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 В 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

```
Misclassified by Random Forest (18 items):
  ..\dataset\crocodile\image_0044.jpg | True: -1, Pred: 1
  ..\dataset\elephant\image_0018.jpg | True: 1, Pred: -1
  ..\dataset\crocodile\image_0003.jpg | True: -1, Pred: 1
  ..\dataset\elephant\image_0008.jpg | True: 1, Pred: -1
  ..\dataset\elephant\image_0028.jpg | True: 1, Pred: -1
  ..\dataset\elephant\image_0033.jpg | True: 1, Pred: -1
  ..\dataset\elephant\image_0052.jpg | True: 1, Pred: -1
  ..\dataset\crocodile\image_0026.jpg | True: -1, Pred: 1
  ..\dataset\crocodile\image_0036.jpg | True: -1, Pred: 1
  ..\dataset\elephant\image_0042.jpg | True: 1, Pred: -1
  ..\dataset\crocodile\image_0027.jpg | True: -1, Pred: 1
  ..\dataset\elephant\image_0047.jpg | True: 1, Pred: -1
  ..\dataset\elephant\image_0058.jpg | True: 1, Pred: -1
  ..\dataset\elephant\image_0031.jpg | True: 1, Pred: -1
  ..\dataset\elephant\image_0049.jpg | True: 1, Pred: -1
  ..\dataset\crocodile\image_0034.jpg | True: -1, Pred: 1
  ..\dataset\crocodile\image_0012.jpg | True: -1, Pred: 1
  ..\dataset\elephant\image_0011.jpg | True: 1, Pred: -1
```

Рисунок 1 – Информация об изображениях, неверно классифицированных случайным лесом

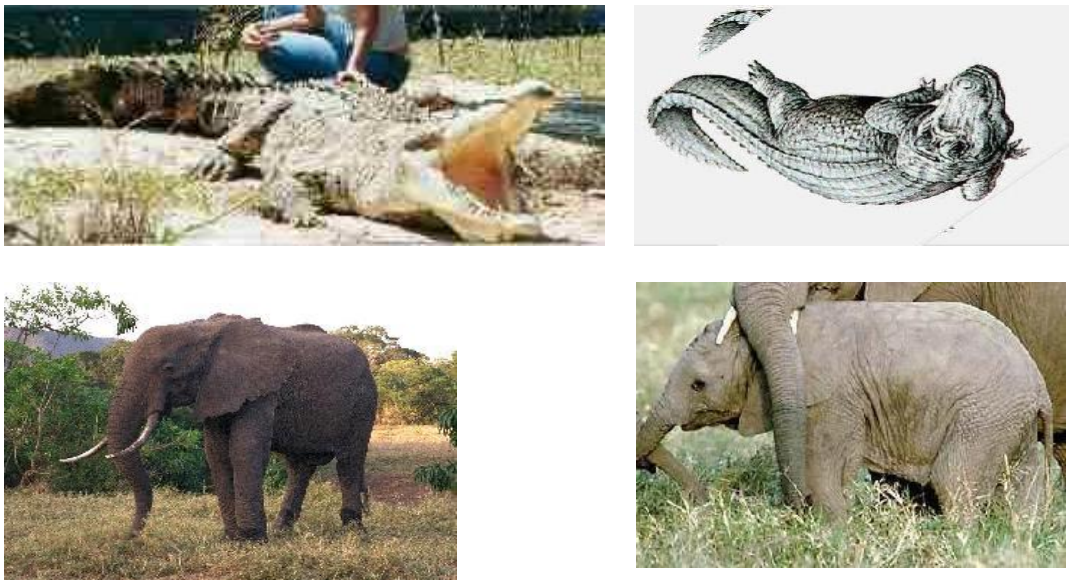

Рисунок 2 – Примеры неверно классифицированных изображений

```
=== Training vocabulary with K-Means...
Collected 44089 descriptors for vocabulary training.
=== Training classifiers...

=== Results (Vocabulary: K-Means, Size: 50) ===
Random Forest error: 0.2384
SVM (RBF) error:     0.2649
```

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

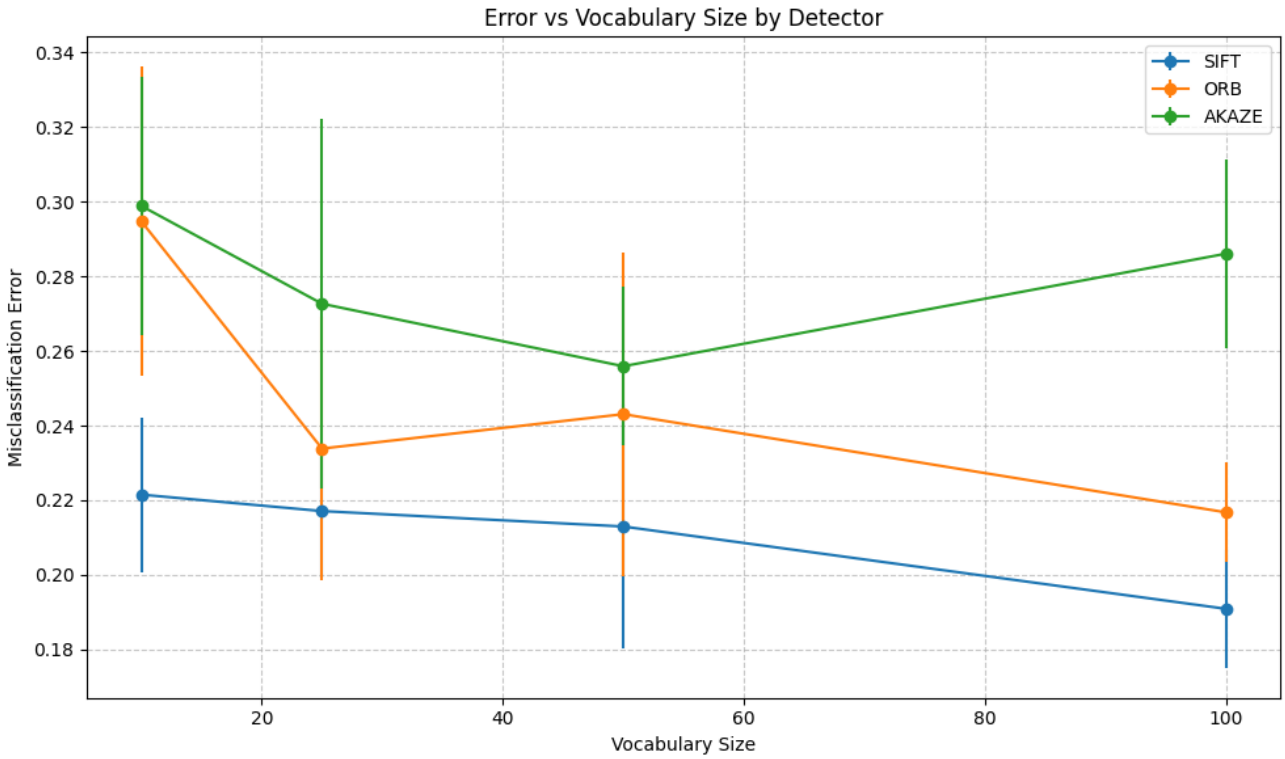

Рисунок 4 – Зависимость ошибки от размеров словаря для разных детекторов

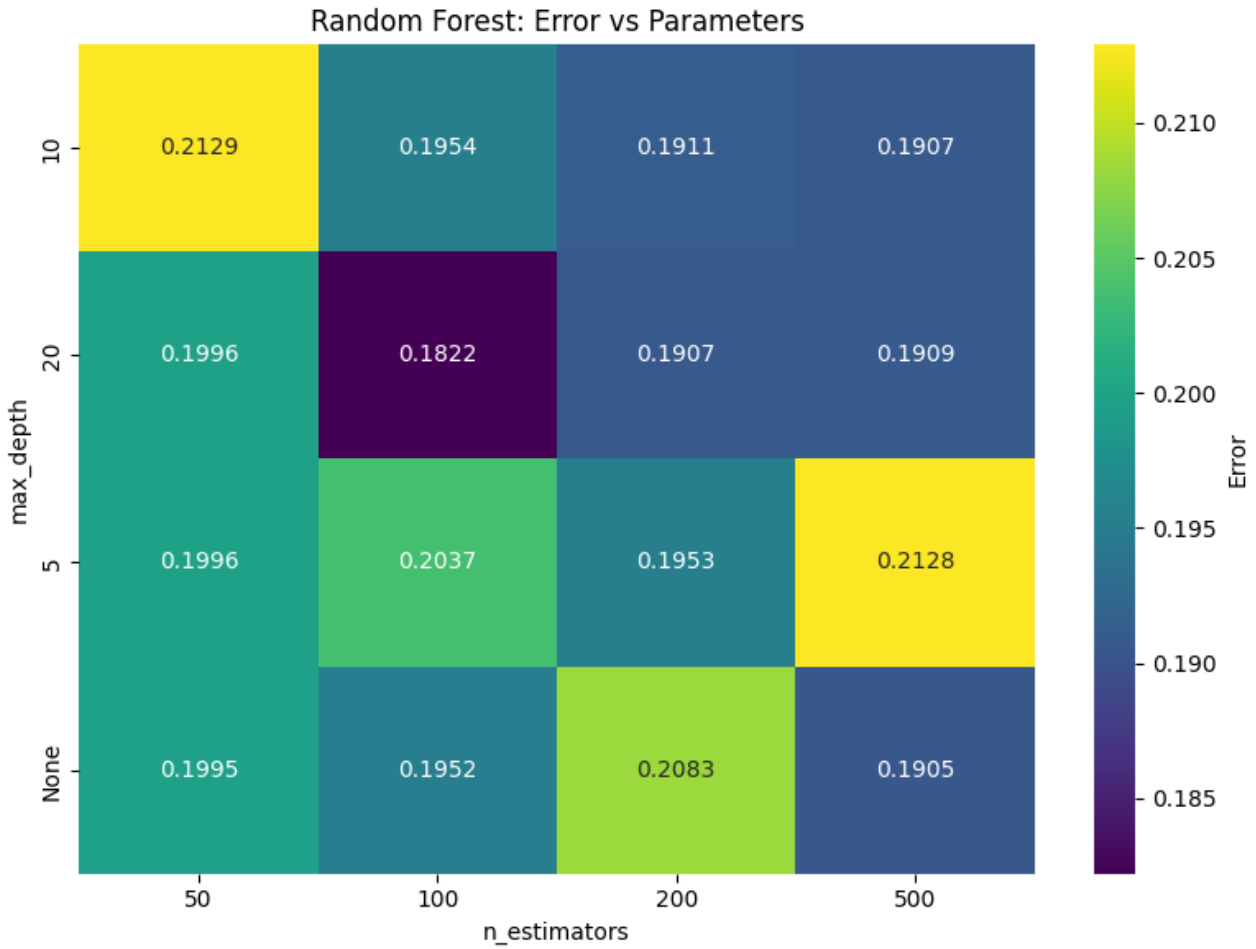

Рисунок 5 – Ошибка при классификации случайным лесом в зависимости от глубины и числа деревьев в ансамбле

```
=== Training with GMM vocabulary...
Collected 44089 descriptors for GMM training.
GMM + Random Forest error: 0.1589
```

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()
```