Цель работы.

Подготовка данных для построения модели нейронной сети распознавания образов.

Задание.

1. Подготовить датасет с дорожными знаками.

1.1. Количество знаков датасета - 10 дорожных знаков на основе "идеальных" векторных изображений. Раздел и коды знаков выдает преподаватель. Разделы "2. Знаки приоритета", "3. Запрещающие знаки", "4. Предписывающие знаки".

1.2. Размер изображения знака 30*30 точек.

1.3. Вносимые искажения (искажение перспективы, фон, шум).

1.3.1. Фон (случайный) из 20 примеров.

1.3.2. Искажение перспективы не более от 10% до 20% - случайная величина (может быть слева или справа).

1.3.3. Шум вида соль и перец до 10%.

2. В отчете привести до 50 примеров полученных изображений (по 5 примеров каждого знака).

Технические требования

1) Объем выборки 1000 примеров (по 100 примеров каждого знака).

2) Последовательность зашумления: искажение перспективы, фон, шум.

Теоретические положения.

Для обучения и тестирования моделей нейронных сетей в задачах распознавания образов требуется репрезентативный и размеченный набор данных. При этом важно, чтобы обучающие изображения учитывали возможные искажения, возникающие в реальных условиях съёмки: изменение перспективы, различный фон, а также наличие шумов.

Использование векторных изображений дорожных знаков в качестве исходных данных позволяет получить «идеальные» изображения без искажений, после чего искусственно смоделировать необходимые искажающие факторы. Такой подход обеспечивает контролируемое формирование датасета и позволяет получить достаточное количество обучающих примеров при ограниченном наборе исходных изображений.

Для корректного наложения знака на фон используется изображение в формате PNG с альфа-каналом. Альфа-канал применяется в качестве маски прозрачности, что позволяет накладывать знак на случайный фон без появления артефактов по краям. Данный способ соответствует рекомендациям технического задания и является стандартным при обработке изображений с прозрачностью.

Разметка датасета выполняется в виде CSV-файла, содержащего соответствие имени изображения и класса дорожного знака. Это необходимо для последующего использования датасета при обучении нейронной сети.

Ход работы

1. Подготовка исходных данных

В качестве исходных изображений использованы 10 векторных изображений дорожных знаков раздела «Запрещающие знаки» (коды 3.1–3.10). Для формирования фона подготовлен набор из 20 фотографий дорожных сцен.

Перед началом генерации выполняется проверка наличия и количества входных данных: 10 SVG-файлов со знаками и 20 фоновых изображений. Это позволяет избежать ошибок при формировании датасета и гарантирует соответствие требованиям задания.

1.1 Конвертация SVG в PNG

На первом этапе выполняется конвертация векторных изображений дорожных знаков из формата SVG в растровый формат PNG с фиксированным размером 30×30 пикселей. Для этого используется библиотека CairoSVG. Размер изображения и наличие альфа-канала дополнительно контролируются программно.

Листинг 1 — Конвертация векторного изображения SVG в растровый формат PNG размером 30×30 пикселей.

```
1. def svg_to_png_30(svg_path, png_path):
2.     cairosvg.svg2png(
3.         url=svg_path,
4.         write_to=png_path,
5.         output_width=30,
6.         output_height=30
7.     )
```

После конвертации изображение считывается и проверяется на соответствие размеру 30×30 пикселей. При отсутствии альфа-канала он добавляется принудительно, что необходимо для дальнейшего наложения знака на фон.

1.2 Искажение перспективы

Для моделирования наклона дорожного знака относительно камеры применяется искажение перспективы. Величина искажения выбирается случайно в диапазоне от 10% до 20% от ширины изображения, направление (влево или вправо) определяется случайным образом.

Листинг 2 — Формирование матрицы перспективного преобразования дорожного знака.

```
strength = random.uniform(0.10, 0.20)
shift = int(30 * strength)

src = np.array([[0, 0], [29, 0], [29, 29], [0, 29]], dtype=np.float32)
dst = np.array([[shift, 0], [29, 0], [29 - shift, 29], [0, 29]], dtype=np.float32)

M = cv2.getPerspectiveTransform(src, dst)
sign_p = cv2.warpPerspective(sign, M, (30, 30))
```

Значение искажения и направление наклона выбираются случайным образом, что позволяет получить разнообразные варианты изображений.

1.3 Наложение знака на фон через маску

После искажения перспективы изображение знака накладывается на случайный фон. Для корректного наложения используется альфа-канал PNG-изображения, который применяется в качестве маски прозрачности. Это позволяет избежать появления артефактов по краям знака.

Листинг 3 — Наложение изображения знака на фон с использованием альфа-канала.

```
sign_bgr = sign_bgra[:, :, :3]
alpha = sign_bgra[:, :, 3] / 255.0

alpha3 = np.dstack([alpha, alpha, alpha])
result = background * (1 - alpha3) + sign_bgr * alpha3
```

Использование маски прозрачности соответствует рекомендациям задания.

1.4 Добавление шума «соль и перец»

На завершающем этапе к изображению добавляется шум типа «соль и перец». Доля зашумлённых пикселей выбирается случайно и не превышает 10%, что соответствует требованиям задания.

Листинг 4 — Добавление шума типа «соль и перец» к изображению.

```
noise_amount = random.uniform(0.0, 0.10)
n = int(30 * 30 * noise_amount)

for i in range(n // 2):
    x, y = random.randint(0, 29), random.randint(0, 29)
    image[y, x] = (255, 255, 255)
```

Количество зашумлённых пикселей не превышает 10%, что соответствует требованиям технического задания.

1.5 Формирование CSV-разметки

Для каждого сгенерированного изображения в CSV-файл сохраняется имя файла и класс дорожного знака. Данная разметка необходима для последующего обучения и тестирования модели нейронной сети.

Листинг 5 — Запись разметки изображения в CSV-файл.

```
writer.writerow(["img_00001.png", "3.1"])
```

1.6 Формирование итогового датасета

После реализации всех этапов обработки изображения выполняется формирование итогового датасета.

Для каждого из 10 дорожных знаков генерируется по 100 изображений, в результате чего общий объём выборки составляет 1000 изображений размером 30×30 пикселей.

Генерация выполняется в цикле, внутри которого последовательно применяются искажение перспективы, наложение на случайный фон и добавление шума.

Для каждого изображения выполняется сохранение файла и запись соответствующей строки в CSV-файл разметки. Дополнительно сохраняются первые 5 изображений каждого знака для использования в отчёте.

Блок-схема алгоритма генерации датасета приведена на рисунке 1.

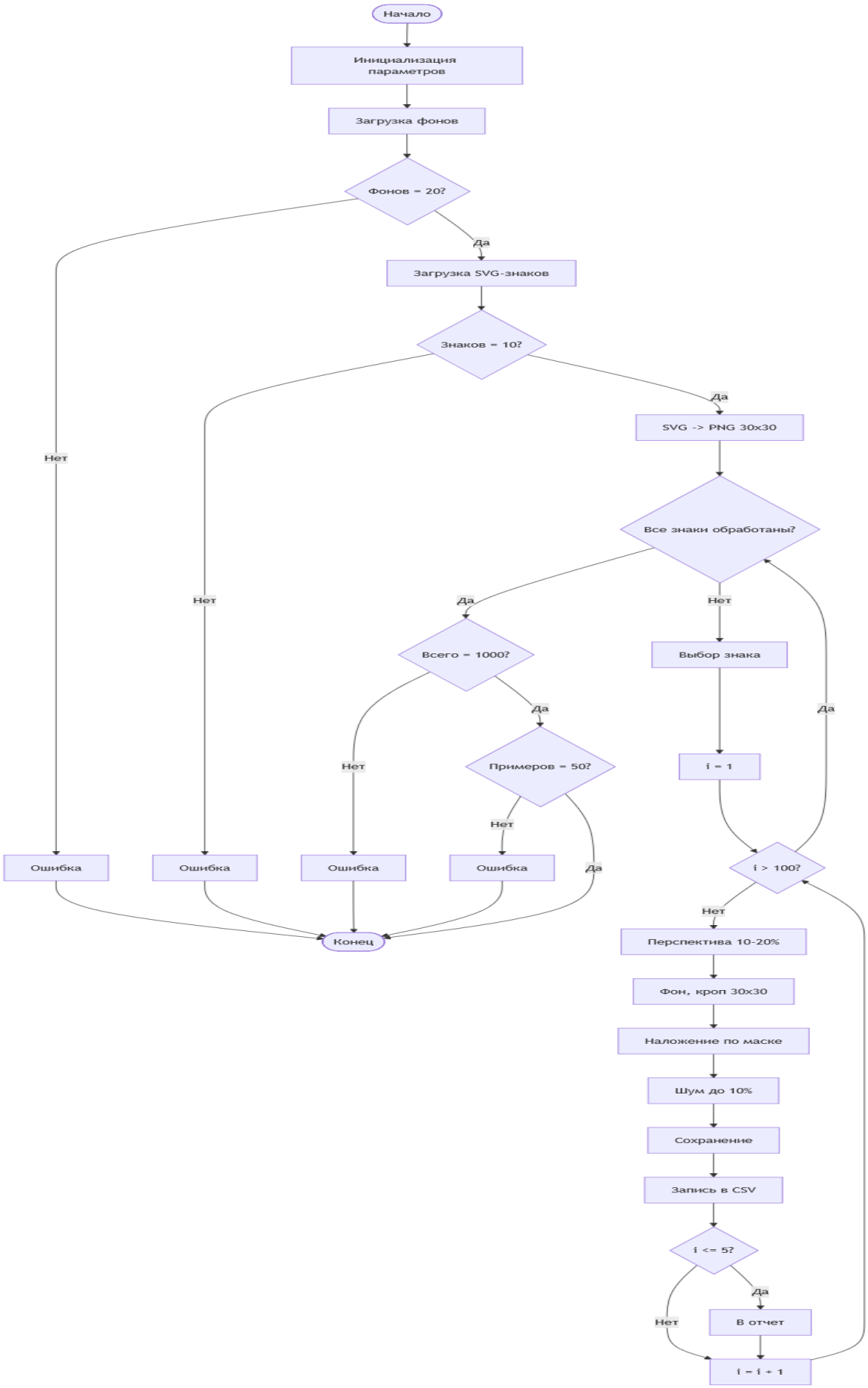

Рисунок 1 — Блок-схема алгоритма генерации датасета дорожных знаков

ВЫВОД

В ходе выполнения индивидуального задания № 2 был разработан программный генератор датасета дорожных знаков на языке Python.
На основе 10 векторных изображений дорожных знаков сформирован набор из 1000 растровых изображений размером 30×30 пикселей с учётом искажений перспективы, случайного фона и шума типа «соль и перец».
Полученный датасет соответствует требованиям технического задания и содержит CSV-разметку, необходимую для обучения и тестирования модели нейронной сети распознавания образов**.**
Таким образом, поставленная цель работы была достигнута.

Приложение 1. Исходный код генератора датасета.

```
# ИДЗ-2 — генератор датасета дорожных знаков
# Мой диапазон: 3.1–3.10 (запрещающие)
#
# По ТЗ:
# 1) SVG -> PNG 30x30 (идеальные знаки)
# 2) Искажения в порядке: перспектива -> фон -> шум
# 3) Перспектива 10..20% (влево/вправо случайно)
# 4) Фон берём случайно из 20 изображений, кроп 30x30
# 5) Наложение PNG на фон через маску (альфа-канал)  <-- важно
# 6) Шум "соль и перец" до 10%
# 7) Датасет: 10 знаков * 100 = 1000 картинок + labels.csv
# 8) Для отчёта: до 50 примеров (по 5 на каждый знак)

import os
import csv
import random

import cv2
import numpy as np
import cairosvg


# ------------------------------------------------------------
# OpenCV не работает с русскими путями.
# Поэтому imdecode/imencode (через байты файла).
# ------------------------------------------------------------


def imread_u(path, flags=cv2.IMREAD_UNCHANGED):
    data = np.fromfile(path, dtype=np.uint8)
    if data.size == 0:
        return None
    return cv2.imdecode(data, flags)


def imwrite_u(path, img):
    ext = os.path.splitext(path)[1].lower()
    ok, buf = cv2.imencode(ext, img)
    if not ok:
        return False
    buf.tofile(path)
    return True


# ------------------------------------------------------------
# Пути (папки рядом с файлом скрипта)
# ------------------------------------------------------------

BASE_DIR = os.path.dirname(os.path.abspath(__file__))

BG_DIR = os.path.join(BASE_DIR, "backgrounds")  # 20 фонов
SIGNS_SVG_DIR = os.path.join(BASE_DIR, "signs", "veselkov")  # 10 svg
SIGNS_PNG_DIR = os.path.join(BASE_DIR, "signs", "veselkov_png")  # 10 png 30x30

OUT_DATASET_DIR = os.path.join(BASE_DIR, "dataset")
OUT_REPORT_DIR = os.path.join(BASE_DIR, "report_samples")
OUT_CSV = os.path.join(OUT_DATASET_DIR, "labels.csv")
```

```
58.
59.
60. # ------------------------------------------------------------
61. # Параметры по ТЗ
62. # ------------------------------------------------------------
63.
64. REQUIRED_BACKGROUNDS = 20
65. REQUIRED_SIGNS = 10
66.
67. FINAL_W, FINAL_H = 30, 30
68.
69. PERSPECTIVE_MIN = 0.10
70. PERSPECTIVE_MAX = 0.20
71.
72. SP_NOISE_MAX = 0.10  # до 10%
73. SAMPLES_PER_SIGN = 100
74. REPORT_PER_SIGN = 5
75.
76. BG_EXTS = (".jpg", ".jpeg", ".png", ".bmp")
77.
78.
79. # ------------------------------------------------------------
80. # Вспомогательные функции
81. # ------------------------------------------------------------
82.
83.
84. def get_backgrounds():
85.     """Собираю фоны и проверяю, что их ровно 20 (из ТЗ)."""
86.     if not os.path.isdir(BG_DIR):
87.         raise RuntimeError(f"Нет папки backgrounds: {BG_DIR}")
88.
89.     bgs = [f for f in os.listdir(BG_DIR) if f.lower().endswith(BG_EXTS)]
90.     bgs.sort()
91.
92.     if len(bgs) != REQUIRED_BACKGROUNDS:
93.         raise RuntimeError(
94.             f"Нужно ровно {REQUIRED_BACKGROUNDS} фонов, найдено: {len(bgs)}"
95.         )
96.
97.     return [os.path.join(BG_DIR, f) for f in bgs]
98.
99.
100. def get_svgs():
101.     """Собираю svg-знаки и проверяю, что их ровно 10 (по ТЗ)."""
102.     if not os.path.isdir(SIGNS_SVG_DIR):
103.         raise RuntimeError(f"Нет папки signs/veselkov: {SIGNS_SVG_DIR}")
104.
105.     svgs = [f for f in os.listdir(SIGNS_SVG_DIR) if f.lower().endswith(".svg")]
106.     svgs.sort()
107.
108.     if len(svgs) != REQUIRED_SIGNS:
109.         raise RuntimeError(
110.             f"Нужно ровно {REQUIRED_SIGNS} SVG знаков, найдено: {len(svgs)}"
111.         )
112.
113.     return [os.path.join(SIGNS_SVG_DIR, f) for f in svgs]
114.
115.
116. def svg_to_png_30(svg_path, png_path):
117.     """Конвертирую SVG -> PNG 30x30 (идеальный знак)."""
```

```
    os.makedirs(os.path.dirname(png_path), exist_ok=True)
    cairosvg.svg2png(
        url=svg_path, write_to=png_path, output_width=FINAL_W, output_height=FINAL_H
    )


def prepare_signs_png():
    """
    Перегенерирую PNG каждый запуск — так проще.
    Заодно контролирую 30x30 и наличие альфы.
    """
    svgs = get_svgs()
    os.makedirs(SIGNS_PNG_DIR, exist_ok=True)

    print(f"[INFO] SVG: {len(svgs)} шт. Конвертирую в PNG 30x30...")

    pngs = []
    for svg in svgs:
        name = os.path.splitext(os.path.basename(svg))[0]
        out_png = os.path.join(SIGNS_PNG_DIR, f"{name}.png")

        svg_to_png_30(svg, out_png)

        img = imread_u(out_png, cv2.IMREAD_UNCHANGED)  # ожидаю BGRA
        if img is None:
            raise RuntimeError(f"Не удалось прочитать PNG: {out_png}")

        if img.shape[:2] != (FINAL_H, FINAL_W):
            raise RuntimeError(
                f"PNG не 30x30: {out_png} -> {img.shape[1]}x{img.shape[0]}"
            )

        # если вдруг альфы нет — добавляю (на всякий)
        if img.ndim == 3 and img.shape[2] == 3:
            alpha = np.full((FINAL_H, FINAL_W, 1), 255, dtype=np.uint8)
            img = np.concatenate([img, alpha], axis=2)
            if not imwrite_u(out_png, img):
                raise RuntimeError(f"Не удалось записать PNG: {out_png}")

        if img.shape[2] != 4:
            raise RuntimeError(
                f"Ожидаю BGRA (4 канала), но получил {img.shape[2]}: {out_png}"
            )

        pngs.append(out_png)
        print(f"[OK] {os.path.basename(svg)} -> {os.path.basename(out_png)}")

    return pngs


def random_bg_crop_30(bg_path):
    """Беру случайный кроп 30x30 из фона. Если фон маленький — ресайз до 30x30."""
    bg = imread_u(bg_path, cv2.IMREAD_COLOR)  # BGR
    if bg is None:
        raise RuntimeError(f"Не удалось прочитать фон: {bg_path}")

    h, w = bg.shape[:2]
    if h < FINAL_H or w < FINAL_W:
        return cv2.resize(bg, (FINAL_W, FINAL_H), interpolation=cv2.INTER_AREA)

```

```
    x = random.randint(0, w - FINAL_W)
    y = random.randint(0, h - FINAL_H)
    return bg[y : y + FINAL_H, x : x + FINAL_W].copy()


def apply_perspective(sign_bgra):
    """
    Искажение перспективы: делаю сдвиг вершин на 10..20% по ширине.
    Направление выбираю случайно (влево/вправо).
    """
    h, w = sign_bgra.shape[:2]

    strength = random.uniform(PERSPECTIVE_MIN, PERSPECTIVE_MAX)
    shift = int(round(w * strength))
    shift = max(1, min(shift, w - 1))

    right = random.choice([True, False])

    src = np.array([[0, 0], [w - 1, 0], [w - 1, h - 1], [0, h - 1]], dtype=np.float32)

    if right:
        dst = np.array(
            [[shift, 0], [w - 1, 0], [w - 1 - shift, h - 1], [0, h - 1]],
            dtype=np.float32,
        )
    else:
        dst = np.array(
            [[0, 0], [w - 1 - shift, 0], [w - 1, h - 1], [shift, h - 1]],
            dtype=np.float32,
        )

    M = cv2.getPerspectiveTransform(src, dst)

    # borderValue=(0,0,0,0) — чтобы края оставались прозрачными
    warped = cv2.warpPerspective(
        sign_bgra,
        M,
        (w, h),
        flags=cv2.INTER_LINEAR,
        borderMode=cv2.BORDER_CONSTANT,
        borderValue=(0, 0, 0, 0),
    )
    return warped


def overlay_with_alpha(bg_bgr, sign_bgra):
    """
    Наложение по альфа-каналу (маска).
    PNG с прозрачностью кладем на фон через маску.
    """
    sign_bgr = sign_bgra[:, :, :3].astype(np.float32)
    alpha = sign_bgra[:, :, 3].astype(np.float32) / 255.0

    bg = bg_bgr.astype(np.float32)
    alpha3 = np.dstack([alpha, alpha, alpha])

    out = bg * (1.0 - alpha3) + sign_bgr * alpha3
    return np.clip(out, 0, 255).astype(np.uint8)


```

```python
def add_salt_pepper(img_bgr, amount):
    """Шум 'соль-перец' (amount = доля пикселей). По до 10%."""
    if amount <= 0:
        return img_bgr

    h, w = img_bgr.shape[:2]
    n = int(round(h * w * amount))
    if n <= 0:
        return img_bgr

    out = img_bgr.copy()
    n_salt = n // 2
    n_pepper = n - n_salt

    ys = np.random.randint(0, h, size=n_salt)
    xs = np.random.randint(0, w, size=n_salt)
    out[ys, xs] = (255, 255, 255)

    ys = np.random.randint(0, h, size=n_pepper)
    xs = np.random.randint(0, w, size=n_pepper)
    out[ys, xs] = (0, 0, 0)

    return out


# ---------------------------------------------------------
# MAIN
# ---------------------------------------------------------


def main():
    backgrounds = get_backgrounds()
    print(f"[OK] Фоны: {len(backgrounds)} шт.")

    signs = prepare_signs_png()
    print(f"[OK] Знаки PNG: {len(signs)} шт.")

    os.makedirs(OUT_DATASET_DIR, exist_ok=True)
    os.makedirs(OUT_REPORT_DIR, exist_ok=True)

    img_index = 1
    report_index = 1

    with open(OUT_CSV, "w", newline="", encoding="utf-8") as f:
        wr = csv.writer(f)
        wr.writerow(["filename", "class"])

        print("[INFO] Генерация датасета 30x30...")

        for sign_path in signs:
            cls = os.path.splitext(os.path.basename(sign_path))[0]

            sign0 = imread_u(sign_path, cv2.IMREAD_UNCHANGED)  # BGRA
            if sign0 is None:
                raise RuntimeError(f"Не удалось прочитать знак: {sign_path}")

            for i in range(1, SAMPLES_PER_SIGN + 1):
                # 1) перспектива
                sign_p = apply_perspective(sign0)

```

```
298.                # 2) фон (случайный кроп 30x30)
299.                bg = random_bg_crop_30(random.choice(backgrounds))
300.
301.                # 3) наложение через маску (альфа)
302.                merged = overlay_with_alpha(bg, sign_p)
303.
304.                # 4) шум до 10% (случайно)
305.                noise_amount = random.uniform(0.0, SP_NOISE_MAX)
306.                merged = add_salt_pepper(merged, noise_amount)
307.
308.                # сохраняю картинку и строку в CSV
309.                fname = f"img_{img_index:05d}.png"
310.                out_path = os.path.join(OUT_DATASET_DIR, fname)
311.
312.                if not imwrite_u(out_path, merged):
313.                    raise RuntimeError(f"Не удалось записать изображение: {out_path}")
314.
315.                wr.writerow([fname, cls])
316.
317.                # первые 5 на каждый знак — в папку отчёта (в сумме 50)
318.                if i <= REPORT_PER_SIGN:
319.                    rep_name = f"sample_{report_index:02d}_{cls}.png"
320.                    rep_path = os.path.join(OUT_REPORT_DIR, rep_name)
321.                    if not imwrite_u(rep_path, merged):
322.                        raise RuntimeError(
323.                            f"Не удалось записать пример для отчёта: {rep_path}"
324.                        )
325.                    report_index += 1
326.
327.                img_index += 1
328.
329.            print(f"[OK] {cls}: {SAMPLES_PER_SIGN} шт.")
330.
331.    # контроль по количеству изображений
332.    total = img_index - 1
333.    if total != REQUIRED_SIGNS * SAMPLES_PER_SIGN:
334.        raise RuntimeError(
335.            f"Ошибка: dataset {total}, ожидалось {REQUIRED_SIGNS * SAMPLES_PER_SIGN}"
336.        )
337.
338.    rep_total = report_index - 1
339.    if rep_total != REQUIRED_SIGNS * REPORT_PER_SIGN:
340.        raise RuntimeError(
341.            f"Ошибка: report_samples {rep_total}, ожидалось {REQUIRED_SIGNS *
REPORT_PER_SIGN}"
342.        )
343.
344.    print(f"[DONE] dataset: {total} изображений -> dataset/")
345.    print(f"[DONE] report_samples: {rep_total} изображений -> report_samples/")
346.    print(f"[DONE] labels.csv -> dataset/labels.csv")
347.    print("Всё готово!")
348.
349.
350. if __name__ == "__main__":
351.    main()
352.
```

Приложение 2. Примеры полученных изображений

(50 изображений по 5 примеров каждого знака).