Глава 4.

ДИАГНОСТИКА ДЕФЕКТОВ НА АВТОМОБИЛЬНЫХ ДОРОГАХ НА ОСНОВЕ АЛГОРИТМОВ КОМПЬЮТЕРНОГО ЗРЕНИЯ

Кутузов Виктор Владимирович
к.т.н., доцент
Зубков Евгений Александрович
Литвинчук Артём Сергеевич
МОУВО «Белорусско-Российский университет»

Аннотация: в настоящее время технологии компьютерного зрения применяются в различных областях, в том числе, и в дорожной отрасли. Их используют для обнаружения дорожных знаков, светофоров, линий дорожной разметки и т.д. В рассматриваемой работе представлено применение методов компьютерного зрения для диагностики дефектов на автомобильных дорогах. Показан по шагам процесс создания датасета, обучения моделей и распознавания дефектов на фотографиях и в видеопотоке. Показаны результаты работы программ. Применение данной методики к оценке технического состояния автомобильных дорог позволяет существенно экономить материальные и трудовые ресурсы.

Ключевые слова: компьютерное зрение, диагностика автомобильных дорог, дефекты, DataSet, Yolo, ResNet.

DIAGNOSTICS OF DEFECTS ON ROADS BASED ON COMPUTER VISION ALGORITHMS

Kutuzov Victor Vladimirovich
Zubkov Evgeny Alexandrovich
Litvinchuk Artem Sergeevich

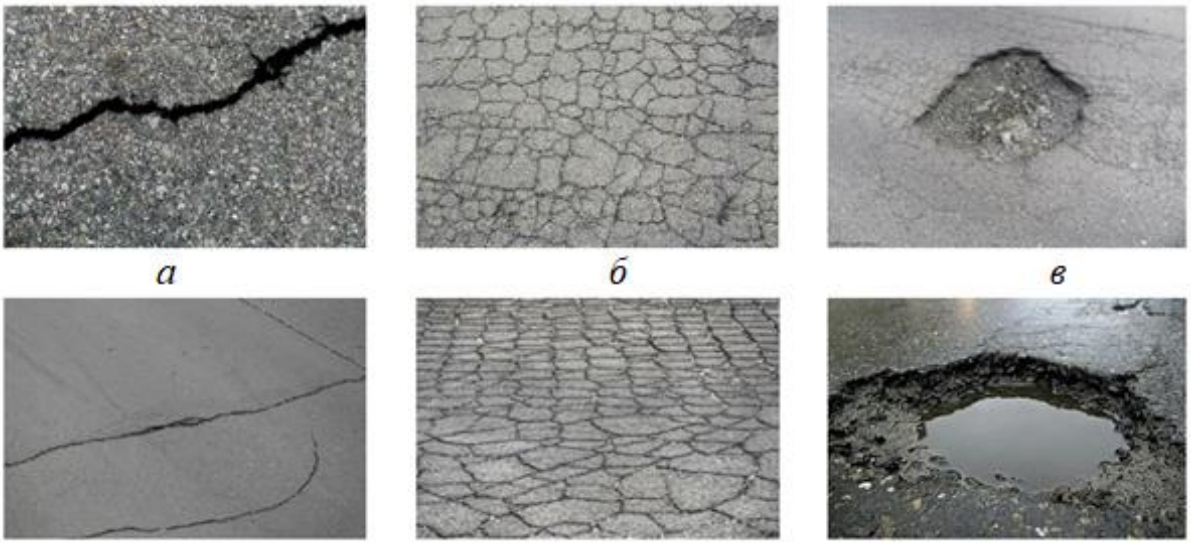
Abstract: computer vision technology is currently used in a variety of applications, including the road sector. They are used for detection of road signs, traffic lights, road markings, etc. In this paper, the application of computer vision methods to diagnose defects on roads is presented. It shows, step by step, the process of data set creation, model training and defect detection on photographs and video stream. The results of the software operation are shown. The application of this methodology to the assessment of the technical condition of motor roads makes it possible to save material and human resources considerably.

Key words: computer vision, road diagnostics, defects, DataSet, Yolo, ResNet.

Обоснование выбора контролируемых параметров

Диагностика является важной составляющей в обеспечении продолжительного срока службы автомобильной дороги. Диагностику выполняют при весеннем и осеннем осмотре, а также для назначения ремонтных, после строительства и реконструкции автомобильных дорог [1-4]. Для автоматизации данного процесса была поставлена задача использования технологии компьютерного зрения [5-11] для определения дефектов на дорогах.

В тоже время определение всех дорожных дефектов достаточно трудоемкий процесс, который занимает огромное количество времени и ресурсов, поэтому в рамках исследования мы прибегли к следующему исключению: мы будем детектировать только самые распространённые дефекты покрытий автомобильных дорог, к которым можно отнести: трещины; сетки трещин и выбоины (ямы) (рис. 1). Данные дефекты составляют более 60% от общего числа дефектов на покрытии автомобильных дорог. Для упрощения обучения модели, а также для создания датасета мы будем рассматривать только эти дефекты (выбоины, сетка трещин, трещины).



а) трещины; б) сетка трещин; в) выбоины

Рис. 1. Основные дефекты автомобильных дорог

Разработка алгоритма диагностики автомобильных дорог на основе алгоритмов компьютерного зрения

Для того, чтобы выполнить поставленную задачу были проделаны следующие шаги (рис. 2).

Шаги 1-6 являются направляющими, именно они определяют исходный код, модель и задачу, которую мы будем решать. Особенно следует выделить 4 и 5 шаг (сбор данных и формирование датасета). От шагов 4 и 5 будет зависеть точность полученной модели, если датасет составлен из большой выборки (более 1000 изображений для каждого типа дефектов), то полученная модель будет более точной, чем при меньшей выборке.

Шаги 7-10 зависят от шагов 1-6, а также будут показывать полученный результат. Данные шаги указывают на результат. В шагах 7-10 следует выделить 7-ой шаг, так как тестирование должно происходить на подготовленной заранее выборке, чтобы получить наиболее достоверные результаты.

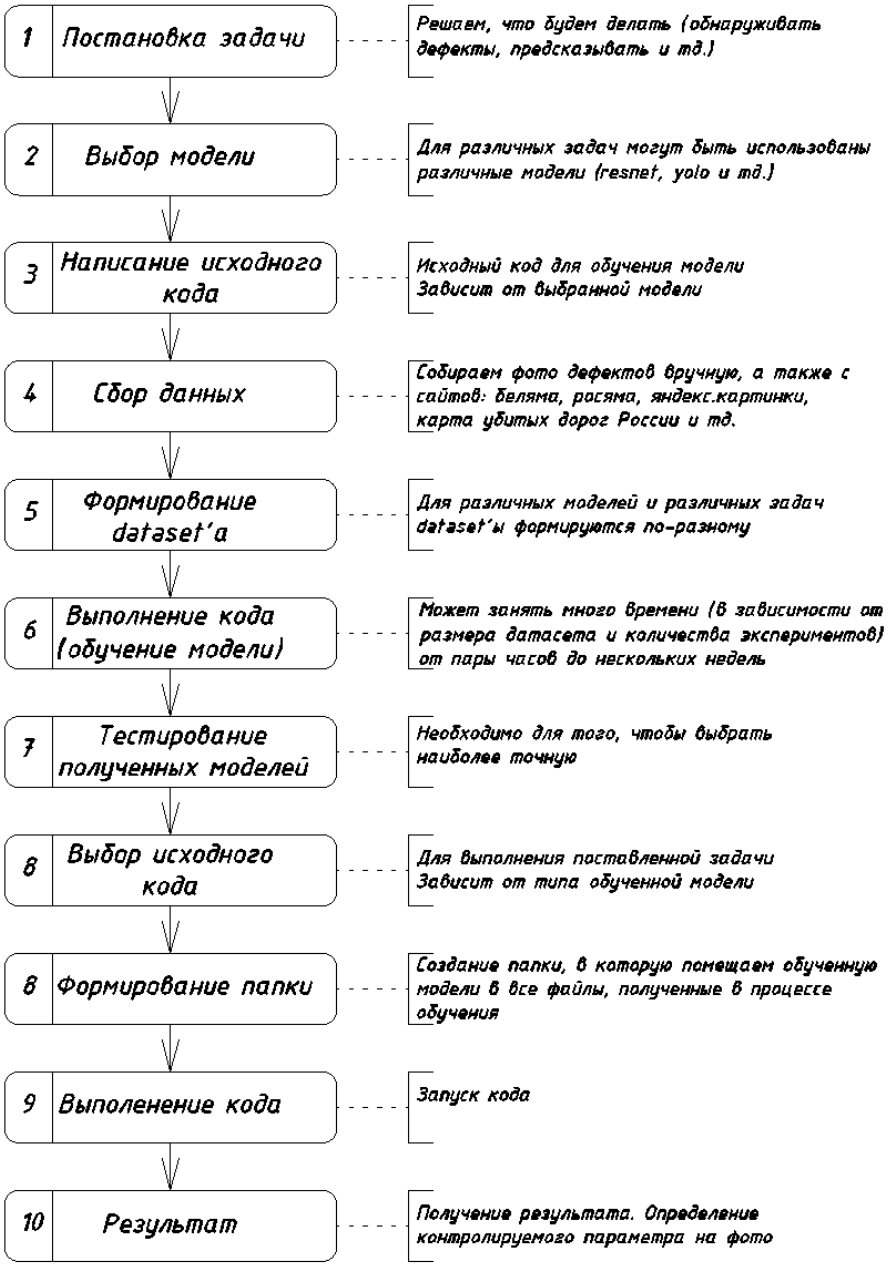


Рис. 2. Блок-схема проведения исследования

В нашем случае мы будем использовать данный алгоритм для предсказания дефектов и для обнаружения. Для каждой из этих задач будем составлять отдельные датасеты, так как обучение модели будет происходить с использованием различных библиотек (для предсказания дефектов мы будем использовать ResNet, а для обнаружения дефектов YOLOv3), а также использовать различные разработанные алгоритмы.

Создание Dataset'a. Обучение модели с использование архитектуры ResNet для предсказания дефекта по фотографии.

Для прогнозирования того или иного дефекта, нужно обучить нашу модель. Для этого мы будем использовать класс **ModelTraining,** преимуществами данного способа является быстрота, точность, а также несколько строк кода. Класс **ModelTraining является классом для обучения моделей.**

Ниже представлен алгоритм последовательности обучения модели по предсказанию дефектов (рис. 3).

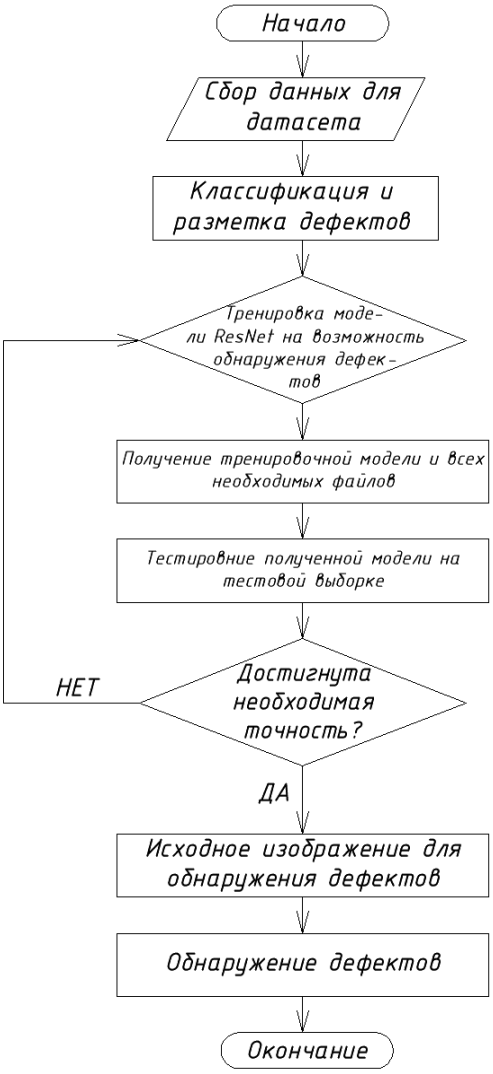


Рис. 3. Блок-схема алгоритма последовательности для обучения модели по предсказанию дефектов

В данном алгоритме (рис.3) мы будем тренировать модель ResNet. Для начала мы должны создать датасет, затем на собранном датасете тренировать модель ResNet далее проверить точность полученной модели, если точность полученной модели недостаточна, то мы продолжаем обучать модель до тех пор, пока не получим требуемую точность. После того как точность достигнута мы должны проверить достоверность результатов на исходном изображении по предсказанию дефектов.

Для разработки ПО использовался язык программирования Python с использованием библиотек tensorflow 1.5.0; h5py 2.10.0; pillow 7.1.2; keras 2.1.5; matplotlib 3.0.3; opencv-python 4.2.0.34; numpy 1.18.4; pip 19.0.3; imageai 2.1.5; scipy 1.4.1 и т.д.

Ключевой библиотекой была ImageAI, которая предоставляет очень мощные, но простые в использовании классы и функции для выполнения обнаружения и отслеживания видео объектов и анализа видео. ImageAI позволяет выполнять все это с помощью современных алгоритмов глубокого обучения, таких как RetinaNet, YOLOv3 и TinyYOLOv3 и др. С ImageAI мы запустили задачи. Ниже приведены классы и их соответствующие функции, доступные для использования. Эти классы могут быть интегрированы в любую традиционную программу python, которую мы разрабатываем.

Класс ModelTraining позволяет обучать любой из 4 поддерживаемых алгоритма глубокого обучения (SqueezeNet, ResNet, InceptionV3 и DenseNet) на собственном наборе данных изображений для создания собственных пользовательских моделей.

В процессе обучения создается файл JSON, который отображает типы объектов в наборе данных изображений и создает множество моделей. Затем мы достигаем максимальной точности модели и выполняем пользовательское предсказание изображения, используя созданную модель и файл JSON.

Для того, чтобы обучить нашу модель предсказывать дорожные дефекты необходимо создав набор данных с изображениями дорожных дефектов. Создаем папку «Дорожные Дефекты (RoadDefects)» далее в этой папке создаем ещё две папки: «тренировка (train)» и «проверка (test)». Папка «тренировка (train)» необходима для обучения, в ней мы также создаем необходимое количество папок для каждого дефекта в отдельности: «выбоины (pothole)», «трещины (crack)», «сетка трещин (the net of cracks)». В каждую из этих папок помещаем необходимые изображение того или иного дефекта. В папке «проверка (test)» создаем аналогичным образом папки с классом для каждого дефекта, эта папка необходима для тестирования данных. Чтобы модель корректно работала необходимо поместить в каждую из папок как можно больше изображений (желательно более 500).

Ниже на рис. 4 можно увидеть, как должны располагаться папки с изображениями в датасете.

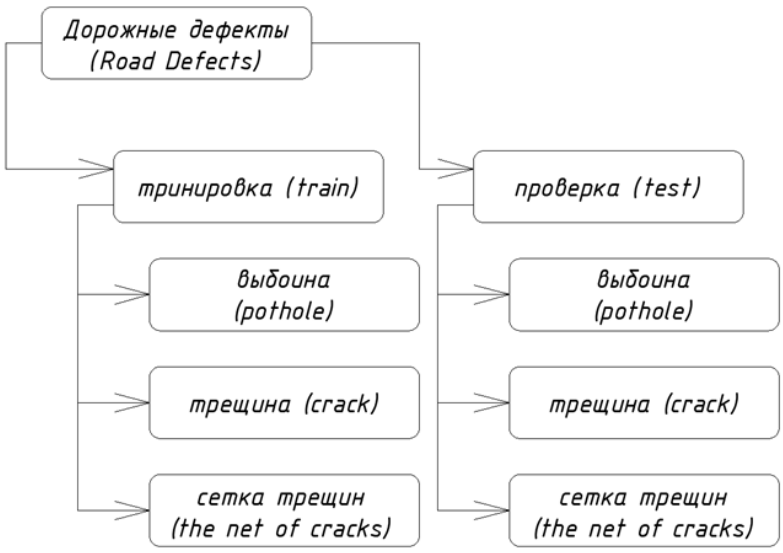


Рис. 4. Схема расположения папков в датасете

После того, как наш датасет готов, мы можем приступать к созданию экземпляра, а именно ModelTraining. Затем нам необходимо задать функции.

.setModelTypeAsResNet (), эта функция устанавливает тип модели обучающего экземпляра, созданного для модели ResNet, что означает, что алгоритм ResNet будет обучаться на собранном наборе данных.

Рассмотрим структуру данного кода. Для того, чтобы код был понятен, мы сделаем комментарии по каждой строке.

Строка 1. from imageai.Prediction.Custom import ModelTraining. Импотртиртируем данные из imageai.

Строка 2. model_trainer = ModelTraining(). Создаем экземпляр.

Строка 3. model_trainer. ModelTypeAsResNet(). Устанавливаем тип модели обучаемого экземпляра.

Строка 4. model_trainer. setDataDirectory("C:/RoadDefects). Задаем путь к папке, где собраны данные.

Строка 5. model_trainer.trainModel (…) Запускает процесс обучения нашей модели.

Ниже приведен код (рис. 5), для обучения нашей модели.

```
1  from imageai.Prediction.Custom import ModelTraining
2
3  model_trainer = ModelTraining()
4  model_trainer.setModelTypeAsResNet()
5  model_trainer.setDataDirectory(r"C:/RoadDefects")
6  model_trainer.trainModel(num_objects=3,
7                           num_experiments=100,
8                           enhance_data=True,
9                           batch_size=80,
10                          show_network_summary=True)
11
```

Рис. 5. Пример кода для обучения модели на подобии RetinaNet

Можно заметить, что в строке под номером 6 существуют различные параметры, которые играют важную роль в обучении модели, рассмотрим подробнее каждый из них:

- num_objects=3. Является количеством классов, используемых при обучении модели. В нашем случае 3 («the net of cracks (сетка трещин)», «crack (трещина)», «pothole (выбоина)»);

- num_experiements=100. Это число раз, когда алгоритм будет обучаться на нашем наборе данных изображений. Точность вашего обучения действительно увеличивается по мере увеличения числа раз, когда он тренируется. Однако она достигает своего пика после определенного количества тренировок; и эта точка зависит от размера и природы набора данных;

- batch_size=80. Количество изображение, на которых модель обучается параллельно. Всегда должно быть кратно 8. В нашем случае 80;

- enhance_data=True. Нужен для трансформирования набора данных изображений, для создания дополнительной выборки. Если этот параметр не прописан в коде, то по умолчанию его значение «True»;

- show_network_summary=True. Отображает структуру алгоритма. Параметр «True» задан по умолчанию;

Далее мы запускаем приложение. Следует отметить, чем больше выборка и чем больше заданное количество экспериментов, тем больше времени займет обучение нашей модели.

При обучении модели было обнаружено 20002 изображения в папке «тренировка (train)», которые относятся к 3-ём классам, а также 5023 изображения в папке «проверка (test», которые также относятся к 3-ем классам. Была также создана модель автоматически «model_class.json» в папке «json». Модель «model_class.json» будет использоваться нами в исходном коде, для достижения максимальной точности. Строка «Epoch 1/100» означает, что наша модель обучает первый эксперимент из 100 (количество экспериментов задаем параметром «num_experiements=…»). Количество партий определяется автоматически и зависит от размера выборки.

После того, как обучение модели завершено, автоматически создаются три папки в RoadDefects: «json», «logs», «models». В папке «json» находится «model_class» (это файл, в котором хранятся простые структуры данных и объекты. Он содержит данные в стандартном формате обмена данными, который является легким, текстовым и удобочитаемым). В папке «logs» находится «events.out.tfevents.1590738095» . В папке «models» находится наша «обученная модель» «model_ex-001_acc-0.3000000.h5».

После завершения обучения пользовательской модели можно использовать класс CustomImagePrediction из библиотеки ImageAI, для выполнения прогнозирования изображений с помощью модели. После обучения модели и получения всех необходимых исходных данных мы должны поместить их в одну папку и сходным кодом. Все это необходимо для того, чтобы код работал корректно, а также для удобства задания пути ко всем необходимым данным. Завершающим этапом служит исходный код (рис. 6).

```
from imageai.Prediction.Custom import CustomImagePrediction
import os

execution_path = os.getcwd()

prediction = CustomImagePrediction()
prediction.setModelTypeAsResNet()
prediction.setModelPath(os.path.join(execution_path,
                                     "model_ex-001_acc-0.300000.h5"))
prediction.setJsonPath(os.path.join(execution_path,
                                    "model_class.json"))
prediction.loadModel(num_objects=3)

predictions, probabilities = prediction.predictImage\
    \(os.path.join(execution_path, "9.jpg"), result_count=5)

for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)

```

Рис. 6. Полученный код для предсказания дефектов

Рассмотри подробнее данный код и что значат отдельные строки:

Строка 1. from imageai.Prediction.Custom import CustomImagePrediction import os. Создание нового экземпляра.

Строка 2. execution_path = os.getcwd().

Строка 3. prediction = CustomImagePrediction(). Устанавливаем тип модели экземпляра распознавания образов, созданного в модели ResNet (выбор модели был выбран заранее).

Строка 4. prediction. SetModelPath(…). Задаем путь к полученной модели с расширение h5 (для удобства помещаем в папку с кодом)

Строка 5. prediction. SetJsonPath(…). Задаем путь к полученному фалу с расширением json (для удобства помещаем в папку с кодом)

Строка 6. Prediction.LoadModel(num_objects=3). Устанавливаем число классов (в нашем случае 3 «выбоина (pothole)», «сетка трещин (the net of cracks)», «трещина (crack)»)

Строка 7. predictions, probabilities = prediction.predictImage (os.path.join (execution_path, "defect.jpg"), result_count=5). Выполняем предсказание. Указываем путь к изображению (для удобства помещаем в папку с кодом)

Для проверки кода мы загружаем фотографию (defect.jpg) с дефектами в папку, где находится код и все исходные данные. Затем запускаем код.

Мы видим, что дефект, изображенный на фото была выявлена как стека трещин с вероятностью 68% и как выбоина с вероятностью 32%. Это свидетельствует о том, что код работает корректно, и данная выборка, была хорошего качества. Если мы хотим повышения точности необходима более большая выборка с дефектами как упоминалось выше, чем больше количество изображений собранно в датасете, тем больше точность, обучаемой модели.

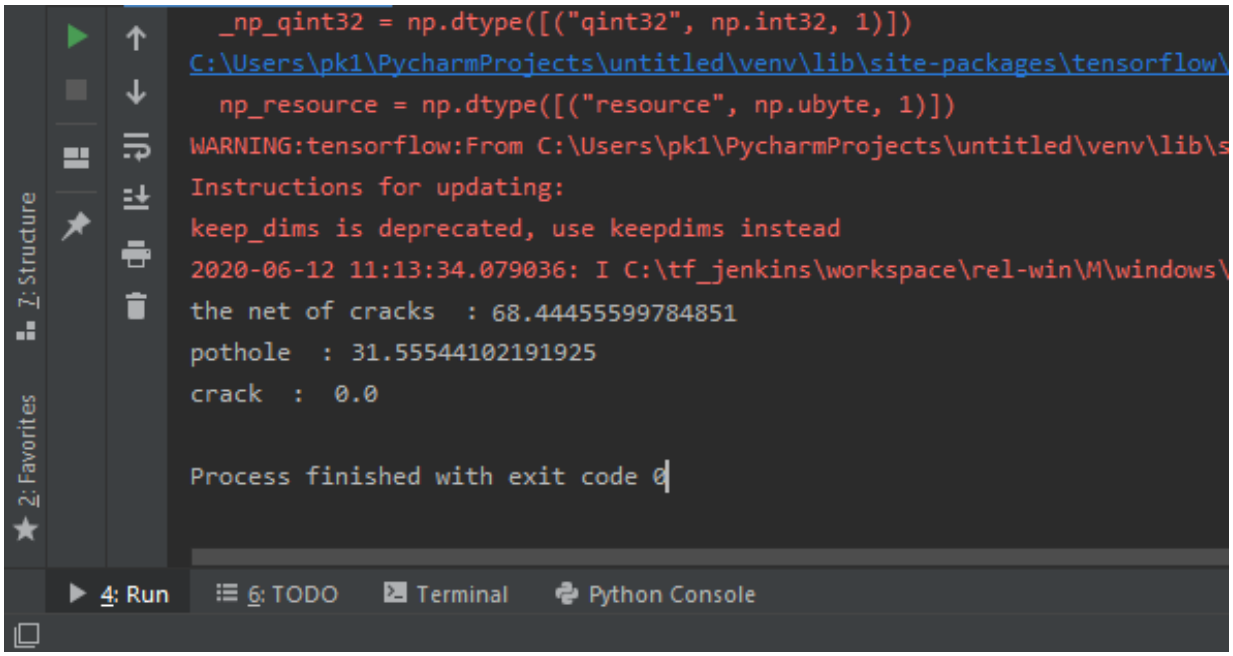


Рис. 7. Результат работы кода

Вывод: данный метод является всего лишь первым этапом (предсказание дефектов по фотографии). Далее мы будем обучать модель распознавать дефекты на фотографии с последующим их выделением.

Создание dataset'а для распознавания дефектов.

Для того, чтобы обучить модель обнаруживать дефекты на фотографиях нам необходимо создать свой уникальный набор изображений в формате Pascal VOC.

Для начала нам нужно выбрать тип объектов, который мы будем определять (в нашем случае, трещины (crack), сетка трещин (the net of the cracks)). Необходимо собрать минимум 200 изображений для каждого объекта, что является минимальной рекомендацией. На рис.8-10 можно набор данных по каждому определяемому дефекту.

Рис. 8. Набор изображений с трещинами

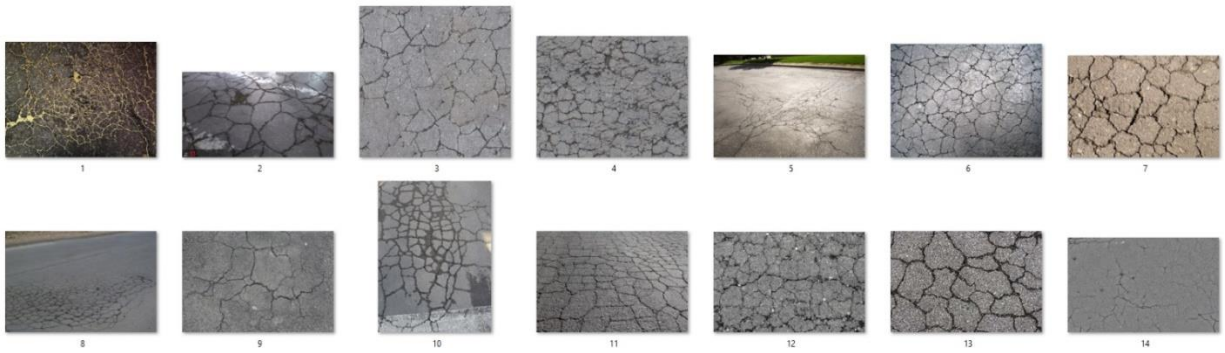


Рис. 9. Набор изображений с сеткой трещин

Рис. 10. Набор изображений с выбоинами

Затем мы должны аннотировать наши объекты, для этого будем использовать инструмент LabelIMG. Чтобы начать пользоваться этим инструментом мы установили следующее инструменты: labelimg (графический инструмент для аннотирования изображений) и PyQt5 (набор расширений графического фреймворка QT для языка программирования Python).

Ниже представлен рис.11 открытого инструмента «labelImg».

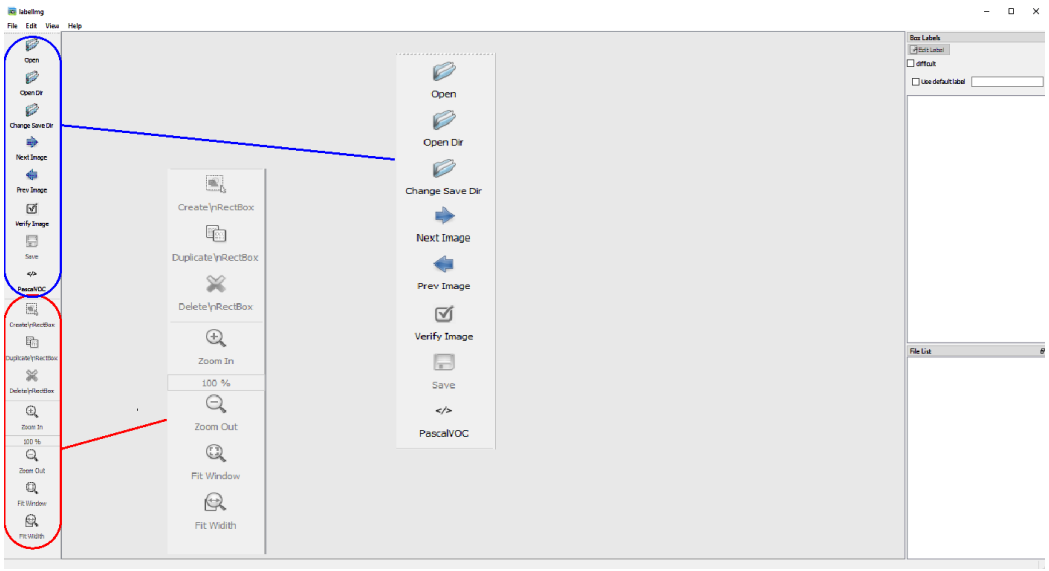


Рис. 11. Открытие инструмента «labaelImg»

Слева мы видим различные кнопки для создания аннотаций. Для открытия фото мы будем пользоваться кнопкой «open», для выделения объекта (в нашем случаи дефект) кнопкой «Creat\nRetBox», для сохранения кнопкой «Save». После того как мы открыли изображение мы должны выделить наш объект (дефект) и создать его название «трещина/crack» как показано на рис.12-13.

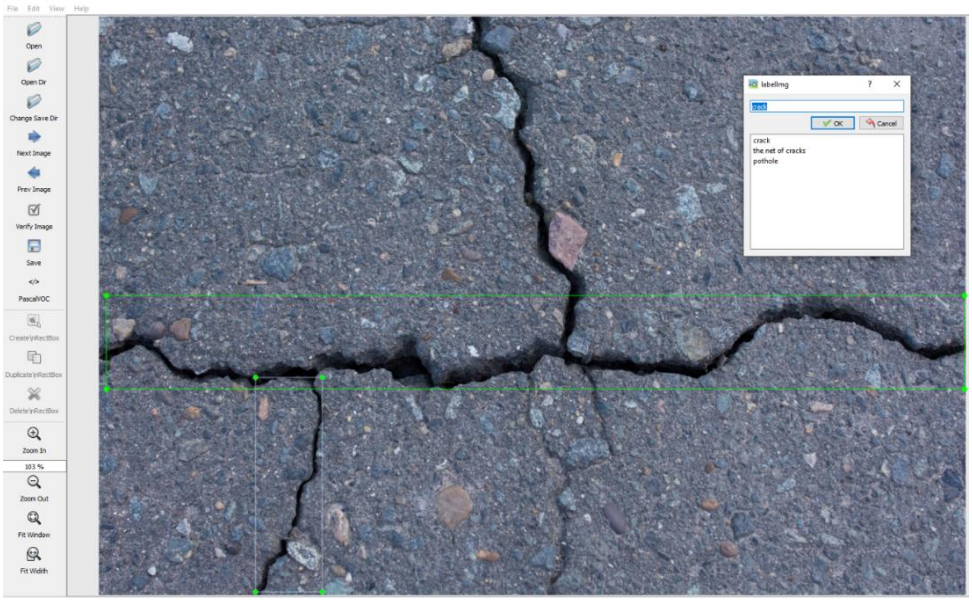


Рис. 12. Процесс создание аннотаций в инструменте labelImg

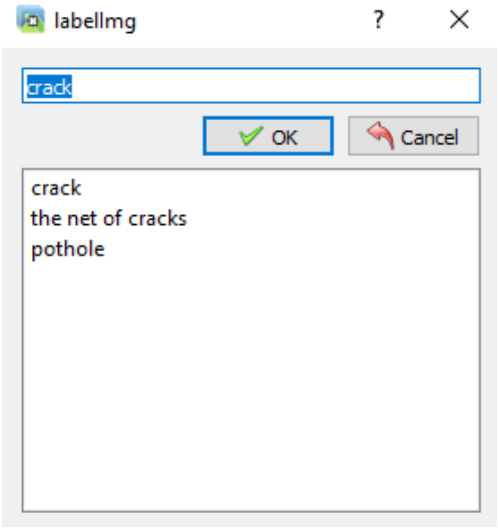


Рис. 13. Процесс создание аннотаций в инструменте labelImg

Далее после того, как создано название объекта (дефекта) мы последовательно выделяем каждый объект (дефект) на изображении и сохраняем аннотатированное изображение с расширение XML. Ниже представлен результат аннотатирования данного изображения (рис.14).

```
<annotation>
        <folder>images</folder>
        <filename>1.jpg</filename>
        <path>D:\new dataset\train\images\1.jpg</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>1280</width>
                <height>852</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>crack</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>428</xmin>
                        <ymin>295</ymin>
                        <xmax>758</xmax>
                        <ymax>761</ymax>
                </bndbox>
        </object>
</annotation>
```

Рис. 14. Результат выполнения (полученный XML файл)

Когда аннотация сохранена, мы должны проделать анологичные действия со всеми изображениями.

- После обработки всех изображений мы должны создать папку «RoadDefects»,

- в ней мы должны создать еще две папки «train» и «validation»,

- в каждой из этих попок мы должны поместить еще по два папки «images» и «annotaitons».

Когда мы создали наш dataset, мы можем перейти к следующему шагу – обучения модели на основании нашего dataset'a.

Разработка кода для обнаружения дефектов по фотографии. Обучение модели с использование архитектуры YOLOv3.

Для того, чтобы определить тот или иной дефект на фотографии нам нужно обучить модель распознавать дефекты. Мы будем обучать нашу модель с использованием архитектуры YOLOv3. Это позволит нам тренировать свою собственную модель на любом наборе изображений.

imageai.Detection.Custom.DetectionModelTrainer – это класс обучения модели обнаружения, который позволяет нам обучать модели обнаружения объектов на наборах данных изображений, которые находятся в формате аннотации Pascal VOC, используя YOLOv3.

Процесс обучения генерирует файл JSON, который отображает имена объектов в вашем наборе данных изображений, а также создает множество моделей.

Для разработки программы был разработан алгоритм (рис.15) её решения.

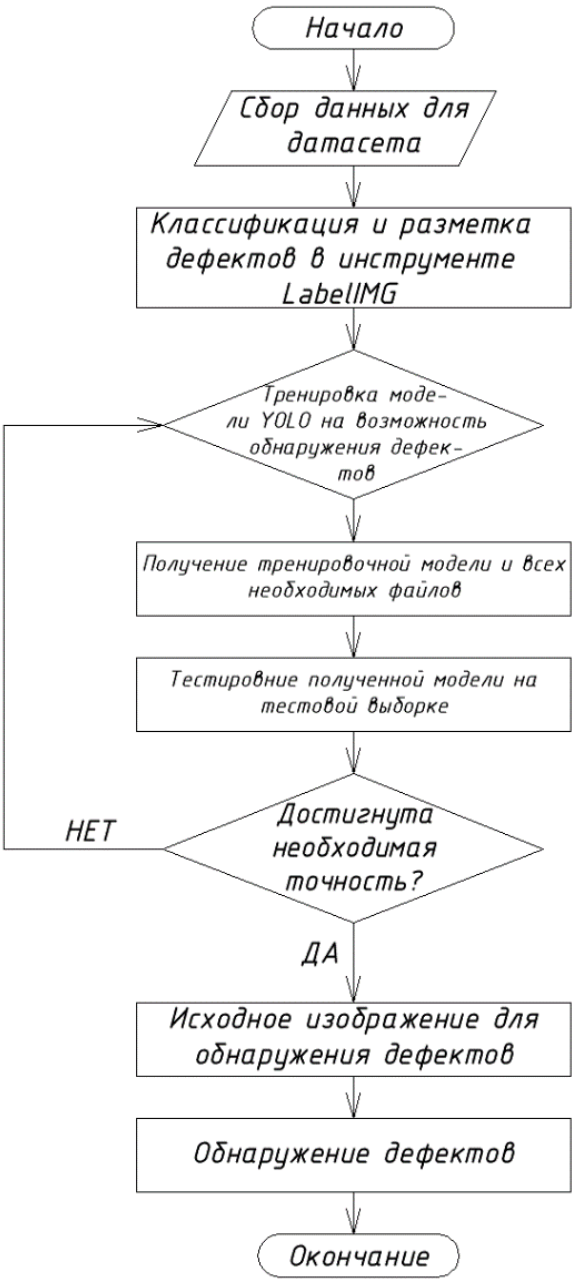


Рис. 15. Блок-схема алгоритма последовательности для обучения модели по обнаружению дефектов

Для выполнения данного алгоритма были задействованы такие же библиотеки, как и для предсказания дефектов, но добавлены еще несколько новых: tensorflow-gpu==1.13.1; yolov3;

Для того, чтобы обучить нашу модель распознавать дорожные дефекты мы использовали собранный dataset а также исходный код для обучения модели (рис. 16).

```
from imageai.Detection.Custom import DetectionModelTrainer

trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory("D:/RoadDefects")
trainer.setTrainConfig(object_names_array=
                       ["crack", "the net of cracks", "pothole"],
                       batch_size=4,
                       num_experiments=20)
trainer.trainModel()
```

Рис. 16. Исходный код для обучения модели

Рассмотрим более подробно отдельные строки в данном коде.

«object_names_array=[«crack», «the net of the cracks», «pothole»]. Подразумевает количество определяемых объектов.

- «batch_size=4». Количество изображение, на которых модель обучается параллельно. В нашем случае 4;

- «num_experiements=20». Это число раз, когда алгоритм будет обучаться на нашем наборе данных изображений. Точность вашего обучения действительно увеличивается по мере увеличения числа раз, когда он тренируется. Однако она достигает своего пика после определенного количества тренировок; и эта точка зависит от размера и природы набора данных;

«train_from_pretrained_model» означает, что наша модель будет обучаться на предтенировонной модели, что дает более высокую точность.

Запускаем код на выполнение. После того как наша модель обучилась в папке с датасетом были автоматически созданы еще несколько папок «cache»,

«json», «logs», «models». В папке «cache» находится файл «detection_train_data.pkl», в папке «json» файл «detection_config.json», в папке «logs» файл «events.out.tfevents.1591179101», в папке «models» находятся обученные модели «detection_model-ex-013--loss-0016.744» количество моделей зависит от количества экспериментов, которые мы задаем. Чем выше количества экспериментов, тем точнее будут модели.

После того как мы получили все данные, мы должны проверить наши модели на точность и выбрать одну из них для дальнейшего использования.

Проверять наши модели мы будем использовать следующий код (рис. 17).

```
1  from imageai.Detection.Custom import DetectionModelTrainer
2
3  trainer = DetectionModelTrainer()
4  trainer.setModelTypeAsYOLOv3()
5  trainer.setDataDirectory("D:/new dataset")
6  metrics = trainer.evaluateModel(model_path="D:/new dataset/models",
7                                  json_path="D:/new dataset/json/detection_config.json",
8                                  iou_threshold=0.5,  object_threshold=0.3,
9                                  nms_threshold=0.5)
10 print(metrics)
```

Рис. 17. Код для проверки моделей

Рассмотрим более подробно строки, использованные в данном коде.

Строка 4. «trainer.setModelTypeAsYOLOv3()»

Строка 5. «trainer.setDataDirectory(“D:/new dataset”)». Задаем путь к папке, где находится датасет

Строка 6. «metrics = trainer.evaluateModel (….)». Задаем путь к папке, где хранятся модели, а также к папке, где расположен файл «detection_config.json»

После этого мы должны «выполнить» код нажав кнопку «run».

Здесь видно (рис. 18), что мы проверяем модель, которая была получена в 12-ом эксперименте, а именно «detection_model-ex-012--loss-0016.914.h»,

далее видно, что мы рассматриваем 3 вида дефектов: трещины (crack); выбоины (pothole); сетка трещин (the net of cracks). Видно, что самый высокий процент детектирования у сетки трещин = 92.56, а самый малый у трещин = 1.05, суммарная точность = 34.54. Следует отметить, что это всего лишь 12-ый эксперимент, который не дал точного определения трещин и выбоин, из этого можно сделать вывод, что наша модель нуждается в дальнейшем обучении. Продолжаем обучение.

В этот раз для избежания «обучения с нуля» мы будем использовать, самую точную из полученных моделей «detection_model-ex-013—loss-0016.744.h5» как предобученную. Для этого нам необходимо поместить предобученную модель в папку с исходным кодом для обучения модели.

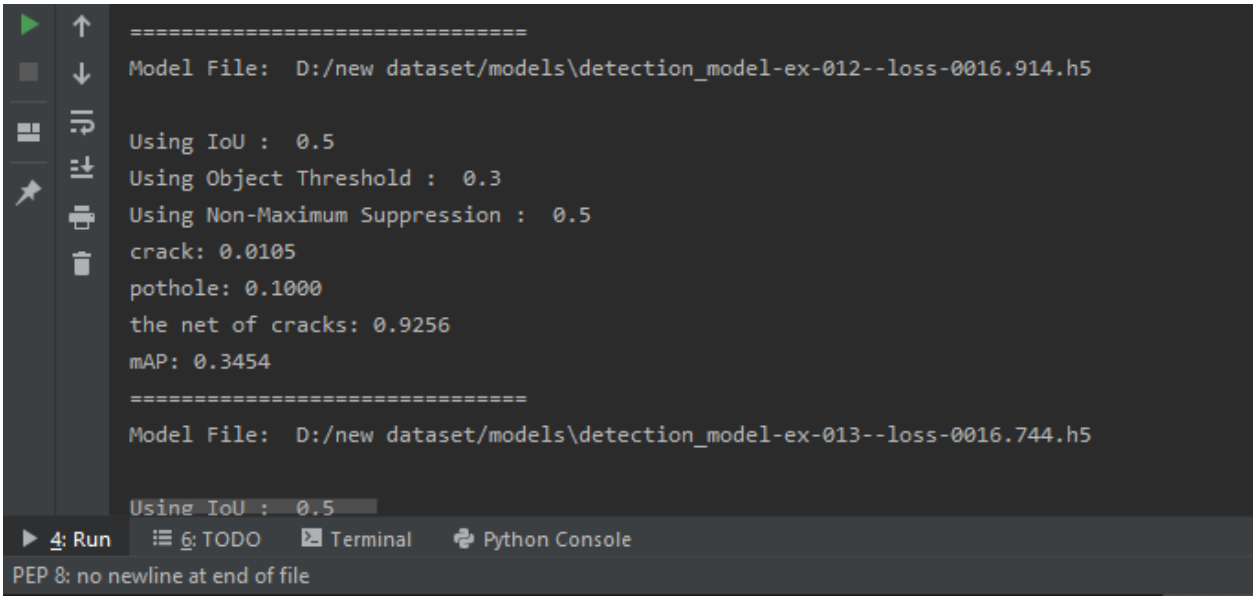


Рис. 18. Пример выполнения кода

Запускаем код. После того как обучения завершено, мы тестируем, полученные модели с использование кода как показано на рис. 19. Результат выполнения можно увидеть на рис. 20.

```
from imageai.Detection.Custom import DetectionModelTrainer

trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory("D:/RoadDefects")
trainer.setTrainConfig(object_names_array=
                       ["crack", "the net of cracks", "pothole"],
                       batch_size=4, num_experiments=100,
                       train_from_pretrained_model=
                       "detection_model-ex-013--loss-0016.744.h5")
trainer.trainModel()
```

Рис. 19. Код для обучения модели с использование предтренированной модели

```
Model File:  D:/install/NEW DATASET1/models\detection_model-ex-055--loss-0019.411.h5

Using IoU :  0.5
Using Object Threshold :  0.3
Using Non-Maximum Suppression :  0.5
crack: 0.8178
pothole: 0.7725
the net of cracks: 0.9321
mAP: 0.8408
==============================
```

Рис. 20. Результат выполнения кода

Исходя из рис.20 видно, что средний процент обнаружения у самой точной модели значительно вырос по сравнению с предыдущим и составляет более 80%, что является достаточно высоким результатом. На этом шаге мы прекращаем наше обучение. В дальнейшем мы будем использовать самую точну. Модель – «detection_model-ex-055—loss-0019.411.h5». Для удобства переименовываем ее в «detect_RoadDefects.h5».

Затем мы должны использовать ее вместе с исходным кодом для детектирования дефектов. Все полученные файлы мы помещаем в одну папку с сходным кодом, а также тестируемое изображение (roaddefect).

Теперь рассмотрим исходный код для обнаружения дефектов (рис. 21).

for defects.py

```
1  from imageai.Detection.Custom import CustomObjectDetection
2
3  detector = CustomObjectDetection()
4  detector.setModelTypeAsYOLOv3()
5  detector.setModelPath("detect_RoadDefects.h5")
6  detector.setJsonPath("detection_config.json")
7  detector.loadModel()
8  detections = detector.detectObjectsFromImage\
9      (input_image="roaddefect.jpg", output_image_path="defect ready.jpg")
10 for detection in detections:
11     print(detection["name"], " : ",
12           detection["percentage_probability"], " : ", detection["box_points"])
```

Рис. 21. Исходный код для определения дефектов с использованием YOLOv3

Рассмотрим более подробно строки из данного кода.

Строка 4. «directot.setModelTypeAsYOLOv3()». Указывает на то, что мы используем обученную модель YOLOv3.

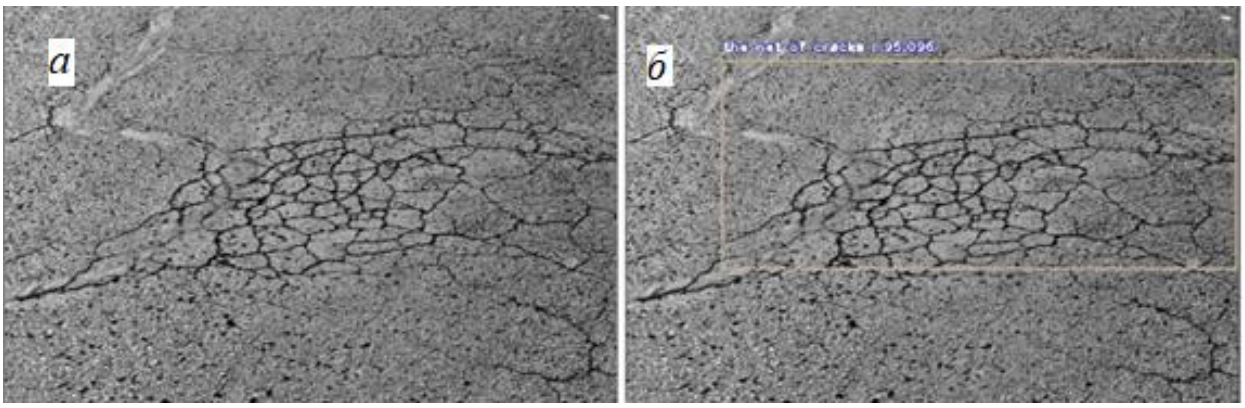
Строка 5. «director.setModelPath(«…»)Задаем путь к папке, где находится обученная модель (в нашем случае модель находится в папке с исходным кодом)

Строка 6. «director.setJsonPath («…»)Задаем путь к папке, где находится полученный файл json (в нашем случае файл находится в папке с исходным кодом)

Строка 8. «detections = director.detectObjectsFromImage (…)Задаем путь к исходному файлу (изображению), задаем путь куда будет сохраняться выходной файл (изображение) в нашем случае входной файл находится в папке с кодом, выходной файл будет сохранен в папке с исходным кодом.

Строка 9. «for detection in detections: print(…)». Задаем что на изображении будет обозначен дефект, его название, а также процент вероятности.

После того как все готово мы загружаем в папку, где расположен код и все необходимые файлы изображение дефекта (roaddefect.jpg) выходное название (defect ready.jpg) мы нажимаем кнопку «run» и наблюдаем процесс выполнения, в результате которого видно, что на изображении (рис. 22) была найдена сетка трещин с вероятностью примерно 95%.



а) входное изображение; б) полученное изображение

Рис. 22. Входные и выходные данные

Результаты работы программы показывают, что, сетка трещин была определена с вероятностью 95% и были обозначены границы сетки трещин.

В тоже время не все результаты работы программы показали такие высокие результаты. При расширении выборки, а также более аккуратное создание аннотаций, что улучшило детектирование. Но стоит отметить, что все также результат не удовлетворял требованием (рис.23) и нуждался в улучшении.

Рис. 23. Результат выполнения на 10-ом эксперименте

На рис.23 видно, что результат выполнения стал более точным по сравнению с предыдущим, однако дефекты определяются не точно и «мелко», и самое главное, что не были детектированы явные дефекты. Для того, чтобы улучшить показатели точности мы еще больше расширили выборку (для повышения точности), выбросили мелкие изображения, а также увеличили число экспериментов для обучаемой модели и использовали предтренированную модель.

Все приведенные выше действия привели к точному детектированию и проценту распознаваемости, результат можно видеть на рис. 24.

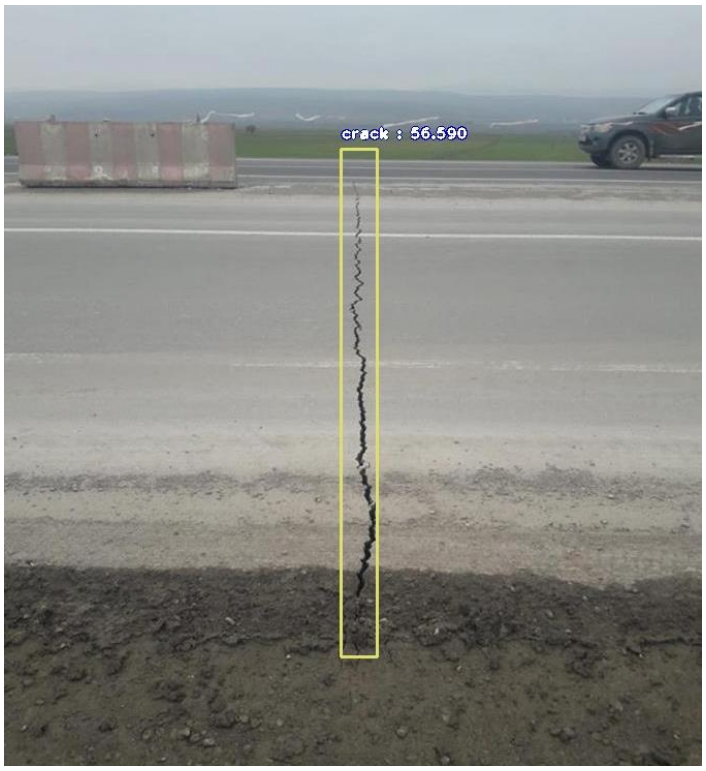


Рис. 24. Результат выполнения

Из всего вышесказанного можно сделать вывод, чтобы получить хороший результат детектирования и процентного обнаружения необходимо собрать большой dataset необходимых дефектов (более 10000 изображений), сделать аннотации точные к каждому изображению, сделать предварительную тренировку (получить предтренерованную модель для более высокой точности), сделать большое количество экспериментов для получения большего количества моделей, а также провести оценку всех полученных моделей для выбора наиболее точный. Придерживаясь этому алгоритму, можно получить достаточно высокие результаты.

Обнаружение дефектов на видео

Одним из самых важных моментов в обнаружении дефектов на дороге, является обнаружение дефектов в динамике (на видеопотоке). Для этого мы будем использовать datset, который был получен ранее, а также полученные

ранее модели. Также для удобства мы переименовали нашу модель в «for video.h5». Для того, чтобы код работ корректно, мы помещаем все файлы в папку с исходным кодом.

Исходный код использует архитектуру YOLOv3 также как для обнаружения дефектов на фото, однако исходный код будет отличаться. (рис. 25)

```
from imageai.Detection.Custom import CustomVideoObjectDetection
import os

execution_path = os.getcwd()

video_detector = CustomVideoObjectDetection()
video_detector.setModelTypeAsYOLOv3()
video_detector.setModelPath("for video.h5")
video_detector.setJsonPath("detection_config.json")
video_detector.loadModel()

video_detector.detectObjectsFromVideo("D:/41.mp4",
                                      output_file_path=os.path.join(execution_path, "road4"),
                                      frames_per_second=30,
                                      minimum_percentage_probability=40,
                                      log_progress=True)

```

Рис. 25. Исходный код для обнаружения дефектов на видео

Рассмотрим более подробно строки из данного кода:

Строка 2. «directot.setModelTypeAsYOLOv3()». Указывает на то, что мы используем обученную модель YOLOv3.

Строка 3. «director.setModelPath(«…»)Задаем путь к папке, где находится обученная модель (в нашем случае модель находится в папке с исходным кодом)

Строка 3. «director.setJsonPath («…»)Задаем путь к папке, где находится полученный файл json (в нашем случае файл находится в папке с исходным кодом)

Строка 3. «video_detector.detectObjectsFromVideo (…)» Задаем путь к исходному файлу (видео), задаем путь куда будет сохраняться выходной файл

(видео) в нашем случае входной файл находится на диске «D», выходной файл будет сохранен в папке с исходным кодом.

Запускаем код. После того, как процесс выполнения кода завершен, мы получаем обработанное видео (рис.26).

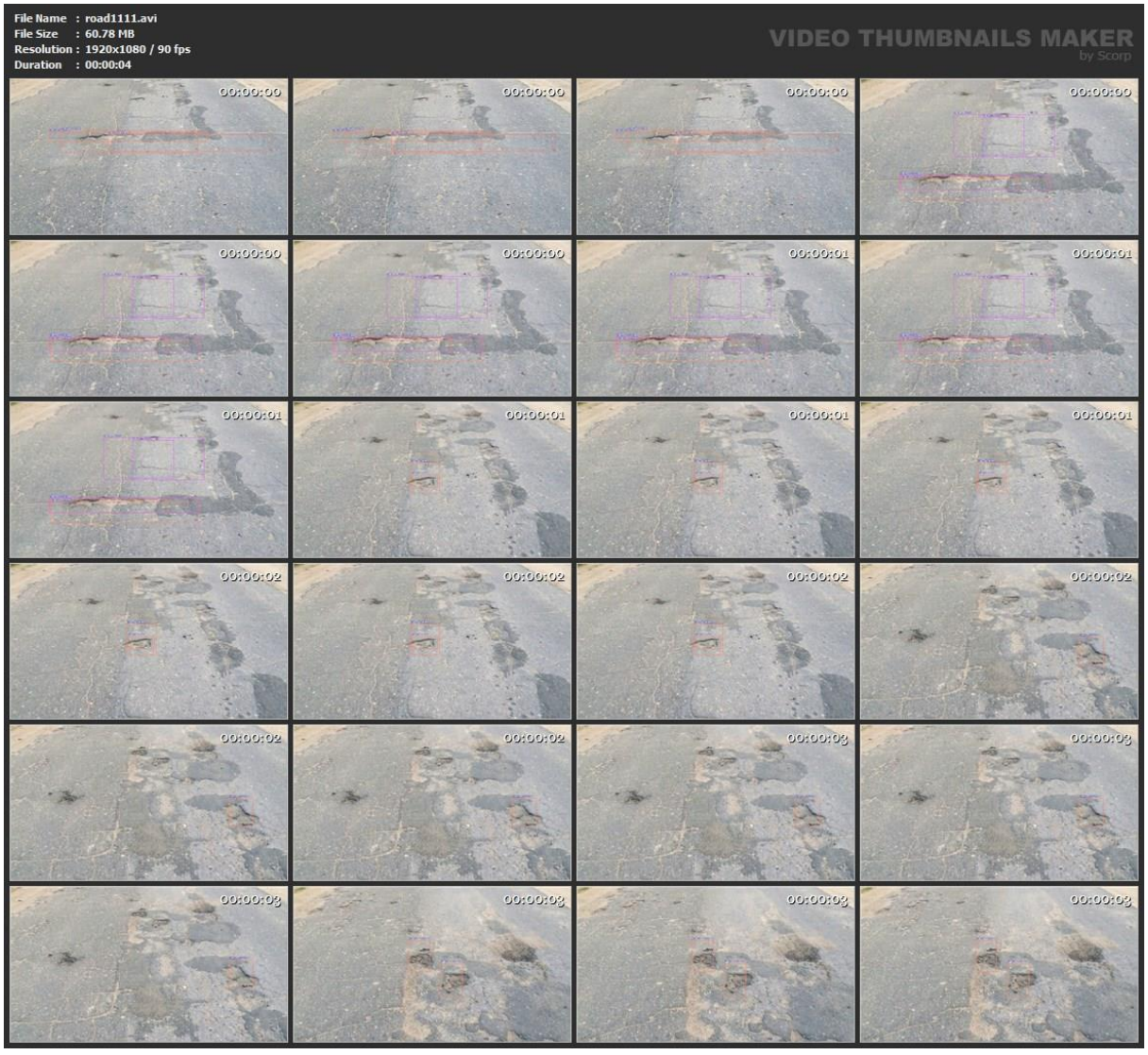


Рис. 26. Раскадровка обработанного видео

Из рис. 26 можно сделать вывод, что теперь наша модель может работать в динамике (обрабатывать видео или онлайн съемку), что являлось нашей целью. Следует также отметить, что точность детектирования достаточно высокая.

Выводы

В заключение можно сказать то, что да технологии компьютерного зрения, нейронных сетей и машинного обучения можно широко применять в дорожной отрасли для различных задач. Данные технологии помогут сократить время и затраты на проведение диагностики, а также увеличат точность и скорость диагностики. Для подтверждения данного тезиса мы провели диагностику дорог, используя методы компьютерного зрения. Первой задачей была разработка алгоритма. В зависимости от алгоритма мы решали различные типы задач по предсказанию и обнаружению дефектов на покрытии автомобильных дорог. Также был предложен алгоритм, упрощающий выполнение задач. Были составлены датасеты с дорожными дефектами для различных типов задач (предсказания и обнаружения). Была выполнена задача по предсказанию дорожных дефектов, также была выполнена задача по обнаружению дорожных дефектов по изображению и самой важной решенной задачей в данной главе являлась обнаружение дорожных дефектов на видеопотоке. Данная задача (обнаружение дефектов на видеопотоке) является наиболее важной, так как обнаружение дефектов в динамике (на видеопотоке) дает нам возможность обнаружения дорожных дефектов в режиме реального времени, что существенно снижает временные и материальные затраты по обнаружению дорожных дефектов на покрытии автомобильных дорог.

Список литературы

1. ТКП 271–2014 (02030). Оценка эксплуатационного состояния и качества содержания до-рожных одежд и дождевой канализации улиц населенных пунктов.; введ. 01.01.2011. – Минск: БНТУ, 2015. – 104 с.

2. ТКП 604–2017 (33200). Автомобильные дороги. Оценка эксплуатационного состояния и качества содержания; введ. 01.09.2017. – Минск: Белдорцентр, 2017. – 64 с.

3. СТБ 1291–2016. Дороги автомобильные и улицы. Требования к эксплуатационному состоянию, допустимому по условиям обеспечения безопасности дорожного движения. – Взамен СТБ 1291–2007; введ. 01.07.2017. – Минск : БелдорНИИ, 2016. – 28 с.

4. Кутузов, В.В. Диагностика автомобильных дорог на основе алгоритмов компьютерного зрения / В. В. Кутузов, А. С. Литвинчук // Новые материалы, оборудование и технологии в промышленности : Материалы Международной научно-технической конференции мо-лодых ученых, Могилев, 24–25 октября 2019 года / Редколлегия: М.Е. Лустенков [и др.]. – Могилев: Межгосударственное образовательное учреждение высшего образования "Бело-русско-Российский университет", 2019. – С. 134.

5. Dr. Adrian Rosebrock. Deep Learning for Computer Vision with Python. Practitioner Bundle – PyimageSearch.com, 2017. – 323 p.

6. OpenCV modules Tutorials [Электронный ресурс]. - Режим доступа: https://docs.opencv.org/. - Дата доступа: 28.10.2022.

7. Video and Live-Feed Detection and Analysis - ImageAI [Электронный ресурс]. - Режим доступа: https://imageai.readthedocs.io/en/latest/video/index.html. - Дата доступа: 28.10.2022.

8. labelImg PyPI [Электронный ресурс]. - Режим доступа: https://pypi.org/project/labelImg/. - Дата доступа: 28.10.2022.

9. Python documentation [Электронный ресурс]. - Режим доступа: https://docs.python.org/3/. - Дата доступа: 28.10.2022.

10. tensorflow [Электронный ресурс]. - Режим доступа: https://www.tensorflow.org. - Дата доступа: 28.10.2022.

11. YOLO: Real-Time Object Detection [Электронный ресурс]. - Режим доступа: https://pjreddie.com/darknet/yolo/. - Дата доступа: 28.10.2022.