Санкт-Петербургский политехнический университет Петра Великого

Институт металлургии, машиностроения и транспорта

Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Объектно-ориентированное программирование
Тема: Модуль аугментации датасетов

Выполнил студент гр. 3331506/10401 Макаров Г.В.

Преподаватель Ананьевский М.С.

«___»_____ 2024 г.

Санкт-Петербург

2024

Введение

Аугментация датасетов для задач детекции объектов — это процесс увеличения размера и разнообразия существующего набора данных путем добавления новых модифицированных изображений и соответствующих им аннотаций. Это важный шаг в задачах обучении моделей машинного обучения в компьютерном зрении, поскольку он помогает улучшить точность и надежность результатов.

Одна из основных проблем, которую решает аугментация, — это малое количество изначальных данных. В реальных бизнес-задачах зачастую инженеры машинного обучения сталкиваются с весьма ограниченным датасетом, расширить который за счет интернет-ресурсов не представляет возможным. За счёт аугментации можно увеличить имеющиеся данные кратно (теоретически - бесконечно).

Еще одна проблема, которую решает аугментация, — это улучшение обобщающей способности моделей. Аугментация позволяет модели лучше адаптироваться к новым данным, которые могут отличаться от тех, на которых она обучалась. Это особенно важно при работе с изображениями, где даже небольшие изменения, такие как поворот или изменение яркости, могут существенно изменить внешний вид объекта. Например, поворот изображения полезен для аэрофотоснимков, где нет привязки к ориентации камеры, а яркость – для задач требующих всепогодную детекцию.

Для решения данной задачи было решено создать модуль на языке программирования Python, реализация всех аугментаций и остального функционала которого будут использовать только библиотеки NumPy, Pillow.

Ссылка на репозиторий - https://gitverse.ru/makarov/augmentor.

Ход работы

Существует несколько форматов хранения датасетов для задач детекции объектов. Самые распространенные из них – COCO, CVAT и YOLO. В рамках задачи будет реализован последний формат, как самый популярный и активный в сообществе компьютерного зрения. Информацию о нём можно найти по ссылке https://docs.ultralytics.com/ru/datasets/detect/.

Реализуем интерфейс AugmentorBase, который будет наследован различными классами под специфичные форматы в дальнейшем. В нём потребуется функционал логгера, аугментации (вызываемой функции для пайплайна и приватной функции для аугментации отдельных сабсетов), сканирования датасетов и сабсетов, а также чтения и записи изображений и аннотаций. Определенные методы будут абстрактными – их реализацию нужно будет проводить уже в дочерних классах. Код **base.py**:

```
from abc import ABC, abstractmethod
import logging
import sys
import os

import numpy as np
from PIL import Image

import augmentor


class AugmentorBase(ABC):
    LOGGING_FORMAT = "%(asctime)s %(levelname)s - %(message)s"
    FLOAT_PRECISION = 4

    def __init__(self, debug: bool = False, log_path: str = None):
        level = logging.INFO if not debug else logging.DEBUG
        if log_path is None:
            logging.basicConfig(stream=sys.stdout,
                                format=self.LOGGING_FORMAT,
                                datefmt='%H:%M:%S',
                                level=level)
        elif log_path.endswith(".txt"):
            Checker.check_log(log_path)
            logging.basicConfig(filename=log_path,
                                filemode="a",
                                format=self.LOGGING_FORMAT,
                                datefmt='%H:%M:%S',
```

```
                    level=level)
        else:
            raise ValueError(f"Log path should be with .txt extension")

        logging.debug("Augmentor class object initialized")

        self.load_path = None
        self.save_path = None

        self.image = None
        self.bboxes = None

    def __del__(self):
        logging.debug("Augmentor class object deleted")
        logging.shutdown()

    def augment(self, *args, **kwargs):
        """
        Augment dataset by modifying images and labels in selected subsets with given augmentation pipeline in arguments
        using *load_path* and *save_path* paths, *train*, *val* and *test* flags from keyword arguments. In the process,
        the original dataset gets copied to *save_path* and augmented images and bounding boxes get saved along with
        unique filenames - *original-name_augmentation+id.original-extension*
        :param args: each augmentation in the pipeline should be passed as 2 consequent arguments - augmentation name
                and corresponding value (e.g. *"rotate", 90* to rotate images by angle 90)
        :param kwargs: *load_path* to directory with dataset for augmentation, *save_path* to nonexistent directory
                    where augmented dataset will be created and saved, *train*, *val* and *test* flags to indicate
                    which subsets are to be augmented
        """
        self.load_path = kwargs.get("load_path", "")
        self.save_path = kwargs.get("save_path", "")

        Checker.check_paths(self.load_path, self.save_path)

        train = kwargs.get("train", True)
        val = kwargs.get("val", True)
        test = kwargs.get("test", True)

        if not train and not val and not test:
            raise ValueError("No subset chosen for augmentation")

        augmentations = {args[i]: [] for i in range(0, len(args), 2)}
        for i in range(0, len(args), 2):
            Checker.check_args(args[i], args[i + 1])
            augmentations[args[i]].append(args[i + 1])

        logging.info(f"{self.load_path.rsplit('/')[-1]} dataset chosen for augmentation")
        logging.debug(f"Augmentations: {augmentations}")
```

```
    os.system(f"cp -r {self.load_path} {self.save_path}")
    os.system(f"find {self.save_path} -name '*.cache' -type f -delete")

    image_paths, label_paths = self.scan(self.load_path, train, val, test)

    logging.info(f"A total amount of "
                 f"{len(image_paths['train']) + len(image_paths['val']) + len(image_paths['test'])} "
                 f"images will be augmented")

    if train:
        self._augment_subset(augmentations, image_paths["train"], label_paths["train"], "train")

    if val:
        self._augment_subset(augmentations, image_paths["val"], label_paths["val"], "val")

    if test:
        self._augment_subset(augmentations, image_paths["test"], label_paths["test"], "test")

    logging.info(f"{self.load_path.rsplit('/')[-1]} dataset has been augmented")

def _augment_subset(self, augmentations: dict, image_paths: list, label_paths: list, subset: str):
    logging.info(f"Augmenting {augmentor.utils.subset_lower(subset)} subset...")

    for augmentation in augmentations:
        count = 0

        for value in augmentations[augmentation]:
            for image_path, label_path in zip(image_paths, label_paths):
                self._image_read(image_path)
                self._bboxes_read(label_path)

                if augmentation in augmentor.augmentations.GEOMETRIC_AUGMENTATIONS:
                    self.image, self.bboxes = (getattr(augmentor.augmentations.geometric, f"{augmentation}")
                                               (self.image, self.bboxes, value))
                elif augmentation in augmentor.augmentations.PHOTOMETRIC_AUGMENTATIONS:
                    self.image = (getattr(augmentor.augmentations.photometric, f"{augmentation}")
                                  (self.image, value))

                if self.image is not None and self.bboxes is not None:
                    load_image_path = image_path.replace(self.load_path, self.save_path)
                    load_label_path = label_path.replace(self.load_path, self.save_path)

                    self._image_write(f"{load_image_path.rsplit('.', 1)[0]}_{augmentation}"
                                      f"{count}.{load_image_path.rsplit('.', 1)[1]}")
                    self._bboxes_write(f"{load_label_path.rsplit('.', 1)[0]}_{augmentation}"
                                       f"{count}.{load_label_path.rsplit('.', 1)[1]}")

            count += 1

@classmethod
@abstractmethod
def scan(cls, path: str, train: bool, val: bool, test: bool) -> tuple[dict, dict]:
    pass
```

```
    @classmethod
    @abstractmethod
    def _scan_subset(cls, path: str, subset: str) -> tuple[list, list]:
        pass

    def _image_read(self, image_path: str):
        logging.debug(f"Reading image {image_path}")

        self.image = np.array(Image.open(image_path).convert("RGB"))

    def _image_write(self, image_path: str):
        logging.debug(f"Writing image {image_path}")

        Image.fromarray(self.image, "RGB").save(image_path)

    @abstractmethod
    def _bboxes_read(self, label_path: str):
        pass

    @abstractmethod
    def _bboxes_write(self, label_path: str):
        pass


class Checker(ABC):
    ALLOWED_AUGMENTATIONS = ("rotate", "flip", "resize", "crop",
                             "saturation", "brightness", "contrast", "blur", "noise")

    @staticmethod
    def check_log(log_path: str):
        if not os.path.isabs(log_path):
            raise ValueError(f"{log_path} is not an absolute path")

        if os.path.exists(log_path):
            raise FileExistsError(f"{log_path} already exists")

    @classmethod
    def check_args(cls, augmentation: str, value: int or float or tuple[int, int, int, int]):
        logging.debug(f"Checking argument {value}...")

        if augmentation not in cls.ALLOWED_AUGMENTATIONS:
            raise ValueError(f"Unknown augmentation {augmentation}")

        if augmentation in (cls.ALLOWED_AUGMENTATIONS[0], cls.ALLOWED_AUGMENTATIONS[1]):
            if type(value) is not int:
                raise TypeError(f"For {augmentation} augmentation value should be integer")

            if augmentation is cls.ALLOWED_AUGMENTATIONS[0] and value not in (-270, -180, -90, 90, 180, 270):
                raise ValueError(f"Cannot rotate by angle {value}")

            if augmentation is cls.ALLOWED_AUGMENTATIONS[1] and value not in (-1, 0, 1):
                raise ValueError(f"Cannot flip by axis {value}")
```

```
        elif augmentation in (cls.ALLOWED_AUGMENTATIONS[2], cls.ALLOWED_AUGMENTATIONS[8]):
            if type(value) is not float:
                raise TypeError(f"For {augmentation} augmentation value should be float")

            if augmentation is cls.ALLOWED_AUGMENTATIONS[2] and 0. >= value > 2.:
                raise ValueError(f"For resize augmentation ratio should be between 0.0 and 2.0")

            if augmentation in (cls.ALLOWED_AUGMENTATIONS[4], cls.ALLOWED_AUGMENTATIONS[5],
                                cls.ALLOWED_AUGMENTATIONS[6]) and 0. >= value > 2.:
                raise ValueError(f"For {augmentation} augmentation factor should be between 0.0 and 2.0")

            if augmentation in (cls.ALLOWED_AUGMENTATIONS[7], cls.ALLOWED_AUGMENTATIONS[8]) and 0. >=
value > 2.:
                raise ValueError(f"For {augmentation} augmentation sigma should be between 0.0 and 2.0")

        elif augmentation is cls.ALLOWED_AUGMENTATIONS[3]:
            if type(value) is not tuple or len(value) != 4 or not all(type(coord) is int for coord in value):
                raise TypeError("For crop augmentation value should be a tuple of 4 integers")

            if (augmentation is cls.ALLOWED_AUGMENTATIONS[3] and
                    value[0] < 0 or value[1] < 0 or value[2] <= 0 or value[3] <= 0):
                raise ValueError(f"Cannot crop with coordinates {value}")

    @staticmethod
    def check_paths(load_path: str, save_path: str):
        logging.debug(f"Checking load path {load_path} and save path {save_path}...")

        if load_path == "":
            raise ValueError("Keyword argument load_path is required")
        if save_path == "":
            raise ValueError("Keyword argument save_path is required")

        if not os.path.isdir(load_path):
            raise NotADirectoryError(f"{load_path} directory is nonexistent")
        if os.path.isdir(save_path):
            raise IsADirectoryError(f"{save_path} directory already exists")

    @staticmethod
    def check_dif(dif: int, subset_lower: str):
        logging.debug("Checking difference between images and labels...")

        if dif == 1:
            logging.warning(f"{subset_lower} label is missing")
        elif dif > 1:
            logging.warning(f"{dif} {subset_lower} labels are missing")
        elif dif == -1:
            logging.warning(f"Redundant {subset_lower} label has been found")
        elif dif < -1:
            logging.warning(f"{dif} redundant {subset_lower} labels have been found")

    @staticmethod
    def check_shape(shape: [int, int, int], coordinates: tuple[int, int, int, int]):
```

```
    height, width, _ = shape

    if coordinates[0] + coordinates[2] > width:
        raise ValueError("Crop zone exceeds processing image width")

    if coordinates[1] + coordinates[3] > height:
        raise ValueError("Crop zone exceeds processing image height")
```

Также для пользователей модуля могут понадобится некоторые утилиты – маленькие функции, расширяющие функционал библиотеки. В рамках данной задачи в **utils.py** хранятся функции форматирования названия сабсетов и функция, рисующая аннотации поверх изображений для дебага:

```
import numpy as np
from PIL import Image, ImageDraw


def subset_upper(subset: str) -> str:
    """
    Take a subset shortened name and return a subset name that starts with uppercase letter. Used for log formatting
    :param subset: *train*, *val* or *test*
    :return: *Training*, *Validation* or *Testing*
    """
    return "Training" if subset == "train" else "Validation" if subset == "val" else "Testing"


def subset_lower(subset: str) -> str:
    """
    Take a subset shortened name and return a subset name with all letters in lowercase. Used for log formatting
    :param subset: *train*, *val* or *test*
    :return: *training*, *validation* or *testing*
    """
    return "training" if subset == "train" else "validation" if subset == "val" else "testing"


def draw_bboxes(image: np.ndarray, bboxes: list) -> np.ndarray:
    """
    Draw bounding boxes on the image for debugging purposes. The image is processed as NumPy array, so additional
    transition to PIL might be needed to see the results. The bounding boxes outline is of red color with pixel width 2
    :param image: image as NumPy array
    :param bboxes: bounding boxes list of (x, y, w, h, class)
    :return: image with drawn bounding boxes on it
    """
    drawn_image = Image.fromarray(image)
    draw = ImageDraw.Draw(drawn_image)

    for bbox in bboxes:
```

```
            x, y, w, h, _class = bbox
            draw.rectangle((x, y, x + w, y + h), outline="red", width=2)

        return np.array(drawn_image)
```

Для чтения датасетов формата YOLO делаем соответствующий файл **yolo.py**, где унаследуем интерфейс и определим его абстрактные методы:

```
import logging
import os

from augmentor.core.base import AugmentorBase, Checker
import augmentor.core.utils as utils


class YOLO(AugmentorBase):
    ALLOWED_IMAGE_FORMATS = (".bmp", ".jpeg", ".jpg", ".png", ".tif", ".tiff", ".webp")
    ALLOWED_LABEL_FORMATS = (".txt",)

    @classmethod
    def scan(cls, path: str, train: bool, val: bool, test: bool) -> tuple[dict, dict]:
        """
        Scan selected subsets in passed *path* of dataset directory depending on the *train*, *val* and *test* flags
        :param path: path to dataset directory
        :param train: whether to scan training subset or not
        :param val: whether to scan validation subset or not
        :param test: whether to scan testing subset or not
        :return: *image_paths* and *label_paths* dictionaries with training, validation or/and testing subsets'
                 corresponding scanned paths
        """
        image_paths = {}
        label_paths = {}

        if train:
            image_paths["train"], label_paths["train"] = cls._scan_subset(path, "train")

        if val:
            image_paths["val"], label_paths["val"] = cls._scan_subset(path, "val")

        if test:
            image_paths["test"], label_paths["test"] = cls._scan_subset(path, "test")

        logging.debug(f"{path} dataset has been scanned")

        return image_paths, label_paths

    @classmethod
    def _scan_subset(cls, path: str, subset: str) -> tuple[list, list]:
        if not (os.path.isdir(path + "/" + subset + "/images") or os.path.isdir(path + "/" + subset + "/labels")):
            raise NotADirectoryError(f"{path}/{subset} is nonexistent or is not a directory")

        images = [path + "/" + subset + "/images/" + image for image in os.listdir(path + "/" + subset + "/images")
```

```
            if image.endswith(cls.ALLOWED_IMAGE_FORMATS)]
        images.sort()

        labels = [path + "/" + subset + "/labels/" + label for label in os.listdir(path + "/" + subset + "/labels")
            if label.endswith(cls.ALLOWED_LABEL_FORMATS)]
        labels.sort()

        difference = len(images) - len(labels)
        Checker.check_dif(difference, utils.subset_lower(subset))

        images_split = [image.rsplit("/", 1)[-1].rsplit(".", 1)[0] for image in images]
        labels_split = [label.rsplit("/", 1)[-1].rsplit(".", 1)[0] for label in labels]
        labels = [labels[labels_split.index(image_split)] if image_split in labels_split
            else None for image_split in images_split]

        logging.info(f"{utils.subset_upper(subset)} subset scanned: "
                f"{len(images)} images | {len(labels) - difference} labels")

        return images, labels

    def _bboxes_read(self, label_path: str):
        logging.debug(f"Reading bounding bboxes from {label_path}")

        bboxes = list()
        height, width, _ = self.image.shape

        if label_path is None:
            self.bboxes = None
        else:
            with open(label_path, "r") as file:
                for line in file:
                    class_, xc, yc, w, h = [int(line.rsplit()[0])] + [float(x) for x in line.rsplit()[1:]]

                    if class_ < 0:
                        logging.warning(f"{label_path} contains bounding box with negative class")

                    if xc < 0 or yc < 0 or w < 0 or h < 0:
                        raise ValueError(f"{label_path} contains bounding box with negative coordinates")

                    w = round(w * width, self.FLOAT_PRECISION)
                    h = round(h * height, self.FLOAT_PRECISION)
                    x = round(xc * width - w / 2, self.FLOAT_PRECISION)
                    y = round(yc * height - h / 2, self.FLOAT_PRECISION)

                    bboxes.append([x, y, w, h, class_])

            self.bboxes = bboxes

    def _bboxes_write(self, label_path: str):
        logging.debug(f"Writing bounding boxes in {label_path}")

        height, width, _ = self.image.shape
```

```
    with open(label_path, "w") as file:
        for bbox in self.bboxes:
            x, y, w, h, class_ = bbox

            w = round(w / width, self.FLOAT_PRECISION)
            h = round(h / height, self.FLOAT_PRECISION)
            xc = round(x / width + w / 2, self.FLOAT_PRECISION)
            yc = round(y / height + h / 2, self.FLOAT_PRECISION)

            file.write(f"{class_} {xc} {yc} {w} {h}\n")
```

Наконец, реализуем аугментации. Они включают в себя 2 вида модифицирования изображений и аннотаций – геометрические и фотометрические. Первые меняют геометрию изображений и требуют редактирования аннотаций (поворот, изменение размера и т.д.), а вторые, соответственно, работают только с фотографией (изменение яркости, зашумление и т.д.). Код **geometric.py**:

```
import numpy as np
from PIL import Image

from augmentor.core.base import Checker


def rotate(image: np.ndarray, bboxes: list, angle: int) -> tuple[np.ndarray, list]:
    """
    Rotate image and bounding boxes by *-270*, *-180*, *-90*, *90*, *180* or *270* degrees counter-clockwise.
If angle
    is *180* or *-180* the image is just flipped along both x and y axes instead
    :param image: image as NumPy array
    :param bboxes: bounding boxes list of (x, y, w, h, class)
    :param angle: angle to rotate by
    :return: rotated image and bounding boxes list
    """
    if angle == 180 or angle == -180:
        return flip(image, bboxes, -1)

    rotated_image = np.rot90(image, k=1)
    rotated_bboxes = [[y, image.shape[1] - x - w, h, w, class_] for [x, y, w, h, class_] in bboxes]

    if angle == 270 or angle == -90:
        return flip(rotated_image, rotated_bboxes, 1)

    return rotated_image, rotated_bboxes


def flip(image: np.ndarray, bboxes: list, axis: int) -> tuple[np.ndarray, list]:
    """
```

```
    Flip image and bounding boxes along given axis. *-1* stands for flipping along both x and y axes, *0* stands
for
    flipping along x-axis, *1* stands for flipping along y-axis
    :param image: image as NumPy array
    :param bboxes: bounding boxes list of (x, y, w, h, class)
    :param axis: axis to flip along
    :return: flipped image and bounding boxes list
    """
    height, width, _ = image.shape

    if axis == 0:
        flipped_image = np.flip(image, 0)
        flipped_bboxes = [[x, height - y - h, w, h, class_] for [x, y, w, h, class_] in bboxes]

    elif axis == 1:
        flipped_image = np.flip(image, 1)
        flipped_bboxes = [[width - x - w, y, w, h, class_] for [x, y, w, h, class_] in bboxes]

    else:
        flipped_image = np.flip(image, (0, 1))
        flipped_bboxes = [[width - x - w, height - y - h, w, h, class_] for [x, y, w, h, class_] in bboxes]

    return flipped_image, flipped_bboxes


def resize(image: np.ndarray, bboxes: list, ratio: float) -> tuple[np.ndarray, list]:
    """
    Resize the image and bounding boxes to the given ratio between 0.0 and 2.0
    :param image: image as NumPy array
    :param bboxes: bounding boxes list of (x, y, w, h, class)
    :param ratio: resize ratio
    :return: resized image and bounding boxes list
    """
    height, width, _ = image.shape

    resized_image = np.array(Image.fromarray(image).resize((int(width * ratio), int(height * ratio))))
    resized_bboxes = [[ratio * x, ratio * y, ratio * w, ratio * h, class_] for [x, y, w, h, class_] in bboxes]

    return resized_image, resized_bboxes


def crop(image: np.ndarray, bboxes: list, coordinates: tuple[int, int, int, int]) -> tuple[np.ndarray, list]:
    """
    Crop the image and bounding boxes using given (x, y, w, h) coordinates. *x*, *y* stand for the top-left corner
    coordinates of the cropped zone, whereas *w*, *h* stand for its width and height. Bounding boxes get into
the
    resulting list if they are within the cropped zone at least partially. Each crop augmentation is followed by an
    image shape checkout through *Checker* abstract class
    :param image: image as NumPy array
    :param bboxes: bounding boxes list of (x, y, w, h, class)
    :param coordinates: cropped zone coordinates
    :return: cropped image and bounding boxes list
    """
```

```
    Checker.check_shape(image.shape, coordinates)

    x_crop, y_crop, w_crop, h_crop = coordinates

    cropped_image = image[y_crop:y_crop + h_crop, x_crop:x_crop + w_crop]
    cropped_bboxes = list()

    for bbox in bboxes:
        x, y, w, h, class_ = bbox

        if x >= x_crop and x + w <= x_crop + w_crop:
            bbox[0] -= x_crop

        if x < x_crop and x + w <= x_crop + w_crop:
            bbox[0] = 0
            bbox[2] += x - x_crop

        if x >= x_crop and x + w > x_crop + w_crop:
            bbox[0] -= x_crop
            bbox[2] = x_crop + w_crop - x

        if y >= y_crop and y + h <= y_crop + h_crop:
            bbox[1] -= y_crop

        if y < y_crop and y + h <= y_crop + h_crop:
            bbox[1] = 0
            bbox[3] += y - y_crop

        if y >= y_crop and y + h > y_crop + h_crop:
            bbox[1] -= y_crop
            bbox[3] = y_crop + h_crop - y

        if (bbox[0] < 0 or bbox[1] < 0 or bbox[2] < 0 or bbox[3] < 0
                or bbox[0] + bbox[2] > w_crop or bbox[1] + bbox[3] > h_crop):
            continue

        cropped_bboxes.append(bbox)

    return cropped_image, cropped_bboxes
```

Код **photometric.py**:

```
import numpy as np
from PIL import Image, ImageEnhance


def saturation(image: np.ndarray, factor: float) -> np.ndarray:
    """
    Adjust image saturation using a ImageEnhance PIL submodule. Factor should be between 0.0 and 2.0. *0.0*
will return
    black and white image, *1.0* will leave it unchanged and *2.0* will increase its saturation to a noticeable
extent
    :param image: image as NumPy array
```

```
    :param factor: saturation factor
    :return: image with adjusted saturation
    """
    return np.array(ImageEnhance.Color(Image.fromarray(image)).enhance(factor))


def contrast(image: np.ndarray, factor: float) -> np.ndarray:
    """
    Adjust image contrat using a ImageEnhance PIL submodule. Factor should be between 0.0 and 2.0. *0.0* will make the
    image solid grey, *1.0* will leave it unchanged and *2.0* will increase its contrast to a noticeable extent
    :param image: image as NumPy array
    :param factor: contrast factor
    :return: image with adjusted contrast
    """
    return np.array(ImageEnhance.Contrast(Image.fromarray(image)).enhance(factor))


def brightness(image: np.ndarray, factor: float) -> np.ndarray:
    """
    Adjust image brightness using a ImageEnhance PIL submodule. Factor should be between 0.0 and 2.0. *0.0* will return
    blackened image, *1.0* will leave it unchanged and *2.0* will increase its brightness to a noticeable extent
    :param image: image as NumPy array
    :param factor: saturation factor
    :return: image with adjusted saturation
    """
    return np.array(ImageEnhance.Brightness(Image.fromarray(image)).enhance(factor))


def blur(image: np.ndarray, sigma: float) -> np.ndarray:
    """
    Blur image using Gaussian blur through x and y along-axes convolutions. The sigma parameter is used to adjust the
    strength of the blurring, it should be between 0.0 and 2.0. The kernel is automatically calculated with size 3
    :param image: image as NumPy array
    :param sigma: standard deviation of the Gaussian distribution
    :return: image with applied Gaussian blur
    """
    kernel = np.exp(-0.5 * (np.arange(-(3 // 2), 3 // 2 + 1) / sigma) ** 2)
    kernel = kernel / np.sum(kernel)

    blurred_image = image.astype(np.float32) / 255.
    blurred_image = np.apply_along_axis(lambda x: np.convolve(x, kernel, mode='same'), 0, blurred_image)
    blurred_image = np.apply_along_axis(lambda y: np.convolve(y, kernel, mode='same'), 1, blurred_image)
    blurred_image = (blurred_image * 255.).astype(np.uint8)

    return blurred_image


def noise(image: np.ndarray, sigma: float) -> np.ndarray:
    """
```

```
    Add Gaussian noise to the image. The sigma parameter is used to adjust the strength of the noising, it should be
    between 0.0 and 2.0. The mean grey value in the probability density function is equal to 0 during calculation
    :param image: image as NumPy array
    :param sigma: standard deviation of the Gaussian distribution
    :return: image with applied Gaussian noise
    """
    gaussian_noise = np.random.normal(0, sigma, image.shape)

    noised_image = image.astype(np.float32) / 255.
    noised_image = np.clip(noised_image + gaussian_noise, 0., 1.)
    noised_image = (noised_image * 255.).astype(np.uint8)

    return noised_image
```

На этом основная часть модуля готова, и остаётся только оформить его в виде устанавливаемого пакета. Для этого используем соответствующую структуру проекта и **pyproject.toml**, который позволит устанавливать и импортировать наш модуль в дальнейшем. Окончательный результат можно увидеть во вложении или в репозитории. Также был оформлен README.md, выбрана лицензия CC BY-NC-SA 4.0 и написан docstring для всех публичных функций и методов модуля.

Примеры аугментаций можно увидеть на рисунках 2 и 3:

Рисунок 1 - Оригинальное изображение

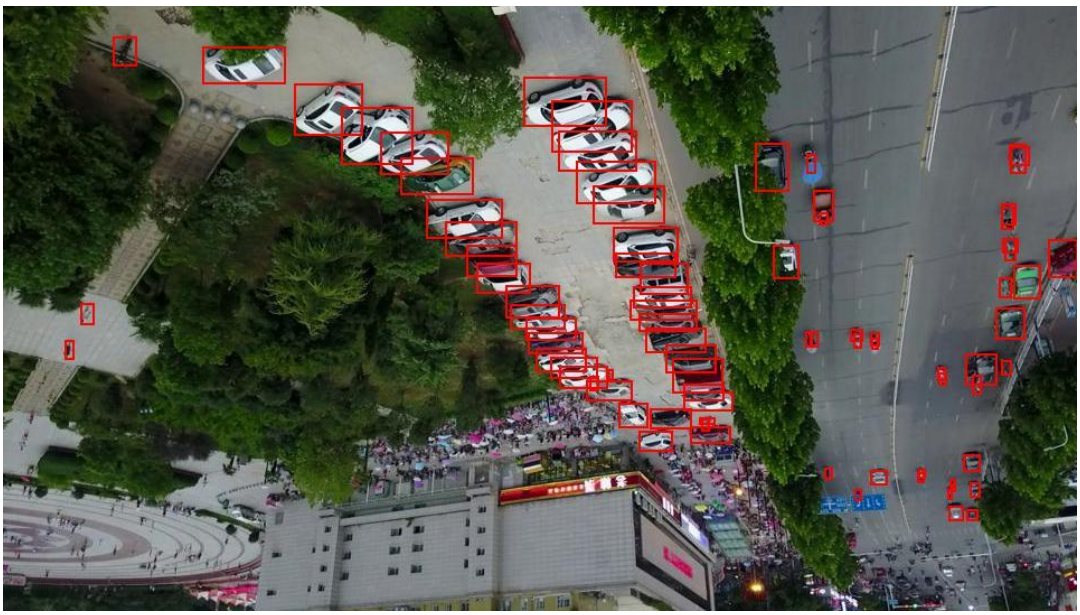

Рисунок 2 – Геометрическая аугментация (поворот вокруг осей x и y)

Рисунок 3 – Фотометрическая аугментация (шум Гаусса)

Заключение

В результате работы был успешно написан модуль, который позволяет решать задачи, связанные с аугментацией датасетов для задач детекции объектов. Модуль был протестирован и показал высокую эффективность в работе. Он может быть использован специалистами компьютерного зрения для расширения малых датасетов в качестве stand-alone проекта. При желании, модуль может быть легко интегрирован в существующие системы и приложения.

В перспективе для модуля требуется написать поддержку форматов датасетов COCO и CVAT for image. Также планируется написать CLI и, возможно, минималистичный GUI. В будущем может быть интегрирована поддержка ориентированных аннотаций (повернутых на определенный угол ограничивающих рамок).