



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»

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
НА ТЕМУ:**

Студент ИУ9-82Б
(Группа)

(Подпись, дата)

Д.С. Головкин
(И.О. Фамилия)

Руководитель ВКР

(Подпись, дата)

Ю.Т. Каганов
(И.О. Фамилия)

Консультант

(Подпись, дата)

(И.О. Фамилия)

Консультант

(Подпись, дата)

(И.О. Фамилия)

Нормоконтролер

(Подпись, дата)

(И.О. Фамилия)

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 необходимо протестировать разработанную программу.

1 Постановка задачи

Программа должна получать на вход набор флагов, по которым будет определяться дальнейшее поведение программы. Базовый вариант использования будет подразумевать использование предзагруженного датасета гиперспектральных изображений и работы с ним. Цель программы состоит в сегментации объекта определенного класса на гиперспектральном изображении. 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 «В», 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, соответственно, 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 объясняется ее простотой и предоставляемым функционалом.

Программа поддерживает следующие параметры запуска:

- `dataset_name` (строковый тип) – устанавливает используемый датасет. При отсутствии 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

```
1 class OnnHyperPipeline(HyperPipeline):
2     """Pipeline class for implementing ONN workflow"""
3
4     def __init__(self):
5         super().__init__()
6         self.dataset: HyperSpectralData
7         self.model: OnnModel
8         self.result: list[Result]
9         self.params: Params
10
11     def add_dataset(self, dataset_name='PaviaU',
12                   load_folder="./datasets"):
13         data, gt, labels, ignored_labels, \
14         rgb_bands, palette, num_of_bands = get_dataset(dataset_name,
15                                                         load_folder)
16         params = {'labels': labels,
17                  'ignored_labels': ignored_labels,
18                  'rgb_bands': rgb_bands,
19                  'palette': palette,
20                  'num_of_bands': num_of_bands}
21         self.dataset = HyperSpectralData(data, gt, **params)
22
23     def get_data_info(self):
24         self.dataset.get_data_info()
25         self.dataset.view_data()
26
27     def specify_target_class(self, target_class: str):
28         self.dataset.specify_target_class(target_class)
29
30     def create_samples(self,
31                       num_samples:int = 3,
32                       sample_height:int = 100,
33                       sample_width:int = 100,
34                       threshold:int = 100):
35         self.dataset.create_samples(num_samples=num_samples,
36                                    sample_height=sample_height,
37                                    sample_width=sample_width,
38                                    threshold=threshold)
39
40     def select_chanel(self, method = "expert", num_of_bands = 4):
41         """method which define strategy of selection of spectral bands"""
42         self.dataset.select_chanel(method, num_of_bands)
43
44     def add_model(self, model):
45         self.model = model
46
```

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

```

1  def run(self,
2      target_class: str,
3      params_path: str = "",
4      optimize_before_run: bool = False):
5      """method to fully setup and run onn models"""
6      self.specify_target_class(target_class)
7
8      if len(params_path) != 0 and os.path.exists(params_path):
9          file_path = params_path
10     else:
11         file_path = check_default_json_file()
12     self.params = Params(file_path)
13
14     self.select_chanel(self.params.band_sel_method,
15                       self.params.num_of_bands)
16     self.create_samples(num_samples=self.params.num_samples,
17                       sample_height=self.params.sample_height,
18                       sample_width=self.params.sample_width,
19                       threshold=self.params.threshold)
20
21     if optimize_before_run:
22         optimize_hyperparams(self)
23         self.params.update(file_path)
24
25     self.model.run(
26         self.dataset,
27         po_num=self.params.po_num,
28         params_sel_method=self.params.params_sel_method,
29         stimul_num=self.params.stimul_num,
30         stimul_sel_method=self.params.stimul_sel_method,
31         find_cont_method=self.params.find_cont_method,
32         osc_params=[self.params.dict[f"stimul_{i}"]
33                     for i in range(self.params.stimul_num)],
34         draw_contours=self.params.draw_contours,
35         level_value=self.params.level_value,
36         k_size=self.params.k_size,
37         max_number_of_iters=self.params.max_number_of_iters,
38         alpha=self.params.alpha,
39         beta=self.params.beta,
40         w1=self.params.w1,
41         w4=self.params.w4,
42         threshold=self.params.threshold,
43
44         ↪ cont_area_threshold_percent=self.params.cont_area_threshold_percent

```

Листинг 3 — Метод eval для обработки и сохранения результатов

```
1 def eval(self, metric:str = "iou"):
2     """evaluate and save the results on model segmantation"""
3     samples_result = evaluate_best_segmentation(
4         self.model.segmented_samples,
5         self.dataset.samples,
6         self.dataset.target_class_id,
7         metric)
8     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_chanel`s), 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

```
1 DATASETS_CONFIG = {
2     'PaviaC': {
3         'urls': ['http://www.ehu.eus/ccwintco/uploads/e/e3/Pavia.mat',
4                 'http://www.ehu.eus/ccwintco/uploads/5/53/Pavia_gt.mat'],
5         'img': 'Pavia.mat',
6         'gt': 'Pavia_gt.mat'
7     },
8     'PaviaU': {
9         'urls': ['http://www.ehu.eus/ccwintco/uploads/e/ee/PaviaU.mat',
10                 'http://www.ehu.eus/ccwintco/uploads/5/50/PaviaU_gt.mat'],
11         'img': 'PaviaU.mat',
12         'gt': 'PaviaU_gt.mat'
13     }
14 }
```

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.

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 экспериментально было выявлено, что лучшие результаты модель показывает при выборе алгоритма 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).

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 представлены два типа осцилляторов: периферийный (см. Листинг 17, 18, 19) и центральный (см. Листинг 20, 21)

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 5 — Основная часть функции загрузчика

```
1 def get_dataset(dataset_name,
2                 target_folder,
3                 datasets=DATASETS_CONFIG):
4     """ Gets the dataset specified by name and return the related
5         components.
6     """
7     palette = None
8     if dataset_name not in datasets.keys():
9         raise ValueError("{} dataset is unknown.".format(dataset_name))
10    dataset = datasets[dataset_name]
11    folder = target_folder + datasets[dataset_name].get('folder',
12                                                         dataset_name+'/')
13    if dataset.get('download', True):
14        # Download the dataset if is not present
15        if not os.path.isdir(folder):
16            os.mkdir(folder)
17        for url in datasets[dataset_name]['urls']:
18            # download the files
19            filename = url.split('/')[-1]
20            if not os.path.exists(folder + filename):
21                with TqdmUpTo(unit='B', unit_scale=True, miniters=1,
22                             desc="Downloading {}".format(filename)) as t:
23                    urlretrieve(url, filename=folder + filename,
24                               reporthook=t.update_to)
25    elif not os.path.isdir(folder):
26        print("WARNING: {} is not downloadable.".format(dataset_name))
27
28    if dataset_name == 'PaviaC':
29        # Load the image
30        img = open_file(folder + 'Pavia.mat')['pavia']
31        rgb_bands = (55, 41, 12)
32        gt = open_file(folder + 'Pavia_gt.mat')['pavia_gt']
33        label_values = ["Undefined", "Water", "Trees", "Asphalt",
34                        "Self-Blocking Bricks", "Bitumen", "Tiles",
35                        "Shadows", "Meadows", "Bare Soil"]
36        ignored_labels = [0]
37        # Filter NaN out
38        nan_mask = np.isnan(img.sum(axis=-1))
39        img[nan_mask] = 0
40        gt[nan_mask] = 0
41        ignored_labels.append(0)
42        ignored_labels = list(set(ignored_labels))
43        num_of_bands = img.shape[2]
44        # Normalization
45        img = np.asarray(img, dtype='float32')
46        return np.array([img]), gt, label_values, ignored_labels, \
47            rgb_bands, palette, num_of_bands
```

Листинг 6 — Класс данных модели

```
1 class HyperSpectralData():
2     """Class for storing hyp data for simple ONN"""
3
4     def __init__(self, data, gt, **params):
5         """
6         Args:
7             data: array of 3D hyperspectral image (may be 1-elem array)
8             gt: 2D array of labels
9             data_augmentation: bool, set to True to perform random flips
10            supervision: 'full' or 'un' supervised algorithms
11        """
12        self.data: np.ndarray = data
13        self.gt: np.ndarray = gt
14        self.labels_to_ignore = params['ignored_labels']
15        self.labels = params['labels']
16        self.rgb = params['rgb_bands']
17        self.num_bands = params['num_of_bands']
18        self.samples: list[Sample]
19        self.samples_labels: np.ndarray
20        self.selected_bands: np.ndarray
21        self.target_class: str
22        self.target_class_id: int
23
24    def select_chanel(self, method = "expert", n = 4):
25        """method which define strategy of selection of spectral bands"""
26        if method == "expert":
27            self.selected_bands = [55, 41, 12]
28        if method == "simple_opt":
29            sel_bands = select_best_spectrums(
30                img = self.data[0],
31                complete_gt = self.gt,
32                target_class_id = self.target_class_id,
33                n = n
34            )
35            self.selected_bands = sel_bands
36        if method == "advanced_opts":
37            self.selected_bands = [55, 41, 12]
```

Листинг 7 — Основная часть основной функции run модели

```
1  if "SelectiveAtt" not in self.modules.keys():
2      raise KeyError("Module SelectiveAtt are not in model modules list")
3
4  for sample in dataset.samples:
5      area_of_interest_mask = self.modules["SelectiveAtt"].run(
6          img=sample.band_img, po_num=po_num,
7          params_sel_method=params_sel_method,
8          stimul_num=stimul_num,
9          stimul_sel_method=stimul_sel_method,
10         osc_params=osc_params,
11         target_brightness=sample.target_brightness)
12
13     if self.modules["SelectiveAtt"].att_method == "separate":
14         iterate_bands_over = area_of_interest_mask
15
16     if self.modules["SelectiveAtt"].att_method == "intersect":
17         iterate_bands_over = sample.band_img
18
19     segmented_on_bands = {}
20     for i, spectral_band_mask in enumerate(iterate_bands_over):
21         area_of_interest = deepcopy(sample.band_img[i])
22         if self.modules["SelectiveAtt"].att_method == "separate":
23             # verify that only current stimul represented in
24             ↪ area_of_interest brightness values
25             area_of_interest[area_of_interest != spectral_band_mask] = -1
26
27         if self.modules["SelectiveAtt"].att_method == "intersect":
28             # verify that only current stimul represented in
29             ↪ area_of_interest brightness values
30             area_of_interest[area_of_interest != area_of_interest_mask] =
31                 -1
```


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

```
1      # crop the area to minimize further calculations
2      area_of_interest = area_of_interest[~np.all(area_of_interest ==
3      ↪ -1, axis=1)]
4      area_of_interest =
5      ↪ area_of_interest[:, ~(area_of_interest==-1).all(0)]
6      try:
7          contours = self.modules["ContourExtr"].run(
8              img=area_of_interest,
9              find_cont_method=find_cont_method,
10             draw_contours=draw_contours,
11             level_value=level_value, k_size=k_size,
12             ↪ cont_area_threshold_percent=cont_area_threshold
13
14     except KeyError:
15         segmented_on_bands[dataset.selected_bands[i]] =
16         ↪ area_of_interest
17         continue
18
19     try:
20         segmented_img = self.modules["Segmentation"].run(
21             img=area_of_interest, gt=sample.labels,
22             contours=contours,
23             max_number_of_iters=max_number_of_iters,
24             increase_value = increase_value,
25             alpha=alpha, beta=beta, w1=w1, w4=w4,
26             threshold=threshold)
27
28     except KeyError:
29         segmented_on_bands[dataset.selected_bands[i]] =
30         ↪ area_of_interest
31         continue
32
33     segmented_on_bands[dataset.selected_bands[i]] = segmented_img
34
35     segmented_samples.append(segmented_on_bands)
36
37     self.segmented_samples = segmented_samples
```

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

```

1  class OnnSelectiveAttentionModule2D(OnnModule):
2
3      def __init__(self, module_name, att_method = "separate"):
4          super().__init__(module_name)
5          self.synchronization_states: list[list] = []
6          self.central_oscillator: CentralOscillatorSelAtt
7          self.periferal_oscillators:
8              ↪ list[list[PeripheralOscillatorSelAtt]]=[]
9          self.stimuls: list[Stimul]
10         self.att_method: str = att_method
11
12     def run(self,
13             img:np.ndarray,
14             po_num:int = 2,
15             params_sel_method:str = "simple_random",
16             stimuls_num:int = 2,
17             stimuls_sel_method:str = "brightness",
18             osc_params:int = [],
19             target_brightness:list = []) -> np.ndarray:
20         """some txt
21         Args:
22         img: 3D np.array of shape CxHxW
23         Return:
24         3D np.array of image shape with 0 representing pixels
25         without att
26         """
27         selected_area = []
28         for i, spectral_band_img in enumerate(img):
29             self.setup_oscillators(img=spectral_band_img,
30                                   po_num=po_num,
31                                   params_sel_method=params_sel_method,
32                                   stimuls_num=stimuls_num,
33                                   stimuls_sel_method=stimuls_sel_method,
34                                   osc_params=osc_params,
35                                   target_brightness=target_brightness[i])
36         new_selection = self.perform_selection()
37         if self.att_method == "separate":
38             selected_area.append(new_selection)
39         if self.att_method == "intersect":
40             if len(selected_area) == 0:
41                 selected_area = new_selection
42             else:
43                 selected_area[selected_area != new_selection] = 0
44
45         selected_area = np.asarray(selected_area)
46
47         if len(selected_area) == 0:
48             return img
49         return selected_area

```

Листинг 10 — Функция setup_oscillators для задания осцилляторов

```

1  def setup_oscillators(self,
2      img: np.array,
3      po_num:int = 2,
4      params_sel_method:str = "simple_random",
5      stimul_num:int = 2,
6      stimul_sel_method:str = "brightness",
7      osc_params:list = [],
8      target_brightness:int = 0):
9      """select area of interest (where approximately target is
10         ↪ located)"""
11      self.periferal_oscillators = []
12      self.stimuls = extract_stimuls(img=img,
13                                     stimul_num=stimul_num,
14                                     method=stimul_sel_method)
15
16      co_freq = np.mean(np.asarray(list(map(lambda x: x.stimul_values,
17                                             ↪ self.stimuls))), dtype="object"))
18      params =
19         ↪ self.generate_oscillators_params(method=params_sel_method,
20                                         ↪ target_brightness=target_
21                                         exp_params=osc_params)
22
23      self.central_oscillator = CentralOscillatorSelAtt(freq=co_freq,
24                                                         phase=1,
25                                                         params=params)
26
27      for i, stimul in enumerate(self.stimuls):
28          self.periferal_oscillators.append([])
29          self.generate_periferal_oscillators(stimul=stimul,
30                                              stimul_id=i,
31                                              alpha=params[i],
32                                              n=po_num)
33
34      self.periferal_oscillators =
35         ↪ np.asarray(self.periferal_oscillators)
36
37  def generate_periferal_oscillators(self, stimul: Stimul, alpha:int,
38         ↪ stimul_id:int, n:int = 2):
39      """some_txt
40         Args:
41             n: number of oscillators per stimul
42         """
43      coef = 1e-4
44      for i in range(min(n, len(stimul.stimul_values))):
45          self.periferal_oscillators[stimul_id].append(
46              PeripheralOscillatorSelAtt(
47                  phase=0,
48                  ↪ freq=stimul.stimul_values.m
49                  ↪ \
50                  + coef *
51                  ↪ stimul.stimul_values[i
52                  alpha=alpha))

```

Листинг 11 — Модуль выделения контуров

```
1 class OnnContourExtractionModule(OnnModule):
2
3 def __init__(self, module_name):
4     super().__init__(module_name)
5     self.contours: np.ndarray
6     self.best_params: Dict[str, Any]
7
8 def run(self,
9         img: np.ndarray,
10        find_cont_method: str = "simple_sobel",
11        draw_contours: bool = False,
12        level_value: str = 0.7,
13        target_class_brightness: float = 100,
14        k_size: int = 3,
15        cont_area_threshold_percent: int = 1):
16     """some txt
17     Args:
18     img: 2D np.array of shape HxW
19     Return:
20     3D np.array of representing contour lines for each spectral band
21     """
22     if find_cont_method == "library":
23         self.extract_cont_library(img, level_value=level_value)
24
25     if find_cont_method == "gabor_grad_scale":
26         self.extract_cont_gabor_grad_scale(img, target_class_brightness)
27
28     if find_cont_method == "simple_sobel":
29         self.extract_cont_simple_sobel(img, k_size)
30
31     if draw_contours:
32         try:
33             show_contours(img, self.contours)
34         except KeyError:
35             print("Extract contours module doesnt detect any contours(
36                 ↪ Try to change params")
37
38     return self.contours
```

Листинг 12 — Основная часть функции выделения контуров

```

1  for i, row in enumerate(temp_image):
2      for j, _ in enumerate(row):
3          grad_ok = 0
4          for grads in gradients:
5              grad_x, grad_y, grad_xx, grad_yy, grad_xxx, grad_yyy = grads
6              grad_p = [grad_x[i, j], grad_y[i, j]]
7              # считаем модуль вектора градиента
8              mag_grad = np.sqrt(grad_p[0]**2 + grad_p[1]**2)
9              angle_grad = np.arctan(grad_p[0] / (grad_p[1] + eps))
10             angle_grad_degree = np.rad2deg(angle_grad)
11             if angle_grad_degree < 0:
12                 angle_grad_degree += 360
13             dir_cos1 = grad_p[0] / (mag_grad + eps)
14             dir_cos2 = grad_p[1] / (mag_grad + eps)
15
16             if ((angle_grad_degree < delta_deg) and (angle_grad_degree >
17 ↪ 360 - delta_deg)) or \
18                 ((angle_grad_degree < 180 + delta_deg) and
19 ↪ (angle_grad_degree > 180 - delta_deg)):
20                 grad_yy_delta_minus = grad_yy[i, j]
21                 grad_yy_delta_plus = grad_yy[i, j]
22             else:
23                 grad_yy_delta_plus = grad_yy[i, min(j + 1,
24 ↪ temp_image.shape[1] - 1)]
25                 grad_yy_delta_minus = grad_yy[i, max(j - 1, 0)]
26
27             if ((angle_grad_degree < 90 + delta_deg) and
28 ↪ (angle_grad_degree > 90 - delta_deg)) or \
29                 ((angle_grad_degree < 270 + delta_deg) and
30 ↪ (angle_grad_degree > 270 - delta_deg)):
31                 grad_xx_delta_minus = grad_xx[i, j]
32                 grad_xx_delta_plus = grad_xx[i, j]
33             else:
34                 grad_xx_delta_plus = grad_xx[min(i + 1,
35 ↪ temp_image.shape[0] - 1), j]
36                 grad_xx_delta_minus = grad_xx[max(i - 1, 0), j]
37
38             dir_ddf_plus = grad_xx_delta_plus*dir_cos1 +
39 ↪ grad_yy_delta_plus*dir_cos2
40             dir_ddf_minus = grad_xx_delta_minus*dir_cos1 +
41 ↪ grad_yy_delta_minus*dir_cos2
42
43             dir_ddd = grad_xxx[i, j]*dir_cos1 + grad_yyy[i, j]*dir_cos2
44             if (mag_grad > const1) and ((dir_ddf_minus * dir_ddf_plus) <
45 ↪ 0) and (dir_ddd < const2):
46                 grad_ok += 1
47         if grad_ok == len(gradients):
48             contour[i, j] += 1

```

Листинг 13 — Функция выделения контуров на основе оператора Собеля

```
1 def extract_cont_simple_sobel(image, k_size):
2
3     blurred_image = cv2.GaussianBlur(image, (k_size, k_size), 0)
4
5     sobel_x = np.array([[ -1,  0,  1], [-2,  0,  2], [-1,  0,  1]])
6     sobel_y = np.array([[ -1, -2, -1], [ 0,  0,  0], [ 1,  2,  1]])
7
8     grad_x = cv2.filter2D(blurred_image, -1, sobel_x)
9     grad_y = cv2.filter2D(blurred_image, -1, sobel_y)
10
11     magnitude = np.sqrt(grad_x**2 + grad_y**2)
12
13     magnitude = cv2.normalize(magnitude, None, 0, 255, cv2.NORM_MINMAX)
14     magnitude = np.uint8(magnitude)
15
16     # _, thresholded = cv2.threshold(magnitude, 50, 255,
17     ↪ cv2.THRESH_BINARY)
18
19     return np.asarray(magnitude)
```

Листинг 14 — Метод инициализации осцилляторов для модуля сегментации

```

1  def setup_oscillators(self):
2      # fill both layers
3      self.layer1 = np.vectorize(lambda x:
4          ↪ PeripheralOscillatorSegmentationL1(
5
6          ↪ freq=np.random.uniform(
7              ↪ 5),
8
9              ↪ phase=np.random.uniform(
10                 ↪ 0.2),
11                 level =
12                 ↪ 1))(self.img)
13
14     self.layer2 = np.vectorize(lambda x:
15         ↪ PeripheralOscillatorSegmentationL2(
16
17         ↪ freq=np.random.uniform(
18             ↪ 5),
19
20             ↪ phase=np.random.uniform(
21                 ↪ 0.2),
22                 level =
23                 ↪ 2))(self.img)
24
25     self.central_oscillator = CentralOscillatorSegmentation(freq=6,
26         ↪ phase=0)
27     #setup inactive border oscillators
28     for i, row in enumerate(self.layer1):
29         for j, _ in enumerate(row):
30             neighbours = self.get_all_neighbours((i, j))
31             self.neighbours_dict[(i, j)] = neighbours
32
33             senders_to_l2 = self.get_senders_to_l2((i, j))
34             self.senders_to_l2_dict[(i, j)] = senders_to_l2
35
36             if self.contours[i, j] > 10:
37                 self.layer1[i, j].disable()
38     #setup begin square, from which segmentation starts
39     begin_part_coord, begin_part_size = extract_square_subarray(self.gt)
40     begin_part_coord[0] += 1
41     begin_part_coord[1] += 1
42     self.begin_part_coord = begin_part_coord
43     self.begin_part_size = begin_part_size
44     #increase freq of begin square oscillators and get s_oscillators
45     s_area_oscillators = []
46     for i, row in enumerate(self.layer1):
47         for j, _ in enumerate(row):
48             if i > begin_part_coord[0] and i < begin_part_coord[0] +
49                 ↪ begin_part_size and \
50                 j > begin_part_coord[1] and j < begin_part_coord[1] +
51                 ↪ begin_part_size:
52                 self.layer1[i, j].freq += self.increase_value
53                 s_area_oscillators.append(self.layer1[i, j])
54
55     self.s_area_oscillators = np.array(s_area_oscillators)

```

Листинг 15 — Основная часть основного метода модуля сегментации

```
1 while self.check_stop_condition():
2     self.central_oscillator.step(s_area_osc=self.s_area_oscillators,
3                                 w1=self.w1,
4                                 alpha=self.alpha,
5                                 g=self.g)
6
7     for i, row in enumerate(self.layer1):
8         for j, _ in enumerate(row):
9             neighbours = self.neighbours_dict[(i, j)]
10            senders_to_l2 = self.senders_to_l2_dict[(i, j)]
11
12            self.layer1[i,
13            ↪ j].step(central_oscillator=self.central_oscillator,
14                    w2 = self.w2,
15                    w3 = self.w3,
16                    point = [i, j],
17                    s_size = self.begin_part_size,
18                    s_start_p = self.begin_part_coord,
19                    neighbours = neighbours[0],
20                    t = self.number_of_iters)
21
22            self.layer2[i, j].step(w4 = self.w4,
23                                   neighbours = neighbours[1],
24                                   senders_to_l2 = senders_to_l2,
25                                   w5 = self.w5,
26                                   point = [i, j],
27                                   beta = self.beta,
28                                   t = self.number_of_iters)
29
30            res_image = np.zeros(self.layer1.shape)
31            self.number_of_iters += 1
32            self.central_oscillator.update()
33            for i, row in enumerate(self.layer1):
34                for j, _ in enumerate(row):
35                    self.layer1[i, j].update()
36                    self.layer2[i, j].update()
```


Листинг 16 — Функция extract_stimuls

```

1  def extract_stimuls(img: np.ndarray, stimul_num: int = 2, method: str =
    ↪ "brightness"):
2      """extract stimul from img based on different methods (brightness,
    ↪ etc.)
3      Args:
4          img: 2D np.array representing sample img
5      Returns:
6          list of Stimul object, each of those consists info about position
    ↪ of stimul
7          and about some internal values (brightness)
8      """
9      num_of_chunks = stimul_num
10     if method == "brightness":
11         brightness_values = np.sort(np.unique(img.flatten()))
12
13         while (len(brightness_values) % num_of_chunks) != 0:
14             brightness_values = brightness_values[:-1]
15
16         stimul_brightnesses = np.array_split(brightness_values,
    ↪ num_of_chunks)
17
18         stimul = list(map(lambda x: Stimul(
19             img=img,
20
    ↪ sub_img=calculate_subimage_from_brightn
21
    ↪ img=dee
22
    ↪ brightn
23             stimul_values=x),
    ↪ stimul_brightnesses))
24     return stimul

```

Листинг 17 — Класс периферийного осциллятора для модуля селективного внимания

```
1 class PeripheralOscillatorSelAtt(Oscillator):
2
3     def __init__(self, freq, phase, alpha):
4         super().__init__(freq, phase)
5         self.alpha = alpha
6
7     def step(self, central_oscillator: CentralOscillatorSelAtt):
8         self.phase += self.freq + self.alpha *
9             ↪ np.sin(central_oscillator.phase - self.phase)
10
11     def get_synchronization_state(self, central_oscillator:
12     ↪ CentralOscillatorSelAtt):
13         tol = 100
14         if are_close(self.phase, central_oscillator.phase, abs_tol=tol):
15             return 1
16         else:
17             return 0
```

Листинг 18 — Класс периферийного осциллятора первого слоя для модуля сегментации

```

1  class
   ↪ PeripheralOscillatorSegmentationL1(PeripheralOscillatorSegmentation):
2
3      def __init__(self, freq, phase, level, state: int = 1):
4          super().__init__(freq, phase, level, state)
5
6      def step(self,
7              central_oscillator: CentralOscillatorSegmentation,
8              w2: callable,
9              w3: callable,
10             point: list,
11             s_size: int,
12             s_start_p: list,
13             neighbours: List[List[Tuple[PeripheralOscillatorSegmentation,
   ↪ list]]],
14             t: int):
15         """some txt"""
16         if self.state == 1:
17             s_i = 1
18             eps = 1e-6
19             i, j = point
20             if i > s_start_p[0] and i < s_start_p[0] + s_size and \
21                 j > s_start_p[1] and j < s_start_p[1] + s_size:
22                 s_i = 0
23
24             active_neighbours = [x for x in neighbours if x[0].state == 1]
25
26             delta_phase = self.phase - s_i *
   ↪ w2(t)*np.sin(central_oscillator.phase - self.phase) + \
27                 w3(t) / (len(active_neighbours) + eps) *
   ↪ sum(list(map(lambda x: np.sin(x[0].phase -
   ↪ self.phase), active_neighbours)))
28             self.delta_phase = delta_phase

```

Листинг 19 — Класс периферийного осциллятора второго слоя для модуля сегментации

```

1  class
2  ↪ PeripheralOscillatorSegmentationL2(PeripheralOscillatorSegmentation):
3
4  def __init__(self, freq, phase, level, state: int = 1):
5      super().__init__(freq, phase, level, state)
6      self.delta_freq:float
7
8  def step(self,
9      w4:float,
10     neibours:List[List[Tuple[PeripheralOscillatorSegmentation,
11     ↪ list]]],
12
13     ↪ senders_to_l2:List[List[Tuple[PeripheralOscillatorSegmentation,
14     ↪ list]]],
15     w5:callable,
16     point:list,
17     beta:float,
18     t:int):
19
20     active_senders_to_l2 = [x for x in senders_to_l2 if x[0].state ==
21     ↪ 1]
22
23     eps = 1e-6
24     neibours_impact = w4 / (len(neibours) + eps) * \
25         sum(list(map(lambda x: np.sin(x[0].phase -
26         ↪ self.phase), neibours)))
27
28     l1_osc_impact = sum(list(map(lambda x: w5(point,x[1]) *
29     ↪ np.sin(x[0].phase - self.phase),
30         active_senders_to_l2))) /
31         (len(active_senders_to_l2) + eps)
32
33     delta_phase = self.freq + neibours_impact + l1_osc_impact
34
35     self.delta_phase = delta_phase
36
37     delta_freq = beta * (self.delta_phase - self.freq)
38     self.delta_freq = delta_freq
39
40 def update(self):
41     self.phase += self.delta_phase
42     self.phase = self.phase // (2 * np.pi)
43     self.freq += self.delta_freq

```

Листинг 20 — Класс центрального осциллятора для модуля селективного внимания

```
1 class CentralOscillatorSelAtt(Oscillator):
2
3     def __init__(self, freq, phase, params: list):
4         super().__init__(freq, phase)
5         self.params = params
6
7     def step(self, periferal_osc: list[np.ndarray[Oscillator]]):
8         """some txt"""
9         delta_phase = self.freq
10        for i, ensemble_po in enumerate(periferal_osc):
11            partial_sum = np.sum(list(map(lambda x: np.sin(x.phase -
12                ↪ self.phase),
13                                         ensemble_po)))
14            delta_phase += self.params[i] * partial_sum /
15                ↪ len(ensemble_po)
16        self.phase += delta_phase
```

Листинг 21 — Класс центрального осциллятора для модуля сегментации

```
1 class CentralOscillatorSegmentation(Oscillator):
2
3     def __init__(self, freq, phase):
4         super().__init__(freq, phase)
5         self.delta_phase: float
6         self.delta_freq: float
7
8     def step(self, s_area_osc: np.ndarray[Oscillator], w1:float,
9         ↪ alpha:float, g:callable):
10        """some txt"""
11        vectorized_g = np.vectorize(g)
12        delta_phase = self.freq + w1 / len(s_area_osc) *
13            ↪ sum(list(map(lambda x: g(x.phase - self.phase), s_area_osc)))
14        self.delta_phase = delta_phase
15
16        delta_freq = alpha * (self.delta_phase - self.freq)
17        self.delta_freq = delta_freq
18
19    def update(self):
20        self.phase += self.delta_phase
21        self.phase = self.phase // (2 * np.pi)
22        self.freq += self.delta_freq
```

Листинг 22 — Класс и функция гиперпараметров для модуля селективного внимания

```
1 class SelAttObjective:
2
3     def __init__(self, min_po_num:int, max_po_num:int,
4                   sample:np.ndarray, sel_att_module,
5                   selected_bands: list, target_class_id: int,
6                   min_stimuls:int, max_stimuls:int):
7         self.min_po_num = min_po_num
8         self.max_po_num = max_po_num
9         self.sample = sample
10        self.target_class_id = target_class_id
11        self.selected_bands = selected_bands
12        self.sel_att_module = sel_att_module
13        self.min_stimuls = min_stimuls
14        self.max_stimuls = max_stimuls
15
16    def __call__(self, trial: optuna.trial):
17        po_num = trial.suggest_int("po_num", self.min_po_num,
18                                   ↪ self.max_po_num)
19        stimuls_num = trial.suggest_int("stimuls_num", self.min_stimuls,
20                                       ↪ self.max_stimuls)
21        params_sel_method =
22            ↪ trial.suggest_categorical("params_sel_method",
23                                       ↪ ["simple_random", "expert"])
24        osc_params = [trial.suggest_int(f"stimul_{i}", 1, 10000) for i in
25                       ↪ range(stimuls_num)]
26        area_of_interest_mask = self.sel_att_module.run(
27            img=self.sample.band_img,
28            po_num=po_num,
29
30            ↪ params_sel_method=params_sel_method,
31            osc_params=osc_params,
32            stimuls_num=stimuls_num,
33
34            ↪ target_brightness=self.sample.target_brightness)
35
36        segmented_on_bands = {}
37        for i, spectral_band_mask in enumerate(area_of_interest_mask):
38            area_of_interest = deepcopy(self.sample.band_img[i])
39            area_of_interest[area_of_interest != spectral_band_mask] = -1
40
41            area_of_interest = area_of_interest[~np.all(area_of_interest
42                ↪ == -1, axis=1)]
43            area_of_interest =
44                ↪ area_of_interest[:, ~(area_of_interest==-1).all(0)]
45
46            segmented_on_bands[self.selected_bands[i]] = area_of_interest
47
48        res = evaluate_best_segmentation([segmented_on_bands],
49                                       ↪ [self.sample], self.target_class_id)
50
51        # self.best_resres[0].sample[self.best_band_id].shape)
52        return res[0].best_score
```

Листинг 23 — Класс и функция гиперпараметров для модуля выделения контуров

```
1 class ContExtrObjective:
2
3     def __init__(self,
4                 min_level:float, max_level:float,
5                 min_k_size:int, max_k_size:int,
6                 img:np.ndarray, cont_extr_module,
7                 cont_area_threshold_percent_min:float = 1,
8                 cont_area_threshold_percent_max:float = 10):
9         self.min_level = min_level
10        self.max_level = max_level
11        self.min_k_size = min_k_size
12        self.max_k_size = max_k_size
13        self.img = img
14        self.cont_area_threshold_percent_min =
15        ↪ cont_area_threshold_percent_min
16        self.cont_area_threshold_percent_max =
17        ↪ cont_area_threshold_percent_max
18        self.cont_extr_module = cont_extr_module
19
20    def __call__(self, trial: optuna.trial):
21        find_cont_method = trial.suggest_categorical("find_cont_method",
22        ↪ ["simple_sobel", "gabor_grad_scale"])
23        self.cont_extr_module.run(img=self.img,
24        ↪ find_cont_method=find_cont_method)
25        return len(self.cont_extr_module.contours)
```

Листинг 24 — Класс и функция гиперпараметров для модуля сегментации

```

1  class SegmentationObjective:
2
3      def __init__(self,
4                  max_number_of_iters, min_number_of_iters,
5                  max_w1, min_w1, max_alpha, min_alpha,
6                  max_beta, min_beta, max_w4, min_w4,
7                  sample, contours, segm_module):
8          self.max_number_of_iters = max_number_of_iters
9          self.min_number_of_iters = min_number_of_iters
10         self.max_w1 = max_w1
11         self.min_w1 = min_w1
12         self.max_alpha = max_alpha
13         self.min_alpha = min_alpha
14         self.max_beta = max_beta
15         self.min_beta = min_beta
16         self.max_w4 = max_w4
17         self.min_w4 = min_w4
18         self.sample = sample
19         self.contours = contours
20         self.segm_module = segm_module
21
22     def __call__(self, trial):
23         number_of_iters = trial.suggest_int("max_number_of_iters",
24                                           self.min_number_of_iters,
25                                           self.max_number_of_iters)
26         w1 = trial.suggest_int("w1", self.min_w1, self.max_w1)
27         alpha = trial.suggest_int("alpha", self.min_alpha, self.max_alpha)
28         beta = trial.suggest_int("beta", self.min_beta, self.max_beta)
29         w4 = trial.suggest_int("w4", self.min_w4, self.max_w4)
30         threshold = trial.suggest_int("threshold_segm", 10, 100)
31         increase_value = trial.suggest_float("increase_value", 0.1, 20)
32         segm_res = self.segm_module.run(img = self.sample.band_img[0],
33                                         gt = self.sample.labels,
34                                         contours = self.contours,
35
36                                         ↪ max_number_of_iters=number_of_iters,
37                                         increase_value=increase_value,
38                                         alpha=alpha,
39                                         beta=beta,
40                                         w1=w1,
41                                         w4=w4,
42                                         threshold=threshold)
43
44         segmented_on_bands = {}
45         segmented_on_bands[0] = segm_res
46         cv2.imwrite("best_sample.png", segm_res)
47         res = evaluate_best_segmentation([segmented_on_bands],
48                                         ↪ [self.sample], 1)
49
50         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 различных функций представлен ниже (см. Листинг 25, 26, 27)

Листинг 25 — Пример юнит-теста для модуля селективного внимания

```
1 def test_check_synchronization_state_2(onn_sel_att_module:
  ↳ OnnSelectiveAttentionModule2D):
2     for ensemble in onn_sel_att_module.periferal_oscillators:
3         for po in ensemble:
4             po.phase = onn_sel_att_module.central_oscillator.phase
5
6     state, state_id = onn_sel_att_module.check_synchronization_state()
7     assert state == "gs"
8     assert state_id != 0
```

Листинг 26 — Пример юнит-теста для модуля выделения контуров

```
1 def test_extr_cont_library(onn_pipeline_fully_set: OnnHyperPipeline):
2     ext_cont_module = OnnContourExtractionModule("ContExtr")
3
4     ↪ ext_cont_module.run(onn_pipeline_fully_set.dataset.samples[0].band_img[
5     assert len(ext_cont_module.contours) > 0
6
7 def test_find_contour_area_1():
8     simple_contour = np.array([[0, 0], [1, 0], [1, 1], [0, 1]])
9     cont_area = find_contour_area(simple_contour)
10
11     assert cont_area == 1
12
13 def test_find_contour_area_2():
14     simple_contour = np.array([[0, 0], [1, 0], [1, 1]])
15     cont_area = find_contour_area(simple_contour)
16
17     assert cont_area == 0.5
18
19 def test_find_contour_area_3(onn_pipeline_fully_set: OnnHyperPipeline):
20     test_img = onn_pipeline_fully_set.dataset.samples[0].band_img[0]
21     ext_cont_module = OnnContourExtractionModule("ContExtr")
22     ext_cont_module.extract_cont_library(test_img)
23
24     max_cont_area = test_img.shape[0] * test_img.shape[1]
25     min_cont_area = 0
26
27     test_sample_cont_area =
28     ↪ find_contour_area(ext_cont_module.contours[0])
29
30     print(ext_cont_module.contours[0])
31     print(test_sample_cont_area)
32
33     assert max_cont_area >= test_sample_cont_area
34     assert min_cont_area <= test_sample_cont_area
```

Листинг 27 — Пример юнит-теста для модуля сегментации

```
1 def test_segm_module(onn_pipeline_fully_set: OnnHyperPipeline):
2     ext_cont_module = OnnContourExtractionModule("ContExtr")
3     contours = ext_cont_module.run(img =
4         ↪ onn_pipeline_fully_set.dataset.samples[0].band_img[0],
5         find_cont_method="simple_sobel",
6         draw_contours=False)
7
8     segm_module = OnnSegmentationModule("Segmentation")
9
10    res_img = segm_module.run(img =
11        ↪ onn_pipeline_fully_set.dataset.samples[0].band_img[0],
12        gt =
13            ↪ onn_pipeline_fully_set.dataset.samples[0].lab
14            contours=contours,
15            w1 = 22,
16            alpha = 0.2,
17            beta = 0.3,
18            w4 = 10,
19            threshold = 40,
20            increase_value = 0.1,
21            max_number_of_iters=20)
22
23    assert np.all(res_img < 256)
```

5 Руководство пользователя

В данном разделе описывается краткое руководство пользователя по запуску разработанной программы. В разделе ??

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 наличии 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
```

5.3 Результат работы программы

Листинг 28 — инструкция по запуску программы

```
1  Программа для анализа гиперспектральных изображений с помощью  
   ↳ осцилляторной нейронной сети  
2  
3  options:  
4    -h, --help          show this help message and exit  
5    --dataset_name {PaviaU,PaviaC,KSC,IndianPines,Botswana}  
6                        Используемый датасет (для загрузки нового  
   ↳ понадобится подключить vpn)  
7    --hyperparams_path HYPERPARAMS_PATH  
8                        путь до файла с гиперпараметрами  
9    --optimize_before_run OPTIMIZE_BEFORE_RUN  
10                       запуск программы с оптимизацией гиперпараметров  
11    --metric {iou,pixelwise}  
12                       метрика, по которой будет оценено качество  
   ↳ сегментации  
13    --segm_output_dir SEGM_OUTPUT_DIR  
14                       путь до директории с результатами сегментации
```

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы была разработана программа анализа гиперспектральных изображений с помощью осцилляторных нейронных сетей. Также была успешно реализована модульная система модели.

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>..