. Laser. Eng., 2009, vol. 47, pp. 371–378.
10. *Valença J., Dias-da-Costa D., Júlio E., Araújo H., Costa H.* Automatic crack monitoring using photogrammetry and image processing. Measurement, 2013, vol. 46, pp. 433–441.
11. *Yeum C.M., Dyke S.J.* Vision-based automated crack detection for bridge inspection. Comput.-Aided Civ. Inf., 2015, vol. 30, pp. 759–770.
12. *Dong L., Tang Z., Li X., Chen Y., Xue J.* Discrimination of mining microseismic events and blasts using convolutional neural networks and original waveform. J. Cent. S. Univ., 2020, vol. 27, pp. 3078–3089.
13. *Yu Y., Wang C., Gu X., Li J.* A novel deep learning-based method for damage identification of smart building structures. Struct. Health Monit., 2019, vol. 18, pp. 143–163.
14. *Su C., Wang W.* Concrete Cracks Detection Using Convolutional Neural Network Based on Transfer Learning. Mathematical Problems in Engineering, 2020, Article ID 7240129, 10 p. DOI: 10.1155/2020/7240129
15. *Pauly L., Hogg D., Fuentes R., Peel H.* Deeper networks for pavement crack detection. In Proc. 34th ISARC, 2017, pp. 479–485.

16. *Maeda H., Sekimoto Y., Seto T., Kashiyama T., Omata Y.* Road damage detection using deep neural networks with images captured through a smartphone. arXiv preprint, 2018, pp. 1–14. URL: https://arxiv.org/abs/1801.09454
17. *Xu H., Su X., Wang Y., Cai H., Cui K., Chen X.* Automatic bridge crack detection using a convolutional neural network. Applied Sciences, 2019, vol. 9, no. 14, p. 2867. DOI: 10.3390/app9142867
18. *Yuan Y., Ge Z., Su X., Guo X., Suo T., Liu Y., Yu Q.* Crack Length Measurement Using Convolutional Neural Networks and Image Processing. Sensors, 2021, vol. 21, p. 5894.
19. *Gehri N., Mata-Falcón J., Kaufmann W.* Automated crack detection and measurement based on digital image correlation. Construction and Building Materials, 2020, vol. 256. p. 119383. DOI: 10.1016/j.conbuildmat.2020.119383
20. *Shelhamer E., Long J., Darrell T.* Fully Convolutional Networks for Semantic Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017, vol. 39, no. 4, pp. 640–651. DOI: 10.1109/TPAMI.2016.2572683
21. *Chen L., Zhu Y., Papandreou G., Schroff F., Adam H.* Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. Proc. European conference on computer vision (ECCV). 2018. pp. 801-818. URL: https://arxiv.org/abs/1802.02611