|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **ИУК «Информатика и управление»** |
| **КАФЕДРА** | **ИУК4 «Программное обеспечение ЭВМ,** |
| **информационные технологии»** | |

**ЛАБОРАТОРНАЯ РАБОТА №4**

**«**Классификация изображений с использованием Bag-Of-Words методов с использованием функционала OpenCV**»**

**ДИСЦИПЛИНА:** «Программные системы распознавания и обработки информации»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил: студент гр. ИУК4-31М | |  |  | ( | Сафронов Н.С, | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |
| Проверил: | |  |  | ( | Гагарин Ю.Е. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |

|  |  |
| --- | --- |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | |
|  | - Балльная оценка:  - Оценка: |

Калуга, 2025

**Цель:**

Изучить bag-of-words подход для классификации изображений с использованием соответствующих функций библиотеки компьютерного зрения OpenCV.

**Задачи:**

1. Изучить bag-of-words подход для классификации изображений и принцип работы базовых операций обработки изображений:

• детектирование ключевых точек на изображении;

• вычисление дескрипторов ключевых точек;

• построение словаря дескрипторов ключевых точек;

• вычисление признакового описания изображений с использованием словаря дескрипторов ключевых точек;

• обучение классификатора "случайный лес" и предсказание категорий новых изображений.

2. Рассмотреть прототипы функций, реализующих перечисленные операции в библиотеке OpenCV.

3. Разработать простые примеры использования указанного набора функций.

4. Разработать консольное приложение, содержащее реализацию bag-of-words подхода для задачи классификации изображений двух категорий.

5. Провести вычислительный эксперимент на основе подмножества изображений из набора данных Caltech-101.

6. Выполнить исследование зависимости ошибки классификации от используемых параметров (типа используемых детекторов и дескрипторов ключевых точек, числа слов B словаре, параметров алгоритма обучения с учителем "случайный лес").

7. Реализовать подход для построения словаря на основе Gaussian Mixture Model и сравнить полученные результаты с ранее реализованным подходом.

**Задание**

1. Добавьте в разработанное приложение вывод информации об изображениях из тестовой выборки, которые были неправильно классифицированы.

2. Добавьте в разработанное приложение возможность использования в качестве используемого классификатора машины опорных векторов с ядром типа Radial Basis Function. Сравните результаты с ранее реализованным подходом.

3. Выполните исследование зависимости ошибки классификации от используемых параметров (типа используемых детекторов и дескрипторов ключевых точек, числа слов в словаре, параметров алгоритма обучения с учителем "случайный лес": числа деревьев в ансамбле, максимальной глубины деревьев, входящих в ансамбль).

4. Реализуйте построение словаря на основе Gaussian Mixture Model и сравните полученные результаты с ранее реализованным подходом.

**Результаты выполнения работы**

**Изображение выглядит как текст, снимок экрана, меню, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.**

**Рисунок 1** – Информация об изображениях, неверно классифицированных случайным лесом

|  |  |
| --- | --- |
|  |  |
|  |  |

**Рисунок 2** – Примеры неверно классифицированных изображений

**Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.**

**Рисунок 3 –** Результаты сравнения случайного леса с машиной опорных векторов с ядром RBF

Изображение выглядит как линия, диаграмма, График, текст

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 4 –** Зависимость ошибки от размеров словаря для разных детекторов

Изображение выглядит как текст, снимок экрана, диаграмма, Красочность

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Рисунок 5 –** Ошибка при классификации случайным лесом в зависимости от глубины и числа деревьев в ансамбле

**Изображение выглядит как текст, Шрифт, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.**

**Рисунок 6 –** Результаты с построением словаря на основе GMM

**Вывод:** в процессе выполнения лабораторной работы был изучен bag-of-words подход для классификации изображений с использованием соответствующих функций библиотеки компьютерного зрения OpenCV.

**Листинг программы**

**# bow.py**

from typing import List, Tuple, Optional

import cv2

import numpy as np

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

from sklearn.mixture import GaussianMixture

class BoWExtractor:

"""Класс для извлечения признаков Bag-of-Words."""

def \_\_init\_\_(self, vocabulary: np.ndarray):

"""."""

if vocabulary.ndim != 2:

raise ValueError("Vocabulary must be 2D array (K, D)")

self.vocabulary = vocabulary.astype(np.float32)

self.matcher = cv2.BFMatcher(cv2.NORM\_L2, crossCheck=False)

def \_\_call\_\_(self, descriptors: Optional[np.ndarray]) -> np.ndarray:

"""Преобразует дескрипторы изображения в гистограмму BoW."""

if descriptors is None or descriptors.shape[0] == 0:

return np.zeros(self.vocabulary.shape[0], dtype=np.float32)

descriptors = descriptors.astype(np.float32)

matches = self.matcher.match(descriptors, self.vocabulary)

indices = [m.trainIdx for m in matches]

hist, \_ = np.histogram(

indices,

bins=np.arange(self.vocabulary.shape[0] + 1),

)

return hist.astype(np.float32)

def create\_feature\_detector(detector\_name: str) -> cv2.Feature2D:

"""Создаёт детектор/дескриптор по имени."""

if detector\_name.upper() == "SIFT":

return cv2.SIFT.create()

elif detector\_name.upper() == "AKAZE":

return cv2.AKAZE.create()

elif detector\_name.upper() == "ORB":

return cv2.ORB.create()

else:

msg = f"Unsupported detector {detector\_name}, only 'SIFT' is supported."

raise ValueError(msg)

def train\_vocabulary\_kmeans(

image\_paths: List[str],

mask: List[bool],

detector: cv2.Feature2D,

vocab\_size: int,

) -> np.ndarray:

"""Обучает словарь дескрипторов с использованием K-Means."""

all\_descriptors = []

for path, use\_for\_vocab in zip(image\_paths, mask):

if not use\_for\_vocab:

continue

img = cv2.imread(path, cv2.IMREAD\_GRAYSCALE)

if img is None:

continue

\_, descriptors = detector.detectAndCompute(img, None)

if descriptors is not None:

all\_descriptors.append(descriptors.astype(np.float32))

if not all\_descriptors:

raise ValueError("No descriptors collected for vocabulary training")

all\_descriptors = np.vstack(all\_descriptors)

msg = f"Collected {all\_descriptors.shape[0]} descriptors for vocabulary "\

"training."

print(msg)

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 100, 0.1)

\_, \_, centers = cv2.kmeans(

data=all\_descriptors,

K=vocab\_size,

bestLabels=None,

criteria=criteria,

attempts=3,

flags=cv2.KMEANS\_RANDOM\_CENTERS

)

return centers

def train\_vocabulary\_gmm(

image\_paths: List[str],

mask: List[bool],

detector: cv2.Feature2D,

vocab\_size: int,

seed: int = 42

) -> np.ndarray:

"""Обучает словарь с использованием Gaussian Mixture Model."""

all\_descriptors = []

for path, use\_for\_vocab in zip(image\_paths, mask):

if not use\_for\_vocab:

continue

img = cv2.imread(path, cv2.IMREAD\_GRAYSCALE)

if img is None:

continue

\_, descriptors = detector.detectAndCompute(img, None)

if descriptors is not None:

all\_descriptors.append(descriptors.astype(np.float32))

if not all\_descriptors:

raise ValueError("No descriptors collected for GMM training")

all\_descriptors = np.vstack(all\_descriptors)

print(f"Collected {all\_descriptors.shape[0]} descriptors for GMM training.")

gmm = GaussianMixture(

n\_components=vocab\_size,

covariance\_type='full',

reg\_covar=1e-3,

random\_state=seed,

max\_iter=100

)

gmm.fit(all\_descriptors)

return gmm.means\_.astype(np.float32)

def extract\_features\_dataset(

image\_paths: List[str],

mask: List[bool],

detector: cv2.Feature2D,

bow\_extractor: BoWExtractor

) -> Tuple[np.ndarray, np.ndarray]:

"""Извлекает признаки для подмножества изображений."""

features = []

selected\_indices = []

for i, (path, selected) in enumerate(zip(image\_paths, mask)):

if not selected:

continue

img = cv2.imread(path, cv2.IMREAD\_GRAYSCALE)

if img is None:

features.append(

np.zeros(bow\_extractor.vocabulary.shape[0], dtype=np.float32)

)

else:

\_, descriptors = detector.detectAndCompute(img, None)

hist = bow\_extractor(descriptors)

features.append(hist)

selected\_indices.append(i)

return np.array(features, dtype=np.float32), np.array(selected\_indices)

def train\_random\_forest(

X: np.ndarray,

y: np.ndarray,

n\_estimators: int = 200,

max\_depth: Optional[int] = None,

seed: int = 42

) -> RandomForestClassifier:

"""Обучает Random Forest."""

clf = RandomForestClassifier(

n\_estimators=n\_estimators,

max\_depth=max\_depth,

random\_state=seed,

n\_jobs=-1

)

clf.fit(X, y)

return clf

def evaluate\_classifier(

y\_true: np.ndarray,

y\_pred: np.ndarray

) -> float:

"""Вычисляет ошибку классификации."""

return 1.0 - accuracy\_score(y\_true, y\_pred)

**# utils.py**

import glob

import os

import random

def get\_jpg\_files\_in\_directory(directory: str) -> list[str]:

"""Получает список всех JPG/JPEG файлов в указанной директории."""

if not os.path.isdir(directory):

raise FileNotFoundError(f"Directory not found: {directory}")

pattern = os.path.join(directory, "\*.jpg")

files = glob.glob(pattern, recursive=False)

pattern\_upper = os.path.join(directory, "\*.JPG")

files += glob.glob(pattern\_upper, recursive=False)

return sorted(files)

def create\_random\_mask(

size: int,

true\_probability: float,

seed: int | None = None,

) -> list[bool]:

"""

Создаёт булевский вектор заданного размера с заданной вероятностью True.

"""

if not (0.0 <= true\_probability <= 1.0):

raise ValueError("true\_probability must be in [0.0, 1.0]")

if seed is not None:

random.seed(seed)

return [random.random() < true\_probability for \_ in range(size)]

**# main.py**

import argparse

import sys

from typing import List, Tuple

import numpy as np

from sklearn.svm import SVC

from .bow import (

train\_random\_forest,

create\_feature\_detector,

train\_vocabulary\_kmeans,

BoWExtractor,

extract\_features\_dataset,

evaluate\_classifier,

train\_vocabulary\_gmm,

)

from .utils import get\_jpg\_files\_in\_directory, create\_random\_mask

def train\_svm\_rbf(

X: np.ndarray,

y: np.ndarray,

C: float = 1.0,

gamma: str = 'scale',

seed: int = 42

) -> SVC:

"""Обучает SVM с RBF ядром."""

clf = SVC(kernel='rbf', C=C, gamma=gamma, random\_state=seed)

clf.fit(X, y)

return clf

def find\_misclassified(

image\_paths: List[str],

y\_true: np.ndarray,

y\_pred: np.ndarray

) -> set[Tuple[str, int, int]]:

"""Находит неправильно классифицированные изображения."""

return set(

(path, true, pred)

for path, true, pred in zip(image\_paths, y\_true, y\_pred)

if true != pred

)

def parse\_args():

parser = argparse.ArgumentParser(

description="Bag-of-Words Image Classifier for two classes."

)

parser.add\_argument(

"folder1",

type=str,

help="Path to first class directory",

)

parser.add\_argument(

"folder2",

type=str,

help="Path to second class directory",

)

parser.add\_argument(

"detector",

type=str,

choices=["SIFT"],

help="Feature detector",

)

parser.add\_argument(

"descriptor",

type=str,

choices=["SIFT", "AKAZE", "ORB"],

help="Descriptor type",

)

parser.add\_argument(

"voc\_size",

type=int,

help="Vocabulary size",

)

parser.add\_argument(

"train\_proportion",

type=float,

help="Train proportion (0.0–1.0)",

)

parser.add\_argument(

"--seed",

type=int,

default=42,

help="Random seed for reproducibility",

)

return parser.parse\_args()

def main():

args = parse\_args()

try:

files1 = get\_jpg\_files\_in\_directory(args.folder1)

files2 = get\_jpg\_files\_in\_directory(args.folder2)

except FileNotFoundError as e:

print(f"Error: {e}", file=sys.stderr)

sys.exit(1)

if not files1 or not files2:

print("Error: One or both directories are empty.", file=sys.stderr)

sys.exit(1)

all\_files = files1 + files2

labels = np.array([1] \* len(files1) + [-1] \* len(files2), dtype=np.int32)

split\_mask = create\_random\_mask(

len(all\_files), args.train\_proportion, seed=args.seed,

)

detector = create\_feature\_detector(args.detector)

print("=== Training vocabulary with K-Means...")

vocab\_kmeans = train\_vocabulary\_kmeans(

all\_files, split\_mask, detector, args.voc\_size,

)

bow\_kmeans = BoWExtractor(vocab\_kmeans)

X\_train, train\_indices = extract\_features\_dataset(

all\_files, split\_mask, detector, bow\_kmeans

)

y\_train = labels[train\_indices]

print("=== Training classifiers...")

rf\_clf = train\_random\_forest(

X\_train, y\_train, n\_estimators=200, seed=args.seed,

)

svm\_clf = train\_svm\_rbf(X\_train, y\_train, C=1.0, seed=args.seed)

test\_mask = [not m for m in split\_mask]

X\_test, test\_indices = extract\_features\_dataset(

all\_files, test\_mask, detector, bow\_kmeans

)

y\_test = labels[test\_indices]

rf\_pred = rf\_clf.predict(X\_test)

svm\_pred = svm\_clf.predict(X\_test)

rf\_error = evaluate\_classifier(y\_test, rf\_pred)

svm\_error = evaluate\_classifier(y\_test, svm\_pred)

print(f"\n=== Results (Vocabulary: K-Means, Size: {args.voc\_size}) ===")

print(f"Random Forest error: {rf\_error:.4f}")

print(f"SVM (RBF) error: {svm\_error:.4f}")

misclassified\_rf = find\_misclassified(

[all\_files[i] for i in test\_indices], y\_test, rf\_pred

)

if misclassified\_rf:

print(

f"\nMisclassified by Random Forest ({len(misclassified\_rf)} "\

"items):"

)

for path, true, pred in misclassified\_rf:

print(f" {path} | True: {true}, Pred: {pred}")

else:

print("\nRandom Forest: No misclassifications!")

print("\n=== Training with GMM vocabulary...")

try:

vocab\_gmm = train\_vocabulary\_gmm(

all\_files, split\_mask, detector, args.voc\_size, seed=args.seed

)

bow\_gmm = BoWExtractor(vocab\_gmm)

X\_train\_gmm, \_ = extract\_features\_dataset(

all\_files, split\_mask, detector, bow\_gmm

)

rf\_gmm = train\_random\_forest(X\_train\_gmm, y\_train, seed=args.seed)

X\_test\_gmm, \_ = extract\_features\_dataset(

all\_files, test\_mask, detector, bow\_gmm

)

pred\_gmm = rf\_gmm.predict(X\_test\_gmm)

gmm\_error = evaluate\_classifier(y\_test, pred\_gmm)

print(f"GMM + Random Forest error: {gmm\_error:.4f}")

except Exception as e:

print(f"GMM training failed: {e}")

print("\nDone.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**# task3.py**

import argparse

import os

from typing import List, Tuple

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

from bow import (

create\_feature\_detector,

train\_vocabulary\_kmeans,

BoWExtractor,

extract\_features\_dataset,

evaluate\_classifier,

)

from sklearn.ensemble import RandomForestClassifier

from utils import get\_jpg\_files\_in\_directory, create\_random\_mask

SEEDS = [42, 101, 2024]

DETECTORS = ["SIFT", "ORB", "AKAZE"]

VOC\_SIZES = [10, 25, 50, 100]

RF\_N\_ESTIMATORS = [50, 100, 200, 500]

RF\_MAX\_DEPTHS = [5, 10, 20, None]

def load\_data(folder\_1: str, folder\_2: str) -> Tuple[List[str], np.ndarray]:

"""Загружает изображения и метки."""

files1 = get\_jpg\_files\_in\_directory(folder\_1)

files2 = get\_jpg\_files\_in\_directory(folder\_2)

all\_files = files1 + files2

labels = np.array([1] \* len(files1) + [-1] \* len(files2), dtype=np.int32)

return all\_files, labels

def run\_single\_experiment(

all\_files: List[str],

labels: np.ndarray,

detector\_name: str,

voc\_size: int,

n\_estimators: int,

max\_depth: int | None,

seed: int,

train\_proportion: float,

) -> float:

"""Выполняет один эксперимент и возвращает ошибку."""

try:

split\_mask = create\_random\_mask(

len(all\_files), train\_proportion, seed=seed,

)

detector = create\_feature\_detector(detector\_name)

vocab = train\_vocabulary\_kmeans(

all\_files, split\_mask, detector, voc\_size,

)

bow = BoWExtractor(vocab)

X\_train, train\_idx = extract\_features\_dataset(

all\_files, split\_mask, detector, bow,

)

y\_train = labels[train\_idx]

X\_test, test\_idx = extract\_features\_dataset(

all\_files,

[not m for m in split\_mask],

detector,

bow,

)

y\_test = labels[test\_idx]

if X\_train.size == 0 or X\_test.size == 0:

return np.nan

clf = RandomForestClassifier(

n\_estimators=n\_estimators,

max\_depth=max\_depth,

random\_state=seed,

n\_jobs=-1

)

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

y\_pred = clf.predict(X\_test)

return evaluate\_classifier(y\_test, y\_pred)

except Exception as e:

print(

f"Failed: {detector\_name}, voc={voc\_size}, n\_est={n\_estimators}, "

f"depth={max\_depth}, seed={seed} | {e}"

)

return np.nan

def run\_detector\_voc\_experiment(

folder\_1: str,

folder\_2: str,

train\_proportion: float,

) -> pd.DataFrame:

"""Исследование: детектор + размер словаря."""

all\_files, labels = load\_data(folder\_1, folder\_2)

results = []

for det in DETECTORS:

for voc in VOC\_SIZES:

errors = []

for seed in SEEDS:

err = run\_single\_experiment(

all\_files, labels, det, voc,

n\_estimators=200, max\_depth=None, seed=seed,

train\_proportion=train\_proportion,

)

if not np.isnan(err):

errors.append(err)

if errors:

results.append({

"detector": det,

"voc\_size": voc,

"mean\_error": np.mean(errors),

"std\_error": np.std(errors)

})

return pd.DataFrame(results)

def run\_rf\_params\_experiment(

folder\_1: str,

folder\_2: str,

train\_proportion: float,

) -> pd.DataFrame:

"""Исследование: параметры Random Forest."""

all\_files, labels = load\_data(folder\_1, folder\_2)

results = []

best\_det = "SIFT"

best\_voc = 50

for n\_est in RF\_N\_ESTIMATORS:

for depth in RF\_MAX\_DEPTHS:

errors = []

for seed in SEEDS:

err = run\_single\_experiment(

all\_files, labels, best\_det, best\_voc,

n\_estimators=n\_est, max\_depth=depth, seed=seed,

train\_proportion=train\_proportion,

)

if not np.isnan(err):

errors.append(err)

if errors:

results.append({

"n\_estimators": n\_est,

"max\_depth": str(depth) if depth is not None else "None",

"mean\_error": np.mean(errors),

"std\_error": np.std(errors)

})

return pd.DataFrame(results)

def plot\_detector\_voc(df: pd.DataFrame):

"""Строит график зависимости ошибки от детектора и размера словаря."""

plt.figure(figsize=(10, 6))

for det in df["detector"].unique():

subset = df[df["detector"] == det]

plt.errorbar(

subset["voc\_size"], subset["mean\_error"],

yerr=subset["std\_error"], label=det, marker="o",

)

plt.xlabel("Vocabulary Size")

plt.ylabel("Misclassification Error")

plt.title("Error vs Vocabulary Size by Detector")

plt.legend()

plt.grid(True, linestyle="--", alpha=0.7)

plt.tight\_layout()

plt.savefig("../results/detector\_voc\_error.png", dpi=150)

plt.show()

def plot\_rf\_params(df: pd.DataFrame):

"""Тепловая карта ошибки для параметров RF."""

df\_pivot = df.pivot(index="max\_depth", columns="n\_estimators",

values="mean\_error")

plt.figure(figsize=(8, 6))

sns.heatmap(df\_pivot, annot=True, fmt=".4f", cmap="viridis",

cbar\_kws={'label': 'Error'})

plt.title("Random Forest: Error vs Parameters")

plt.tight\_layout()

plt.savefig("../results/rf\_params\_error.png", dpi=150)

plt.show()

def parse\_args():

parser = argparse.ArgumentParser(

description="Bag-of-Words Image Classifier for two classes."

)

parser.add\_argument(

"folder1",

type=str,

help="Path to first class directory",

)

parser.add\_argument(

"folder2",

type=str,

help="Path to second class directory",

)

parser.add\_argument(

"train\_proportion",

type=float,

help="Train proportion (0.0–1.0)",

)

return parser.parse\_args()

def main():

args = parse\_args()

print("Running detector + vocabulary size experiment...")

df1 = run\_detector\_voc\_experiment(

folder\_1=args.folder1,

folder\_2=args.folder2,

train\_proportion=args.train\_proportion,

)

os.makedirs('../results', exist\_ok=True)

df1.to\_csv("../results/detector\_voc\_results.csv", index=False)

print(df1)

plot\_detector\_voc(df1)

print("\nRunning Random Forest parameters experiment...")

df2 = run\_rf\_params\_experiment(

folder\_1=args.folder1,

folder\_2=args.folder2,

train\_proportion=args.train\_proportion,

)

df2.to\_csv("../results/rf\_params\_results.csv", index=False)

print(df2)

plot\_rf\_params(df2)

if \_\_name\_\_ == "\_\_main\_\_":

main()