МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет математики и компьютерных наук
Кафедра математических и компьютерных методов

КУРСОВАЯ РАБОТА

РАЗРАБОТКА СИСТЕМЫ ГЕНЕРАЦИИ И ВЕРИФИКАЦИИ СИНТЕТИЧЕСКИХ ДАННЫХ ДЛЯ ЗАДАЧ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ

Работу выполнила_____О. К. Марчук
(подпись)

Направление подготовки 02.03.01 Математика и компьютерные науки курс 3

Направленность (профиль) Математическое и компьютерное моделирование

Научный руководитель
ст.преподаватель_____А. П. Невечеря
(подпись, дата)

Нормоконтролер
ст. лаборант_____В. В. Писоцкая
(подпись, дата)

Краснодар
2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сбор и разметка достаточного количества реальных данных часто связана с большими временными или материальными затратами. Чтобы добавить данных для обучения классификатора, используются генераторы, но изображения, выводимые этими генераторами, необходимо верифицировать - определить, сохраняются ли в них статистические свойства, характерные для реальных данных.

Объект исследования – классифицируюшие способности свёрточных ИНС на заданном классе изображений.

Предмет исследования – влияние синтетических данных на эффективность свёрточного классификатора дефектов на металлических листах.

Цель работы: проверить гипотезу о том, что датасет, состоящий из фотографий листов металла можно дополнить или частично заменить синтетическим, что приведёт к возрастанию точности.

Задачи:

- разработка генератора изображений;
- разработка верификатора этих изображений;
- проведение A/B тестирования как метода верификации.

В ходе работы будут проведены следующие действия:

- анализ задачи и поиск информации о разработке аддонов на Blender;
- изучение архитектур свёрточных нейросетей;
- подтверждение или опровержение гипотезы о дополнении датасета синтетическими данными.

Требования к программе:

- имплементация нескольких методов разметки;
- получение метрик модели;
- наличие графического интерфейса, интегрированного в Blender.

1 Анализ условия задачи и разработка алгоритма решения

Постановка задачи предполагает написание двух алгоритмов: первая программа генерирует процедурные изображения листов металла с различными дефектами, вторая их верифицирует. В силу простоты имплементации программа-генератор будет разрабатываться в виде аддона Blender, что позволяет создавать свои операторы и работать с различными контекстами. Для реализации верификатора используются модули `Keras` и `Tensorflow`.

Генератор должен содержать несколько типов дефектов, возможность их комбинирования на одном изображении, настройки для количества изображений, учёта карты шероховатостей, нормалей и смещения. Также каждому сгенерированному изображению должна соответствовать маска дефектов, которая будет использоваться как разметка для обучения модели.

Верификатор должен соединить реальный и синтетический датасеты в разных пропорциях и провести несколько итераций обучения свёрточной нейронной сети на полученных данных.

Базовый лист металла и отдельные дефекты реализованы как группы нод в материале `Blender`. Дефекты получаются путём применения математических операций к процедурным шумам, и изображения получаются различными ввиду псевдослучайности зерна шума. Чёрно-белые маски дефектов будут вынесены в отдельный материал.

Для обучения сети будет использована метрика `Intersection over Union`, функция потерь `dice_loss` и архитектура:

```
inputs = Input(shape=input_shape)
conv1 = Conv2D(32, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2(weight_decay))(inputs)
pool1 = MaxPooling2D((2, 2))(conv1)
conv2 = Conv2D(64, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2(weight_decay))(pool1)
pool2 = MaxPooling2D((2, 2))(conv2)
conv3 = Conv2D(128, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2(weight_decay))(pool2)
conv3 = Dropout(0.3)(conv3)
up1 = UpSampling2D((2, 2))(conv3)
```

```
concat1 = Concatenate()([up1, conv2])
conv4 = Conv2D(64, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2(weight_decay))(concat1)
up2 = UpSampling2D((2, 2))(conv4)
concat2 = Concatenate()([up2, conv1])
conv5 = Conv2D(32, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2(weight_decay))(concat2)
outputs = Conv2D(1, (1, 1), activation='sigmoid')(conv5)
```

Модель будет обучаться в контейнере среды Docker, в котором будет скомпилирован Tensorflow. В качестве базового образа взят tensorflow-latest. Для поддержки GPU также необходимо будет установить nvidia-container-toolkit (система-хост – Manjaro Linux).

Структура программы-верификатора (символы «→» здесь обозначают запуск каждого файла и перенос аргументов между ними):

```
main.py → docker_runner.py(real_path, synth_path, epoch, batch,
test_size) → model.py(real_path, synth_path, epoch, batch,
test_size) → utils.py(real_path, synth_path) → model.py →
main.py(results)
```

Структура программы-генератора состоит из трёх файлов, организованных в виде аддона blender:

```
aluminum_gen.py, contours.py, aluminum_generator.blend
```

В последнем файле `.blend` хранится сцена со всеми материалами, необходимыми для генерации синтетического датасета.

2 Программная формализация задачи: генератор

Прежде чем приступить к написанию программ, нужно выбрать среду программирования, где будет реализована задача. 3D редактор Blender предлагает большое количество возможностей для скриптинга и создания дополнений (аддонов).

Модуль bpy является официальным API Blender, cv2 – модуль OpenCV для обработки изображений в задачах компьютерного зрения, numpy – модуль, имплементирующий массивы и быстрые операции с ними. Остальные импорты – встроенные модули Python.

```
import bpy
import os
import random
import bpy
from datetime import datetime
import cv2 as cv
import numpy as np
import sys
import importlib
```

Заголовок аддона:

```
bl_info = {
    "name": "Data Generator",
    "author": "practice_kubsu_marchuk",
    "version": (1, 0),
    "blender": (3, 6, 2),
    "location": "Output Properties > Data Generator",
    "description": "Генерация синтетических данных для
определения дефектов алюминиевых листов",
    "category": "Output",
}
```

Далее объявляется класс DATA_GENERATOR_PT_Panel, который будет отрисовывать элементы интерфейса генератора:

```
class DATA_GENERATOR_PT_Panel(bpy.types.Panel):
    bl_label = "Data Generator"
    bl_idname = "DATA_GENERATOR_PT_Panel"
    bl_space_type = "PROPERTIES"
    bl_region_type = "WINDOW"
    bl_context = "output"
```

```
    def draw(self, context):
        layout = self.layout
        scene = context.scene
        output_props = scene.data_generator_props
```

Полный список из девятнадцати дефектов приведён в приложениях. Далее объявляется класс DATA_GENERATOR_Properties с некоторыми свойствами, которые сможет настраивать пользователь:

```
class DATA_GENERATOR_Properties(bpy.types.PropertyGroup):
    dataset_size: bpy.props.IntProperty(
        name="Количество изображений",
        default=1,
        min=0,
        max=10000,
        step=1)
```

dataset_size – количество изображений в синтетическом датасете.

```
    max_tries: bpy.props.IntProperty(
        name="Количество попыток перерендера",
        default=5,
        min=0,
        max=100,
        step=1)
```

max_tries - Если в битмаске дефекта менее 100 не чёрных пикселей, то сменить свойство Randomness и перерендерить битмаску указанное пользователем количество раз. Предотвращает появление изображений, где дефект не присутствует из-за случайных шумов.

```
    output_path: bpy.props.StringProperty(
        name="Вывод",
        subtype="FILE_PATH")
    use_roughness: bpy.props.BoolProperty(
        name="Рендерить шероховатости (Roughness)",
default=False)
    use_normal: bpy.props.BoolProperty(
        name="Рендерить неровности (Normal)", default=False)
    use_displace: bpy.props.BoolProperty(
        name="Рендерить неровности (Displacement)",
default=False)
```

Чекбоксы для рендера шероховатостей и неровностей управляют дополнительными параметрами генератора – если они выбраны, то рендер займёт больше времени, но изображение получится более высокого качества.

```
    find_contours: bpy.props.BoolProperty(
        name="Найти контуры", default=False)
    find_bound_box: bpy.props.BoolProperty(
        name="Найти ограничивающие прямоугольники",
default=False)
```

Также в генераторе имплементированы два дополнительных варианта разметки, кроме битмасок – поиск контуров и поиск ограничивающих прямоугольников. Однако, в этой работе используются только маски, так как ограничивающие прямоугольники показали себя неэффективными для данной задачи.

Оператор рендера – главная функция, которую вызывает аддон:

```
class DATA_GENERATOR_OT_RenderOperator(bpy.types.Operator):
    bl_idname = "object.data_generator"
    bl_label = "Render"

    def execute(self, context):
        """ ОПЕРАТОР """

        props = context.scene.data_generator_props

        path = dir_handler(bpy.path.abspath(props.output_path))
        present_list = get_present_list(props)
        transfer_values_to_bitmaps(present_list,
random_only=False)
        inverted_bump_list, bump_list =
get_adder_list(present_list)
        unlink_all("aluminum")
        enable_nodes(present_list, "color_adder")
        enable_nodes(inverted_bump_list, "inverted_bump_adder")
        enable_nodes(bump_list, "bump_adder")
        enable_effects(props.use_roughness, props.use_normal,
props.use_displace)

        nodes_alum =
bpy.data.materials["aluminum"].node_tree.nodes
        base_node = nodes_alum.get("metal_base")

        render_start = datetime.now()

        for image_idx in range(props.dataset_size):
```

```
            base_node.inputs[0].default_value =
round(random.uniform(-100, 100), 3)
            transfer_values_to_bitmaps(present_list,
random_only=True)
            render_bitmaps(path, present_list, image_idx,
props.max_tries, props.find_contours, props.find_bound_box)
            render_aluminum(path, image_idx)

        render_end = datetime.now()
        print(f"Total time: {render_end - render_start}")

        return {"FINISHED"}
```

Готовый интерфейс генератора изображён на рис. 1.

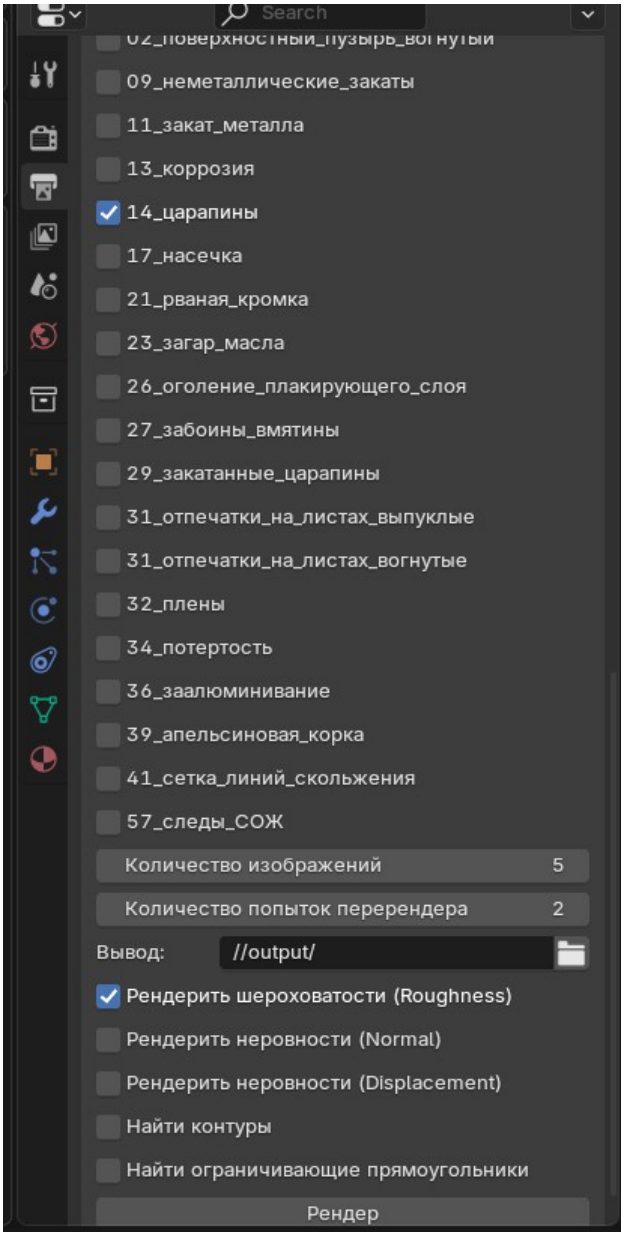

Рисунок 1 – Интерфейс генератора

Настройки, относящиеся к отдельным дефектам, выведены в группы нодов. При генерации каждого изображения у активных дефектов обновляется как минимум один параметр (`Randomness`), и поэтому необходимо переносить значения настроек на материал битмасок. После рандомизации, переноса, включения специальных эффектов и подключения необходимых групп нодов рендерится битмаска для каждого дефекта в отдельности, а потом – сам материал алюминия со всеми выбранными дефектами.

Функция переноса значений:

```
def transfer_values_to_bitmaps(present_list: list,
random_only=True):
    """ Перенести установленные пользователем значения из alum в
bitmaps, all - перенос всех параметров """

    nodes_bit = bpy.data.materials["bitmaps"].node_tree.nodes
    nodes_alum = bpy.data.materials["aluminum"].node_tree.nodes
    for name in present_list:
        node, node_to_copy = nodes_bit.get(name),
nodes_alum.get(name)
        randomize_node(node_to_copy)
        if not random_only:
            end = 1
        else:
            end = len(node.inputs) - 1
        for i in range(0, len(node.inputs) - end):
            node.inputs[i].default_value =
node_to_copy.inputs[i].default_value
```

Все цветовые каналы групп дефектов подключаются к центральной ноде `color_adder`, которая складывает отдельные шумы между собой. То же самое происходит и с выводами неровностей, для них существует `bump_adder` и `inverted_bump_adder`. Инвертированные неровности создают иллюзию выпуклости, например, полос коррозии, а обычные неровности выглядят вдавленными в металл.

```
def get_adder_list(present_list: list):
    nodes = bpy.data.materials["aluminum"].node_tree.nodes
    inverted_bump_list = []
    for name in present_list:
        node = nodes.get(name)
```

```
        if node.inputs[-3].default_value < 0:
            inverted_bump_list.append(name)

    bump_list = [i for i in present_list if i not in
inverted_bump_list]
    return inverted_bump_list, bump_list
```

В функции `render_bitmaps` происходит переключение материала алюминия на материал, содержащий битмаски дефектов. По этим же битмаскам выполняется поиск контуров и ограничивающих прямоугольников.

```
def render_bitmaps(path: str, present_list: list, image_indx:
int, max_tries: int,
                   find_contours: bool, find_bound_box: bool):
    """ Отрендерить и сохранить bitmaps """

    obj = bpy.data.objects["Plane"]
    obj.active_material = bpy.data.materials["bitmaps"]
    nodes = bpy.data.materials["bitmaps"].node_tree.nodes
    links = bpy.data.materials["bitmaps"].node_tree.links
    adder =
bpy.data.materials["bitmaps"].node_tree.nodes["adder"]
    unlink_all("bitmaps")

    for name in present_list:
        node = nodes.get(name)
        link = links.new(node.outputs[0], adder.inputs[0])
        path_to_bitmap = os.path.join(path, "bitmaps",
("bitmap%d_%s.jpg" % (image_indx, name)))
        path_to_csv = os.path.join(path, "images",
("contours%d_%s.csv" % (image_indx, name)))
        bpy.context.scene.render.filepath = path_to_bitmap
        bpy.ops.render.render(write_still=True)
        limit_blank_defects(path_to_bitmap, node,
max_tries=max_tries)
        links.remove(link)

        # CONTOURS!
        if find_contours:
            contours.contours_csv(idx=image_indx,
image_path=path_to_bitmap, path_to_csv=path_to_csv)
        if find_bound_box:
            contours.bound_box_csv(idx=image_indx,
image_path=path_to_bitmap, path_to_csv=path_to_csv)
```

После объявления остальных утилитарных функций аддон регистрируется в системе `Blender`:

```
classes = [
    DATA_GENERATOR_PT_Panel,
    DATA_GENERATOR_Properties,
    DATA_GENERATOR_OT_RenderOperator
]

def register():
    for cls in classes:
        bpy.utils.register_class(cls)

    bpy.types.Scene.data_generator_props =
bpy.props.PointerProperty(type=DATA_GENERATOR_Properties)


def unregister():
    for cls in classes:
        bpy.utils.unregister_class(cls)

    del bpy.types.Scene.data_generator_props


if __name__ == "__main__":
    import_dir = os.path.dirname(bpy.data.filepath)
    if import_dir not in sys.path:
        sys.path.append(import_dir)

    import contours
    importlib.reload(contours)

    register()
```

2.1 Программная формализация задачи: верификатор

Разработка верификатора начинается с файла `main.py`, в котором будет задаваться интерфейс программы и вызов контейтера `Docker`. Объявляется класс `VerifierGUI`:

```
import threading
import tkinter as tk
from tkinter import filedialog, font, scrolledtext
from docker_runner import DockerManager

class VerifierGUI:
    def __init__(self):
        self.root = tk.Tk()
        self.docker_manager = DockerManager()

        tk.Misc.rowconfigure(self.root, [i for i in range(6)],
weight=1)
        tk.Misc.columnconfigure(self.root, [i for i in
range(2)], weight=1)
        label_font = font.Font(family="Verdana", size=12)
        entry_font = font.Font(family="Verdana", size=11)
        button_font = font.Font(family="Verdana", size=15)

        self.root.title("Synth_Verifier")
```

Задаются стили для кнопок, ярлыков и полей:

```
        self.dataset_real_lbl = tk.Label(self.root,
text="Датасет из реальных данных:", font=label_font)
        self.dataset_real_entry = tk.Entry(self.root,
font=entry_font)
        self.dataset_real_btn = tk.Button(self.root,
text=u"\U0001F4C2", font=button_font,
                                          bg="grey",
command=lambda : self.ask_dir("real"))

        self.dataset_synthetic_lbl = tk.Label(self.root,
text="Датасет для верификации:", font=label_font)
        self.dataset_synthetic_entry = tk.Entry(self.root,
font=entry_font)
        self.dataset_synthetic_btn = tk.Button(self.root,
text=u"\U0001F4C2", font=button_font,
                                               bg="grey",
command=lambda : self.ask_dir("synthetic"))

        self.epoch_count_lbl = tk.Label(self.root,
text="Количество эпох:", font=label_font)
        self.epoch_count_entry = tk.Entry(self.root,
```

```
font=entry_font)

        self.batch_count_lbl = tk.Label(self.root, text="Размер
batch:", font=label_font)
        self.batch_count_entry = tk.Entry(self.root,
font=entry_font)

        self.test_size_lbl = tk.Label(self.root, text="Размер
тестового датасета:", font=label_font)
        self.test_size_entry = tk.Entry(self.root,
font=entry_font)

        self.results_text = scrolledtext.ScrolledText(self.root,
font=label_font)
        self.results_text.config(wrap=tk.WORD, state="disabled",
font="Verdana", width=70, height=20)
        self.start_btn = tk.Button(self.root, text="Начать
верификацию",

font=font.Font(family="Verdana", size=12),
command=self.run_container)
```

Далее элементы интерфейса расставляются в `grid` (весь интерфейс изображён на рис. 2), и объявляются функции для вызова файлового браузера и для запуска контейнера, которому передаются значения `real_path`, `synthetic_path`, `epoch_count`, `batch_count`, `test_size` из полей.

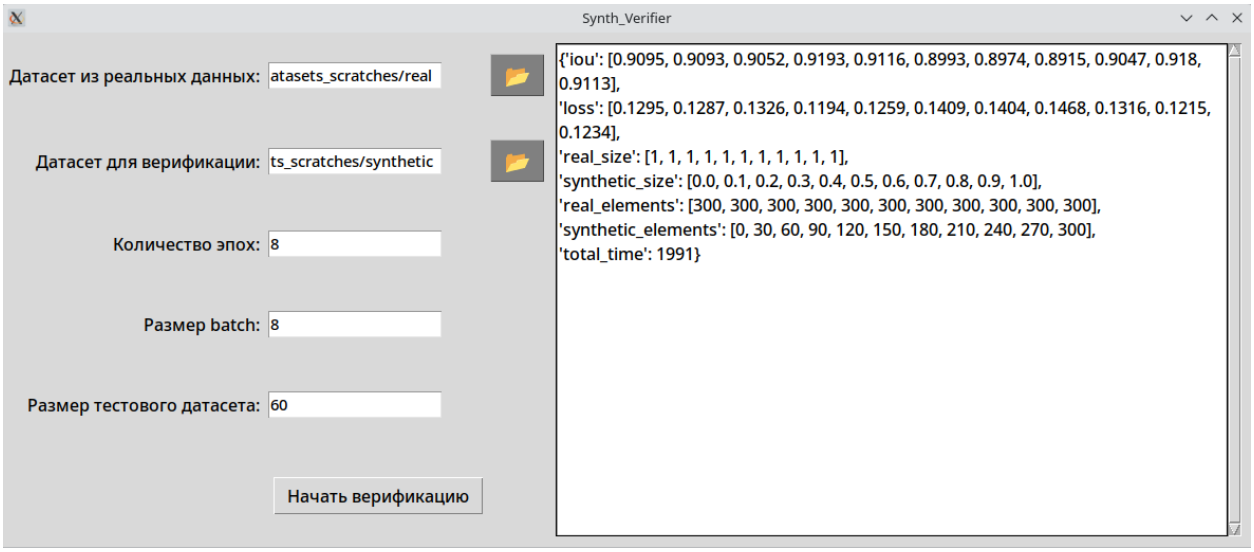

Рисунок 2 – Интерфейс верификатора

Контейнер будет запущен в отдельном потоке, чтобы `UI` оставался активен даже во время обучения сети:

```
    def ask_dir(self, dataset_type):
        file_path = filedialog.askdirectory()
```

```
        if dataset_type =="real":
            self.dataset_real_entry.delete(0, tk.END)
            self.dataset_real_entry.insert(0, file_path)
            return
        self.dataset_synthetic_entry.delete(0, tk.END)
        self.dataset_synthetic_entry.insert(0, file_path)

    def run_container(self):
        real_path, synthetic_path = self.get_paths()
        batch_count, epoch_count = self.batch_count_entry.get(),
self.epoch_count_entry.get()
        test_size = self.test_size_entry.get()

        args = [real_path, synthetic_path, epoch_count,
batch_count, test_size]
        self.update_output("Контейнер запущен...")

        threading.Thread(

target=self.docker_manager.run_script(args_for_model=args,
callback=self.update_output),
            args=("model.py", self.update_output),
            daemon=True).start()
```

Вывод логов из контейнера отображается в интерфейсе:

```
    def update_output(self, text):
        self.results_text.config(state="normal")
        self.results_text.insert(tk.END, "".join(text))
        self.results_text.see(tk.END)
        self.results_text.config(state="disabled")

        self.root.update_idletasks()

gui = VerifierGUI()
```

Следующий файл: `docker_runner.py`, но сначала нужно построить образ на основе `tensorflow:latest-gpu` с помощью `Dockerfile`:

```
FROM tensorflow/tensorflow:latest-gpu
RUN apt-get update
RUN apt-get install -y libgl1-mesa-glx
WORKDIR /tf
RUN python -m venv .venv1
RUN pip install --upgrade pip
RUN pip install opencv-python
RUN pip install keras
RUN pip install pandas
RUN pip install scikit-learn
```

В docker_runner.py включена поддержка GPU и образ не перестраивается каждый раз, если он уже есть в системе. После завершения работы программы контейнер этого образа удалится сам. Логи пишутся в объект callback, который до этого был указан как self.results_text.

```
import docker
import os


class DockerManager:
    def __init__(self):
        self.client = docker.from_env()
        self.image_name = "tensorflow-container:latest"
        self.host_dir = os.getcwd()

    def ensure_image_exists(self):
        try:
            self.client.images.get(self.image_name)
            print("Image exists")
        except docker.errors.ImageNotFound:
            print("Building image...")
            self.client.images.build(path=".",
tag=self.image_name)

    def run_script(self, args_for_model, callback=None):
        command = ["python", f"/tf/model.py"] + args_for_model
        command = " ".join(command)
        # real_path, synthetic_path, epoch_count, batch_count,
test_size
        container = self.client.containers.run(
            image=self.image_name,
            command=command,
            user=f"{os.getuid()}:{os.getgid()}",
            runtime="nvidia",

device_requests=[docker.types.DeviceRequest(count=-1,
capabilities=[['gpu']])],
            volumes={self.host_dir: {'bind': '/tf', 'mode':
'rw'}},
            working_dir="/tf",
            detach=True,
            stdout=True,
            stderr=True
        )

        for line in container.logs(stream=True):
            if callback:
                callback(line.decode('utf-8').strip())

        container.remove()
```

Аналог команды zsh для запуска такого же контейнера:

```
docker run -u $(id -u):$(id -g) --gpus all -it -v
/home/yara/Documents/otherstuff_v3.0/tech_prog/aluminum_generato
r/:/tf tensorflow-container:latest
```

Файл model.py приведён также в сокращённом виде. Для предотвращения переобучения имплементирована функция ReduceLROnPlateau и learning_rate понижен до 0,001. В обычном режиме программа проводит тест с замещением реального датасета синтетическим:

```
real_dataset_path, synthetic_dataset_path, epoch_count,
batch_count, test_size = sys.argv[1:] "total_time": -1}
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=2, min_lr=1e-6)
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

X_synthetic_full, y_synthetic_full =
utils.load_dataset(synthetic_dataset_path)
X_real_full, y_real_full = utils.load_dataset(real_dataset_path)

for i in range(11):
    real_size = round(1 - i/10, 1)
    synthetic_size = round(i/10, 1)

    real_params = X_real_full, y_real_full, real_size
    synthetic_params = X_synthetic_full, y_synthetic_full,
synthetic_size

    X_mixed, y_mixed, X_test, y_test =
utils.get_mixed_data(real_params, synthetic_params, test_size)

    real_elements_num, synthetic_elements_num =
utils.get_number_of_elements(real_params, synthetic_params)
    model = build_segmentation_model()
    model.compile(optimizer=Adam(learning_rate=1e-3),
                  loss=utils.dice_loss,
```

Формула (1) для dice_loss, вычисляемая на основе истинных и предсказанных значений для класса пикселей:

$$loss = 1 - 2\frac{\mathrm{sum}\left(y_{true} \cdot y_{pred}\right)}{\mathrm{sum}\left(y_{true}\right) + \mathrm{sum}\left(y_{pred}\right)} \quad (1)$$

```
metrics=[keras.metrics.BinaryIoU(target_class_ids=(0, 1),
threshold=0.5, name=None, dtype=None)])
    history = model.fit(X_mixed, y_mixed,
                        validation_data=(X_test, y_test),
                        epochs=epoch_count,
                        batch_size=batch_count,
                        verbose=1,
                        callbacks=[reduce_lr, early_stopping])
```

Так как класса всего два (есть дефект или нет дефекта), то используется метрика `BinaryIoU – Intersection over Union`. Чем выше количество перекрывающихся пикселей на предсказанной и на истинной битмаске, тем выше значение этой метрики.

```
    folder_name = "pred" + str(real_size)
    pred_folder_names.append(folder_name)
    for image_file in os.listdir(os.path.join(real_dataset_path,
"images"))[::10]:
        utils.visualize_predictions(real_dataset_path, model,
image_file, folder_name, threshold=0.7)
```

Предсказания модели сохраняем в отдельные папки для каждой итерации обучения.

```
    loss, iou = model.evaluate(X_test, y_test, batch_size=8)
    loss, iou = round(loss, 4), round(iou, 4)

    results_dict["iou"].append(iou)
    results_dict["loss"].append(loss)
    results_dict["real_size"].append(real_size)
    results_dict["synthetic_size"].append(synthetic_size)

results_dict["synthetic_elements"].append(synthetic_elements_num
)
    results_dict["real_elements"].append(real_elements_num)
    print(results_dict)


time_end = time.time()
results_dict["total_time"] = round(time_end - time_start)
print(results_dict)
```

Все результаты находятся в словаре `results_dict`, по ним и были построены графики в приложении.

2.2 Поиск и подготовка тестового датасета

Для того, чтобы оценить синтетические изображения листов металла, нужно сначала найти реальные. Большое количество уже готовых датасетов можно импортировать из фреймворка `Tensorflow`, но в нём нет изображений с листами металла. Эти датасеты находятся на интернет-ресурсе `kaggle` – там был найден датасет `NEU Metal Surface Defects Database`, который используется в этой работе в качестве реальных данных. Он содержит шесть типов дефектов, но для работы с гипотезой достаточно всего одного.

Триста изображений царапин размечены в `.csv` файлах, но не до конца – на многих листах, где очевидно присутствует дефект, файл разметки не выделяет отдельный объект. С помощью `openCV` выделим битмаски из изображений:

```
import cv2 as cv
import os

dataset_path = "example_datasets_scratches/real/"
im_dir = os.listdir(f"{dataset_path}/images")
for image_name in im_dir:
    binary_name = image_name.replace("image", "bitmap")
    im_path = f"{dataset_path}/images/" + image_name
    im = cv.imread(im_path)
    im_grayscale = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
    blur_image = im_grayscale
    blur_image = cv.GaussianBlur(src=im_grayscale,
dst=blur_image, ksize=(3, 3), sigmaX=1, sigmaY=1)

    thresh = cv.adaptiveThreshold(blur_image, 255,
        cv.ADAPTIVE_THRESH_GAUSSIAN_C,
        cv.THRESH_BINARY, 41, -2)
    _, thresh = cv.threshold(thresh, 128, 255, cv.THRESH_BINARY)
    cv.imwrite(f"{dataset_path}/bitmaps/{binary_name}", thresh)
```

Сохраним полученные битмаски. В отличие от синтетических, они не являются абсолютно точными и напрямую зависят от коэффициентов функции `adaptiveThreshold`, подобранных эмпирическим путём.

2.3 Исследование результатов работы верификатора

Были проведены следующие эксперименты:

- дополнение датасета, состоящего из реальных данных, синтетическим датасетом в разных пропорциях;
- замена части реальных данных их синтетическими аналогами;
- подтверждение или опровержение гипотезы о дополнении датасета синтетическими данными.

При замене реальных данных синтетическими (90 % реальных данных, 10 % синтетических) наблюдалось консистентное незначительное повышение метрики `Intersection over Union` и более значительное понижение значения функции потерь (`dice_loss`). При дальнейшем уменьшении количества реальных данных точность распознавания начинает падать ниже тех значений, которые наблюдались при 100 % реального датасета и 0 % синтетического.

Значения, выведенные в результате эксперимента с заменой (усреднённые данные трёх запусков программы), изображены на графике на рис. 3:

```
iou = [0.8536, 0.8776, 0.8732, 0.8596, 0.8465, 0.8221, 0.8328,
0.7950, 0.7393, 0.7376, 0.6980]
loss = [0.1923, 0.1660, 0.1734, 0.1886, 0.2009, 0.2353, 0.2209,
0.2711, 0.3637, 0.3674, 0.4445]
synthetic_size = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
0.9, 1.0]
```

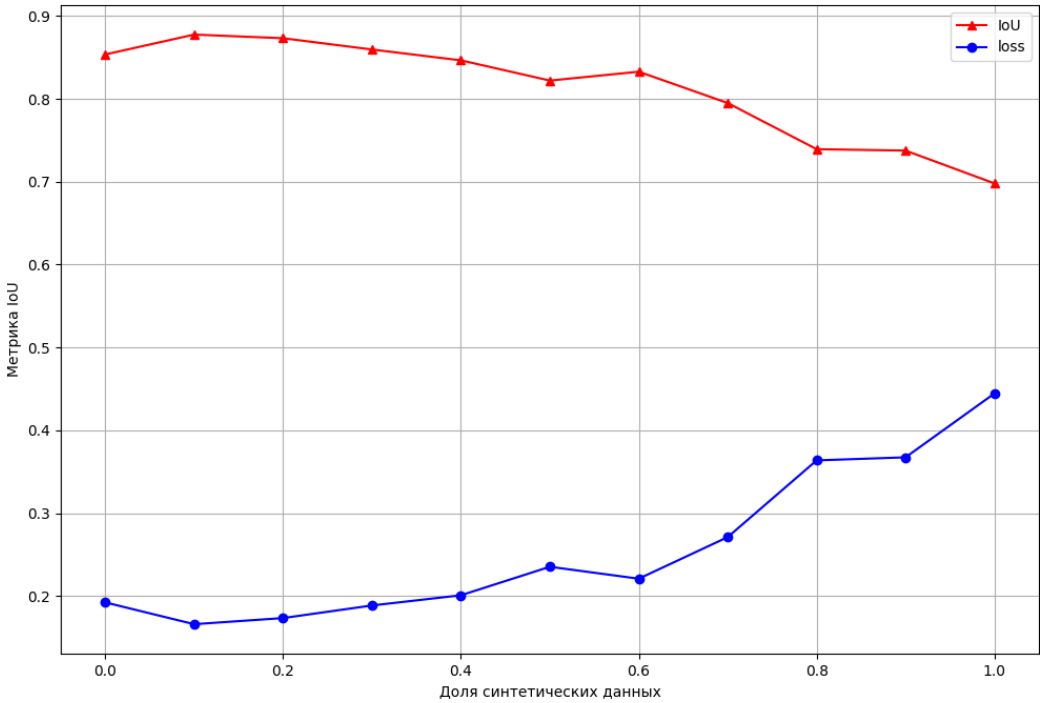

Рисунок 3 – График зависимости IoU от доли синтетических данных в тесте с заменой

График, полученный в ходе эксперимента с дополнением реального датасета синтетическими данными, изображён на рис. 4. При данной архитектуре достигается переобучение, и значительных изменений метрик в среднем не выявлено:

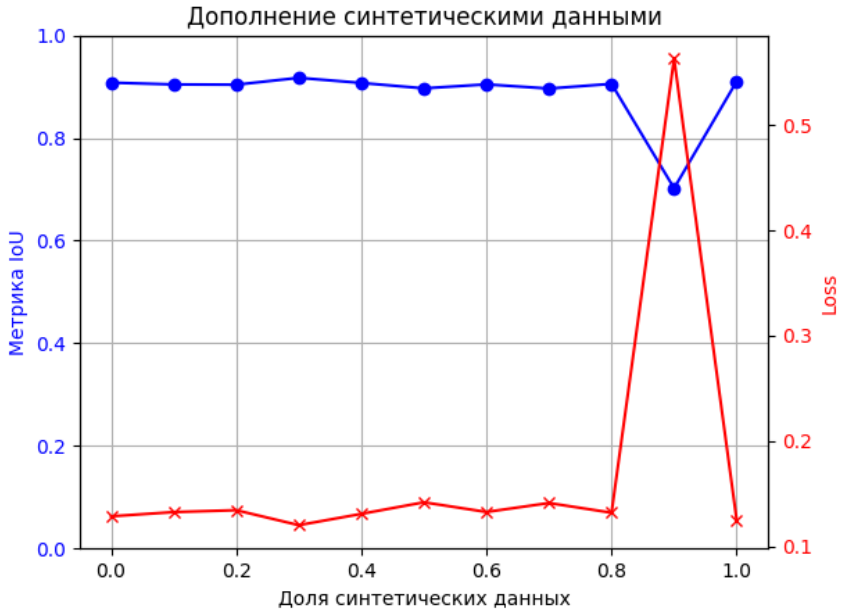

Рисунок 4 – График зависимости IoU от доли синтетических данных в тесте с дополнением

Результаты предсказаний модели на рис. 5 представлены красным цветом, зелёным представлены истинная битмаска.

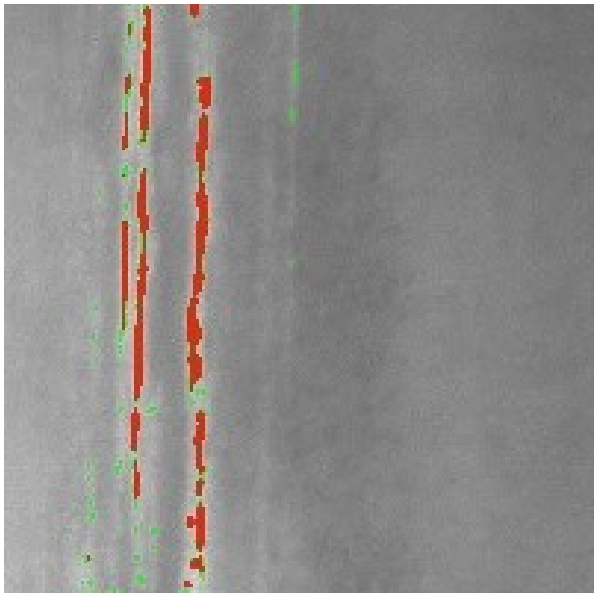

Рисунок 5 – Пример предсказанных дефектов

ЗАКЛЮЧЕНИЕ

В ходе работы выполнены следующие задачи:

- разработан генератор синтетических данных;
- разработан верификатор сгенерированного датасета;
- проведено A/B тестирование с использованием реального датасета и получены данные о достоверности синтетических данных.

Также можно доработать программу, добавив:

- автоматическую настройку параметров генерации для конкретного датасета;
- более точный метод оценки датасетов;
- поддержку распределённых вычислений для одновременного обучения нескольких итераций модели.

Перспективы развития: даже при незначительной настройке генератора пользователем можно добиться небольшого, но устойчивого повышения точности распознавания дефектов. Если существует такой алгоритм, который на вход принимает текущие значения метрик и выводит набор параметров для активных дефектов и окружения, то можно достичь ещё большего роста метрик.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Blender Python API: официальный сайт. – URL: https://docs.blender.org/api/current/index.html, свободный – Загл. с экрана (Дата обращения: 15.05.2025г.)

2) Kaggle | Metal Surface Defects Dataset: официальный сайт. – Датасет реальных фото листов металла. – URL: https://www.kaggle.com/datasets/fantacher/neu-metal-surface-defects-data, свободный – Загл. с экрана (Дата обращения: 15.05.2025г.)

3) Python 3.13.3 Documentation: официальный сайт. – Документация языка Python. – URL: https://docs.python.org/3/index.html, свободный – Загл. с экрана (Дата обращения: 15.05.2025г.)

4) API Documentation | Tensorflow v2.16.1: официальный сайт. – Документация модуля Tensorflow. – URL: https://www.tensorflow.org/api_docs, свободный – Загл. с экрана (Дата обращения: 15.05.2025г.)

5) tensorflow/tensorflow | Docker Image | Docker Hub: официальный сайт. – Образ Docker фреймворка Tensorflow. – URL: https://hub.docker.com/r/tensorflow/tensorflow, свободный – Загл. с экрана (Дата обращения: 15.05.2025г.)

ПРИЛОЖЕНИЕ А

main.py

```
import threading
import tkinter as tk
from tkinter import filedialog, font, scrolledtext
from docker_runner import DockerManager

class VerifierGUI:
    def __init__(self):
        self.root = tk.Tk()
        self.docker_manager = DockerManager()

        tk.Misc.rowconfigure(self.root, [i for i in range(6)],
weight=1)
        tk.Misc.columnconfigure(self.root, [i for i in
range(2)], weight=1)
        label_font = font.Font(family="Verdana", size=12)
        entry_font = font.Font(family="Verdana", size=11)
        button_font = font.Font(family="Verdana", size=15)

        self.root.title("Synth_Verifier")

        self.dataset_real_lbl = tk.Label(self.root,
text="Датасет из реальных данных:", font=label_font)
        self.dataset_real_entry = tk.Entry(self.root,
font=entry_font)
        self.dataset_real_btn = tk.Button(self.root,
text=u"\U0001F4C2", font=button_font,
                                          bg="grey",
command=lambda : self.ask_dir("real"))

        self.dataset_synthetic_lbl = tk.Label(self.root,
text="Датасет для верификации:", font=label_font)
        self.dataset_synthetic_entry = tk.Entry(self.root,
font=entry_font)
        self.dataset_synthetic_btn = tk.Button(self.root,
text=u"\U0001F4C2", font=button_font,
                                               bg="grey",
command=lambda : self.ask_dir("synthetic"))

        self.epoch_count_lbl = tk.Label(self.root,
text="Количество эпох:", font=label_font)
        self.epoch_count_entry = tk.Entry(self.root,
font=entry_font)

        self.batch_count_lbl = tk.Label(self.root, text="Размер
batch:", font=label_font)
        self.batch_count_entry = tk.Entry(self.root,
font=entry_font)
```

```
        self.test_size_lbl = tk.Label(self.root, text="Размер
тестового датасета:", font=label_font)
        self.test_size_entry = tk.Entry(self.root,
font=entry_font)

        self.results_text = scrolledtext.ScrolledText(self.root,
font=label_font)
        self.results_text.config(wrap=tk.WORD, state="disabled",
font="Verdana", width=70, height=20)
        self.start_btn = tk.Button(self.root, text="Начать
верификацию",

font=font.Font(family="Verdana", size=12),
command=self.run_container)

        self.dataset_real_lbl.grid(column=0, row=0, sticky="E",
pady=5, padx=5)
        self.dataset_real_entry.grid(column=1, row=0,
columnspan=3, sticky="W")
        self.dataset_real_btn.grid(column=2, row=0, sticky="W")

        self.dataset_synthetic_lbl.grid(column=0, row=1,
sticky="E", pady=5, padx=5)
        self.dataset_synthetic_entry.grid(column=1, row=1,
columnspan=3, sticky="W")
        self.dataset_synthetic_btn.grid(column=2, row=1,
sticky="W")

        self.epoch_count_lbl.grid(column=0, row=2, sticky="E",
pady=5, padx=5)
        self.epoch_count_entry.grid(column=1, row=2, sticky="W")

        self.batch_count_lbl.grid(column=0, row=3, sticky="E",
pady=5, padx=5)
        self.batch_count_entry.grid(column=1, row=3, sticky="W")

        self.test_size_lbl.grid(column=0, row=4, sticky="E",
pady=5, padx=5)
        self.test_size_entry.grid(column=1, row=4, sticky="W")

        self.results_text.grid(column=3, row=0, padx=10,
pady=10, rowspan=6)
        self.start_btn.grid(column=1, row=5, padx=5, pady=10)

        self.root.mainloop()

    def get_paths(self):
        real_dataset_path = self.dataset_real_entry.get()
        synthetic_dataset_path =
self.dataset_synthetic_entry.get()
        return real_dataset_path, synthetic_dataset_path

    def ask_dir(self, dataset_type):
```

```
        file_path = filedialog.askdirectory()
        if dataset_type =="real":
            self.dataset_real_entry.delete(0, tk.END)
            self.dataset_real_entry.insert(0, file_path)
            return
        self.dataset_synthetic_entry.delete(0, tk.END)
        self.dataset_synthetic_entry.insert(0, file_path)

    def run_container(self):
        real_path, synthetic_path = self.get_paths()
        batch_count, epoch_count = self.batch_count_entry.get(),
self.epoch_count_entry.get()
        test_size = self.test_size_entry.get()

        args = [real_path, synthetic_path, epoch_count,
batch_count, test_size]
        self.update_output("Контейнер запущен...")

        threading.Thread(

target=self.docker_manager.run_script(args_for_model=args,
callback=self.update_output),
            args=("model.py", self.update_output),
            daemon=True).start()

    def update_output(self, text):
        self.results_text.config(state="normal")
        self.results_text.insert(tk.END, "".join(text))
        self.results_text.see(tk.END)
        self.results_text.config(state="disabled")

        self.root.update_idletasks()

gui = VerifierGUI()
```

ПРИЛОЖЕНИЕ Б

docker_runner.py

```
import docker
import os


class DockerManager:
    def __init__(self):
        self.client = docker.from_env()
        self.image_name = "tensorflow-container:latest"
        self.host_dir = os.getcwd()

    def ensure_image_exists(self):
        try:
            self.client.images.get(self.image_name)
        except docker.errors.ImageNotFound:
            print("Building image...")

self.client.images.build(path=".",tag=self.image_name)

    def run_script(self, args_for_model, callback=None):
        command = ["python", f"/tf/model.py"] + args_for_model
        command = " ".join(command)
        # real_path, synthetic_path, epoch_count, batch_count,
test_size
        container = self.client.containers.run(
            image=self.image_name,
            command=command,
            user=f"{os.getuid()}:{os.getgid()}",
            runtime="nvidia",

device_requests=[docker.types.DeviceRequest(count=-1,
capabilities=[['gpu']])],
            volumes={self.host_dir: {'bind': '/tf', 'mode':
'rw'}},
            working_dir="/tf",
            detach=True,
            stdout=True,
            stderr=True
        )

        for line in container.logs(stream=True):
            if callback:
                callback(line.decode('utf-8').strip())

        container.remove()
```

ПРИЛОЖЕНИЕ В

model.py

```
import os
import sys
import time

import keras
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,
UpSampling2D, Concatenate, Dropout
from tensorflow.keras.optimizers import Adam
from keras.src.callbacks import ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.regularizers import l2

import utils


def build_segmentation_model(input_shape=(200, 200, 1),
weight_decay=1e-4):
    inputs = Input(shape=input_shape)
    conv1 = Conv2D(32, (3, 3), activation='relu',
padding='same', kernel_regularizer=l2(weight_decay))(inputs)
    pool1 = MaxPooling2D((2, 2))(conv1)
    conv2 = Conv2D(64, (3, 3), activation='relu',
padding='same',
                   kernel_regularizer=l2(weight_decay))(pool1)
    pool2 = MaxPooling2D((2, 2))(conv2)
    conv3 = Conv2D(128, (3, 3), activation='relu',
padding='same',
                   kernel_regularizer=l2(weight_decay))(pool2)
    conv3 = Dropout(0.3)(conv3)
    up1 = UpSampling2D((2, 2))(conv3)
    concat1 = Concatenate()([up1, conv2])
    conv4 = Conv2D(64, (3, 3), activation='relu',
padding='same',
                   kernel_regularizer=l2(weight_decay))(concat1)
    up2 = UpSampling2D((2, 2))(conv4)
    concat2 = Concatenate()([up2, conv1])
    conv5 = Conv2D(32, (3, 3), activation='relu',
padding='same',
                   kernel_regularizer=l2(weight_decay))(concat2)
    outputs = Conv2D(1, (1, 1), activation='sigmoid')(conv5)
    return Model(inputs=inputs, outputs=outputs)


real_dataset_path, synthetic_dataset_path, epoch_count,
batch_count, test_size = sys.argv[1:]
epoch_count, batch_count, test_size = int(epoch_count),
int(batch_count), int(test_size)
```

```
time_start = time.time()

real_size = 1.0
results_dict = {"iou": [], "loss": [],
                "real_size": [], "synthetic_size": [],
                "real_elements": [], "synthetic_elements": [],
"total_time": -1}
pred_folder_names = []
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=2, min_lr=1e-6)
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

X_synthetic_full, y_synthetic_full =
utils.load_dataset(synthetic_dataset_path)
X_real_full, y_real_full = utils.load_dataset(real_dataset_path)

for i in range(11):
    real_size = round(1 - i/10, 1)
    synthetic_size = round(i/10, 1)

    real_params = X_real_full, y_real_full, real_size
    synthetic_params = X_synthetic_full, y_synthetic_full,
synthetic_size

    X_mixed, y_mixed, X_test, y_test =
utils.get_mixed_data(real_params, synthetic_params, test_size)

    real_elements_num, synthetic_elements_num =
utils.get_number_of_elements(real_params, synthetic_params)
    model = build_segmentation_model()
    model.compile(optimizer=Adam(learning_rate=1e-3),
                  loss=utils.dice_loss,

metrics=[keras.metrics.BinaryIoU(target_class_ids=(0, 1),
threshold=0.5, name=None, dtype=None)])
    history = model.fit(X_mixed, y_mixed,
                        validation_data=(X_test, y_test),
                        epochs=epoch_count,
                        batch_size=batch_count,
                        verbose=1,
                        callbacks=[reduce_lr, early_stopping])

    folder_name = "pred" + str(real_size)
    pred_folder_names.append(folder_name)
    for image_file in os.listdir(os.path.join(real_dataset_path,
"images"))[::10]:
        utils.visualize_predictions(real_dataset_path, model,
image_file, folder_name, threshold=0.7)

    loss, iou = model.evaluate(X_test, y_test, batch_size=8)
    loss, iou = round(loss, 4), round(iou, 4)
```

```
    results_dict["iou"].append(iou)
    results_dict["loss"].append(loss)
    results_dict["real_size"].append(real_size)
    results_dict["synthetic_size"].append(synthetic_size)

results_dict["synthetic_elements"].append(synthetic_elements_num
)
    results_dict["real_elements"].append(real_elements_num)
    print(results_dict)


time_end = time.time()
results_dict["total_time"] = round(time_end - time_start)
print(results_dict)
```

ПРИЛОЖЕНИЕ Г

utils.py

```
import os
import re

import cv2 as cv
import numpy as np
import tensorflow as tf


def dice_coefficient(y_true, y_pred, smooth=1e-6):
    y_true_f = tf.keras.backend.flatten(y_true)
    y_pred_f = tf.keras.backend.flatten(y_pred)
    intersection = tf.keras.backend.sum(y_true_f * y_pred_f)
    return (2. * intersection + smooth) /
(tf.keras.backend.sum(y_true_f) + tf.keras.backend.sum(y_pred_f)
+ smooth)


def dice_loss(y_true, y_pred):
    return 1 - dice_coefficient(y_true, y_pred)


def load_dataset(dataset_path, image_size=(200, 200)):
    images = []
    masks = []
    image_dir = os.path.join(dataset_path, "images")
    mask_dir = os.path.join(dataset_path, "bitmaps")
    for img_file in os.listdir(image_dir):
        img_path = os.path.join(image_dir, img_file)
        image = cv.imread(img_path, cv.IMREAD_GRAYSCALE)
        image = cv.resize(image, image_size)
        image = image.astype(np.float32) / 255.0
        image = np.expand_dims(image, axis=-1)

        base_name = os.path.splitext(img_file)[0]
        mask_path = os.path.join(mask_dir,
f"{base_name.replace('image', 'bitmap')}.jpg")

        if os.path.exists(mask_path):
            mask = cv.imread(mask_path, cv.IMREAD_GRAYSCALE)
            mask = cv.resize(mask, image_size)
            _, mask = cv.threshold(mask, 127, 1,
cv.THRESH_BINARY)
            mask = mask.astype(np.float32)
            mask = np.expand_dims(mask, axis=-1)

            images.append(image)
            masks.append(mask)
```

```
    return np.array(images), np.array(masks)


def visualize_predictions(real_dataset_path, model, image_name,
folder, threshold=0.01):
    index = re.search(r'image(\d+)', image_name).group(1)
    image_name = "image" + index + ".jpg"
    image_path = os.path.join(real_dataset_path, "images/")
    mask_path = os.path.join(real_dataset_path, "bitmaps/")

    img = cv.imread(image_path + image_name,
cv.IMREAD_GRAYSCALE)
    h, w = img.shape[:2]
    img_color = cv.cvtColor(img, cv.COLOR_GRAY2BGR)
    true_mask = cv.imread(mask_path +
image_name.replace("image", "bitmap"), cv.IMREAD_GRAYSCALE)
    true_mask = cv.resize(true_mask, (w, h))
    _, true_mask = cv.threshold(true_mask, 127, 255,
cv.THRESH_BINARY)
    img_input = cv.resize(img, (200, 200))
    img_input = img_input.astype(np.float32) / 255.0
    img_input = np.expand_dims(img_input, axis=(0, -1))
    pred_mask = model.predict(img_input)[0]
    pred_mask = (pred_mask > threshold).astype(np.uint8) * 255
    pred_mask = cv.resize(pred_mask, (w, h))
    _, pred_mask = cv.threshold(pred_mask, 127, 255,
cv.THRESH_BINARY)
    true_mask_color = np.zeros((h, w, 3), dtype=np.uint8)
    pred_mask_color = np.zeros((h, w, 3), dtype=np.uint8)

    true_mask_color[true_mask > 0] = [0, 255, 0]
    pred_mask_color[pred_mask > 0] = [0, 0, 255]

    overlay = img_color.copy()
    overlay[true_mask > 0] = overlay[true_mask > 0] * 0.3 +
true_mask_color[true_mask > 0] * 0.7
    overlay[pred_mask > 0] = overlay[pred_mask > 0] * 0.2 +
pred_mask_color[pred_mask > 0] * 0.8

    filename = os.path.splitext(os.path.basename(image_name))[0]

cv.imwrite(f"{real_dataset_path}predictions/{folder}/{filename}
_overlay.jpg", overlay)


def get_number_of_elements(real_params: tuple, synthetic_params:
tuple):
    X_real_full, _, real_size = real_params
    X_synthetic_full, _, synthetic_size = synthetic_params
    real_elements_num = int(len(X_real_full) * real_size)
    synthetic_elements_num = int(len(X_synthetic_full) *
synthetic_size)
    return real_elements_num, synthetic_elements_num
```

```
def get_mixed_data(real_params: tuple, synthetic_params: tuple,
test_size=60):
    X_real_full, y_real_full, real_size = real_params
    X_synthetic_full, y_synthetic_full, synthetic_size =
synthetic_params

    real_elements_num, synthetic_elements_num =
get_number_of_elements(real_params, synthetic_params)

    test_index_list = np.random.choice(len(X_real_full),
test_size, replace=False)
    X_test = X_real_full[test_index_list]
    y_test = y_real_full[test_index_list]

    real_index_list = np.random.choice(len(X_real_full),
real_elements_num, replace=False)
    X_real, y_real = X_real_full[real_index_list],
y_real_full[real_index_list]

    synthetic_index_list =
np.random.choice(len(X_synthetic_full), synthetic_elements_num,
replace=False)
    X_synthetic, y_synthetic =
X_synthetic_full[synthetic_index_list],
y_synthetic_full[synthetic_index_list]

    X_mixed = np.concatenate((X_real, X_synthetic))
    y_mixed = np.concatenate((y_real, y_synthetic))

    return X_mixed, y_mixed, X_test, y_test
```

ПРИЛОЖЕНИЕ Д

aluminum_gen.py

```
import os
import random
import bpy
from datetime import datetime
import cv2 as cv
import numpy as np
import sys
import importlib

bl_info = {"name": "Data Generator",
    "author": "practice_kubsu_marchuk",
    "version": (1, 0),
    "blender": (3, 6, 2),
    "location": "Output Properties > Data Generator",
    "description": "Генерация синтетических данных для
определения дефектов алюминиевых листов",
    "category": "Output"}


class DATA_GENERATOR_PT_Panel(bpy.types.Panel):
    bl_label = "Data Generator"
    bl_idname = "DATA_GENERATOR_PT_Panel"
    bl_space_type = "PROPERTIES"
    bl_region_type = "WINDOW"
    bl_context = "output"

    def draw(self, context):
        layout = self.layout
        scene = context.scene
        output_props = scene.data_generator_props

        layout.prop(output_props, "defect_02")
        layout.prop(output_props, "defect_02_2")
        layout.prop(output_props, "defect_09")
        layout.prop(output_props, "defect_11")
        layout.prop(output_props, "defect_13")
        layout.prop(output_props, "defect_14")
        layout.prop(output_props, "defect_17")
        layout.prop(output_props, "defect_21")
        layout.prop(output_props, "defect_23")
        layout.prop(output_props, "defect_26")
        layout.prop(output_props, "defect_27")
        layout.prop(output_props, "defect_29")
        layout.prop(output_props, "defect_31")
        layout.prop(output_props, "defect_31_2")
        layout.prop(output_props, "defect_32")
        layout.prop(output_props, "defect_34")
        layout.prop(output_props, "defect_36")
```

```
        layout.prop(output_props, "defect_39")
        layout.prop(output_props, "defect_41")
        layout.prop(output_props, "defect_57")
        layout.prop(output_props, "dataset_size")
        layout.prop(output_props, "max_tries")
        layout.prop(output_props, "output_path")
        layout.prop(output_props, "use_roughness")
        layout.prop(output_props, "use_normal")
        layout.prop(output_props, "use_displace")
        layout.prop(output_props, "find_contours")
        layout.prop(output_props, "find_bound_box")
        layout.operator("object.data_generator", text="Рендер")


class DATA_GENERATOR_Properties(bpy.types.PropertyGroup):
    defect_02: bpy.props.BoolProperty(
        name="02_поверхностный_пузырь_выпуклый", default=False)
    defect_02_2: bpy.props.BoolProperty(
        name="02_поверхностный_пузырь_вогнутый", default=False)
    defect_09: bpy.props.BoolProperty(
        name="09_неметаллические_закаты", default=False)
    defect_11: bpy.props.BoolProperty(
        name="11_закат_металла", default=False)
    defect_13: bpy.props.BoolProperty(
        name="13_коррозия", default=False)
    defect_14: bpy.props.BoolProperty(
        name="14_царапины", default=False)
    defect_17: bpy.props.BoolProperty(
        name="17_насечка", default=False)
    defect_21: bpy.props.BoolProperty(
        name="21_рваная_кромка", default=False)
    defect_23: bpy.props.BoolProperty(
        name="23_загар_масла", default=False)
    defect_26: bpy.props.BoolProperty(
        name="26_оголение_плакирующего_слоя", default=False)
    defect_27: bpy.props.BoolProperty(
        name="27_забоины_вмятины", default=False)
    defect_29: bpy.props.BoolProperty(
        name="29_закатанные_царапины", default=False)
    defect_31: bpy.props.BoolProperty(
        name="31_отпечатки_на_листах_выпуклые", default=False)
    defect_31_2: bpy.props.BoolProperty(
        name="31_отпечатки_на_листах_вогнутые", default=False)
    defect_32: bpy.props.BoolProperty(
        name="32_плены", default=False)
    defect_34: bpy.props.BoolProperty(
        name="34_потертость", default=False)
    defect_36: bpy.props.BoolProperty(
        name="36_заалюминивание", default=False)
    defect_39: bpy.props.BoolProperty(
        name="39_апельсиновая_корка", default=False)
    defect_41: bpy.props.BoolProperty(
        name="41_сетка_линий_скольжения", default=False)
```

```
    defect_57: bpy.props.BoolProperty(
        name="57_следы_СОЖ", default=False)
    dataset_size: bpy.props.IntProperty(
        name="Количество изображений",
        default=1,
        min=0,
        max=10000,
        step=1)
    max_tries: bpy.props.IntProperty(
        name="Количество попыток перерендера",
        default=5,
        min=0,
        max=100,
        step=1,
        description="Если в bitmap дефекта менее 100 не чёрных
пикселей, "
                    "то сменить Randomness и перерендерить
bitmap указанное количество раз")
    output_path: bpy.props.StringProperty(
        name="Вывод",
        subtype="FILE_PATH")
    use_roughness: bpy.props.BoolProperty(
        name="Рендерить шероховатости (Roughness)",
default=False)
    use_normal: bpy.props.BoolProperty(
        name="Рендерить неровности (Normal)", default=False)
    use_displace: bpy.props.BoolProperty(
        name="Рендерить неровности (Displacement)",
default=False)
    find_contours: bpy.props.BoolProperty(
        name="Найти контуры", default=False)
    find_bound_box: bpy.props.BoolProperty(
        name="Найти ограничивающие прямоугольники",
default=False)


class DATA_GENERATOR_OT_RenderOperator(bpy.types.Operator):
    bl_idname = "object.data_generator"
    bl_label = "Render"

    def execute(self, context):
        """ ОПЕРАТОР """

        props = context.scene.data_generator_props

        path = dir_handler(bpy.path.abspath(props.output_path))
        present_list = get_present_list(props)
        transfer_values_to_bitmaps(present_list,
random_only=False)
        inverted_bump_list, bump_list =
get_adder_list(present_list)
        unlink_all("aluminum")
        enable_nodes(present_list, "color_adder")
```

```
        enable_nodes(inverted_bump_list, "inverted_bump_adder")
        enable_nodes(bump_list, "bump_adder")
        enable_effects(props.use_roughness, props.use_normal,
props.use_displace)

        nodes_alum =
bpy.data.materials["aluminum"].node_tree.nodes
        base_node = nodes_alum.get("metal_base")
        render_start = datetime.now()
        for image_idx in range(props.dataset_size):
            base_node.inputs[0].default_value =
round(random.uniform(-100, 100), 3)
            transfer_values_to_bitmaps(present_list,
random_only=True)
            render_bitmaps(path, present_list, image_idx,
props.max_tries, props.find_contours, props.find_bound_box)
            render_aluminum(path, image_idx)
        render_end = datetime.now()
        print(f"Total time: {render_end - render_start}")

        return {"FINISHED"}

def dir_handler(output_path: str) -> str:
    """ Создать папки для вывода отрендеренных изображений и
bitmaps """
    now = datetime.now()
    folder = now.strftime("%H-%M-%S %d.%m.%Y")
    path = os.path.join(output_path, folder)
    os.mkdir(path)
    os.mkdir(os.path.join(path, "images"))
    os.mkdir(os.path.join(path, "bitmaps"))
    return path

def enable_effects(use_roughness: bool, use_normal: bool,
use_displace: bool):
    """ Подсоединить ноды Roughness и Normal map к шейдеру
Principled BSDF """
    links = bpy.data.materials["aluminum"].node_tree.links
    nodes = bpy.data.materials["aluminum"].node_tree.nodes
    principled = nodes["Principled BSDF"]
    if use_roughness:
        links.new(nodes["roughness_bump"].outputs[0],
principled.inputs[2])
    else:
        if nodes["roughness_bump"].outputs[0].is_linked:
links.remove(nodes["roughness_bump"].outputs[0].links[0])

    if use_normal:
        links.new(nodes["roughness_bump"].outputs[1],
principled.inputs[5])
    else:
        if nodes["roughness_bump"].outputs[1].is_linked:
```

```
links.remove(nodes["roughness_bump"].outputs[1].links[0])

    if use_displace:
        links.new(nodes["roughness_bump"].outputs[2],
nodes["Material Output"].inputs[2])
    else:
        if nodes["roughness_bump"].outputs[2].is_linked:

links.remove(nodes["roughness_bump"].outputs[2].links[0])

def get_present_list(props) -> list:
    """ Найти список отмеченных дефектов """

    prop_dict = {"02_поверхностный_пузырь_выпуклый":
props.defect_02,
                 "02_поверхностный_пузырь_вогнутый":
props.defect_02_2,
                 "09_неметаллические_закаты": props.defect_09,
"11_закат_металла": props.defect_11,
                 "13_коррозия": props.defect_13,
                 "14_царапины": props.defect_14, "17_насечка":
props.defect_17, "21_рваная_кромка": props.defect_21,
                 "23_загар_масла": props.defect_23,
"26_оголение_плакирующего_слоя": props.defect_26,
                 "27_забоины": props.defect_27,
"29_закатанные_царапины": props.defect_29,
                 "31_отпечатки_на_листах_выпуклые":
props.defect_31,
                 "31_отпечатки_на_листах_вогнутые":
props.defect_31_2,
                 "32_плены": props.defect_32, "34_потертость":
props.defect_34,
                 "36_заалюминивание": props.defect_36,
"39_апельсиновая_корка": props.defect_39,
                 "41_сетка_линий_скольжения": props.defect_41,
                 "57_следы_СОЖ": props.defect_57}

    present_list = []
    for defect, prop in prop_dict.items():
        if prop:
            present_list.append(defect)
        else:
            continue

    return present_list

def enable_nodes(nodes_to_link: list, adder: str):
    """ Подключить отмеченные ноды """

    nodes = bpy.data.materials["aluminum"].node_tree.nodes
    links = bpy.data.materials["aluminum"].node_tree.links
    adder_node =
bpy.data.materials["aluminum"].node_tree.nodes[adder]
```

```
    for name in nodes_to_link:
        if adder == "color_adder":
            output_socket = 0
        else:
            output_socket = 1
        node = nodes.get(name)
        for socket in range(len(adder_node.inputs)):
            if adder_node.inputs[socket].is_linked:
                continue
            else:
                links.new(node.outputs[output_socket],
adder_node.inputs[socket])
                break

def transfer_values_to_bitmaps(present_list: list,
random_only=True):
    """ Перенести установленные пользователем значения из alum в
bitmaps, all - перенос всех параметров """
    nodes_bit = bpy.data.materials["bitmaps"].node_tree.nodes
    nodes_alum = bpy.data.materials["aluminum"].node_tree.nodes
    for name in present_list:
        node, node_to_copy = nodes_bit.get(name),
nodes_alum.get(name)
        randomize_node(node_to_copy)
        if not random_only:
            end = 1
        else:
            end = len(node.inputs) - 1
        for i in range(0, len(node.inputs) - end):
            node.inputs[i].default_value =
node_to_copy.inputs[i].default_value

def get_adder_list(present_list: list):
    nodes = bpy.data.materials["aluminum"].node_tree.nodes
    inverted_bump_list = []
    for name in present_list:
        node = nodes.get(name)
        if node.inputs[-3].default_value < 0:
            inverted_bump_list.append(name)
    bump_list = [i for i in present_list if i not in
inverted_bump_list]
    return inverted_bump_list, bump_list


def randomize_node(node):
    node.inputs[0].default_value = round(random.uniform(-100,
100), 3)

def unlink_all(mat):
    """ Отключить все ноды от входов adder в материале mat """
    links = bpy.data.materials[mat].node_tree.links
    if mat == "bitmaps":
        adder_list = ["adder"]
```

```
    else:
        adder_list = ["color_adder", "bump_adder",
"inverted_bump_adder"]
    for add in adder_list:
        adder = bpy.data.materials[mat].node_tree.nodes[add]
        for i in range(len(adder.inputs)):
            if adder.inputs[i].is_linked:
                links.remove(adder.inputs[i].links[0])

def render_aluminum(path_to_alum: str, image_indx: int):
    bpy.context.scene.render.filepath =
os.path.join(path_to_alum, "images", ("image%d.jpg" %
image_indx))
    obj = bpy.data.objects["Plane"]
    obj.active_material = bpy.data.materials["aluminum"]
    bpy.ops.render.render(write_still=True)

def render_bitmaps(path: str, present_list: list, image_indx:
int, max_tries: int,
                   find_contours: bool, find_bound_box: bool):
    """ Отрендерить и сохранить bitmaps """

    obj = bpy.data.objects["Plane"]
    obj.active_material = bpy.data.materials["bitmaps"]
    nodes = bpy.data.materials["bitmaps"].node_tree.nodes
    links = bpy.data.materials["bitmaps"].node_tree.links
    adder =
bpy.data.materials["bitmaps"].node_tree.nodes["adder"]
    unlink_all("bitmaps")
    for name in present_list:
        node = nodes.get(name)
        link = links.new(node.outputs[0], adder.inputs[0])
        path_to_bitmap = os.path.join(path, "bitmaps",
("bitmap%d_%s.jpg" % (image_indx, name)))
        path_to_csv = os.path.join(path, "images",
("contours%d_%s.csv" % (image_indx, name)))
        bpy.context.scene.render.filepath = path_to_bitmap
        bpy.ops.render.render(write_still=True)
        limit_blank_defects(path_to_bitmap, node,
max_tries=max_tries)
        links.remove(link)
        if find_contours:
            contours.contours_csv(idx=image_indx,
image_path=path_to_bitmap, path_to_csv=path_to_csv)
        if find_bound_box:
            contours.bound_box_csv(idx=image_indx,
image_path=path_to_bitmap, path_to_csv=path_to_csv)


def limit_blank_defects(path_to_image: str, node, max_tries:
int):
    """Предотвратить сохранение изображений, где дефект не
виден,
```

```
    т.е. перерендерить до max_tries раз """
    tries, whites = 0, 0
    while tries <= max_tries:
        im = cv.imread(path_to_image)
        whites = np.sum(im >= 0)
        if whites >= 100:
            break
        else:
            tries += 1
            randomize_node(node)
            bpy.ops.render.render(write_still=True)


classes = [DATA_GENERATOR_PT_Panel, DATA_GENERATOR_Properties,
    DATA_GENERATOR_OT_RenderOperator]

def register():
    for cls in classes:
        bpy.utils.register_class(cls)

    bpy.types.Scene.data_generator_props =
bpy.props.PointerProperty(type=DATA_GENERATOR_Properties)

def unregister():
    for cls in classes:
        bpy.utils.unregister_class(cls)

    del bpy.types.Scene.data_generator_props

if __name__ == "__main__":
    import_dir = os.path.dirname(bpy.data.filepath)
    if import_dir not in sys.path:
        sys.path.append(import_dir)
    import contours
    importlib.reload(contours)
    register()
```