МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования «Южно-Уральский государственный университет»

(национальный исследовательский университет)

Высшая школа электроники и компьютерных наук

Кафедра системного программирования

Обработка и анализ изображений на Python

(название практической работы)

ОТЧЕТ

по практической работе № 3

по дисциплине «Интеллектуальные системы и технологии»

Выполнил:

студент группы КЭ–401

/ Д.А. Иванов /

(подпись)

« » 2024 г.

Проверил:

преподаватель кафедры СП

/ С.М. Абдуллаев /

(подпись)

« » 2024 г.

Оглавление

[Введение 3](#_Toc179658834)

[1. Обработка изображений на Python 4](#_Toc179658835)

[2. Анализ изображений 8](#_Toc179658836)

[библиографический список 12](#_Toc179658837)

# Введение

Обработка и анализ изображений являются важными областями современной компьютерной науки и информационных технологий. С развитием цифровых устройств и увеличением объемов визуальных данных обработка изображений стала неотъемлемой частью различных индустрий, включая медицину, промышленность, безопасность и развлекательный сектор.

Python, благодаря своей простоте и богатому набору библиотек, зарекомендовал себя как один из ведущих языков программирования для задач обработки изображений. Инструменты, такие как OpenCV, Pillow, scikit-image и TensorFlow, позволяют разработчикам эффективно реализовывать как простые, так и сложные алгоритмы обработки и анализа изображений.

Цель данной работы — исследовать методы и технологии обработки и анализа изображений с использованием Python. В рамках исследования будут рассмотрены основные концепции этой области, изучены возможности ключевых библиотек и фреймворков, а также проведены практические эксперименты, демонстрирующие применение этих инструментов в решении реальных задач.

# Обработка изображений на Python

Обработка изображений относится к различным методам, которые позволяют компьютерам понимать и изменять цифровые изображения. Он включает в себя анализ пиксельной информации для выполнения таких операций, как идентификация объектов, обнаружение краев, регулировка яркости/контрастности, применение фильтров, распознавание текста и т. д.

Python — популярный язык для обработки изображений благодаря обширным библиотекам, простому синтаксису и активному сообществу разработчиков. Ключевые библиотеки, такие как OpenCV, PIL/Pillow, scikit-image и другие, позволяют работать с изображениями в Python.

Обработка изображений основана на анализе пиксельных данных из цифровых изображений для идентификации и изменения элементов в них. Ключевые понятия включают в себя:

1. Получение изображения: Захват или импорт изображений с помощью камер, сканеров и т.д.
2. Предварительная обработка: Преобразование изображений перед анализом (изменение размера, поворот, удаление шумов и т.д.).
3. Обнаружение функций: Определение пикселей/областей интереса, таких как края, углы или объекты.
4. Анализ: Извлечение значимой информации из изображений с помощью обнаруженных признаков.
5. Манипуляция: Трансформация изображений на основе извлеченной информации (фильтрация, морфинг и т.д.).

**Основы работы с изображениями в Python**

Для работы с изображениями была выбрана библиотека OpenCV. Далее будут представлены различные функции этой библиотеки по обработке изображений.

Прежде всего необходимо подключить библиотеку и загрузить изображение. На листинге 1 представлен соответствующий код.

Листинг 1 – Импорт библиотеки и загрузка изображения

import cv2

img = cv2.imread('image.jpg')

Библиотека OpenCV предоставляет простые методы для работы с изображениями, вот некоторые из них:

1. для сохранения образа на диск после обработки, есть функция cv2.imwrite();
2. для изменения размера изображений – функция cv2.resize();
3. для поворота изображений – функция cv2.rotate.

Также при обработке изображений может понадобиться применение различных эффектов.

Применение эффектов размытия может быть полезно для уменьшения шума изображения. OpenCV предоставляет несколько техник размытия:

1. Линейные фильтры - Простое усреднение окрестностей пикселей. Легко наносится, но дает неестественный вид.
2. Размытие по Гауссу — использует ядро Гаусса для создания более естественного размытия. Регулируемый размер ядра позволяет контролировать интенсивность размытия. Полезно для сглаживания шума с сохранением краев.

Это сглаживает изображение, избегая при этом артефактов искажения.

На листинге 2 представлен пример применения размытия по Гауссу 15x15 в OpenCV Python.

Листинг 2 – Размытие по Гауссу

image = cv2.imread('image.jpg')

blurred = cv2.GaussianBlur(image, (15, 15), 0)

cv2.imwrite('blurred.jpg', blurred)

Повышение резкости позволяет лучше сфокусировать изображения. Сверточные фильтры подчеркивают грани и мелкие детали.

Некоторые варианты фильтра повышения резкости OpenCV:

1. Маскировка нерезкости — повышает контраст краев для воспринимаемой резкости.
2. Фильтры Лапласа — обнаруживает быстрые изменения значений пикселей для выделения краев.
3. Фильтры высоких частот - сохраняют детали высоких частот при подавлении низких частот.

Это позволяет выявить более мелкие детали для повышения четкости.

На листинге 3 представлен пример маскировки нерезкости в OpenCV:

Листинг 3 – Маскировка нерезкости

import cv2

import numpy as np

image = cv2.imread('image.jpg')

kernel = np.array([[0, -1, 0],

[-1, 5,-1],

[0, -1, 0]])

sharpened = cv2.filter2D(image, -1, kernel)

При обработке изображений довольно важным этапом является обнаружение краев. Для этих целей широко используется алгоритм Canny. Он применяет гауссово сглаживание для уменьшения шума, вычисляет градиенты интенсивности для выделения краев, а затем подавляет слабые или разъединенные края.

На листинге 4 представлен пример использования этого алгоритма.

Листинг 4 – Алгоритм Canny

image = cv2.imread('image.jpg')

edges = cv2.Canny(image, 100, 200)

cv2.imwrite('canny\_edges.jpg', edges)

Также необходимо использовать методы сегментации, например Thresholding Techniques (методы установления пороговых значений). Этот метод преобразует изображение в оттенках серого в двоичное изображение, устанавливая значения пикселей выше порогового значения на белый и значения ниже на черный. При этом изображение разделяется на области переднего плана и заднего плана.

Это были основные методы обработки изображений, предоставляемые библиотекой OpenCV.

**Подготовка набора данных**

Изучив основные методы обработки изображений, появляется возможность самостоятельно подготовить набор данных для классификации. Рассмотрим пример для распознания рукописных цифр.

Можно написать множество цифры и сделать фотографию каждой. Далее, чтобы разделить все цифры и привести их к одному виду можно сделать следующие преобразования:

1. с помощью специальных алгоритмов определить края цифр на изображении;
2. определить координаты крайних точек изображения и кадрировать изображение по этим координатам;
3. привести все изображения к одному размеру.

После этих шагов можно приступать к обучению классификатора на подготовленном наборе данных.

# Анализ изображений

В этом примере показано, как scikit-learn может быть использован для распознавания изображений рукописных цифр от 0 до 9.

На листинге 5 представлен код для подключения необходимых библиотек.

Листинг 5 – Импорт необходимых библиотек

import matplotlib.pyplot as plt

from sklearn import datasets, metrics, svm

from sklearn.model\_selection import train\_test\_split

Набор данных о цифрах состоит из изображений цифр размером 8x8 пикселей. Атрибут dataset.images хранит массивы 8x8 значений оттенков серого для каждого изображения. Мы будем использовать эти массивы для визуализации первых 4 изображений. Атрибут dataset.target хранит цифру, которую представляет каждое изображение, и она включена в заголовок 4 графиков, изображенных на рисунке 1. На листинге 6 представлен соответствующий код.

Листинг 6 – Вывод изображений с заголовками

digits = datasets.load\_digits()

\_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))

for ax, image, label in zip(axes, digits.images, digits.target):

ax.set\_axis\_off()

ax.imshow(image, cmap=plt.cm.gray\_r, interpolation="nearest")

ax.set\_title("Training: %i" % label)

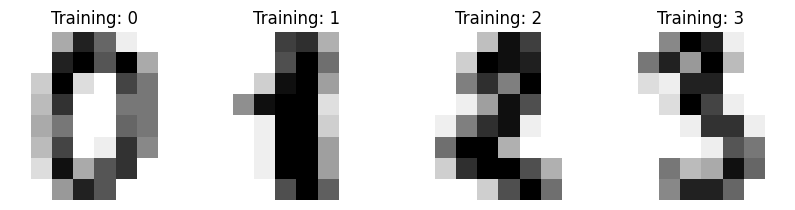


Рисунок 1 – Вывод изображений с заголовками

Чтобы применить классификатор к этим данным, нам нужно сгладить изображения, превратив каждый двумерный массив значений оттенков серого из формы (8, 8) в форму (64,). Впоследствии весь набор данных будет иметь форму (n\_samples, n\_features), где n\_samples — количество изображений, а n\_features — общее количество пикселей в каждом изображении.

Затем мы можем разделить данные на подмножества обучения и тестирования и подогнать классификатор опорных векторов к выборкам поездов. Подогнанный классификатор впоследствии может быть использован для прогнозирования значения цифры для выборок в тестовом подмножестве.

На листинге 7 представлен соответствующий код.

Листинг 7 – Применение классификатора

# flatten the images

n\_samples = len(digits.images)

data = digits.images.reshape((n\_samples, -1))

# Create a classifier: a support vector classifier

clf = svm.SVC(gamma=0.001)

# Split data into 50% train and 50% test subsets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

data, digits.target, test\_size=0.5, shuffle=False

)

# Learn the digits on the train subset

clf.fit(X\_train, y\_train)

# Predict the value of the digit on the test subset

predicted = clf.predict(X\_test)

На рисунке 2 визуализированы первые 4 тестовых образца и их прогнозируемое значение в заголовке.

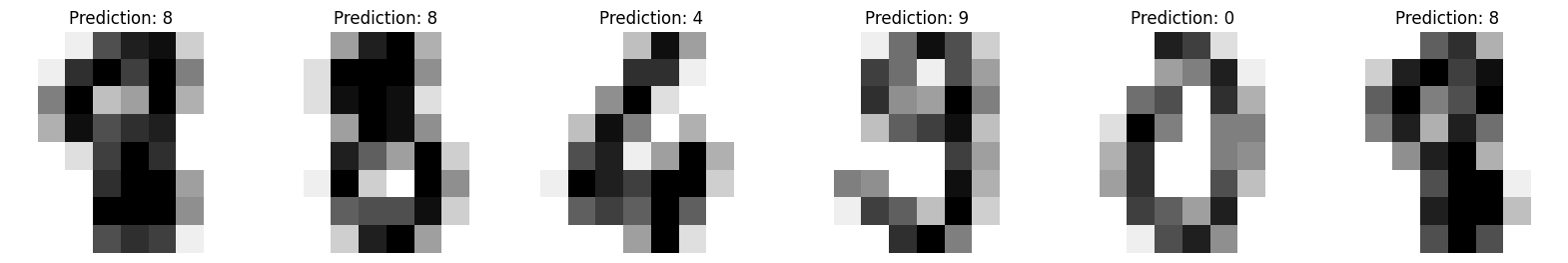


Рисунок 2 – Применение классификатора

На рисунке 3 можно увидеть текстовый отчёт основных метрик классификации.

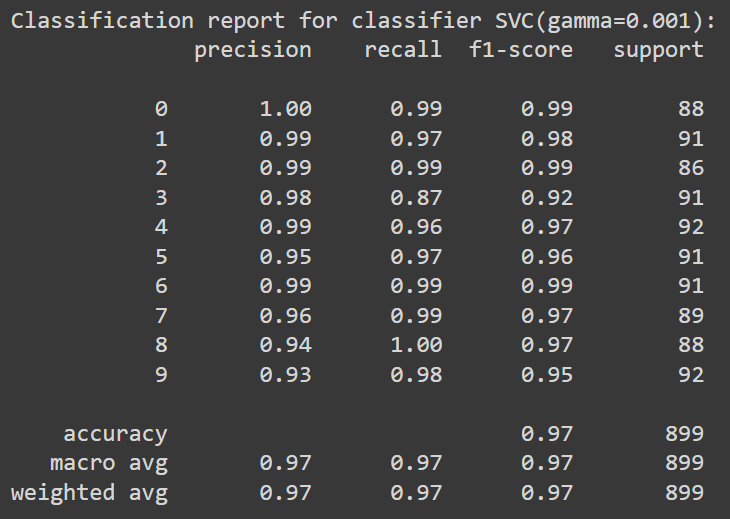


Рисунок 3 – Отчёт о классификации

Можно построить матрицу неточностей истинных значений цифр и прогнозируемых значений цифр. Матрица изображена на рисунке 4.

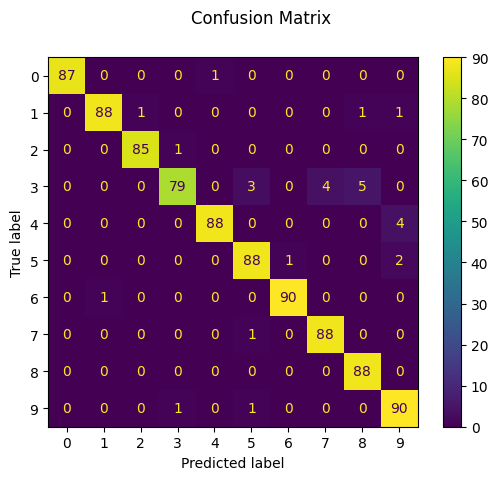


Рисунок 4 – Матрица неточностей

**Оценка этапов классификации изображений**

1. Визуализация изображений: это важный первый шаг для понимания данных, с которыми вы работаете. Он помогает проверить, что изображения загружены правильно, и дает представление о том, как выглядят цифры.
2. Сглаживание изображений: этот шаг необходим для многих алгоритмов машинного обучения, которые ожидают одномерный ввод. Однако при этом теряется пространственная информация, которая может быть важна для задач классификации изображений. Более продвинутые методы, такие как сверточные нейронные сети (CNN), могут работать с двумерными данными напрямую.
3. Разделение данных на обучающую и тестовую выборки: это ключевой шаг для оценки производительности модели на невидимых данных. Он помогает предотвратить переобучение и дает более реалистичную оценку того, насколько хорошо модель будет обобщаться.
4. Обучение классификатора на основе метода опорных векторов (SVC): SVC могут быть эффективны для задач распознавания цифр, особенно с небольшими наборами данных. Они хорошо находят четкие границы разделения между классами. Однако они могут плохо масштабироваться на очень больших наборах данных и могут быть вычислительно затратными.
5. Прогнозирование значений для тестовой выборки: на этом этапе обученная модель применяется к новым, невидимым данным. Это необходимо для оценки производительности модели и ее способности к обобщению.
6. Создание отчета о классификации: это дает комплексное представление о производительности модели по различным метрикам (точность, полнота, F1-мера). Особенно полезно для понимания того, насколько хорошо модель работает с каждым классом цифр.
7. Построение матрицы ошибок: это отличный инструмент визуализации, который показывает, где модель делает правильные прогнозы и где она путает разные цифры. Он может выявить закономерности в неправильных классификациях, что может направить дальнейшие улучшения.

# библиографический список

1. Recognizing hand-written digits: [https://scikit-learn.org/stable/ auto\_examples/classification/plot\_digits\_classification.html#sphxglr-auto-examples-classification-plot-digits-classification-py](https://scikit-learn.org/stable/%20auto_examples/classification/plot_digits_classification.html#sphxglr-auto-examples-classification-plot-digits-classification-py)
2. <https://scikit-learn.org/stable/auto_examples/exercises/> plot\_digits\_classification\_exercise.htm
3. <https://dataheadhunters.com/academy/how-to-do-image-processing-in-python-step-by-step-guide/>
4. <https://scikit-image.org/docs/stable/user_guide/index.html>
5. https://github.com/sightmachine/SimpleCV