Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

ОТЁТ ПО ПРЕДДИПЛОМНОЙ ПРАКТИКЕ

Студент _____
(Фамилия, Имя, Отчество)

Группа _____

Тип практики _____

Название предприятия МГТУ им. Н.Э. Баумана

Студент _____ _____ _____
(Группа) (Подпись, дата) (И.О. Фамилия)

Рекомендуемая оценка _____

Руководитель практики _____ _____
(Подпись, дата) (И.О. Фамилия)

Оценка _____

2024 г.

АННОТАЦИЯ

Темой данной работы является «Разработка программы анализа гиперспектральных изображений с помощью осцилляторных нейронных сетей». Работа состоит из **??** страниц и содержит **??** разделов, **??** листингов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данной работы – разработка программы для сегментации целевых объектов на гиперспектральном изображении с помощью осцилляторных нейронных сетей. Необходимо разработать стратегию выбора спектральных каналов для улучшения работы программы. Также необходимо реализовать возможность использования различных модулей и их комбинаций для обработки изображений. В конечном итоге необхождимо протестировать разработанную программу.

1 Постановка задачи

Программа должна получать на вход набор флагов, по которым будет определяться дальнейшее поведение программы. Базовый вариант использование будет подразумевать использование предзагруженного датасета гиперспектральных изобравжений и работы с ним. Цель программы состоит в сегментации обхекта определенного класса на гиперспектральном изображдении. Из предоставленного изображения случайным образом выбирается его часть, которая в дальнейшем будет обработана. Программа должна вывести маску сегментированного изображения, изначальное изображение, их пересечение и результат по одной из метрик, которые можно будет выбрать с помощью флагов.

2 Характеристика предприятия

Производственная практика проходила с 13 мая по 27 мая 2024 года в МГТУ им. Н.Э. Баумана. МГТУ им. Н.Э. Баумана является одним из ведущих технических вузов страны. Университет ведёт подготовку инженеров с 1830 года. На сегодняшний день в университете обучают по большому количеству направлений. Выпускники МГТУ становятся очень востребованными у работодателей на рынке труда. Ключевой особенностью обучения является сочетание теоретических и практических знаний. Университет занимается научными исследованиями и разработками. Университет стал одним из единственных в России, которому агентство «Эксперт РА» присвоило рейтинговый класс «B», означающий «очень высокий» уровень подготовки выпускников. Ежегодно университет входит в топ-300 мирового рейтинга QS, и входит в пятерку лучших среди российских университетов. Кафедра осуществляет подготовку дипломированных специалистов с квалификацией «Математик, системный программист» по программе высшего профессионального образования по специальности «Прикладная математика и информатика» со специализацией «Системное программирование». Подготовка специалистов осуществляется по математическому и программному обеспечению высокотехнологичных областей техники и современных информационных технологий с уклоном в высокоэффективное программирование, предусматривающее углублённые знания алгоритмов и приёмов решения задач по прикладной математике и программированию.

3 Реализация программы

В данном разделе приведена реализация программы, выполняющей анализ гиперспектральных изображений с помощью осцилляторной нейронной сети.

В качестве языка программирования был выбран язык программирования Python. Данный выбор обусловлен существованием большого множества подключаемых библиотек, поддержкой ООП, а также чрезвычайной простотой особенностей языка.

3.1 Структура программы

Программа разрабатывалась в репозитории с поддержкой системы контроля версий git.

Исходный код программы расположен в папке по пути `source/`. Функция main точки входа в программу расположена в файле `main.py`. В файле `dataset_loader.py` и `datasets.py` расположены, соответсвенно, загрузчик различных датасетов и класс самого датасета, реализующий различные функции взаимодействия с данными. В файлах `onn_model.py`, `GaborGradScale.py` и `stimul.py` расположены классы, отвечающие за логику работы различных частей модели осцилляторной нейронной сети. В файле `oscillators.py` описаны классы, реализующие логику работы составных элементов сети. В файле `pipelines.py` содержится класс для удобного управления процессом инициализации и работы нейронной сети. В файле `hyperparams_opt.py` содержатся функции и классы для проведения подбора гиперпараметров модели. В файле `utils.py` расположены вспомогательные функции и классы.

В папке по пути `tests/` расположены юнит-тесты программы.

В папке по пути `hyperparams/` расположены json файлы конфигурации параметров модели.

3.2 Параметры запуска программы

Для поддержки параметров запуска программы и их парсинга использовалась библиотека argparse [1]. Выбор этой библиотеки объяняется ее простотой и предоставляемым функционалом.

Программа поддерживает следующие параметры запуска:

— `dataset_name` (строковый тип) – устанавливает используемый датасет. При отсутсвии выбранного датасета в директории программы производится его загрузка;
— `hyperparams_path` (строковый тип) – путь до файла с гиперпараметрами;
— `optimize_before_run` (булевый тип) – запуск программы с оптимизацией гиперпараметров;
— `metric` (строковый тип) – метрика, по которой будет оценено качество сегментации;
— `segm_filedir` (строковый тип) – путь до директории с выходным файлом с результатами сегментации, без имя файла и расширения.

3.3 Конвейер программы

Общий конвейер программы находится в файле `pipelines.py` и описывается классом `OnnHyperPipeline` (см. Листинг 1). В этом классе расположены все методы для работы с данными и с моделями. Четыре основных метода это `add_dataset` (см. Листинг 1), `create_samples` (см. Листинг 1), `run` (см. Листинг 2) и `eval` (см. Листинг 3).

Метод `add_dataset` выполняет вызов метода, отвечающего за загрузку датасета в классе `dataset_loader`. Метод `create_samples` выполняет разбиение полученного гиперспектрального изображения из датасета посредством вызова метода класса `HyperSpectralData` и далее вспомогательной функции `build_dataset`. Метод `run` выполняет полную подготовку к запуску модели (вызов методов `add_dataset` и `create_samples`), а также осуществляет загрузку гиперпараметров из указанного в аргументах запуска файла и выполняет вызов функции `hyperparams_opt` для оптимизации гиперпараметров. Метод `eval` отвечает за обработку и сохранение результатов сегментации модели. Он вызывает вспомогательную функцию `evaluate_best_segmentation`.

Листинг 1 — Класс конвейера и некоторые методы

```
class OnnHyperPipeline(HyperPipeline):
"""Pipeline class for implementing ONN workflow"""

    def __init__(self):
        super().__init__()
        self.dataset: HyperSpectralData
        self.model: OnnModel
        self.result: list[Result]
        self.params: Params

    def add_dataset(self, dataset_name='PaviaU',
                            load_folder="./datasets"):
        data, gt, labels, ignored_labels, \
        rgb_bands, palette, num_of_bands = get_dataset(dataset_name,
                                                       load_folder)
        params = {'labels': labels,
                'ignored_labels': ignored_labels,
                'rbg_bands': rgb_bands,
                'palette': palette,
                'num_of_bands': num_of_bands}
        self.dataset = HyperSpectralData(data, gt, **params)

    def get_data_info(self):"
        self.dataset.get_data_info()
        self.dataset.view_data()

    def specify_target_class(self, target_class: str):
        self.dataset.specify_target_class(target_class)

    def create_samples(self,
                    num_samples:int = 3,
                    sample_height:int = 100,
                    sample_width:int = 100,
                    threshold:int = 100):
        self.dataset.create_samples(num_samples=num_samples,
                                    sample_height=sample_height,
                                    sample_width=sample_width,
                                    threshold=threshold)

    def select_chanels(self, method = "expert", num_of_bands = 4):
        """method which define strategy of selection of spectral bands"""
        self.dataset.select_chanels(method, num_of_bands)

    def add_model(self, model):
        self.model = model

```

Листинг 2 — Метод run для полной настройки и запуска модели с помощью конвейера

```
def run(self,
        target_class: str,
        params_path: str = "",
        optimize_before_run:bool = False):
    """method to fully setup and run onn models"""
    self.specify_target_class(target_class)

    if len(params_path) != 0 and os.path.exists(params_path):
        file_path = params_path
    else:
        file_path = check_default_json_file()
    self.params = Params(file_path)

    self.select_chanels(self.params.band_sel_method,
                        self.params.num_of_bands)
    self.create_samples(num_samples=self.params.num_samples,
                        sample_height=self.params.sample_height,
                        sample_width=self.params.sample_width,
                        threshold=self.params.threshold)

    if optimize_before_run:
        optimize_hyperparams(self)
        self.params.update(file_path)

    self.model.run(
        self.dataset,
        po_num=self.params.po_num,
        params_sel_method=self.params.params_sel_method,
        stimuls_num=self.params.stimuls_num,
        stimuls_sel_method=self.params.stimuls_sel_method,
        find_cont_method=self.params.find_cont_method,
        osc_params=[self.params.dict[f"stimul_{i}"]
                    for i in range(self.params.stimuls_num)],
        draw_contours=self.params.draw_contours,
        level_value=self.params.level_value,
        k_size=self.params.k_size,
        max_number_of_iters=self.params.max_number_of_iters,
        alpha=self.params.alpha,
        beta=self.params.beta,
        w1=self.params.w1,
        w4=self.params.w4,
        threshold=self.params.threshold,

        ↪  cont_area_threshold_percent=self.params.cont_area_threshold_percent
```

Листинг 3 — Метод `eval` для обработки и сохранения результатов

```
def eval(self, metric:str = "iou"):
    """evaluate and save the results on model segmantation"""
    samples_result = evaluate_best_segmentation(
                                    self.model.segmented_samples,
                                    self.dataset.samples,
                                    self.dataset.target_class_id,
                                    metric)
    self.result = samples_result
```

3.4 Загрузка и обработка датасета

За загрузку датасета отвечается функция `get_dataset`. В данной функции описаны различные параметры датасетов и реализована их загрузка в корректном формате. Для уменьшения объема будут приведен только некоторые из реализованных обработчиков параметров датасета (см. Листинг 4, 5).

Класс датасета `HyperSpectralData` в своих методах не содержит программной логики (за исключением метода `select_chanels`), а непосредственно вызывает вспомогательные функции из файла `utils.py`. При этом в атрибутах класса содержится вся необходимая информация о данных (см. Листинг 6)

Листинг 4 — Некоторые параметры данных для загрузки

```
DATASETS_CONFIG = {
    'PaviaC': {
        'urls': ['http://www.ehu.eus/ccwintco/uploads/e/e3/Pavia.mat',
                 'http://www.ehu.eus/ccwintco/uploads/5/53/Pavia_gt.mat'],
        'img': 'Pavia.mat',
        'gt': 'Pavia_gt.mat'
        },
    'PaviaU': {
        'urls': ['http://www.ehu.eus/ccwintco/uploads/e/ee/PaviaU.mat',
                'http://www.ehu.eus/ccwintco/uploads/5/50/PaviaU_gt.mat'],
        'img': 'PaviaU.mat',
        'gt': 'PaviaU_gt.mat'
        }}
```

Листинг 5 — Основная часть функции загрузчика

```
def get_dataset(dataset_name,
                target_folder,
                datasets=DATASETS_CONFIG):
    """ Gets the dataset specified by name and return the related
        components.
    """
    palette = None
    if dataset_name not in datasets.keys():
        raise ValueError("{} dataset is unknown.".format(dataset_name))
    dataset = datasets[dataset_name]
    folder = target_folder + datasets[dataset_name].get('folder',
                                                        dataset_name+'/')
    if dataset.get('download', True):
        # Download the dataset if is not present
        if not os.path.isdir(folder):
            os.mkdir(folder)
        for url in datasets[dataset_name]['urls']:
            # download the files
            filename = url.split('/')[-1]
            if not os.path.exists(folder + filename):
                with TqdmUpTo(unit='B', unit_scale=True, miniters=1,
                              desc="Downloading {}".format(filename)) as t:
                    urlretrieve(url, filename=folder + filename,
                                reporthook=t.update_to)
    elif not os.path.isdir(folder):
    print("WARNING: {} is not downloadable.".format(dataset_name))

    if dataset_name == 'PaviaC':
        # Load the image
        img = open_file(folder + 'Pavia.mat')['pavia']
        rgb_bands = (55, 41, 12)
        gt = open_file(folder + 'Pavia_gt.mat')['pavia_gt']
        label_values = ["Undefined", "Water", "Trees", "Asphalt",
                        "Self-Blocking Bricks", "Bitumen", "Tiles",
                        "Shadows", "Meadows", "Bare Soil"]
        ignored_labels = [0]
    # Filter NaN out
    nan_mask = np.isnan(img.sum(axis=-1))
    img[nan_mask] = 0
    gt[nan_mask] = 0
    ignored_labels.append(0)
    ignored_labels = list(set(ignored_labels))
    num_of_bands = img.shape[2]
    # Normalization
    img = np.asarray(img, dtype='float32')
    return np.array([img]), gt, label_values, ignored_labels, \
           rgb_bands, palette, num_of_bands
```

Листинг 6 — Класс данных модели

```python
class HyperSpectralData():
    """Class for storing hyp data for simple ONN"""

    def __init__(self, data, gt, **params):
        """
        Args:
            data: array of 3D hyperspectral image (may be 1-elem array)
            gt: 2D array of labels
            data_augmentation: bool, set to True to perform random flips
            supervision: 'full' or 'un' supervised algorithms
        """
        self.data: np.ndarray = data
        self.gt: np.ndarray = gt
        self.labels_to_ignore = params['ignored_labels']
        self.labels = params['labels']
        self.rgb = params['rbg_bands']
        self.num_bands = params['num_of_bands']
        self.samples: list[Sample]
        self.samples_labels: np.ndarray
        self.selected_bands: np.ndarray
        self.target_class: str
        self.target_class_id: int

    def select_chanels(self, method = "expert", n = 4):
        """method which define strategy of selection of spectral bands"""
        if method == "expert":
            self.selected_bands = [55, 41, 12]
        if method == "simple_opt":
            sel_bands = select_best_spectrums(
                img = self.data[0],
                complete_gt = self.gt,
                target_class_id = self.target_class_id,
                n = n
            )
            self.selected_bands = sel_bands
        if method == "advanced_opts":
            self.selected_bands = [55, 41, 12]
```

3.5 Модель

Класс модели и метод `run` (см. Листинг 7, 8) построен таким образом, чтобы можно было использовать различные модули, как составные части модели.

Листинг 7 — Основная часть основной функции run модели

```
if "SelectiveAtt" not in self.modules.keys():
    raise KeyError("Module SelectiveAtt are not in model modules list")

for sample in dataset.samples:
    area_of_interest_mask = self.modules["SelectiveAtt"].run(
                                img=sample.band_img, po_num=po_num,
                                params_sel_method=params_sel_method,
                                stimuls_num=stimuls_num,
                                stimuls_sel_method=stimuls_sel_method,
                                osc_params=osc_params,
                                target_brightness=sample.target_brightness)

    if self.modules["SelectiveAtt"].att_method == "separate":
        iterate_bands_over = area_of_interest_mask

    if self.modules["SelectiveAtt"].att_method == "intersect":
        iterate_bands_over = sample.band_img

    segmented_on_bands = {}
    for i, spectral_band_mask in enumerate(iterate_bands_over):
        area_of_interest = deepcopy(sample.band_img[i])
        if self.modules["SelectiveAtt"].att_method == "separate":
            # verify that only current stimul represented in
            ↪  area_of_interest brightness values
            area_of_interest[area_of_interest != spectral_band_mask] = -1

        if self.modules["SelectiveAtt"].att_method == "intersect":
            # verify that only current stimul represented in
            ↪  area_of_interest brightness values
            area_of_interest[area_of_interest != area_of_interest_mask] =
            ↪  -1
```

Листинг 8 — Основная часть основной функции run модели (продолжение)

```
        # crop the area to minimuze further calculations
        area_of_interest = area_of_interest[~np.all(area_of_interest ==
        ↪  -1, axis=1)]
        area_of_interest =
        ↪  area_of_interest[:,~(area_of_interest==-1).all(0)]
        try:
            contours = self.modules["ContourExtr"].run(
                            img=area_of_interest,
                            find_cont_method=find_cont_method,
                            draw_contours=draw_contours,
                            level_value=level_value, k_size=k_size,

                            ↪  cont_area_threshold_percent=cont_area_threshold
        except KeyError:
            segmented_on_bands[dataset.selected_bands[i]] =
            ↪  area_of_interest
            continue

        try:
            segmented_img = self.modules["Segmentation"].run(
                                img=area_of_interest, gt=sample.labels,
                                contours=contours,
                                max_number_of_iters=max_number_of_iters,
                                increase_value = increase_value,
                                alpha=alpha, beta=beta, w1=w1, w4=w4,
                                threshold=threshold)
        except KeyError:
            segmented_on_bands[dataset.selected_bands[i]] =
            ↪  area_of_interest
            continue

        segmented_on_bands[dataset.selected_bands[i]] = segmented_img

    segmented_samples.append(segmented_on_bands)

self.segmented_samples = segmented_samples
```

3.5.1 Модули осцилляторной сети

Алгоритм сегментации основан на модели осцилляторной сети с последовательным применением 3-х модулей:

1. Модуль селективного внимания, выделяющий область потенциального интереса на изображении (см. Листинг 9, 10);

2. Модуль выделения контуров. Можно выбрать различные алгоритмы выделения контуров (см. Листинг 11, 12, 13), но эксперементально было выявлено, что лучшие результаты модель показывает при выборе алгоримта на основе оператора Собеля (см. Листинг 14);
3. Модуль сегментации изображения (см. Листинг 14, 15).

Также стоит отметить функцию `extract_stimuls`, которая играет важную роль в модуле селективного внимания. При некорректно заданных стимулах фокус внимания скорее всего не будет сформирован и работа модуля не будет эффективной (см. Листинг 16).

Листинг 9 — Класс `OnnSelectiveAttentionModule2D` модуля селективного внимания с основной функцией `run`

```
class OnnSelectiveAttentionModule2D(OnnModule):

    def __init__(self, module_name, att_method = "separate"):
        super().__init__(module_name)
        self.synchronization_states: list[list] = []
        self.central_oscillator: CentralOscillatorSelAtt
        self.periferal_oscillators:
        ↪  list[list[PeripheralOscillatorSelAtt]]=[]
        self.stimuls: list[Stimul]
        self.att_method: str = att_method

    def run(self,
            img:np.ndarray,
            po_num:int = 2,
            params_sel_method:str = "simple_random",
            stimuls_num:int = 2,
            stimuls_sel_method:str = "brightness",
            osc_params:int = [],
            target_brightness:list = []) -> np.ndarray:
        """some txt
        Args:
            img: 3D np.array of shape CxHxW
        Return:
            3D np.array of image shape with 0 representing pixels
            without att
        """
        selected_area = []
        for i, spectral_band_img in enumerate(img):
            self.setup_oscillators(img=spectral_band_img,
                                   po_num=po_num,
                                   params_sel_method=params_sel_method,
                                   stimuls_num=stimuls_num,
                                   stimuls_sel_method=stimuls_sel_method,
                                   osc_params=osc_params,
                                   target_brightness=target_brightness[i])
            new_selection = self.perform_selection()
            if self.att_method == "separate":
                selected_area.append(new_selection)
            if self.att_method == "intersect":
                if len(selected_area) == 0:
                    selected_area = new_selection
                else:
                    selected_area[selected_area != new_selection] = 0
        selected_area = np.asarray(selected_area)
        if len(selected_area) == 0:
            return img
        return selected_area
```

Листинг 10 — Функция `setup_oscillators` для задания осцилляторов

```
def setup_oscillators(self,
                          img: np.array,
                          po_num:int = 2,
                          params_sel_method:str = "simple_random",
                          stimuls_num:int = 2,
                          stimuls_sel_method:str = "brightness",
                          osc_params:list = [],
                          target_brightness:int = 0):
        """select area of interest (where approximately target is
        ↪  located)"""
        self.periferal_oscillators = []
        self.stimuls = extract_stimuls(img=img,
                                       stimuls_num=stimuls_num,
                                       method=stimuls_sel_method)
        co_freq = np.mean(np.asarray(list(map(lambda x: x.stimul_values,
        ↪  self.stimuls)), dtype="object"))
        params = self.generate_oscillators_params(
                                    method=params_sel_method,
                                    target_brightness=target_brightness,
                                    exp_params=osc_params)
        self.central_oscillator = CentralOscillatorSelAtt(freq=co_freq,
                                                           phase=1,
                                                           params=params)
        for i, stimul in enumerate(self.stimuls):
            self.periferal_oscillators.append([])
            self.generate_periferal_oscillators(stimul=stimul,
                                                stimul_id=i,
                                                alpha=params[i],
                                                n=po_num)
        self.periferal_oscillators =
        ↪  np.asarray(self.periferal_oscillators)

def generate_periferal_oscillators(self, stimul: Stimul, alpha:int,
↪  stimul_id:int, n:int = 2):
    """some_txt
    Args:
        n: number of oscillators per stimul
    """
    coef = 1e-4
    for i in range(min(n, len(stimul.stimul_values))):
        self.periferal_oscillators[stimul_id].append(
                                    PeripheralOscillatorSelAtt(
                                        phase=0,
                                        freq=stimul.stimul_values.mean()\
                                             + coef *
                                             ↪  stimul.stimul_values[i],
                                        alpha=alpha))
```

Листинг 11 — Модуль выделения контуров

```python
class OnnContourExtractionModule(OnnModule):

def __init__(self, module_name):
    super().__init__(module_name)
    self.contours: np.ndarray
    self.best_params: Dict[str, Any]

def run(self,
        img: np.ndarray,
        find_cont_method:str = "simple_sobel",
        draw_contours:bool = False,
        level_value:str = 0.7,
        target_class_brightness: float = 100,
        k_size:int = 3,
        cont_area_threshold_percent:int = 1):
    """some txt
    Args:
        img: 2D np.array of shape HxW
    Return:
        3D np.array of representing contour lines for each spectral band
    """
    if find_cont_method == "library":
        self.extract_cont_library(img, level_value=level_value)

    if find_cont_method == "gabor_grad_scale":
        self.extract_cont_gabor_grad_scale(img, target_class_brightness)

    if find_cont_method == "simple_sobel":
        self.extract_cont_simple_sobel(img, k_size)

    if draw_contours:
        try:
            show_contours(img, self.contours)
        except KeyError:
            print("Extract contours module doesnt detect any contours(
            ↪  Try to change params")

    return self.contours
```

Листинг 12 — Основная часть функции выделения контуров

```
for i, row in enumerate(temp_image):
    for j, _ in enumerate(row):
        grad_ok = 0
        for grads in gradients:
            grad_x, grad_y, grad_xx, grad_yy, grad_xxx, grad_yyy = grads
            grad_p = [grad_x[i, j], grad_y[i, j]]
            # считаем модуль вектора градиента
            mag_grad = np.sqrt(grad_p[0]**2 + grad_p[1]**2)
            angle_grad = np.arctan(grad_p[0] / (grad_p[1] + eps))
            angle_grad_degree = np.rad2deg(angle_grad)
            if angle_grad_degree < 0:
                angle_grad_degree += 360
            dir_cos1 = grad_p[0] / (mag_grad + eps)
            dir_cos2 = grad_p[1] / (mag_grad + eps)

            if ((angle_grad_degree < delta_deg) and (angle_grad_degree > 360 - delta_deg)) or \
                ((angle_grad_degree < 180 + delta_deg) and (angle_grad_degree > 180 - delta_deg)):
                grad_yy_delta_minus = grad_yy[i, j]
                grad_yy_delta_plus = grad_yy[i, j]
            else:
                grad_yy_delta_plus = grad_yy[i, min(j + 1, temp_image.shape[1] - 1)]
                grad_yy_delta_minus = grad_yy[i, max(j - 1, 0)]

            if ((angle_grad_degree < 90 + delta_deg) and (angle_grad_degree > 90 - delta_deg)) or \
                ((angle_grad_degree < 270 + delta_deg) and (angle_grad_degree > 270 - delta_deg)):
                grad_xx_delta_minus = grad_xx[i, j]
                grad_xx_delta_plus = grad_xx[i, j]
            else:
                grad_xx_delta_plus = grad_xx[min(i + 1, temp_image.shape[0] - 1), j]
                grad_xx_delta_minus = grad_xx[max(i - 1, 0), j]

            dir_ddf_plus = grad_xx_delta_plus*dir_cos1 + grad_yy_delta_plus*dir_cos2
            dir_ddf_minus = grad_xx_delta_minus*dir_cos1 + grad_yy_delta_minus*dir_cos2

            dir_dddf = grad_xxx[i, j]*dir_cos1 + grad_yyy[i, j]*dir_cos2
            if (mag_grad > const1) and ((dir_ddf_minus * dir_ddf_plus) < 0) and (dir_dddf < const2):
                grad_ok += 1
        if grad_ok == len(gradients):
            contour[i, j] += 1
```

Листинг 13 — Функция выделения контуров на основе оператора Собеля

```python
def extract_cont_simple_sobel(image, k_size):

    blurred_image = cv2.GaussianBlur(image, (k_size, k_size), 0)

    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

    grad_x = cv2.filter2D(blurred_image, -1, sobel_x)
    grad_y = cv2.filter2D(blurred_image, -1, sobel_y)

    magnitude = np.sqrt(grad_x**2 + grad_y**2)

    magnitude = cv2.normalize(magnitude, None, 0, 255, cv2.NORM_MINMAX)
    magnitude = np.uint8(magnitude)

    # _, thresholded = cv2.threshold(magnitude, 50, 255,
    ↪  cv2.THRESH_BINARY)

    return np.asarray(magnitude)
```

Листинг 14 — Метод инициализации осцилляторов для модуля сегментации

```
def setup_oscillators(self):
    # fill both layers
    self.layer1 = np.vectorize(lambda x:
                                PeripheralOscillatorSegmentationL1(
                                        freq=np.random.uniform(4, 5),
                                        phase=np.random.uniform(0, 0.2),
                                        level = 1))(self.img)
    self.layer2 = np.vectorize(lambda x:
                                PeripheralOscillatorSegmentationL2(
                                        freq=np.random.uniform(4, 5),
                                        phase=np.random.uniform(0, 0.2),
                                        level = 2))(self.img)
    self.central_oscillator = CentralOscillatorSegmentation(freq=6,
    ↪  phase=0)
    #setup inactive border oscillators
    for i, row in enumerate(self.layer1):
        for j, _ in enumerate(row):
            neibours = self.get_all_neibours((i, j))
            self.neibours_dict[(i, j)] = neibours

            senders_to_l2 = self.get_senders_to_l2((i, j))
            self.senders_to_l2_dict[(i, j)] = senders_to_l2

            if self.contours[i, j] > 10:
                self.layer1[i, j].disable()
    #setup begin square, from which segmentation starts
    begin_part_coord, begin_part_size = extract_square_subarray(self.gt)
    begin_part_coord[0] += 1
    begin_part_coord[1] += 1
    self.begin_part_coord = begin_part_coord
    self.begin_part_size = begin_part_size
    #increase freq of begin square oscillators and get s_oscillators
    s_area_oscillators = []
    for i, row in enumerate(self.layer1):
        for j, _ in enumerate(row):
            if i > begin_part_coord[0] and i < begin_part_coord[0] +
            ↪  begin_part_size and \
                j > begin_part_coord[1] and j < begin_part_coord[1] +
                ↪  begin_part_size:
                self.layer1[i, j].freq += self.increase_value
                s_area_oscillators.append(self.layer1[i, j])
    self.s_area_oscillators = np.array(s_area_oscillators)
```

Листинг 15 — Основная часть основного метода модуля сегменатации

```
while self.check_stop_condition():
    self.central_oscillator.step(s_area_osc=self.s_area_oscillators,
                                 w1=self.w1,
                                 alpha=self.alpha,
                                 g=self.g)

    for i, row in enumerate(self.layer1):
        for j, _ in enumerate(row):
            neibours = self.neibours_dict[(i, j)]
            senders_to_l2 = self.senders_to_l2_dict[(i, j)]

            self.layer1[i,
            ↪  j].step(central_oscillator=self.central_oscillator,
                                 w2 = self.w2,
                                 w3 = self.w3,
                                 point = [i, j],
                                 s_size = self.begin_part_size,
                                 s_start_p = self.begin_part_coord,
                                 neibours = neibours[0],
                                 t = self.number_of_iters)

            self.layer2[i, j].step(w4 = self.w4,
                                 neibours = neibours[1],
                                 senders_to_l2 = senders_to_l2,
                                 w5 = self.w5,
                                 point = [i, j],
                                 beta = self.beta,
                                 t = self.number_of_iters)

    res_image = np.zeros(self.layer1.shape)
    self.number_of_iters += 1
    self.central_oscillator.update()
    for i, row in enumerate(self.layer1):
        for j, _ in enumerate(row):
            self.layer1[i, j].update()
            self.layer2[i, j].update()
```

Листинг 16 — Функция `extract_stimuls`

```
def extract_stimuls(img: np.ndarray, stimuls_num:int = 2, method:str =
↪  "brightness"):
    """extract stimuls from img based on different methods (brightness,
    ↪  etc.)
    Args:
        img: 2D np.array representing sample img
    Returns:
        list of Stimul object, each of those consists info about position
        ↪  of stimul
        and about some internal values (brightness)
    """
    num_of_chunks = stimuls_num
    if method == "brightness":
        brightness_values = np.sort(np.unique(img.flatten()))

        while (len(brightness_values) % num_of_chunks) != 0:
            brightness_values = brightness_values[:-1]

        stimuls_brightnesses = np.array_split(brightness_values,
        ↪  num_of_chunks)

        stimuls = list(map(lambda x:
                           Stimul(
                            img=img,
                            sub_img=calculate_subimage_from_brightness(
                                                img=deepcopy(img),
                                                brightness_values=x),
                            stimul_values=x), stimuls_brightnesses))
    return stimuls
```

3.5.2 Осцилляторы

Задание параметров осцилляторов было затронуто ранее (см. Раздел **??**). Каждый осциллятор задается своим уравнением динамики, в зависимости от его типа и используемого модуля. В программе представленны два типа осцилляторов: периферийный (см. Листинг 17, 18, 19) и центральный (см. Листинг 20, 21)

Листинг 17 — Класс периферийного осциллятора для модуля селективного внимания

```python
class PeripheralOscillatorSelAtt(Oscillator):

    def __init__(self, freq, phase, alpha):
        super().__init__(freq, phase)
        self.alpha = alpha

    def step(self, central_oscillator: CentralOscillatorSelAtt):
        self.phase += self.freq + self.alpha *
        ↪  np.sin(central_oscillator.phase - self.phase)

    def get_synchonization_state(self, central_oscillator:
    ↪  CentralOscillatorSelAtt):
        tol = 100
        if are_close(self.phase, central_oscillator.phase, abs_tol=tol):
            return 1
        else:
            return 0
```

Листинг 18 — Класс периферийного осциллятора первого слоя для модуля сегментации

```
class PeripheralOscillatorSegmentationL1(PeripheralOscillatorSegmentation):

    def __init__(self, freq, phase, level, state: int = 1):
        super().__init__(freq, phase, level, state)

    def step(self,
             central_oscillator: CentralOscillatorSegmentation,
             w2:callable,
             w3:callable,
             point:list,
             s_size:int,
             s_start_p:list,
             neibours:List[List[Tuple[PeripheralOscillatorSegmentation, list]]],
             t:int):
        """some txt"""
        if self.state == 1:
            s_i = 1
            eps = 1e-6
            i, j = point
            if i > s_start_p[0] and i < s_start_p[0] + s_size and \
            j > s_start_p[1] and j < s_start_p[1] + s_size:
                s_i = 0

            active_neibours = [x for x in neibours if x[0].state == 1]

            delta_phase = self.phase - s_i *
             w2(t)*np.sin(central_oscillator.phase - self.phase) + \
                        w3(t) / (len(active_neibours) + eps) *
                         sum(list(map(lambda x: np.sin(x[0].phase -
                         self.phase), active_neibours)))
            self.delta_phase = delta_phase
```

Листинг 19 — Класс периферийного осциллятора второго слоя для модуля сегментации

```
class
↪  PeripheralOscillatorSegmentationL2(PeripheralOscillatorSegmentation):

    def __init__(self, freq, phase, level, state: int = 1):
        super().__init__(freq, phase, level, state)
        self.delta_freq:float

    def step(self,
             w4:float,
             neibours:List[List[Tuple[PeripheralOscillatorSegmentation,
             ↪  list]]],

             ↪  senders_to_l2:List[List[Tuple[PeripheralOscillatorSegmentation,
             ↪  list]]],
             w5:callable,
             point:list,
             beta:float,
             t:int):

        active_senders_to_l2 = [x for x in senders_to_l2 if x[0].state ==
        ↪  1]

        eps = 1e-6
        neibours_impact = w4 / (len(neibours) + eps) * \
                          sum(list(map(lambda x: np.sin(x[0].phase -
                          ↪  self.phase), neibours)))

        l1_osc_impact = sum(list(map(lambda x: w5(point,x[1]) *
        ↪  np.sin(x[0].phase - self.phase),
                                     active_senders_to_l2))) /
                                     ↪  (len(active_senders_to_l2) + eps)

        delta_phase = self.freq + neibours_impact + l1_osc_impact

        self.delta_phase = delta_phase

        delta_freq = beta * (self.delta_phase - self.freq)
        self.delta_freq = delta_freq

    def update(self):
        self.phase += self.delta_phase
        self.phase = self.phase // (2 * np.pi)
        self.freq += self.delta_freq
```

Листинг 20 — Класс центрального осциллятора для модуля селективного внимания

```
class CentralOscillatorSelAtt(Oscillator):

    def __init__(self, freq, phase, params: list):
        super().__init__(freq, phase)
        self.params = params

    def step(self, periferal_osc: list[np.ndarray[Oscillator]]):
        """some txt"""
        delta_phase = self.freq
        for i, ensemble_po in enumerate(periferal_osc):
            partial_sum = np.sum(list(map(lambda x: np.sin(x.phase -
            ↪  self.phase),
                                          ensemble_po)))
            delta_phase += self.params[i] * partial_sum /
            ↪  len(ensemble_po)
        self.phase += delta_phase
```

Листинг 21 — Класс центрального осциллятора для модуля сегментации

```
class CentralOscillatorSegmentation(Oscillator):

    def __init__(self, freq, phase):
        super().__init__(freq, phase)
        self.delta_phase: float
        self.delta_freq: float

    def step(self, s_area_osc: np.ndarray[Oscillator], w1:float,
    ↪  alpha:float, g:callable):
        """some txt"""
        vectorized_g = np.vectorize(g)
        delta_phase = self.freq + w1 / len(s_area_osc) *
        ↪  sum(list(map(lambda x: g(x.phase - self.phase), s_area_osc)))
        self.delta_phase = delta_phase

        delta_freq = alpha * (self.delta_phase - self.freq)
        self.delta_freq = delta_freq

    def update(self):
        self.phase += self.delta_phase
        self.phase = self.phase // (2 * np.pi)
        self.freq += self.delta_freq
```

3.6 Оптимизатор гиперпараметров

Так как в модели присутствует большое количество гиперпараметров, был реализован отдельный модуль (см. Листинг 22, 23, 24) на основе библиотеки optuna [2] с помощью которого выполнялся подбор наиболее подходящих параметров на основе результата модели по заданной метрике.

Листинг 22 — Класс и функция гиперпараметров для модуля селективного внимания

```
class SelAttObjective:

    def __init__(self, min_po_num:int, max_po_num:int,
                 sample:np.ndarray, sel_att_module,
                 selected_bands: list, target_class_id: int,
                 min_stimuls:int, max_stimuls:int):
        self.min_po_num = min_po_num
        self.max_po_num = max_po_num
        self.sample = sample
        self.target_class_id = target_class_id
        self.selected_bands = selected_bands
        self.sel_att_module = sel_att_module
        self.min_stimuls = min_stimuls
        self.max_stimuls = max_stimuls

    def __call__(self, trial: optuna.trial):
        po_num = trial.suggest_int("po_num", self.min_po_num,
        ↪  self.max_po_num)
        stimuls_num = trial.suggest_int("stimuls_num", self.min_stimuls,
        ↪  self.max_stimuls)
        params_sel_method =
        ↪  trial.suggest_categorical("params_sel_method",
        ↪  ["simple_random", "expert"])
        osc_params = [trial.suggest_int(f"stimul_{i}", 1, 10000) for i in
        ↪  range(stimuls_num)]
        area_of_interest_mask = self.sel_att_module.run(
                                    img=self.sample.band_img,
                                    po_num=po_num,
                                    params_sel_method=params_sel_method,
                                    osc_params=osc_params,
                                    stimuls_num=stimuls_num,

                                    ↪  target_brightness=self.sample.target_br
        segmented_on_bands = {}
        for i, spectral_band_mask in enumerate(area_of_interest_mask):
            area_of_interest = deepcopy(self.sample.band_img[i])
            area_of_interest[area_of_interest != spectral_band_mask] = -1

            area_of_interest = area_of_interest[~np.all(area_of_interest
            ↪  == -1, axis=1)]
            area_of_interest =
            ↪  area_of_interest[:,~(area_of_interest==-1).all(0)]

            segmented_on_bands[self.selected_bands[i]] = area_of_interest
        res = evaluate_best_segmentation([segmented_on_bands],
        ↪  [self.sample], self.target_class_id)
        return res[0].best_score
```

Листинг 23 — Класс и функция гиперпараметров для модуля выделения контуров

```
class ContExtrObjective:

    def __init__(self,
                 min_level:float, max_level:float,
                 min_k_size:int, max_k_size:int,
                 img:np.ndarray, cont_extr_module,
                 cont_area_threshold_percent_min:float = 1,
                 cont_area_threshold_percent_max:float = 10):
        self.min_level = min_level
        self.max_level = max_level
        self.min_k_size = min_k_size
        self.max_k_size = max_k_size
        self.img = img
        self.cont_area_threshold_percent_min =
        ↪  cont_area_threshold_percent_min
        self.cont_area_threshold_percent_max =
        ↪  cont_area_threshold_percent_max
        self.cont_extr_module = cont_extr_module

    def __call__(self, trial: optuna.trial):
        find_cont_method = trial.suggest_categorical("find_cont_method",
        ↪  ["simple_sobel", "gabor_grad_scale"])
        self.cont_extr_module.run(img=self.img,
        ↪  find_cont_method=find_cont_method)
        return len(self.cont_extr_module.contours)
```

Листинг 24 — Класс и функция гиперпараметров для модуля сегментации

```
class SegmentationObjective:

    def __init__(self,
                 max_number_of_iters, min_number_of_iters,
                 max_w1, min_w1, max_alpha, min_alpha,
                 max_beta, min_beta, max_w4, min_w4,
                 sample, contours, segm_module):
        self.max_number_of_iters = max_number_of_iters
        self.min_number_of_iters = min_number_of_iters
        self.max_w1 = max_w1
        self.min_w1 = min_w1
        self.max_alpha = max_alpha
        self.min_alpha = min_alpha
        self.max_beta = max_beta
        self.min_beta = min_beta
        self.max_w4 = max_w4
        self.min_w4 = min_w4
        self.sample = sample
        self.contours = contours
        self.segm_module = segm_module

    def __call__(self, trial):
        number_of_iters = trial.suggest_int("max_number_of_iters",
                                            self.min_number_of_iters,
                                            self.max_number_of_iters)
        w1 = trial.suggest_int("w1", self.min_w1, self.max_w1)
        alpha = trial.suggest_int("alpha", self.min_alpha,self.max_alpha)
        beta = trial.suggest_int("beta", self.min_beta, self.max_beta)
        w4 = trial.suggest_int("w4", self.min_w4, self.max_w4)
        threshold = trial.suggest_int("threshold_segm", 10, 100)
        increase_value = trial.suggest_float("increase_value", 0.1, 20)
        segm_res = self.segm_module.run(
                                    img = self.sample.band_img[0],
                                    gt = self.sample.labels,
                                    contours = self.contours,
                                    max_number_of_iters=number_of_iters,
                                    increase_value=increase_value,
                                    alpha=alpha,
                                    beta=beta,
                                    w1=w1,
                                    w4=w4,
                                    threshold=threshold)
        segmented_on_bands = {}
        segmented_on_bands[0] = segm_res
        cv2.imwrite("best_sample.png", segm_res)
        res = evaluate_best_segmentation([segmented_on_bands],
        ↪  [self.sample], 1)

        return res[0].best_score
```

4 Тестирование программы

В данном разделе описывается юнит-тестирование разработанной программы сегментации изображений. Для юнит-тестирования использовалась библиотека pytest [3]. Пример тестов для различиных функций представлен ниже (см. Листинг 25, 26, 27)

Листинг 25 — Пример юнит-теста для модуля селективного внимания

```
def test_check_synchronization_state_2(onn_sel_att_module:
↪   OnnSelectiveAttentionModule2D):
    for ensemble in onn_sel_att_module.periferal_oscillators:
        for po in ensemble:
            po.phase = onn_sel_att_module.central_oscillator.phase

    state, state_id = onn_sel_att_module.check_synchronization_state()
    assert state == "gs"
    assert state_id != 0
```

Листинг 26 — Пример юнит-теста для модуля выделения контуров

```
def test_extr_cont_library(onn_pipeline_fully_set: OnnHyperPipeline):
    ext_cont_module = OnnContourExtractionModule("ContExtr")

    ↪  ext_cont_module.run(onn_pipeline_fully_set.dataset.samples[0].band_img[
    assert len(ext_cont_module.contours) > 0

def test_find_contour_area_1():
    simple_contour = np.array([[0, 0], [1, 0], [1, 1], [0, 1]])
    cont_area = find_contour_area(simple_contour)

    assert cont_area == 1

def test_find_contour_area_2():
    simple_contour = np.array([[0, 0], [1, 0], [1, 1]])
    cont_area = find_contour_area(simple_contour)

    assert cont_area == 0.5


def test_find_contour_area_3(onn_pipeline_fully_set: OnnHyperPipeline):
    test_img = onn_pipeline_fully_set.dataset.samples[0].band_img[0]
    ext_cont_module = OnnContourExtractionModule("ContExtr")
    ext_cont_module.extract_cont_library(test_img)

    max_cont_area = test_img.shape[0] * test_img.shape[1]
    min_cont_area = 0

    test_sample_cont_area =
    ↪  find_contour_area(ext_cont_module.contours[0])

    print(ext_cont_module.contours[0])
    print(test_sample_cont_area)

    assert max_cont_area >= test_sample_cont_area
    assert min_cont_area <= test_sample_cont_area

```

Листинг 27 — Пример юнит-теста для модуля сегментации

```
def test_segm_module(onn_pipeline_fully_set: OnnHyperPipeline):
    ext_cont_module = OnnContourExtractionModule("ContExtr")
    contours = ext_cont_module.run(img =
    ↪  onn_pipeline_fully_set.dataset.samples[0].band_img[0],
                                   find_cont_method="simple_sobel",
                                   draw_contours=False)

    segm_module = OnnSegmentationModule("Segmentation")

    res_img = segm_module.run(img =
    ↪  onn_pipeline_fully_set.dataset.samples[0].band_img[0],
                              gt =
                              ↪  onn_pipeline_fully_set.dataset.samples[0].lab
                              contours=contours,
                              w1 = 22,
                              alpha = 0.2,
                              beta = 0.3,
                              w4 = 10,
                              threshold = 40,
                              increase_value = 0.1,
                              max_number_of_iters=20)

    assert np.all(res_img < 256)
```

5 Руководство пользователя

В данном разделе описывается краткое руководство пользователя по запуску разработанной программы. В разделе **??** представлены требования, необходимые для запуска и работы программы. В разделе **??** описано, как необходимо запускать программу и приведен пример. В разделе **??** приведены примеры результата работы и их интерпретация

5.1 Программные требования

Для запуска программы необходимо установить интерпретатор языка `Python` версии не ниже 3.10.11 и менеджер пакетов `pip`. Перед запуском программы в консоли необходимо выполнить команду `pip install -r requirements.txt`. При наличие на компьютере установленного `Python2` может потребоваться явно указать, какой менеджер пакетов использовать: `pip3 install -r requirements.txt`

5.2 Запуск программы

Программа является консольным приложением. Каждый параметр запуска указывается с двумя тире перед названием параметра. В листинге 28 представлена инструкция по запуску программы с информацией о каждом параметре. Пример команды запуска программы:
`python -m source.main -dataset_name PaviaU -hyperparams_path ./hyperparams/model_baseline.json -optimize_before_run False -metric iou -segm_output_dir ./segm_results`

Листинг 28 — Инструкция по запуску программы

```
Программа для анализа гиперспектральных изображений с помощью
↪  осцилляторной нейронной сети

options:
  -h, --help            show this help message and exit
  --dataset_name {PaviaU,PaviaC,KSC,IndianPines,Botswana}
                        Используемый датасет (для загрузки нового
                        ↪  понадобиться подключить vpn)
  --hyperparams_path HYPERPARAMS_PATH
                        путь до файла с гиперпараметрами
  --optimize_before_run OPTIMIZE_BEFORE_RUN
                        запуск программы с оптимизацией гиперпараметров
  --metric {iou,pixelwise}
                        метрика, по которой будет оценено качество
                        ↪  сегментации
  --segm_output_dir SEGM_OUTPUT_DIR
                        путь до директории с результатами сегментации
```

5.3 Результат работы программы

После успешного завершения выполнения программы в выбранной директории `segm_output_dir` появится 2 файла.

Один из них, `segm_res.png` содержит в себе 2 соединенных горизонтально изображения (см. Рисунок 1). В левой часть будет располагаться исходное изображение в формате `RGB`. В правой части изображение представляет исходной изображение, на которое наложено 2 маски различных цветов. Зеленая маска отвечает за те пиксели, который были классифицированы как целевой класс при работе модели, красная маска, это истинно верные пиксели, представляющие целевой класс.

Второй файл будет называться `metric_res.json` (см. Листинг 29). В нем будет 2 поля, `metric_value` и `best_band`. Значением первого поля будет являться результат сегментации по выбранной метрике, а значением второго поля - номер спектрального канала, в котором была продемонстрировано лучшее значение метрики.

Рисунок 1 — Пример маски сегментации в результате работы программы

Листинг 29 — Пример выходного `json` файла

```
{
    "metric_value": 0.01668372569089048,
    "best_band": 41
}
```

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы была разработана программа анализа гиперспектральных изображений с помощью осцилляторных нейронных сетей. Также была успешно реализована модульная система модели и одна из возможных стратегий выбора спектральных каналов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Библиотека argparse. [Электронный ресурс]. – URL: https://docs.python.org/3/library/argparse.html..
2. Библиотека optuna. [Электронный ресурс]. – URL: https://optuna.readthedocs.io/en/stable/index.html..
3. Библиотека pytest. [Электронный ресурс]. – URL: https://docs.pytest.org/en/latest/contents.html..