Санкт-Петербургский государственный университет

Факультет прикладной математики – процессов управления

Лабораторная работа №5

**Отчет**

по дисциплине «Алгоритмы и структуры данных»

**Реализация метода 2DCCA в приложении к обработке изображений лиц**

Вариант 2

Автор работы: Дацык Р.В.

Группа: 22.Б15-пу

Преподаватель: Дик А. Г.

Санкт-Петербург, 2024

**Оглавление**

[**1.** **Цель работы** 3](#_Toc163080782)

[**2.** **Задачи** 3](#_Toc163080783)

[**3.** **Теоретическая часть** 3](#_Toc163080784)

[**4.** **Алгоритм метода** 4](#_Toc163080785)

[**5.** **Описание программы** 5](#_Toc163080786)

[**6.** **Рекомендации пользователю** 6](#_Toc163080787)

[**7.** **Рекомендации программиста** 7](#_Toc163080788)

[**8.** **Контрольный пример** 7](#_Toc163080789)

[**9.** **Вывод** 9](#_Toc163080790)

[**10.** **Список использованной литературы** 9](#_Toc163080791)

[**11.**  **Приложение А** 10](#_Toc163080792)

# **Цель работы**

Исследовать алгоритмы проекции цифровых изображений в собственные подпространства, используя линейные методы PCA, LDA, PLS и CCA. Реализовать каскадные и параллельные алгоритмы 2D CCA для набора изображений в форматах видимого света и инфракрасного излучения, выполнить взаимную реконструкцию пар семантически несвязанных объектов, используя методы PLS и CCA.

# **Задачи**

Для достижения цели поставлены следующие задачи:

1. Изучить основные принципы метода 2D CCA.
2. Собрать или подготовить набор данных в необходимом формате.
3. Реализовать метод 2D CCA.
4. Выполнить редукцию размерности признакового пространства (РРПП).
5. Разработать программу для реконструкции пар семантически несвязанных объектов с использованием методов PLS и CCA.
6. Провести исследование корреляционных связей в наборах данных с применением методов PLS и CCA.
7. Протестировать программу.
8. Проанализировать результаты.
9. Сделать выводы по проведенной работе.

# **Теоретическая часть**

2D Canonical Correlation Analysis (2D CCA) — это статистический метод, применяемый для исследования взаимосвязей между двумя наборами переменных. Он расширяет традиционный метод канонической корреляции (CCA) на двумерные данные. Основная цель 2D CCA заключается в нахождении линейных комбинаций переменных из обоих наборов, чтобы максимизировать корреляцию между их проекциями. Это позволяет определить двумерные направления в пространствах переменных, которые максимизируют корреляцию между ними.

Этот метод широко используется в различных областях, включая обработку сигналов, изображений, биоинформатику, финансовый анализ и другие сферы, где требуется исследовать связь между двумя наборами переменных.

2D Partial Least Squares (2D PLS) — метод множественной регрессии, используемый для поиска линейных зависимостей между двумя наборами переменных. Основная цель PLS — найти проекционные матрицы, которые преобразуют данные в собственное подпространство с максимальной ковариацией. После определения этих матриц выполняется редукция размерности признакового пространства, что уменьшает сложность данных и предотвращает переобучение модели.

Взаимная реконструкция в парах семантически несвязанных объектов — метод, используемый для анализа и взаимодействия между двумя различными наборами данных или объектами, не имеющими явной семантической связи. Этот метод помогает выявить скрытые структуры или паттерны, существующие между этими наборами данных.

# **Алгоритм метода**

На рисунке 4.1 изображен алгоритм для методов 2D CCA и 2D PLS.

В приложении А прикреплен листинг кода с подробными комментариями.

На рисунках 4.2-4.4 представлена блок-схема алгоритма.

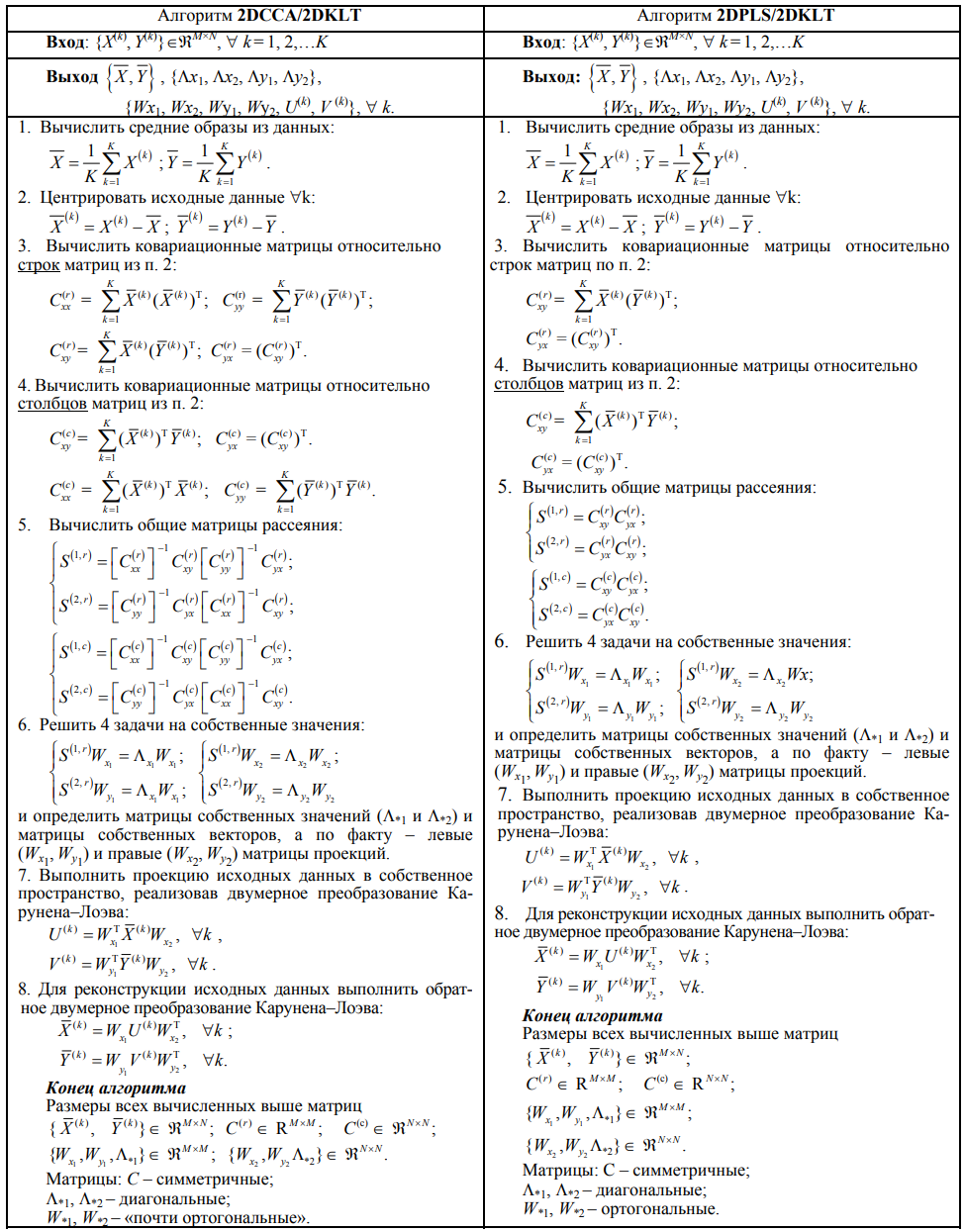


Рисунок 4.1. Алгоритм методов 2D CCA и 2D PLS

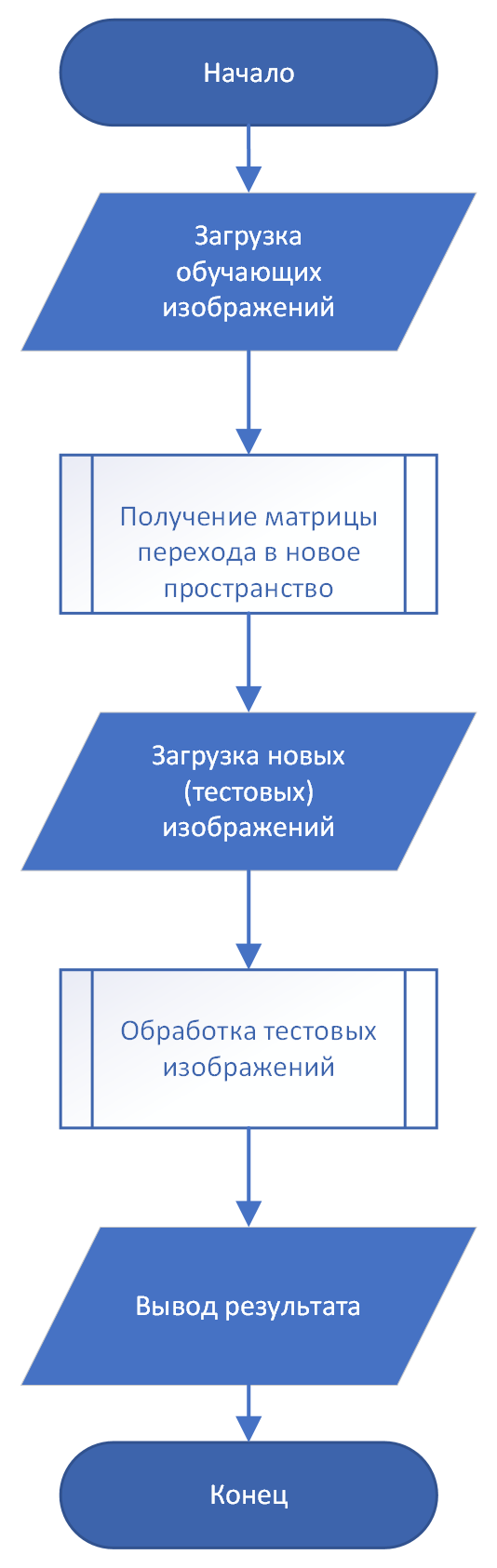


Рисунок 4.2. Блок-схема основной программы

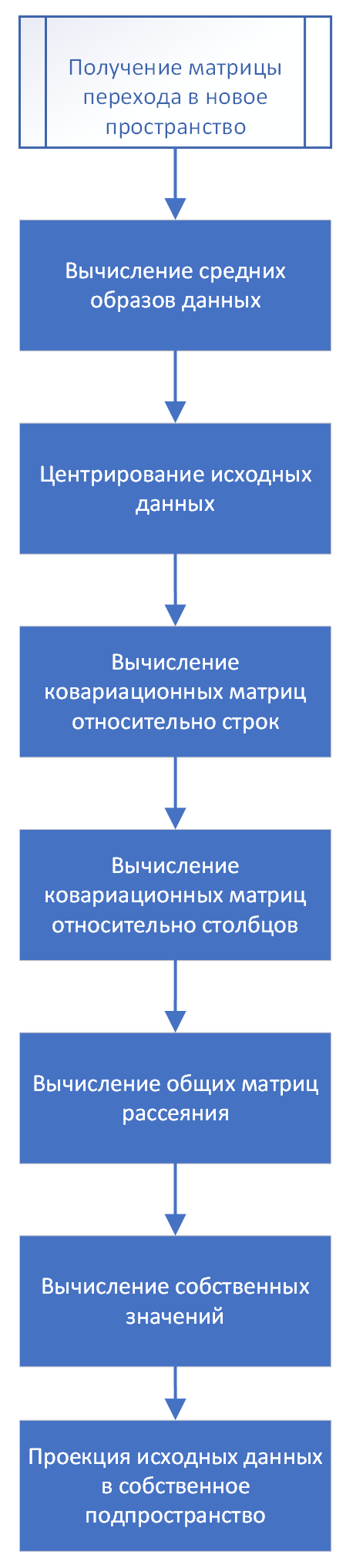


Рисунок 4.3 Блок-схема подпрограммы получения матриц перехода в новое пространство

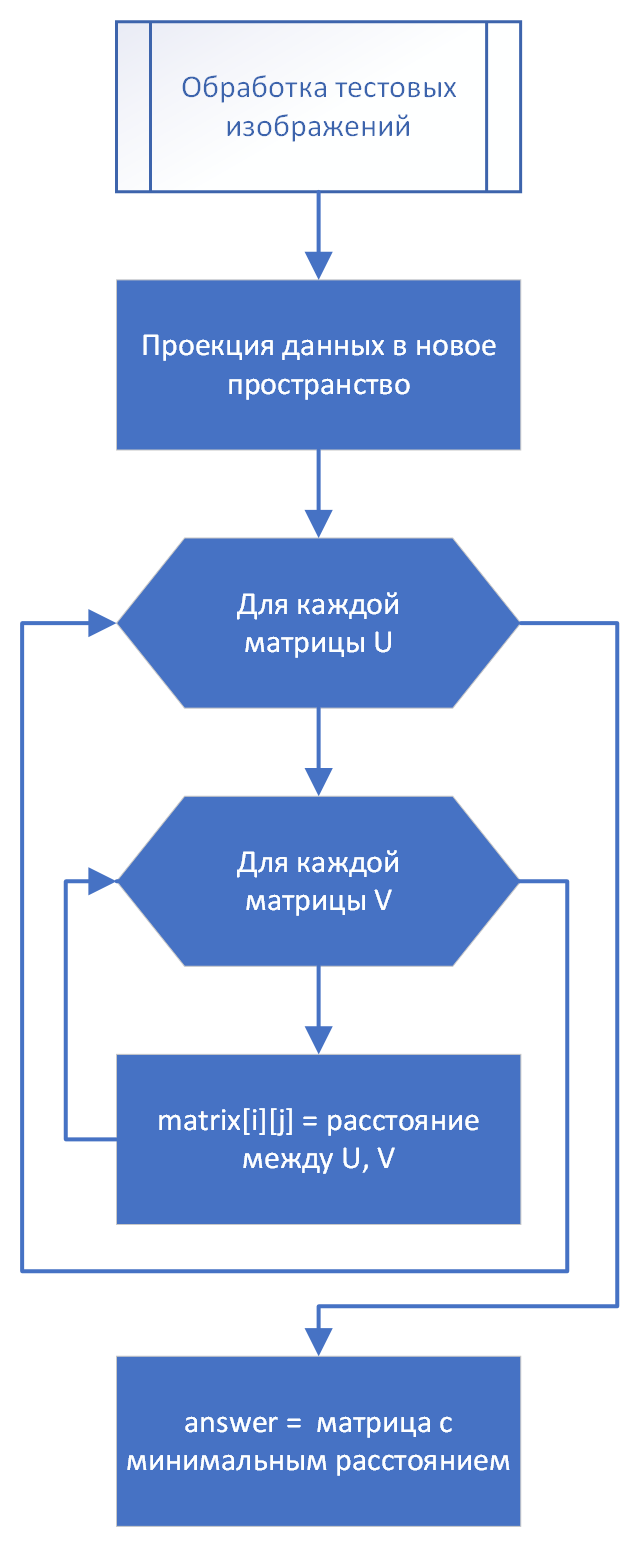


Рисунок 4.4 Блок-схема подпрограммы обработки тестовых изображений

# **Описание программы**

Программа реализована на языке python 3.10 с использованием следующих пакетов: numpy, cv2, matplotlib, os, sys, tkinter, multiprocessing.

В программе используются 3 класса и n функций: классы помогают в отображении интерфейса и реализации алгоритмов; функции нужны для возможности распараллеливания и загрузки изображений. В таблицах 5.1-5.2 представлено описание всех классов и функций.

Таблица 5.1. Описание функций класса 2D CCA/2D PLS.

|  |  |  |
| --- | --- | --- |
| Имя функции | Аргументы функции | Назначение |
| \_\_init\_\_ | self, num\_of\_components, X, Y | Инициализация класса |
| cov | self, A, B, row | Вычисление ковариационных матриц. В зависимости от значения row=True/False вычисляется по строке или столбцу |
| fit | self, X, Y | Обучение алгоритма на исходных данных |
| transform | self, X | Преобразование тестовых данных в новое пространство |
| predict | self, X | Предсказание для новых данных |
| multiproccesing | self, X | Параллельная обработка данных |

Таблица 5.2. Описание функций класса GUI.

|  |  |  |
| --- | --- | --- |
| Имя функции | Аргументы функции | Назначение |
| \_\_init\_\_ | self, title, algorithm | Инициализация класса |
| make\_widget | self | Создание основного виджета |
| add\_directory | self, role | Указание пути к папке (role=обучающий, тестовый) |
| start | self | Запуск алгоритма и вывод результатов |

# **Рекомендации пользователю**

Для успешного запуска программы необходимо устройство с операционной системой Linux, macOS или Windows, среда разработки, поддерживающая запуск python 3.10.

Интерфейс предлагает пользователю выбрать папки с обучающим и тестовым наборами данных.

Алгоритм запустится после нажатия кнопки «Запустить алгоритм». Пользователю выведется сообщение об успешном выполнении программы и результат работы.

# **Рекомендации программиста**

Для запуска программы необходима 64-битная операционная система Windows, Linux или macOS. Для работы с кодом необходима среда разработки, совместимая с python 3.10, библиотеки numpy, cv2, matplotlib, os, sys, tkinter, multiprocessing. Установка необходимых библиотек возможна с помощью ввода в консоль следующих команд:

pip install opencv-python

pip install matplotlib

pip install multiprocessing

pip install numpy

pip install tkinter

Исходный код программы доступен по ссылке:

<https://github.com/LisaNota/spbu-2DCCA>

# **Контрольный пример**

В данном разделе представлен контрольный пример, демонстрирующий работу программы.

Интерфейс при запуске изображен на рисунке 8.1.

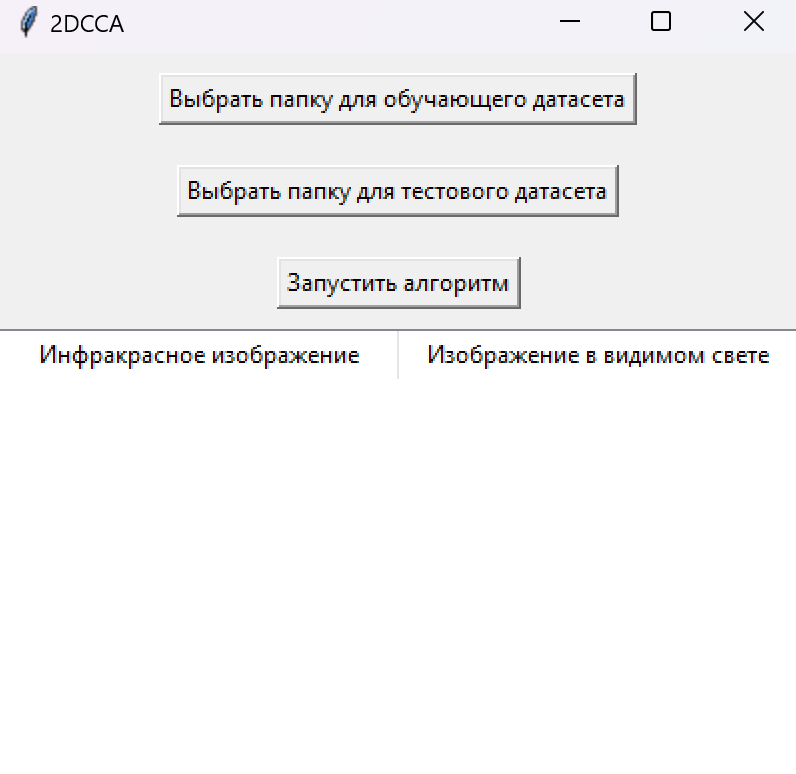


Рисунок 8.1. Интерфейс программы.

Выберем папки с помощью соответсвующих кнопок. После успешной загрузки кнопка «Запустить алгоритм» программа начнет свою работу.

Обучающие и тестовые изображения приведены на рисунках 8.2-8.3.





Рисунок 8.2. Пример изображений из обучающих данных





Рисунок 8.3. Пример изображений из тестовых данных

Результат работы программы изображен на рисунке 8.4.

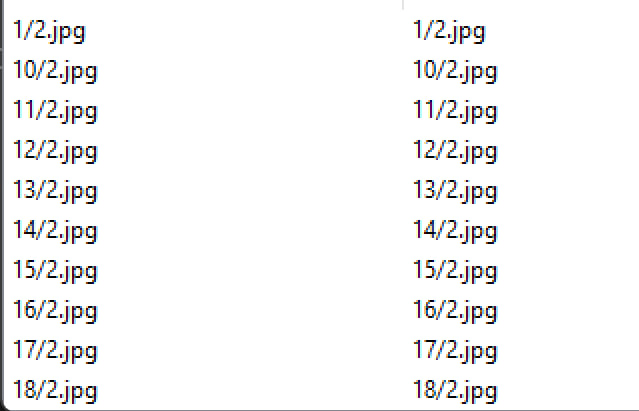


Рисунок 8.4. Результат работы программы

В результате работы программы сопоставляются изображения в видимом свете и в инфракрасном.

Сравним каскадную и параллельную реализации PLS и CCA, график представлен на рисунке 8.5.

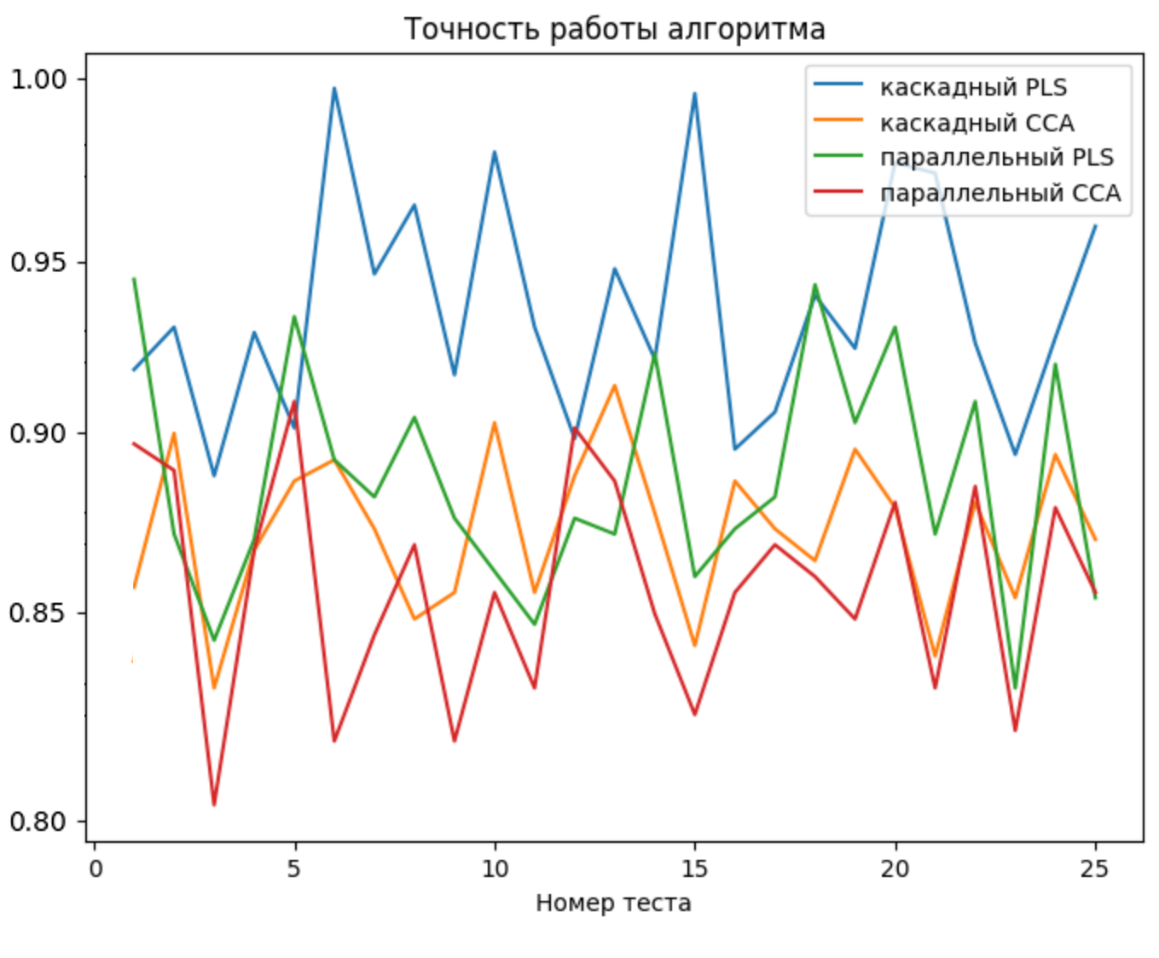


Рисунок 8.5. График точности работы алгоритмов.

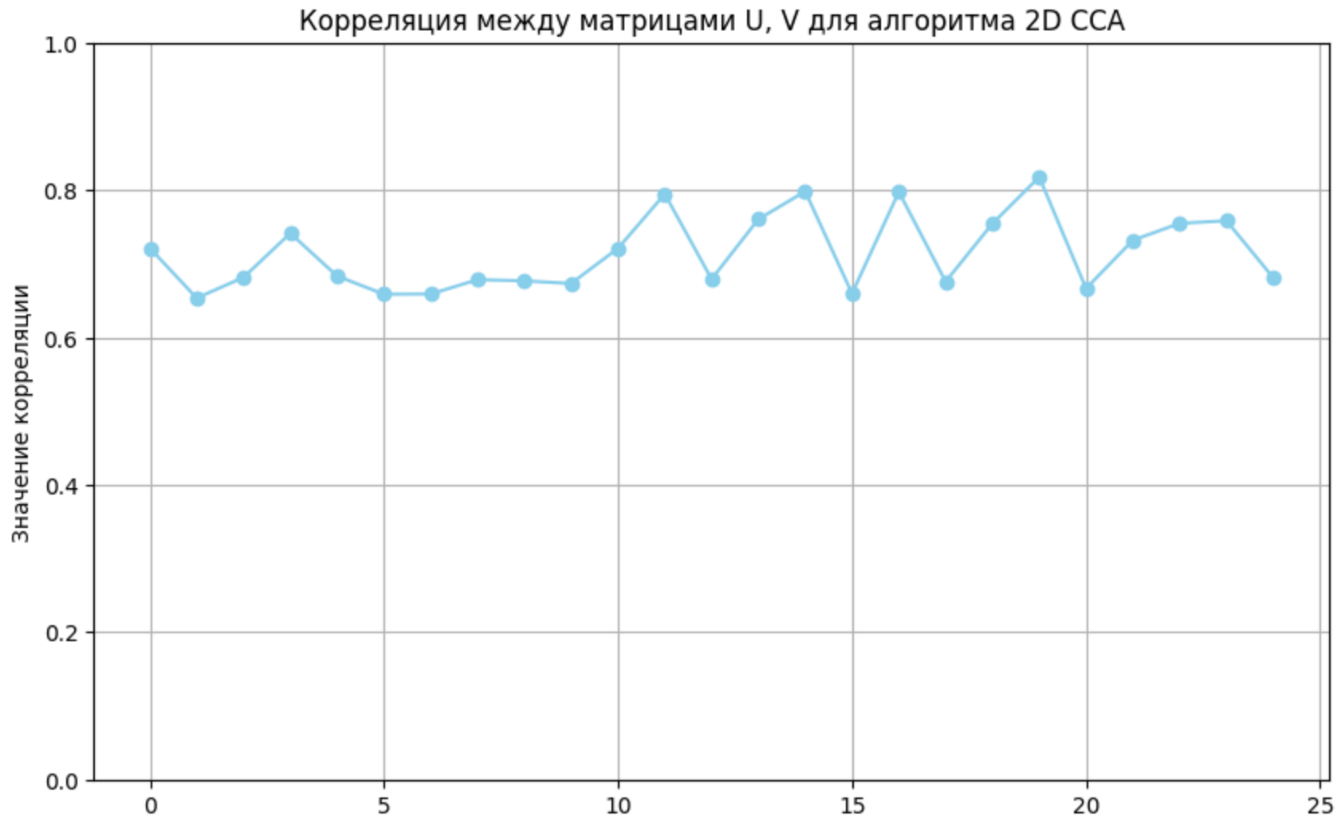
Проанализируем полученные данные. Из графика видно, что каскадный PLS показывает себя лучше других алгоритмов. Параллельный PLS и каскадный CCA показали примерно одинаковую точность, параллельный CCA отработал несколько хуже, хотя абсолютная точность всех алгоритмов не падала ниже 0.8.

Измерим время работы алгоритмов в таблице 8.1.

Таблица 8.1. Время работы алгоритмов.

|  |  |  |  |
| --- | --- | --- | --- |
| Каскадный CCA | Параллельный CCA | Каскадный PLS | Параллельный PLS |
| 3.3 сек | 2.7 сек | 3.2 сек | 2.6 сек |

Результаты показывают, что алгоритм PLS в среднем быстрее, чем алгоритм CCA. При этом в обоих алгоритмах параллельная версия значительно быстрее каскадной.

 Исследуем корреляционные связи в наборах данных. Для этого построим график корреляций среди матриц U, V. Результат изображен на рисунке 8.6.

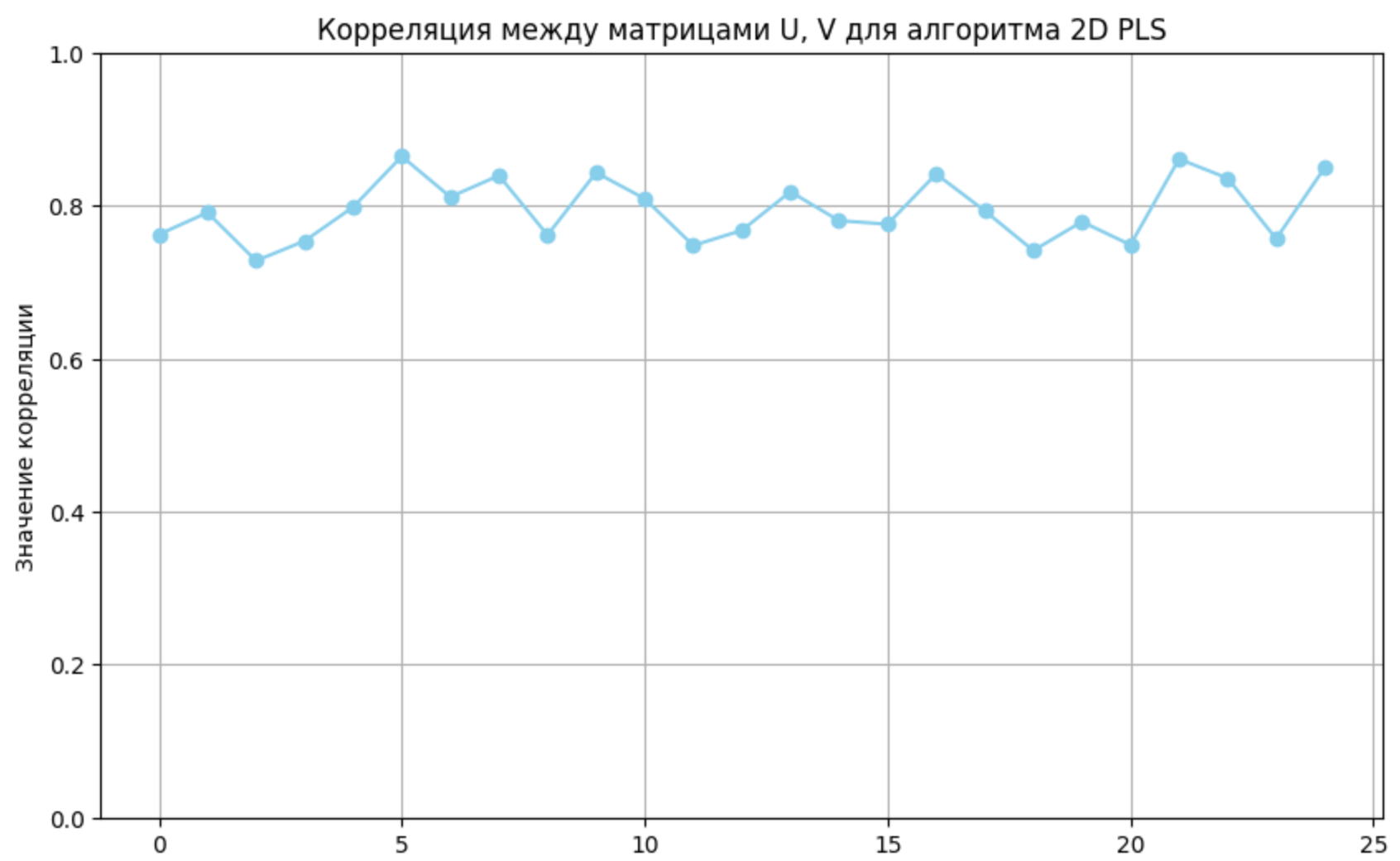


Рисунок 8.6. График корреляций матриц U, V.

Графики показывают, что алгоритм PLS эффективнее для выполнения задачи реконструкции связей семантически несвязанных объектов.

# **Вывод**

В ходе выполнения данной работы изучены особенности алгоритмов 2D CCA и 2D PLS, реализована программа, позволяющая с помощью каскадных и параллельных алгоритмов сопоставить изображения в инфракрасном и видимом свете.

Проведено исследование алгоритмов, их сравнение по точности и времени. Были исследованы корреляционные связи в наборах данных.

# **Список использованной литературы**

1. Г.А. Кухарев, Е.И. Каменская, Ю.Н. Матвеев, Л.Н. Щеголева «Методы обработки и распознавания изображения лиц в задачах биометрии» - 2013
2. Г.А. Кухарев, Л.Н. Щеголева «Методы двумерной проекции цифровых изображений в собственные пространства: особенности реализации и применение» // Компьютерная оптика. - 2018. - №4
3. Статья о работе с tkinter: <https://pythonru.com/uroki/obuchenie-python-gui-tkinter>

# **Приложение А**

**class** **CCA**(Algorithm):

"""

num\_of\_components (int): количество главных компонент в PCA

X, Y: два набора исходных данных, состоящие из K матриц

"""

**def** **\_\_init\_\_**(self, num\_of\_components:int, X: list, Y: list) -> **None**:

"""

distance (int): расстояние между матрицами

bar\_X, bar\_Y: средние образы из данных

"""

self.distance = np.linalg.norm(X - Y)

self.num\_of\_components = num\_of\_components

self.bar\_X, self.bar\_Y = **None**, **None**

self.reg\_coef\_C = 10\*\*(-4) *# Регуляризационный коэффициент для матриц корреляции*

self.reg\_coef\_S = 5 \* 10\*\*(-4) *# Регуляризационный коэффициент для матриц сингулярных значений*

self.U = **None**

self.V = **None**

self.W = **None**

**def** **cov**(self, A: np.ndarray, B: np.ndarray, row: bool) -> np.ndarray:

"""

type (str): row, column - вычисление ковариации по строке/столбцу

"""

**if** row:

C = np.sum([A[i] @ B[i].T **for** i **in** range(len(A))], axis=0)

**else**:

C = np.sum([A[i].T @ B[i] **for** i **in** range(len(A))], axis=0)

**return** C

**def** **transform**(self, matrix\_set: np.ndarray, isX: bool) -> np.ndarray:

result = list()

**for** i **in** range(len(matrix\_set)):

**if** isX:

**for** i **in** range(len(matrix\_set)):

result.append(self.W["x"][0].T @ (matrix\_set[i] - self.bar\_X) @ self.W["x"][1])

**else**:

**for** i **in** range(len(matrix\_set)):

result.append(self.W["y"][0].T @ (matrix\_set[i] - self.bar\_Y) @ self.W["y"][1])

**return** np.array(result)

**def** **fit**(self, X: list, Y:list) -> **None**:

"""

обучение алгоритма на исходных данных

"""

*# вычисление средних образов*

self.bar\_X = np.mean(X, axis=0)

self.bar\_Y = np.mean(Y, axis=0)

*# централизация*

X = X - self.bar\_X

Y = Y - self.bar\_Y

*# корреляции по строкам и столбцам*

matrices = [(X, X), (Y, Y), (X, Y), (Y, X)]

variables = [(**True**, **True**), (**True**, **True**), (**True**, **False**), (**True**, **False**)]

*# ковариации*

covarianсу\_matrices = []

**for** data, r\_flag **in** zip(matrices, variables):

covarianсу\_matrices.append(self.cov(data[0], data[1], row=r\_flag[0]))

covarianсу\_matrices.append(self.cov(data[1], data[0], row=r\_flag[1]))

C\_rxx, C\_ryy, C\_rxy, C\_ryx, C\_cxx, C\_cyy, C\_cxy, C\_cyx = covarianсу\_matrices

*# регуляризация*

covarianсу\_matrices = [C\_rxx, C\_ryy, C\_cxx, C\_cyy]

**for** i **in** range(len(covarianсу\_matrices)):

covarianсу\_matrices[i] += self.reg\_coef\_C \* np.identity(covarianсу\_matrices[i].shape[0])

C\_rxx, C\_ryy, C\_cxx, C\_cyy = covarianсу\_matrices

*# Вычисление матриц сингулярных значений*

S\_matrices = []

covarianсу\_matrices = [C\_rxx, C\_ryy, C\_cxx, C\_cyy]

cross\_covarianсу\_matrices = [C\_rxy, C\_ryx, C\_cxy, C\_cyx]

**for** i **in** range(2):

**for** j **in** range(2):

S\_matrices.append(np.linalg.inv(covarianсу\_matrices[i\*2+j]) @ cross\_covarianсу\_matrices[i\*2+j] @ np.linalg.inv(covarianсу\_matrices[(i+1)%2\*2+(j+1)%2]) @ cross\_covarianсу\_matrices[(i+1)%2\*2+(j+1)%2])

*# Регуляризация матриц сингулярных значений*

**for** matrix **in** S\_matrices:

matrix += self.reg\_coef\_S \* np.identity(matrix.shape[0])

*# Вычисление собственных значений и векторов сингулярных значений*

eigenvalues = []

eigenvectors = []

**for** matrix **in** S\_matrices:

l, v = np.linalg.eig(matrix)

eigenvalues.append(l)

eigenvectors.append(v)

l\_1r, l\_1c, l\_2r, l\_2c = eigenvalues

*# Нормализация векторов сингулярных значений*

**for** i **in** range(len(eigenvalues)):

sorted\_indices = eigenvalues[i].argsort()[::-1]

eigenvectors[i] = eigenvectors[i][:, sorted\_indices] / np.linalg.norm(eigenvectors[i][:, sorted\_indices])

v\_x1, v\_y1, v\_x2, v\_y2 = eigenvectors

self.W = {"x": [**None**, **None**],"y": [**None**, **None**]}

self.W["x"][0] = v\_x1.T[:, l\_1r.argsort()[-self.num\_of\_components:][::-1]]

self.W["x"][1] = v\_x2.T[:, l\_1c.argsort()[-self.num\_of\_components:][::-1]]

self.W["y"][0] = v\_y1.T[:, l\_2r.argsort()[-self.num\_of\_components:][::-1]]

self.W["y"][1] = v\_y2.T[:, l\_2c.argsort()[-self.num\_of\_components:][::-1]]

*# Преобразование данных X и Y*

self.U = self.transform(X + self.X\_c, isX=**True**)

self.V = self.transform(Y + self.Y\_c, isX=**False**)

**def** **predict**(self, matrixes: list, isX: bool = True) -> tuple[int, float]:

*# Преобразование входных данных с использованием метода transform*

matrixes\_transform = self.transform(matrixes, isX=isX)

*# Выбор обучающих данных в зависимости от типа входных данных (X или Y)*

matrixes\_train = self.U **if** isX **else** self.V

*# Инициализация матрицы расстояний между преобразованными входными данными и обучающими данными*

matrix\_distantion = np.zeros((len(matrixes), len(matrixes\_train)), dtype="float64")

*# Вычисление расстояний между каждой парой преобразованных входных данных и обучающих данных*

**for** udx, m\_transform **in** enumerate(matrixes\_transform):

**for** vdx, m\_train **in** enumerate(matrixes\_train):

*# Использование пользовательской функции расстояния для вычисления расстояния между векторами*

distantion = self.distantion(m\_transform, m\_train)

matrix\_distantion[udx][vdx] = distantion

*# Выбор индекса и значения расстояния в зависимости от флага isMax*

results = list()

**for** vector **in** matrix\_distantion:

**if** self.isMax:

results.append([vector.argmax(), vector.max()])

**else**:

results.append([vector.argmin(), vector.min()])

**return** results

Листинг 4.1. Класс 2D CCA

**Class** **PLS**(Algorithm):

"""

num\_of\_components (int): количество главных компонент в PCA

X, Y: два набора исходных данных, состоящие из K матриц

"""

**def** **\_\_init\_\_**(self, num\_of\_components:int, X: list, Y: list) -> **None**:

"""

distance (int): расстояние между матрицами

bar\_X, bar\_Y: средние образы из данных

"""

self.distance = np.linalg.norm(X - Y)

self.num\_of\_components = num\_of\_components

self.bar\_X, self.bar\_Y = **None**, **None**

self.reg\_coef\_C = 10\*\*(-4) *# Регуляризационный коэффициент для матриц корреляции*

self.reg\_coef\_S = 5 \* 10\*\*(-4) *# Регуляризационный коэффициент для матриц сингулярных значений*

self.U = **None**

self.V = **None**

self.W = **None**

**def** **cov**(self, A: np.ndarray, B: np.ndarray, row: bool) -> np.ndarray:

"""

type (str): row, column - вычисление ковариации по строке/столбцу

"""

**if** row:

C = np.sum([A[i] @ B[i].T **for** i **in** range(len(A))], axis=0)

**else**:

C = np.sum([A[i].T @ B[i] **for** i **in** range(len(A))], axis=0)

**return** C

**def** **transform**(self, matrix\_set: np.ndarray, isX: bool) -> np.ndarray:

result = list()

**for** i **in** range(len(matrix\_set)):

**if** isX:

**for** i **in** range(len(matrix\_set)):

result.append(self.W["x"][0].T @ (matrix\_set[i] - self.bar\_X) @ self.W["x"][1])

**else**:

**for** i **in** range(len(matrix\_set)):

result.append(self.W["y"][0].T @ (matrix\_set[i] - self.bar\_Y) @ self.W["y"][1])

**return** np.array(result)

**def** **fit**(self, X: list, Y:list) -> **None**:

"""

обучение алгоритма на исходных данных

"""

*# вычисление средних образов*

self.bar\_X = np.mean(X, axis=0)

self.bar\_Y = np.mean(Y, axis=0)

*# централизация*

X = X - self.bar\_X

Y = Y - self.bar\_Y

*# корреляции по строкам и столбцам*

matrices = [(X, X), (Y, Y), (X, Y), (Y, X)]

variables = [(**True**, **True**), (**True**, **True**), (**True**, **False**), (**True**, **False**)]

*# ковариации*

covarianсу\_matrices = []

**for** data, r\_flag **in** zip(matrices, variables):

covarianсу\_matrices.append(self.cov(data[0], data[1], row=r\_flag[0]))

covarianсу\_matrices.append(self.cov(data[1], data[0], row=r\_flag[1]))

C\_rxx, C\_ryy, C\_rxy, C\_ryx, C\_cxx, C\_cyy, C\_cxy, C\_cyx = covarianсу\_matrices

*# регуляризация*

covarianсу\_matrices = [C\_rxx, C\_ryy, C\_cxx, C\_cyy]

**for** i **in** range(len(covarianсу\_matrices)):

covarianсу\_matrices[i] += self.reg\_coef\_C \* np.identity(covarianсу\_matrices[i].shape[0])

C\_rxx, C\_ryy, C\_cxx, C\_cyy = covarianсу\_matrices

*# Вычисление матриц сингулярных значений*

S\_matrices = []

covarianсу\_matrices = [C\_rxx, C\_ryy, C\_cxx, C\_cyy]

cross\_covarianсу\_matrices = [C\_rxy, C\_ryx, C\_cxy, C\_cyx]

**for** i **in** range(2):

**for** j **in** range(2):

S\_matrices.append(np.linalg.inv(covarianсу\_matrices[i\*2+j]) @ cross\_covarianсу\_matrices[i\*2+j] @ np.linalg.inv(covarianсу\_matrices[(i+1)%2\*2+(j+1)%2]) @ cross\_covarianсу\_matrices[(i+1)%2\*2+(j+1)%2])

*# Регуляризация матриц сингулярных значений*

**for** matrix **in** S\_matrices:

matrix += self.reg\_coef\_S \* np.identity(matrix.shape[0])

*# Вычисление собственных значений и векторов сингулярных значений*

eigenvalues = []

eigenvectors = []

**for** matrix **in** S\_matrices:

l, v = np.linalg.eig(matrix)

eigenvalues.append(l)

eigenvectors.append(v)

l\_1r, l\_1c, l\_2r, l\_2c = eigenvalues

*# Нормализация векторов сингулярных значений*

**for** i **in** range(len(eigenvalues)):

sorted\_indices = eigenvalues[i].argsort()[::-1]

eigenvectors[i] = eigenvectors[i][:, sorted\_indices] / np.linalg.norm(eigenvectors[i][:, sorted\_indices])

v\_x1, v\_y1, v\_x2, v\_y2 = eigenvectors

self.W = {"x": [**None**, **None**],"y": [**None**, **None**]}

self.W["x"][0] = v\_x1.T[:, l\_1r.argsort()[-self.num\_of\_components:][::-1]]

self.W["x"][1] = v\_x2.T[:, l\_1c.argsort()[-self.num\_of\_components:][::-1]]

self.W["y"][0] = v\_y1.T[:, l\_2r.argsort()[-self.num\_of\_components:][::-1]]

self.W["y"][1] = v\_y2.T[:, l\_2c.argsort()[-self.num\_of\_components:][::-1]]

*# Преобразование данных X и Y*

self.U = self.transform(X + self.X\_c, isX=**True**)

self.V = self.transform(Y + self.Y\_c, isX=**False**)

**def** **predict**(self, matrixes: list, isX: bool = True) -> tuple[int, float]:

*# Преобразование входных данных с использованием метода transform*

matrixes\_transform = self.transform(matrixes, isX=isX)

*# Выбор обучающих данных в зависимости от типа входных данных (X или Y)*

matrixes\_train = self.U **if** isX **else** self.V

*# Инициализация матрицы расстояний между преобразованными входными данными и обучающими данными*

matrix\_distantion = np.zeros((len(matrixes), len(matrixes\_train)), dtype="float64")

*# Вычисление расстояний между каждой парой преобразованных входных данных и обучающих данных*

**for** udx, m\_transform **in** enumerate(matrixes\_transform):

**for** vdx, m\_train **in** enumerate(matrixes\_train):

*# Использование пользовательской функции расстояния для вычисления расстояния между векторами*

distantion = self.distantion(m\_transform, m\_train)

matrix\_distantion[udx][vdx] = distantion

*# Выбор индекса и значения расстояния в зависимости от флага isMax*

results = list()

**for** vector **in** matrix\_distantion:

**if** self.isMax:

results.append([vector.argmax(), vector.max()])

**else**:

results.append([vector.argmin(), vector.min()])

**return** results

Листинг 4.2. Класс 2D PLS

**import** sys

sys.path.append('C:/Projects/2DCCA')

**import** tkinter **as** tk

**from** tkinter **import** ttk

**from** tkinter **import** filedialog

**from** tkinter.messagebox **import** askyesno

**from** tkinter.simpledialog **import** askstring

**class** **GUI**(tk.Tk):

**def** **\_\_init\_\_**(self, title, algo: Algorithm):

tk.Tk.\_\_init\_\_(self)

self.title(title)

self.algo = algo(distantion=Correlatin.distantion, isMax=**False**)

self.directories = {"train": **None**, "test": **None**}

self.create\_widgets()

**def** **create\_widgets**(self):

self.train\_button = tk.Button(self, text="Выбрать папку для обучающего датасета", command=**lambda**: self.add\_directory("train"))

self.train\_button.pack(pady=10)

self.test\_button = tk.Button(self, text="Выбрать папку для тестового датасета", command=**lambda**: self.add\_directory("test"))

self.test\_button.pack(pady=10)

self.run\_button = tk.Button(self, text="Запустить алгоритм", command=self.start\_algorithm)

self.run\_button.pack(pady=10)

self.tree = ttk.Treeview(self, columns=("infrared", "visible"), show="headings")

self.tree.heading("visible", text="Изображение в видимом свете")

self.tree.heading("infrared", text="Инфракрасное изображение")

self.tree.pack(side="top", fill="both", expand=**True**)

**def** **add\_directory**(self, role: str) -> **None**:

file\_path = filedialog.askdirectory()

self.directories[role] = file\_path

**def** **add\_row**(self, value1, value2):

self.tree.insert("", "end", values=(value1, value2))

**def** **start\_algorithm**(self):

links\_train = Images.get\_links\_for\_gui(self.directories["train"])

X = Images.get\_pictures(links\_train[0])

Y = Images.get\_pictures(links\_train[1])

count\_RRPP = int(askstring(title="РРПП", prompt="Количество компонент для РРПП"))

self.algo.d = count\_RRPP

self.algo.fit(X, Y, withRRPP=**True**)

isX = askyesno(title="Определение", message="Инфракрасное -> видимый свет")

"{visible\_link.split('/')[-2]}/{visible\_link.split('/')[-1]}")

Листинг 4.3. Класс GUI