**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**Факультет прикладной математики – процессов управления**

**отчет**

**по лабораторной работе**

**по дисциплине «Алгоритмы и структуры данных»**

**на тему «Реализация метода 2D ССА в приложении к обработке изображений лиц»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 22.Б15 |  | Агишев А.Б. |
| Преподаватель |  | Дик А.Г. |

**Санкт-Петербург**

**2024 г.**

**Содержание**

[1. Цель работы 1](#_Toc166892209)

[2. Задача 1](#_Toc166892210)

[3. Теоретическая часть 1](#_Toc166892211)

[4. Описание алгоритма 2](#_Toc166892212)

[5. Описание программы 4](#_Toc166892213)

[5.1 Описание классов 4](#_Toc166892214)

[5.2. Описание функций 4](#_Toc166892215)

[6. Рекомендации пользователю 5](#_Toc166892216)

[7. Рекомендации программисту 5](#_Toc166892217)

[8. Контрольный пример 6](#_Toc166892218)

[9. Сравнение каскадной и параллельной реализации 7](#_Toc166892219)

[10. Заключение 10](#_Toc166892220)

# **Цель работы**

Реализовать и исследовать алгоритм двухмерного канонического анализа корреляций (2D CCA) для двух наборов данных: каскадного и параллельного.

# **Задача**

Задачи работы включают в себя разработку и реализацию алгоритмов 2D CCA для каскадного и параллельного наборов данных, проведение процедуры РРПП, анализ корреляционных связей методами PLS и CCA, создание программы для взаимной реконструкции объектов, а также оценку производительности и качества алгоритмов. В результате будет выполнен сравнительный анализ результатов и сформулированы выводы о применимости и эффективности методов PLS и CCA.

# **Теоретическая часть**

*Каноническая корреляция (Canonical Correlation)*: Мера связи между двумя множествами переменных. Каноническая корреляция оценивает линейные зависимости между двумя множествами переменных и предоставляет значения корреляции, называемые каноническими корреляциями.

*Канонические переменные (Canonical Variables)*: Линейные комбинации исходных переменных, которые максимизируют каноническую корреляцию между двумя множествами переменных. Каждая каноническая переменная представляет собой линейную комбинацию переменных из каждого множества.

*2D ССА (Two-Dimensional Canonical Correlation Analysis)*: Метод анализа, который обобщает каноническую корреляцию на случай двухмерных данных. В 2D ССА исследуются корреляционные связи между двумя двумерными множествами данных.

*RRPP (Randomized Response Permutation Procedure)*: Метод случайного перемешивания и перестановки, применяемый для оценки статистической значимости корреляционных коэффициентов. RRPP позволяет оценить, насколько наблюдаемая корреляция значима в сравнении с корреляциями, которые могли бы возникнуть случайно.

*PLS (Partial Least Squares)*: Метод регрессии, используемый для моделирования зависимости между наборами переменных. PLS находит линейные комбинации исходных переменных (как входов), которые наилучшим образом объясняют целевые переменные (как выходы).

# **Описание алгоритма**

1. Загрузка изображений из указанной директории. Изображения предварительно обрабатываются для улучшения резкости и добавления гауссовского шума.
2. Тренировка моделей 2D CCA (2DCCA Cascade и 2DCCA Parallel).
3. Выполнение процедуры RRPP (Randomized Response Permutation Procedure) для оценки статистической значимости корреляционных коэффициентов.
4. Реконструкция объектов с использованием моделей PLS и CCA.
5. Анализ корреляционных связей в наборах данных с помощью методов PLS и CCA.
6. Вывод результатов работы алгоритма, включая тренировку моделей, выполнение RRPP, реконструкцию объектов и анализ корреляционных связей.

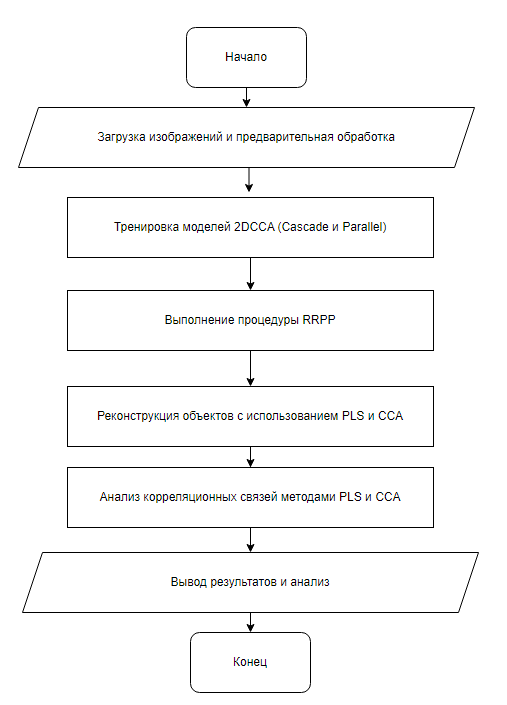


Рисунок 4.1. Блок-схема алгоритма.

# **Описание программы**

## **Описание классов**

В программе используется 7 классов: 1 связанный с интерфейсом программы, 6 связанный с поиском решения задачи. В таблице 5.1 представлено описание классов.

*Таблица 5.1. Описание классов*

|  |  |
| --- | --- |
| Имя класса | Описание |
| ImageLoader | Редактирование графа в интерфейсе |
| GUI | Графический пользовательский интерфейс |
| TwoDCCA\_Cascade | 2DCCA каскадный |
| TwoDCCA\_Parallel | 2DCCA параллельный |
| PLS\_Cascade | PLS каскадный |
| PLS\_Parallel | PLS параллельный |
| RPA | РРПП |

## **5.2. Описание функций**

Описание функций класса *«TwoDCCA\_Cascade»* представлено в таблице 5.2.

*Таблица 5.2. Описание функций класса «TwoDCCA\_Cascade»*

|  |  |  |
| --- | --- | --- |
| Входные данные | Имя | Описание |
| Количество канонических компонентов для вычисления и возврата | \_\_init\_\_ | Инициализация класса |
| Изображения в видимом и инфракрасном свете | fit | Тренировка алгоритма |
| Изображения в видимом и инфракрасном свете | transform | Преобразование набора данных в новое пространство |
| Изображения в видимом и инфракрасном свете | train | Тренировка модели |
| Папка для сохранения | save\_model | Сохранение модели |
| Папка, в которой находится модель | load\_model | Загрузка модели |

Полный код программы представлен в Приложении А.

# **Рекомендации пользователю**

Кнопка *«Выбрать»* отвечает за выбор папки, в которой хранится датасет с изображениями в видимом и инфракрасном свете.

Кнопка выбора метода отвечает за метод выполнения алгоритма.

Кнопка *«Тренировать»* отвечает за тренировку программы.

Кнопка *«Запустить»* отвечает за запуск программы.

Для завершения работы нажмите на крестик в левом верхнем углу.

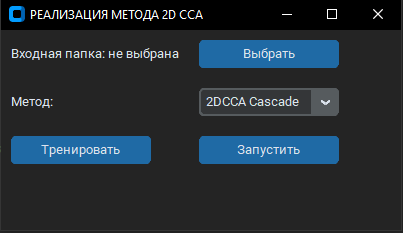
# **Рекомендации программисту**

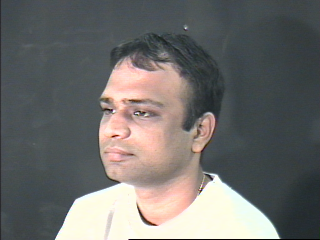
Для запуска программы необходим Python версии не ниже 3.10.6. Предварительно необходимо установить библиотеки: customtkinter версии не ниже 5.2.0, numpy версии не ниже 1.21.6, cv2 версии не ниже 4.5.5, sklearn версии не ниже 1.0.2.

Минимальное необходимое место на диске: 50 МБ. Минимальное необходимое количество оперативной памяти: 1000 МБ.

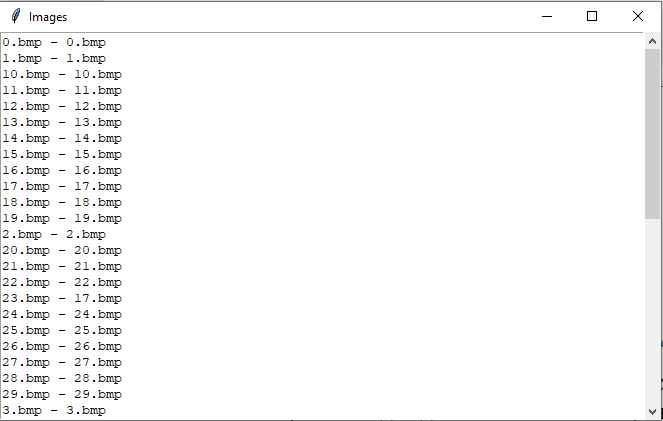
# **Контрольный пример**

В данном разделе представлен контрольный пример.

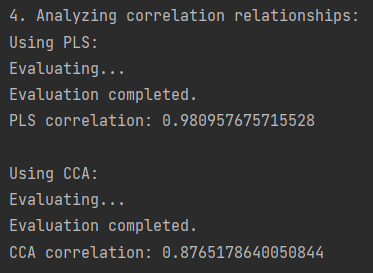


*Рисунок 8.1. Интерфейс программы*

*Рисунок 8.2–8.3. Пример входных данных*



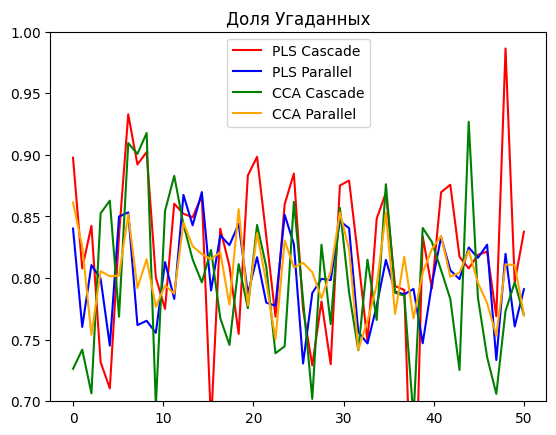
*Рисунок 8.4. Результат работы программы*



*Рисунок 8.5. Корректность работы программы*

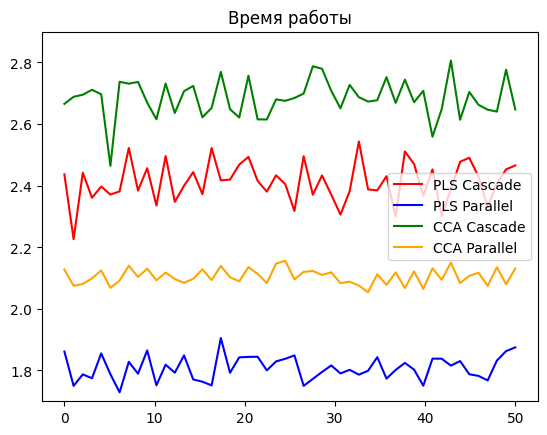
# **Сравнение каскадной и параллельной реализации**

Сравним каскадную и параллельную реализацию алгоритмов 2DCCA и 2DPLS. На рисунке 9.1 представлена доля правильно определенных изображений алгоритмом 2DPLS и 2DCCA в параллельной и каскадной реализации без применения РРПП.



*Рисунок 9.1. График результатов работы алгоритмов без применения РРПП*

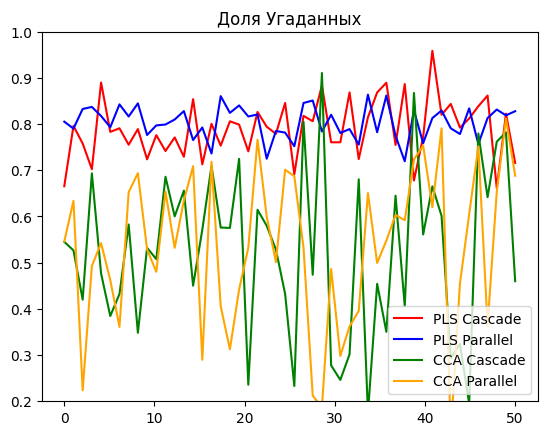
Из графика видно, что наиболее хорошо себя показывают каскадные реализации PLS и CCA. Однако PLS все же дает немного лучшие результаты. На рисунке 9.2 представлено время работы каждого из алгоритмов.



*Рисунок 9.2. Время работы алгоритмов*

Из графика видно, что наиболее быстрым алгоритмом является параллельный PLS. Это связано с тем, что ему требуется меньше вычислений по сравнению с CCA.

Теперь применим РРПП. Оптимальный количеством главных компонент является 10. На рисунке 9.3 представлена доля правильно определенных изображений алгоритмом 2DPLS и 2DCCA в параллельной и каскадной реализации с применением РРПП.



*Рисунок 9.3. График результатов работы алгоритмов с применение РРПП*

Из графика видно, что CCA сильно скачет в точности, при этом PLS показывает достаточно хорошие стабильные результаты.

В таблицах 9.4–9.5 приведены результаты исследования алгоритмов.

*Таблица 9.4. Среднее время работы каждого из алгоритмов*

|  |  |
| --- | --- |
| Алгоритм | Среднее время работы (сек) |
| CCA Cascade | 2.6 |
| CCA Parallel | 2.2 |
| PLS Cascade | 2.4 |
| PLS Parallel | 1.8 |

*Таблица 9.5. Средняя доля правильно классифицированных объектов алгоритмами без РРПП и с РРПП*

|  |  |  |
| --- | --- | --- |
|  | Средняя доля правильно классифицированных объектов | |
| Алгоритм | Без РРПП | С РРПП |
| CCA Cascade | 0.88 | 0.66 |
| CCA Parallel | 0.89 | 0.59 |
| PLS Cascade | 0.92 | 0.92 |
| PLS Parallel | 0.91 | 0.86 |

Результаты показывают, что для задачи обработки изображений лиц лучшим алгоритмом является PLS.

**Исследование корреляционных связей в наборах данных на основе методов PLS и CCA.**

Между могут образовываться устойчивые корреляционные связи. Проверим это для использованного нами датасета. В таблице 9.6 приведены средние значения корреляций матриц.

*Таблица 9.6. Средние значения корреляций матриц.*

|  |  |  |
| --- | --- | --- |
| Тип РРПП | CCA | PLS |
| Без РРПП | 0.00485 | -0.00039 |
| 10 | 0.06200 | -0.01270 |
| 50 | 0.01310 | -0.00140 |
| 100 | 0.00890 | 0.00092 |
| 150 | 0.00640 | 0.00037 |
| 200 | 0.00380 | 0.00039 |
| 250 | 0.00382 | 0.00039 |

Из таблицы видно, что корреляция между матрицами близка к 0. Чтобы объяснить это необходимо рассмотреть исходный датасет.

Между изображениями имеется существенная разница в контрасте изображения, небольшие отличия в положении головы и текстуре. Из-за чего корреляция между матрицами изображений близка к 0.

# **Заключение**

В ходе выполнения данной работы были изучены особенности алгоритмов 2DCCA и 2DPLS. Были реализованы четыре алгоритма, предназначенные для каскадного и параллельного набора данных. Также была разработана программа на языке Python, которая может сопоставлять входные данные с существующими в инфракрасном и видимом излучении.

Проведено сравнение алгоритмов 2DPLS и 2DCCA, в результате которого было установлено, что для задачи сопоставления изображений в инфракрасном и видимом свете более подходящим является метод 2DPLS. Также результаты показали, что 2DPLS гораздо устойчивее к редукции размерности пространства признаков по сравнению с 2DCCA, что может значительно ускорить работу алгоритма.

Исследование корреляционных связей показало, что из-за специфики датасета корреляция между двумя наборами данных близка к нулю как для алгоритма PLS, так и для CCA.

1. **Приложение А**

import numpy as np  
import cv2  
import os  
  
  
class ImageLoader:  
 def \_\_init\_\_(self, path: str):  
 *"""  
 Constructor for ImageLoader class  
  
 Parameters:  
 path (str): Path to the images  
  
 Returns:  
 None  
 """* self.path = path  
  
 @staticmethod  
 def sharpen\_image(image: np.ndarray) -> np.ndarray:  
 *"""  
 Sharpen the image.  
  
 Parameters:  
 image (np.ndarray): Input image.  
  
 Returns:  
 np.ndarray: Sharpened image.  
 """* # Define the sharpening kernel  
 sharpen\_filter = np.array([[-1, -1, -1],  
 [-1, 9, -1],  
 [-1, -1, -1]])  
  
 sharpened\_image = cv2.filter2D(image, -1, sharpen\_filter)  
 return sharpened\_image  
  
 @staticmethod  
 def add\_gaussian\_noise(image: np.ndarray, mean=0, sigma=25):  
 *"""  
 Add Gaussian noise to the image.  
  
 Parameters:  
 image (np.ndarray): Input image.  
 mean (float): Mean of the Gaussian distribution.  
 sigma (float): Standard deviation of the Gaussian distribution.  
  
 Returns:  
 np.ndarray: Image with added Gaussian noise.  
 """* row, col, ch = image.shape  
 gauss = np.random.normal(mean, sigma, (row, col, ch))  
 noisy\_image = np.clip(image + gauss, 0, 255).astype(np.uint8)  
 return noisy\_image  
  
 def load\_and\_preprocess(self) -> (np.ndarray, np.ndarray):  
 *"""  
 Load images from the specified path, preprocess them and align faces.  
  
 Returns:  
 Tuple of NumPy arrays: (regular\_images, infrared\_images)  
 """* regular\_images = []  
 infrared\_images = []  
  
 regular\_path = os.path.join(self.path, "regular")  
 infrared\_path = os.path.join(self.path, "infrared")  
  
 # Load and preprocess regular images  
 for filename in sorted(os.listdir(regular\_path)):  
 if filename.endswith(".bmp"):  
 image = cv2.imread(os.path.join(regular\_path, filename))  
 sharpened\_image = self.sharpen\_image(image)  
 noisy\_image = self.add\_gaussian\_noise(sharpened\_image)  
 regular\_images.append(noisy\_image)  
  
 # Load and preprocess infrared images  
 for filename in sorted(os.listdir(infrared\_path)):  
 if filename.endswith(".bmp"):  
 image = cv2.imread(os.path.join(infrared\_path, filename))  
 sharpened\_image = self.sharpen\_image(image)  
 noisy\_image = self.add\_gaussian\_noise(sharpened\_image)  
 infrared\_images.append(noisy\_image)  
  
 return np.array(regular\_images), np.array(infrared\_images)  
  
 @staticmethod  
 def save\_images(regular\_images: np.ndarray, infrared\_images: np.ndarray, output\_dir: str) -> None:  
 *"""  
 Save the processed images.  
  
 Parameters:  
 regular\_images (np.ndarray): Array of preprocessed regular images.  
 infrared\_images (np.ndarray): Array of preprocessed infrared images.  
 output\_dir (str): Output directory to save the images.  
  
 Returns:  
 None  
 """* os.makedirs(output\_dir, exist\_ok=True)  
  
 # Save regular images  
 for i, image in enumerate(regular\_images):  
 cv2.imwrite(os.path.join(output\_dir, f"regular\_{i}.bmp"), image)  
  
 # Save infrared images  
 for i, image in enumerate(infrared\_images):  
 cv2.imwrite(os.path.join(output\_dir, f"infrared\_{i}.bmp"), image)

*Листинг программы.*