AI Capstone HW1 Report:
Anime Character Face Recognition

Author: 司徒立中 (111550159) | GitHub Link

1. Introduction

Image recognition plays an important role in many applications, from facial recognition to object detection. The accuracy of these systems depends on several factors, including the dataset used for training, the learning strategies applied, and the choice of feature extraction methods.

In this work, I focus on the process of collecting a dataset and analyzing how different learning strategies affect performance. I experiment with methods such as K-Means, AdaBoost, and ResNet to compare their effectiveness in classification tasks. Additionally, we explore how different feature extraction techniques influence model performance, helping us understand which combinations work best.

Through this study, we aim to gain insights into the impact of various approaches and provide useful observations for improving image recognition models.

2. Dataset

In my dataset collection, the process can be divided into five parts, including web crawler, first human filter, anime face detection, second human filter, and deduplication with nearest neighbor in feature space.

To begin with, most anime character artworks are distributed on Twitter (now called X) and Pixiv. However, Twitter's API requires payment to access more data from each post, and Pixiv is

filled with a large number of AI-generated artworks. As a result, collecting these illustrations without paying is quite difficult.

To address this issue, I referred to AniWho [1] and learned from their strategy. I utilized an API for a website called Danbooru [2], which hosts a vast collection of anime character images. With this API, I can use specific tags to search for particular characters, such as Ninomae Ina'nis, Hakos Baelz, Koseki Bijou, and more. For this data collection, I decided to gather 1,000 images for each of 20 distinct Hololive EN members.

When the web crawler has finished collecting data, the dataset requires some level of cleanup because some files might not be in the correct image format, or the images may not contain the intended character. Due to the complexity of the data, I attempt to clean up these files manually.

After cleaning up the dataset, it is helpful to use an existing anime face detection model to crop bounding boxes from each image, reducing unnecessary information while maintaining diversity. For this, I chose the implementation of LAFD [4] due to its efficiency and low GPU resource requirements. To obtain higher-quality anime face images, I adjusted the confidence threshold and the input image size.

It is noticeable that the number of images increases from 13,142 to 31,137, as each image may contain multiple faces. Thus, it is essential to determine whether the detected facial images belong to the corresponding character. Once again, I will manually filter the correct face images from the cropped bounding boxes.

In the end, some face images may be duplicated since the same artworks might be uploaded to different websites or have multiple versions. To solve this problem, I came up with an idea similar to the FID score: using InceptionV3 to obtain embeddings in the feature space. Then, we calculate the cosine similarity between these embeddings. If the similarity exceeds a certain threshold ($\geq$0.99), the corresponding images will be grouped for further deduplication.

3. Methods

In this section, the methods are divided into two parts: learning strategies and feature extraction, which are the two main factors related to the final performance.

In this study, we employ three distinct learning strategies: K-Means, AdaBoost, and ResNet. K-Means is an unsupervised clustering algorithm that partitions the data into K clusters based on feature similarity, serving as a baseline for grouping samples without prior label knowledge. In this dataset with 20 categories, the value of K is ranging from 1 to 20 and be determined with highest accuracy. AdaBoost, on the other hand, is an ensemble method that iteratively combines weak learners to form a strong classifier, placing emphasis on misclassified instances to progressively improve performance. Meanwhile, ResNet leverages deep convolutional architectures with residual connections to effectively train very deep networks, achieving state-of-the-art results in many image classification tasks.

For feature extraction, our approach integrates three techniques: HOG, Color Histogram, and the CLIP image encoder. The HOG (Histogram of Oriented Gradients) method captures local shape and texture information by computing the distribution of gradient orientations, while the Color Histogram quantifies the distribution of color intensities across different channels, offering a straightforward yet effective descriptor of image content. Additionally, the CLIP image encoder utilizes a pre-trained model to map images into a semantically rich feature space that aligns with natural language, enabling a deeper and more contextual representation.

4. Analysis

In this section, we evaluate the performance of different learning strategies and feature extraction methods. We compare three algorithms: K-Means (unsupervised learning), Adaboost (supervised learning), and ResNet (deep learning). Each algorithm is tested with various feature extractors, including HOG, Color, CLIP, and None (raw image features).

The performance is measured using accuracy metrics, as shown in the tables below. Additionally, cross-validation is applied to ensure the reliability of the evaluation, particularly for supervised learning models. The results highlight the impact of different feature extractors on each algorithm's performance, with CLIP showing relatively strong results for K-Means and Adaboost, while ResNet achieves the highest accuracy when trained on raw image features.

Algorithm		K-Means	Adaboost	ResNet
Learning Type		Unsupervised	Supervised	Supervised
Category		Machine Learning	Machine Learning	Deep Learning
Feature Extractor	HOG	0.1649	0.2152	-
	Color	0.3110	0.3140	-
	CLIP	0.5718	0.5680	-
	None	-	-	0.9761

Cross Validation	CLIP + K-Means	CLIP + Adaboost	ResNet
Fold1	0.5596	0.5472	0.9718
Fold2	0.5073	0.4769	0.9718
Fold3	05372	0.5607	0.9731
Fold4	0.5547	0.5475	0.9636

CLIP + K-Means

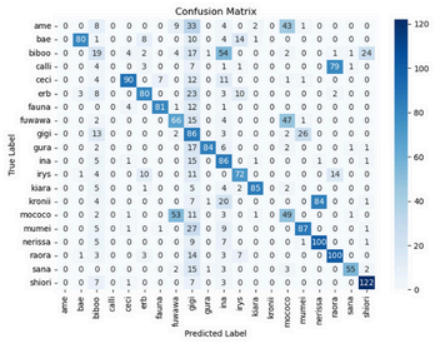
Class	Precision	Recall	F1-Score	Support
ame	0.0	0.0	0.0	100
bae	0.94	0.68	0.79	118
biboo	0.19	0.15	0.17	129
calli	0.0	0.0	0.0	96
ceci	0.88	0.71	0.79	126
erb	0.75	0.62	0.68	129
fauna	0.91	0.82	0.86	99
fuwawa	0.48	0.49	0.49	135
gigi	0.24	0.65	0.35	132
gura	0.94	0.37	0.53	113
ina	0.38	0.78	0.51	110
irys	0.68	0.64	0.66	112
kiara	0.94	0.62	0.75	104
kronii	0.0	0.0	0.0	117
mococo	0.32	0.4	0.36	121
mumei	0.74	0.66	0.7	131
nerissa	0.54	0.81	0.65	123
raora	0.51	0.76	0.61	131
sana	0.93	0.68	0.79	81
shiori	0.8	0.87	0.83	140

CLIP + Adaboost

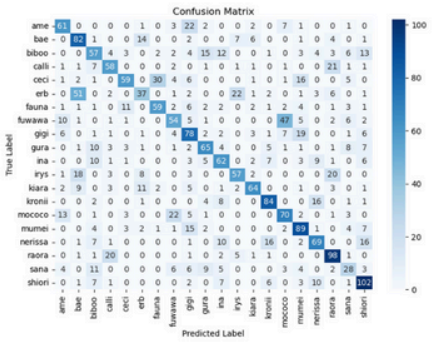
Class	Precision	Recall	F1-Score	Support
ame	0.61	0.61	0.61	100
bae	0.49	0.69	0.57	118
biboo	0.47	0.44	0.46	129
calli	0.59	0.6	0.6	96
ceci	0.7	0.47	0.56	126
erb	0.49	0.29	0.36	129
fauna	0.63	0.6	0.61	99
fuwawa	0.53	0.4	0.46	135
gigi	0.46	0.59	0.52	132
gura	0.6	0.58	0.59	131
ina	0.54	0.56	0.55	140
irys	0.61	0.51	0.55	112
kiara	0.75	0.62	0.67	104
kronii	0.68	0.72	0.7	117
mococo	0.6	0.58	0.59	120
mumei	0.57	0.63	0.6	131
nerissa	0.62	0.65	0.63	123
raora	0.6	0.75	0.66	131
sana	0.41	0.35	0.37	81
shiori	0.61	0.73	0.66	140

ResNet

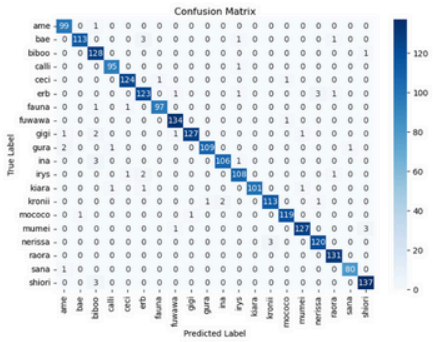
Class	Precision	Recall	F1-Score	Support
ame	0.96	0.99	0.98	100
bae	0.99	0.96	0.97	118
biboo	0.93	0.99	0.96	129
calli	0.98	0.99	0.98	96
ceci	0.98	0.98	0.98	126
erb	0.95	0.95	0.95	129
fauna	0.98	0.98	0.98	99
fuwawa	0.98	0.99	0.99	135
gigi	0.99	0.96	0.97	132
gura	0.99	0.96	0.97	113
ina	0.98	0.96	0.97	110
irys	0.96	0.96	0.96	112
kiara	1.0	0.97	0.97	104
kronii	0.97	0.97	0.97	117
mococo	0.98	0.98	0.98	121
mumei	0.98	0.97	0.97	131
nerissa	0.98	0.97	0.97	123
raora	0.98	1.0	0.99	131
sana	0.99	0.98	0.98	81
shiori	0.97	0.98	0.98	140



(K-Means with CLIP)



(Adaboost with CLIP)



(ResNet18)

5. Experiments

Q1. How are the results affected by the amount of training data?

Training Data	CLIP + K-Means	CLIP + Adaboost	ResNet
100%	0.5718	0.5680	0.9761
75%	0.5143	0.5445	0.9693
50%	0.5198	0.4631	0.9616
25%	0.4989	0.2888	0.9536
10%	0.4248	0.1819	0.9046

As the training data decreases, all models show performance degradation. ResNet remains the most stable, with only a slight drop from 0.9761 to 0.9046. K-Means declines gradually, while Adaboost is highly sensitive to data reduction. Notably, its accuracy drops sharply from 0.4631 at 50% data to 0.2888 at 25%, and further to 0.1819 at 10%, indicating a strong dependence on sufficient training data.

Q2. How are the results affected by dimensionality reduction?

HOG with PCA	K-Means	Adaboost
No PCA (dim=8100)	0.1649	0.2152
PCA (dim=1000)	0.1521	0.2522
PCA (dim=200)	0.1530	0.2714
PCA (dim=50)	0.1692	0.2659
PCA (dim=16)	0.1679	0.2382

The results indicate that dimensionality reduction using PCA affects the performance of both K-Means and Adaboost differently. For K-Means, reducing the HOG features to 50 dimensions yields the best performance, while for Adaboost, reducing them to 200 dimensions is sufficient.

However, as the dimensionality decreases further, performance starts to drop, indicating that while PCA can help remove noise and redundancy, excessive reduction may lead to information loss. Overall, the impact of PCA depends on the specific algorithm used, and the optimal reduced dimension can be determined through extensive experiments.

Q3. How are the results affected by feature extraction?

Feature Extractor	K-Means	Adaboost
HOG	0.1649	0.2152
Color	0.3110	0.3140
CLIP	0.5718	0.5680

Based on the table, regardless of whether K-Means or AdaBoost is used, CLIP features clearly outperform the other two feature extraction methods (HOG and Color). According to the properties of my dataset, the differences between each character are primarily based on color features and structural details.

Meanwhile, HOG only generates features relying on edges and gradients, resulting in the worst performance. In contrast, Color Histogram can capture the overall color distribution, making it more effective than HOG. However, it still lacks the ability to encode structural and semantic information, which explains why CLIP achieves the best results by leveraging deep representations that encompass both visual and contextual features.

Q4. How are the results affected by random seed?

Random Seed	CLIP + K-Means	CLIP + Adaboost	ResNet
42	0.5718	0.5680	0.9761
48763	0.5718	0.5680	0.9740
114514	0.5718	0.5680	0.9723
228922	0.5718	0.5680	0.9702

The results indicate that the choice of random seed has little impact on the performance. K-Means and Adaboost produce identical performance regardless of the random seed because both methods use machine learning techniques to find the optimal solution. In contrast, ResNet exhibits variations in performance because it involves stochastic processes such as weight initialization, data shuffling, and batch ordering during training, which are influenced by the random seed. Since different random seeds lead to different performance outcomes, if the goal is to achieve higher accuracy, the random seed can be considered a hyperparameter for fine-tuning.

6. Discussion

Based on my experiments, most of the results and observed behaviors are in line with my expectations, including that ResNet outperforms other methods and that appropriate dimensionality reduction is beneficial. However, unexpectedly, Adaboost is highly sensitive when the dataset is inadequate compared with others.

The factors such as dataset size, feature extraction methods, and dimensionality reduction all play crucial roles in determining model performance. In particular, the dataset's focus on color and structural details naturally favors methods (like CLIP) that capture higher-level semantic information, while simpler features (like HOG) struggle.

If more time were available, I would run additional experiments, such as training an SVM to compare its performance against K-Means and Adaboost. I would also explore explainable AI techniques—like feature attribution or saliency maps—to better understand how each model makes its decisions, particularly for deep learning models like ResNet.

Through these experiments, I have learned the importance of carefully choosing feature extraction methods in traditional machine learning, and striking a balance in dimensionality reduction to avoid losing critical information. In addition, I found that scikit-learn is very helpful for existing training methods compared to PyTorch, as it allows for cleaner and more concise code.

7. Reference

Related Work

[1] AniWho : A Quick and Accurate Way to Classify Anime Character Faces in Images

[2] Danbooru - Anime Image Board

[3] LFFD: A Light and Fast Face Detector for Edge Devices

[4] An Implementation of Light (and Fast) Anime Face Detector

Library/Package Usage

[5] Pytorch

[6] Scikit Learn (sklearn)

Others

[7] ChatGPT

8. Appendix (Codes)

crawler.py

```python
import argparse
import json
import os
import requests
import time
from tqdm import tqdm

def main(config_file, character_name):
    with open(config_file, "r", encoding="utf-8") as f:
        configs = json.load(f)

    config = configs.get(character_name)
    if not config:
        print(f"Configuration for character '{character_name}' not found.")
        return

    tag = config.get("tag")
    download_folder = config.get("download_folder")
    os.makedirs(download_folder, exist_ok=True)

    total_posts_to_retrieve = 1000  # Desired number of posts
    limit = 100  # Maximum number of posts per request
    total_pages = (total_posts_to_retrieve + limit - 1) // limit  # Calculate total pages needed

    total_downloaded = 0

    for page in range(1, total_pages + 1):
        print(f"Downloading page {page}...")
        params = {
            "tags": tag,
            "page": page,
            "limit": limit
        }
        response = requests.get("https://danbooru.donmai.us/posts.json", params=params)
        posts = response.json()

        if not posts:
            break

        for post in tqdm(posts, desc=f"Page {page}", unit="image"):
            file_url = post.get("sample_file_url") or post.get("file_url")
            if file_url:
                image_name = os.path.basename(file_url)
                image_path = os.path.join(download_folder, image_name)
                if os.path.exists(image_path):
                    continue
                response = requests.get(file_url, stream=True)
                total_size = int(response.headers.get('content-length', 0))
                with open(image_path, "wb") as f:
                    for data in tqdm(response.iter_content(1024), total=total_size//1024, unit='KB', desc=image_name):
                        f.write(data)
                time.sleep(1)  # Delay to avoid excessive requests
                total_downloaded += 1

    print(f"Total images downloaded: {total_downloaded}")

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Danbooru Image Downloader")
    parser.add_argument("--config", type=str, required=True, help="Path to the configuration file")
    parser.add_argument("--cha", type=str, required=True, help="Character name")
    args = parser.parse_args()

    main(args.config, args.cha)
```

runs.py

```python
import os
import json
import cv2
from tqdm import tqdm
from core.detector import LFFDDetector

"""
    !!! IMPORTANT !!!
    Disable auto-tuning
    You might experience a major slow-down if you run the model on images with varied resolution / aspect ratio.
    This is because MXNet is attempting to find the best convolution algorithm for each input size,
    so we should disable this behavior if it is not desirable.
"""
os.environ["MXNET_CUDNN_AUTOTUNE_DEFAULT"] = "0"
os.environ["CUDA_VISIBLE_DEVICES"] = "1"
CONFIG_PATH = "configs/anime.json"

def anime_face_detect(folder):
    new_root_folder = 'face_dataset'
    new_folder = os.path.basename(folder)
    new_path = os.path.join(new_root_folder, new_folder)

    if not os.path.exists(new_path):
        os.makedirs(new_path)

    with open(CONFIG_PATH, "r") as f:
        config = json.load(f)
        detector = LFFDDetector(config, use_gpu=True)

    WIDTH = HEIGHT = 256
    files = os.listdir(folder)
    for file in tqdm(files, desc=f"Processing {new_folder}", unit="file"):
        image = cv2.imread(os.path.join(folder, file))
        boxes = detector.detect(image)

        idx = 1
        for box in boxes:
            x1, y1 = box['xmin'], box['ymin']
            x2, y2 = box['xmax'], box['ymax']

            face_image = image[y1:y2, x1:x2]
            face_image = cv2.resize(face_image, (WIDTH, HEIGHT))

            save_path = f"{new_path}/{file}-{idx}.jpg"
            cv2.imwrite(save_path, face_image)
            idx += 1

    return 0

if __name__ == "__main__":

    root_folder = 'dataset'
    for folder in os.listdir(root_folder):
        path = os.path.join(root_folder, folder)
        if not os.path.isdir(path): continue
        anime_face_detect(path)
        print(f"{folder} has finished.")

    print("Complete!")
```

similarity.py

```python
import os
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from torchvision.models import inception_v3
from PIL import Image
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt

# 1. Create InceptionV3 feature extractor by removing the final classification layer
class InceptionV3FeatureExtractor(nn.Module):
    def __init__(self):
        super(InceptionV3FeatureExtractor, self).__init__()
        # Set aux_logits to True as required by the pretrained model
        inception = inception_v3(pretrained=True, aux_logits=True)
        # Use selected layers up to the final AdaptiveAvgPool2d layer
        self.features = nn.Sequential(
            inception.Conv2d_1a_3x3,
            inception.Conv2d_2a_3x3,
            inception.Conv2d_2b_3x3,
            nn.MaxPool2d(kernel_size=3, stride=2),
            inception.Conv2d_3b_1x1,
            inception.Conv2d_4a_3x3,
            nn.MaxPool2d(kernel_size=3, stride=2),
            inception.Mixed_5b,
            inception.Mixed_5c,
            inception.Mixed_5d,
            inception.Mixed_6a,
            inception.Mixed_6b,
            inception.Mixed_6c,
            inception.Mixed_6d,
            inception.Mixed_6e,
            nn.AdaptiveAvgPool2d((1, 1))
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)  # Flatten to [batch, feature_dim]
        return x

# 2. Preprocess image: resize, convert to tensor, and normalize
def preprocess_image(image_path, device):
    transform = transforms.Compose([
        transforms.Resize((299, 299)),
        transforms.ToTensor(),  # Convert to float tensor and normalize pixel values to [0,1]
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225]),
    ])
    image = Image.open(image_path).convert('RGB')
    image_tensor = transform(image)
    # Add batch dimension to [1, 3, 299, 299]
    return image_tensor.unsqueeze(0).to(device)

# 3. Extract image feature vectors
def extract_features(model, image_path, device):
    model.eval()
    with torch.no_grad():
        image_tensor = preprocess_image(image_path, device)
        features = model(image_tensor)  # Expected output shape: [1, feature_dim]
    return features.cpu().numpy()  # Keep as 2D array [1, feature_dim]

# 4. Group similar images based on cosine similarity
def group_similar_images(folder, model, device, threshold=0.8):
    # Support webp format as well
    image_files = [f for f in os.listdir(folder) if f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif', '.webp'))]
```

```
    image_files = [f for f in os.listdir(folder) if f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif', '.webp'))]
    features_dict = {}
    for file in image_files:
        file_path = os.path.join(folder, file)
        try:
            feat = extract_features(model, file_path, device)
            features_dict[file] = feat
        except Exception as e:
            print(f"Error reading {file}: {e}")

    groups = []
    used = set()
    files = list(features_dict.keys())
    for i in range(len(files)):
        if files[i] in used:
            continue
        group = [files[i]]
        used.add(files[i])
        for j in range(i+1, len(files)):
            if files[j] in used:
                continue
            # Use cosine similarity since extract_features returns a 2D array
            sim = cosine_similarity(features_dict[files[i]], features_dict[files[j]])[0][0]
            if sim >= threshold:
                group.append(files[j])
                used.add(files[j])
        groups.append(group)
    return groups

# 5. Display grouped images, aligned horizontally with titles below each image
def display_groups(folder, groups):
    for idx, group in enumerate(groups, start=1):
        if len(group) <= 1:
            continue
        fig, axes = plt.subplots(1, len(group), figsize=(2 * len(group), 2))
        if len(group) == 1:
            axes = [axes]
        fig.suptitle(f"Group {idx}")
        for file in group:
            print(file)

        for ax, file in zip(axes, group):
            image_path = os.path.join(folder, file)
            try:
                img = Image.open(image_path).convert('RGB')
                ax.imshow(img)
                ax.set_title(file, fontsize=4, y=-0.12)  # y<0 places title below the image
            except Exception as e:
                print(f"Error reading {file}: {e}")
            ax.axis('off')
        plt.subplots_adjust(wspace=0.15, hspace=0.45)
        plt.show()

if __name__ == '__main__':
    folder_path = 'face_dataset/ame'  # Modify folder path as needed
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = InceptionV3FeatureExtractor().to(device)
    groups = group_similar_images(folder_path, model, device, threshold=0.99)
    display_groups(folder_path, groups)
```

split_data.py

```python
import os
import glob
import random
import pandas as pd

random_seed = 42
random.seed(random_seed)

dataset_dir = "face_dataset"

train_data = []
test_data = []

for class_name in os.listdir(dataset_dir):
    class_dir = os.path.join(dataset_dir, class_name)
    if not os.path.isdir(class_dir):
        continue

    image_files = []
    for ext in ['*.png', '*.jpg', '*.jpeg', '*.bmp']:
        image_files.extend(glob.glob(os.path.join(class_dir, ext)))

    random.shuffle(image_files)

    split_index = int(len(image_files) * 0.8)
    train_images = image_files[:split_index]
    test_images = image_files[split_index:]

    train_data.extend([(img_path, class_name) for img_path in train_images])
    test_data.extend([(img_path, class_name) for img_path in test_images])

df_train = pd.DataFrame(train_data, columns=['path', 'label'])
df_test = pd.DataFrame(test_data, columns=['path', 'label'])

df_train = df_train.sample(frac=1, random_state=random_seed).reset_index(drop=True)
df_test = df_test.sample(frac=1, random_state=random_seed).reset_index(drop=True)

df_train.to_csv('train.csv', index=False)
df_test.to_csv('test.csv', index=False)

print("train.csv and test.csv has been saved.")
```

main.py

```python
import argparse
import pandas as pd
import numpy as np
import os
import random
from features import FEATURE_METHODS
from models import (
    load_features_labels,
    train_adaboost,
    train_kmeans,
    train_resnet,
    predict_resnet,
    predict_traditional,
)
from sklearn.decomposition import PCA
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score


def parse_args():
    parser = argparse.ArgumentParser(
        description="Train and evaluate models based on the specified feature extraction method and model type"
    )
    parser.add_argument(
        "--feature",
        type=str,
        default="hog",
        choices=["hog", "clip", "resnet", "color", "hog_color"],
        help="Select feature extraction method: hog, clip, resnet, color, hog_color (effective only for traditional ML models)",
    )
    parser.add_argument(
        "--model",
        type=str,
        default="adaboost",
        choices=["adaboost", "kmeans", "resnet"],
        help="Select training model: adaboost, kmeans (traditional ML) or resnet (deep learning finetune)",
    )
    parser.add_argument(
        "--train_csv",
        type=str,
        default="train.csv",
        help="Path to training CSV file (used as the full dataset in CV mode)",
    )
    parser.add_argument(
        "--test_csv",
        type=str,
        default="test.csv",
        help="Path to testing CSV file (used in non-CV mode)",
    )
    parser.add_argument(
        "--result_csv",
        type=str,
        default="result.csv",
        help="Path for output prediction results CSV (used in non-CV mode)",
    )
    parser.add_argument(
        "--seed",
        type=int,
        default=42,
        help="Random seed for reproducibility",
    )
    parser.add_argument(
        "--train_subset_ratio",
        type=float,
        default=1.0,
        help="Training subset ratio (0-1] to experiment with varying amounts of data",
    )
    parser.add_argument(
```

```
        "--use_pca",
        type=int,
        default=0,
        help="If greater than 0, apply PCA to reduce traditional model features to the specified number of dimensions",
    )
    parser.add_argument(
        "--use_class_weight",
        action="store_true",
        help="Use class weighting (only applicable for ResNet)",
    )
    parser.add_argument(
        "--augment",
        action="store_true",
        help="Enable data augmentation (only applicable for ResNet; disabled by default)",
    )
    parser.add_argument(
        "--n_folds",
        type=int,
        default=1,
        help="Number of folds for cross-validation (if >1, CV mode is enabled using train_csv as the full dataset)",
    )
    return parser.parse_args()


def set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    try:
        import torch

        torch.manual_seed(seed)
        if torch.cuda.is_available():
            torch.cuda.manual_seed_all(seed)
    except ImportError:
        pass


def save_results(results, result_csv):
    """
    Save the results list (each item: (path, pred_label)) to a CSV file.
    """
    df = pd.DataFrame(results, columns=["path", "pred_label"])
    df.to_csv(result_csv, index=False)
    print(f"Prediction results have been saved to {result_csv}")


def run_traditional_model_cv(feature_func, model_name, df_all, args):
    """
    Traditional ML models use StratifiedKFold CV.
    For each fold, the output filename is determined based on the result_csv parameter.
    For example, if result_csv = XXXX.csv, then the output for fold 1 will be XXXX_fold1.csv.
    """
    X_all, y_all, paths_all = load_features_labels(args.train_csv, feature_func)

    if args.train_subset_ratio < 1.0:
        total = X_all.shape[0]
        idx = np.random.choice(total, int(total * args.train_subset_ratio), replace=False)
        X_all = X_all[idx]
        y_all = np.array(y_all)[idx]
        paths_all = [paths_all[i] for i in idx]
        print(f"Using {len(y_all)} samples (originally {total} samples)")

    if args.use_pca > 0:
        print(f"Using PCA to reduce dimensions to {args.use_pca}")
        pca = PCA(n_components=args.use_pca)
        X_all = pca.fit_transform(X_all)
        print("PCA explained variance ratio:", pca.explained_variance_ratio_)

    skf = StratifiedKFold(n_splits=args.n_folds, shuffle=True, random_state=args.seed)
    fold = 1

    for train_idx, val_idx in skf.split(X_all, y_all):
```

```
    for train_idx, val_idx in skf.split(X_all, y_all):
        print(f"=== Fold {fold} ===")
        X_train, y_train = X_all[train_idx], np.array(y_all)[train_idx]
        X_val, y_val = X_all[val_idx], np.array(y_all)[val_idx]
        paths_val = [paths_all[i] for i in val_idx]

        if model_name == "adaboost":
            model = train_adaboost(X_train, y_train, X_val, y_val)
            preds = model.predict(X_val)
        elif model_name == "kmeans":
            model = train_kmeans(X_train, y_train)
            raw_preds = model.predict(X_val)
            # Map clusters to labels based on majority voting
            from collections import Counter

            unique_clusters = np.unique(raw_preds)
            mapping = {}
            for cluster in unique_clusters:
                indices = np.where(raw_preds == cluster)[0]
                labels_in_cluster = y_val[indices]
                mapping[cluster] = Counter(labels_in_cluster).most_common(1)[0][0]
            preds = [mapping[p] for p in raw_preds]
        else:
            raise ValueError(f"Unknown model: {model_name}")

        acc = accuracy_score(y_val, preds)
        print(f"Fold {fold} Accuracy: {acc}")

        # Determine result filename based on result_csv parameter
        if args.result_csv:
            base_name = os.path.splitext(os.path.basename(args.result_csv))[0]
            result_csv = f"{base_name}_fold{fold}.csv"
        else:
            result_csv = f"result_fold{fold}.csv"

        results = list(zip(paths_val, preds))
        save_results(results, result_csv)
        fold += 1
...
```

main.py

```python
...

def run_resnet_cv(label2idx, args):
    """
    ResNet CV mode: use StratifiedKFold to split train_csv,
    apply train_subset_ratio to each training fold, and save each fold's results.
    For example, if result_csv = XXXX.csv, then each fold is saved as XXXX_fold{i}.csv.
    """
    df_all = pd.read_csv(args.train_csv)
    skf = StratifiedKFold(n_splits=args.n_folds, shuffle=True, random_state=args.seed)
    fold = 1

    for train_idx, val_idx in skf.split(df_all, df_all["label"]):
        print(f"=== Fold {fold} ===")
        train_fold = df_all.iloc[train_idx]
        val_fold = df_all.iloc[val_idx]

        # Subsample training fold if needed
        if args.train_subset_ratio < 1.0:
            total = len(train_fold)
            train_fold = train_fold.sample(frac=args.train_subset_ratio, random_state=args.seed)
            print(f"Fold {fold}: Using {len(train_fold)} training samples (originally {total} samples)")

        temp_train_csv = f"temp_train_fold{fold}.csv"
        temp_val_csv = f"temp_val_fold{fold}.csv"
        train_fold.to_csv(temp_train_csv, index=False)
        val_fold.to_csv(temp_val_csv, index=False)

        labels = train_fold["label"].unique()
        local_label2idx = {label: idx for idx, label in enumerate(labels)}
        model = train_resnet(temp_train_csv, temp_val_csv, local_label2idx,
                             use_class_weight=args.use_class_weight, augment=args.augment)
        results = predict_resnet(model, temp_val_csv, local_label2idx)

        df_val = pd.read_csv(temp_val_csv)
        df_pred = pd.DataFrame(results, columns=["path", "pred_label"])
        df_merged = pd.merge(df_val, df_pred, on="path", how="inner")
        acc = accuracy_score(df_merged["label"], df_merged["pred_label"])
        print(f"Fold {fold} ResNet test accuracy: {acc}")

        if args.result_csv:
            base_name = os.path.splitext(os.path.basename(args.result_csv))[0]
            result_csv = f"{base_name}_fold{fold}.csv"
        else:
            result_csv = f"result_fold{fold}.csv"

        save_results(results, result_csv)
        os.remove(temp_train_csv)
        os.remove(temp_val_csv)
        fold += 1


def run_single_mode(feature_func, label2idx, args):
    """
    Non-CV mode: run traditional ML or ResNet training based on the model parameter.
    """
    if args.model in ["adaboost", "kmeans"]:
```

```
    print(f"Using {args.feature} for feature extraction and training {args.model}...")
    X_train, y_train, _ = load_features_labels(args.train_csv, feature_func)
    X_test, y_test, paths = load_features_labels(args.test_csv, feature_func)

    if args.train_subset_ratio < 1.0:
        total = X_train.shape[0]
        idx = np.random.choice(total, int(total * args.train_subset_ratio), replace=False)
        X_train = X_train[idx]
        y_train = np.array(y_train)[idx]
        print(f"Using {len(y_train)} training samples (originally {total} samples)")

    if args.use_pca > 0:
        print(f"Using PCA to reduce dimensions to {args.use_pca}")
        pca = PCA(n_components=args.use_pca)
        X_train = pca.fit_transform(X_train)
        X_test = pca.transform(X_test)
        print("PCA explained variance ratio:", pca.explained_variance_ratio_)

    if args.model == "adaboost":
        model = train_adaboost(X_train, y_train, X_test, y_test)
        preds = model.predict(X_test)
    elif args.model == "kmeans":
        model = train_kmeans(X_train, y_train)
        raw_preds = model.predict(X_test)
        from collections import Counter

        unique_clusters = np.unique(raw_preds)
        mapping = {}
        for cluster in unique_clusters:
            indices = np.where(raw_preds == cluster)[0]
            labels_in_cluster = np.array(y_test)[indices]
            mapping[cluster] = Counter(labels_in_cluster).most_common(1)[0][0]
        preds = [mapping[p] for p in raw_preds]

    acc = accuracy_score(y_test, preds)
    print("Traditional ML model test accuracy:", acc)
    results = list(zip(paths, preds))
    save_results(results, args.result_csv)

elif args.model == "resnet":
    print("Using deep learning ResNet finetune for training and prediction...")
    train_df = pd.read_csv(args.train_csv)

    # Subsample training data if train_subset_ratio < 1.0
    if args.train_subset_ratio < 1.0:
        total = len(train_df)
        train_df = train_df.sample(frac=args.train_subset_ratio, random_state=args.seed)
        print(f"Using {len(train_df)} training samples (originally {total} samples)")

    labels = train_df["label"].unique()
    local_label2idx = {label: idx for idx, label in enumerate(labels)}

    # Save the subsampled training data temporarily
    temp_train_csv = "temp_train_resnet.csv"
    train_df.to_csv(temp_train_csv, index=False)
    model = train_resnet(temp_train_csv, args.test_csv, local_label2idx,
                         use_class_weight=args.use_class_weight, augment=args.augment)
    results = predict_resnet(model, args.test_csv, local_label2idx)

    # Calculate accuracy using test CSV
```

```
        test_df = pd.read_csv(args.test_csv)
        df_pred = pd.DataFrame(results, columns=["path", "pred_label"])
        df_merged = pd.merge(test_df, df_pred, on="path", how="inner")
        acc = accuracy_score(df_merged["label"], df_merged["pred_label"])
        print("ResNet test accuracy:", acc)
        os.remove(temp_train_csv)
        save_results(results, args.result_csv)

    else:
        raise ValueError(f"Unknown model selection: {args.model}")


def main():
    args = parse_args()
    set_seed(args.seed)

    # For non-CV mode, set the feature extraction method (only applicable to traditional ML models)
    feature_func = FEATURE_METHODS[args.feature]

    if args.n_folds > 1:
        print(f"Enabling {args.n_folds}-fold cross validation using {args.train_csv} as the full dataset.")
        df_all = pd.read_csv(args.train_csv)
        if args.model in ["adaboost", "kmeans"]:
            run_traditional_model_cv(feature_func, args.model, df_all, args)
        elif args.model == "resnet":
            run_resnet_cv({}, args)  # Label mapping will be created within each fold
        else:
            raise ValueError(f"Unknown model selection: {args.model}")
    else:
        # Non-CV mode
        run_single_mode(feature_func, {}, args)


if __name__ == "__main__":
    main()
```

models.py

```python
import numpy as np
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score

def load_features_labels(csv_file, feature_extraction_func):
    """
    Read a CSV file (which must contain "path" and "label" columns), extract features
    using the provided feature_extraction_func for each row, and return (X, y, paths).
    """
    df = pd.read_csv(csv_file)
    X, y, paths = [], [], []
    for idx, row in df.iterrows():
        feat = feature_extraction_func(row["path"])
        if feat is not None:
            X.append(feat)
            y.append(row["label"])
            paths.append(row["path"])
    return np.array(X), np.array(y), paths

def train_adaboost(X_train, y_train, X_test, y_test):
    """
    Train an AdaBoostClassifier and print the test set accuracy.
    """
    clf = AdaBoostClassifier(n_estimators=50, random_state=42)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print("AdaBoost test accuracy:", acc)
    return clf

def train_kmeans(X_train, y_train):
    """
    Train a KMeans clustering model. The number of clusters is set to the number
    of unique labels in the training data.
    """
    num_clusters = len(np.unique(y_train))
    print(f"KMeans number of clusters: {num_clusters}")
    kmeans = KMeans(n_clusters=num_clusters, random_state=42)
    kmeans.fit(X_train)
    print("KMeans clustering completed")
    return kmeans

# ----------------------------
# Deep Learning: ResNet Finetuning (with progress bars, class weighting, and data augmentation options)
# ----------------------------
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.models as models
from torch.utils.data import Dataset, DataLoader
from PIL import Image
from tqdm import tqdm
```

```
from sklearn.utils.class_weight import compute_class_weight

class ImageDataset(Dataset):
    def __init__(self, csv_file, transform=None, label2idx=None):
        self.data = pd.read_csv(csv_file)
        self.transform = transform
        if label2idx is None:
            labels = self.data["label"].unique()
            self.label2idx = {label: idx for idx, label in enumerate(labels)}
        else:
            self.label2idx = label2idx

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        row = self.data.iloc[idx]
        image = Image.open(row["path"]).convert("RGB")
        if self.transform:
            image = self.transform(image)
        label = self.label2idx[row["label"]]
        return image, label
...
```

models.py

```python
...

def train_resnet(train_csv, test_csv, label2idx, num_epochs=5, lr=1e-4, batch_size=32,
                 num_workers=4, use_class_weight=False, augment=True):
    """
    Finetune a pretrained ResNet18 model and report test set accuracy.
    Displays progress bars for training and evaluation. Adjust settings with
    use_class_weight and augment parameters.
    """
    device = "cuda" if torch.cuda.is_available() else "cpu"

    # Define train transforms based on augmentation option
    if augment:
        train_transform = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
        ])
    else:
        train_transform = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
        ])
    test_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])

    train_dataset = ImageDataset(train_csv, transform=train_transform, label2idx=label2idx)
    test_dataset = ImageDataset(test_csv, transform=test_transform, label2idx=label2idx)

    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers)
    test_loader  = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_workers=num_workers)

    # Load a pretrained ResNet18 and replace the final fully connected layer
    model = models.resnet18(pretrained=True)
    num_features = model.fc.in_features
    model.fc = nn.Linear(num_features, len(label2idx))
    model = model.to(device)

    # Set up loss function with optional class weighting
    if use_class_weight:
        df_train = pd.read_csv(train_csv)
        classes = np.array(list(label2idx.keys()))
        class_weights = compute_class_weight('balanced', classes=classes, y=df_train['label'])
        class_weights_tensor = torch.tensor(class_weights, dtype=torch.float).to(device)
        criterion = nn.CrossEntropyLoss(weight=class_weights_tensor)
        print("Using class weighting:", class_weights)
    else:
        criterion = nn.CrossEntropyLoss()

    optimizer = optim.Adam(model.parameters(), lr=lr)

    print("Starting ResNet finetuning...")
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        pbar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}", leave=False)
        for inputs, labels_batch in pbar:
```

```
            inputs = inputs.to(device)
            labels_batch = labels_batch.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels_batch)
            loss.backward()
            optimizer.step()
            running_loss += loss.item() * inputs.size(0)
            pbar.set_postfix(loss=f"{loss.item():.4f}")
        epoch_loss = running_loss / len(train_dataset)
        print(f"Epoch {epoch+1}/{num_epochs}, Loss: {epoch_loss:.4f}")

    # Evaluation phase
    model.eval()
    correct = 0
    total = 0
    for inputs, labels_batch in tqdm(test_loader, desc="Evaluating", leave=False):
        inputs = inputs.to(device)
        labels_batch = labels_batch.to(device)
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        total += labels_batch.size(0)
        correct += (preds == labels_batch).sum().item()
    acc = correct / total
    print("ResNet test accuracy:", acc)
    return model

def predict_resnet(model, test_csv, label2idx, batch_size=32, num_workers=4):
    """
    Use the finetuned ResNet model to predict images listed in test_csv.
    Returns a list of (path, pred_label) tuples. The predicted labels are the original class names.
    """
    device = "cuda" if torch.cuda.is_available() else "cpu"
    inv_label_map = {v: k for k, v in label2idx.items()}
    test_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])
    test_dataset = ImageDataset(test_csv, transform=test_transform, label2idx=label2idx)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_workers=num_workers)
    results = []
    model.eval()
    pbar = tqdm(test_loader, desc="Predicting", leave=False)
    all_paths = pd.read_csv(test_csv)["path"].tolist()
    start_idx = 0
    for inputs, _ in pbar:
        inputs = inputs.to(device)
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        batch_size_actual = len(preds)
        batch_paths = all_paths[start_idx:start_idx + batch_size_actual]
        start_idx += batch_size_actual
        for path, pred in zip(batch_paths, preds.cpu().numpy()):
            results.append((path, inv_label_map[pred]))
    return results

def predict_traditional(model, X_test, paths, y_test, method="adaboost"):
    """
    Make predictions with a traditional ML model.
    For KMeans, use a simple mapping based on the assumption that cluster labels roughly correspond to the original labels.
    Returns (preds, accuracy).
    """
    if method == "adaboost":
        preds = model.predict(X_test)
    elif method == "kmeans":
        raw_preds = model.predict(X_test)
        # Build a mapping from clusters to labels. Note: This is a simple example that assumes
```

```
        # build a mapping from clusters to labels. Note: this is a simple example that assumes
        # the cluster label roughly corresponds to the original label. For a robust evaluation,
        # supervised metrics should be used.
        from collections import Counter
        unique_clusters = np.unique(raw_preds)
        mapping = {cluster: str(cluster) for cluster in unique_clusters}
        preds = [mapping[p] for p in raw_preds]
    else:
        preds = model.predict(X_test)
    acc = accuracy_score(y_test, preds)
    print(f"{method} test accuracy:", acc)
    return preds, acc
```

features.py

```python
import os
import numpy as np
from PIL import Image
import random

# HOG feature extraction (using skimage)
from skimage.feature import hog
from skimage import color

def extract_hog_features(image_path):
    """
    Read an image, convert it to grayscale, and extract HOG features.
    """
    try:
        norm_path = os.path.normpath(image_path)
        image = Image.open(norm_path).convert("RGB")
        image_np = np.array(image)
        gray = color.rgb2gray(image_np)
        features = hog(
            gray,
            pixels_per_cell=(16, 16),
            cells_per_block=(2, 2),
            feature_vector=True
        )
        return features
    except Exception as e:
        print(f"HOG extract error ({image_path}): {e}")
        return None

# CLIP feature extraction
import torch
import clip

# Load CLIP model and preprocessor (initialized at the first call)
_device = "cuda" if torch.cuda.is_available() else "cpu"
_clip_model, _clip_preprocess = clip.load("ViT-B/32", device=_device)

def extract_clip_features(image_path, model=_clip_model, preprocess=_clip_preprocess):
    """
    Use CLIP's image encoder to convert an image into a feature vector.
    """
    try:
        norm_path = os.path.normpath(image_path)
        image = Image.open(norm_path).convert("RGB")
        image_input = preprocess(image).unsqueeze(0).to(_device)
        with torch.no_grad():
```

```
            features = model.encode_image(image_input)
        return features.cpu().numpy().flatten()
    except Exception as e:
        print(f"CLIP extract error ({image_path}): {e}")
        return None

# Color-based non-deep learning feature extraction: Color Histogram
def extract_color_histogram(image_path, bins=8):
    """
    Calculate histograms for the RGB channels (each with 8 bins), normalize them,
    and concatenate them into a single feature vector of length 24.
    """
    try:
        norm_path = os.path.normpath(image_path)
        image = Image.open(norm_path).convert("RGB")
        image_np = np.array(image)
        hist_r, _ = np.histogram(image_np[:, :, 0], bins=bins, range=(0, 256))
        hist_g, _ = np.histogram(image_np[:, :, 1], bins=bins, range=(0, 256))
        hist_b, _ = np.histogram(image_np[:, :, 2], bins=bins, range=(0, 256))
        hist = np.concatenate([hist_r, hist_g, hist_b]).astype(float)
        hist /= (hist.sum() + 1e-7)
        return hist
    except Exception as e:
        print(f"Color histogram extract error ({image_path}): {e}")
        return None

# Dictionary mapping feature extraction method names to functions
FEATURE_METHODS = {
    "hog": extract_hog_features,
    "clip": extract_clip_features,
    "color": extract_color_histogram
}
```

stats.py

```python
import argparse
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report

def parse_args():
    parser = argparse.ArgumentParser(
        description="Compare and visualize test.csv with the prediction result CSV."
    )
    parser.add_argument(
        '--test_csv',
        type=str,
        default='test.csv',
        help="Path to the original test CSV (must contain 'path' and 'label' columns)."
    )
    parser.add_argument(
        '--result_csv',
        type=str,
        default='result.csv',
        help="Path to the prediction results CSV (must contain 'path' and 'pred_label' columns)."
    )
    parser.add_argument(
        '--out_fig',
        type=str,
        default='confusion_matrix.png',
        help="Output path for the confusion matrix image."
    )
    return parser.parse_args()

def main():
    args = parse_args()

    # Read the test data and prediction results
    df_test = pd.read_csv(args.test_csv)
    df_result = pd.read_csv(args.result_csv)

    # Merge data based on the 'path' column
    df = pd.merge(df_test, df_result, on="path", how="inner")

    # Check if the merged DataFrame contains the required 'label' and 'pred_label' columns
    if "label" not in df.columns or "pred_label" not in df.columns:
        print("CSV column error. Please ensure test.csv contains 'label' and result.csv contains 'pred_label'.")
        return

    y_true = df["label"]
    y_pred = df["pred_label"]

    # Compute the confusion matrix
    labels = sorted(y_true.unique())
    cm = confusion_matrix(y_true, y_pred, labels=labels)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=labels, yticklabels=labels)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("Confusion Matrix")
    plt.tight_layout()
    plt.savefig(args.out_fig)
```

```
    plt.savefig(args.out_fig)
    plt.show()

    # Print the detailed classification report
    report = classification_report(y_true, y_pred, labels=labels)
    print("Classification Report:\n", report)

if __name__ == "__main__":
    main()
```